Protein significance analysis of mass spectrometry-based proteomics experiments with R and MSstats (v3.6)

Meena Choi, Tsung-Heng Tsai & Erik Verschueren

November 9, 2016

Contents

1. Statistical relative protein quantification: SRM, DDA and DIA experiments **1**
1.1 Applicability . . . 1
1.2 Statistical functionalities . . . 2
1.3 Interoperability with existing computational tools . . . 2
1.4 Availability . . . 2
1.5 Overview of the functionalities . . . 3
1.6 Troubleshooting . . . 4

2. Allowable data formats **4**
2.1 SRM with stable isotope labeled reference peptides . . . 4
2.2 Label-free DDA . . . 7
2.2 Label-free DIA . . . 8

3. Prerequisites and setting for MSstats analysis **8**

4. DDA analysis with MSstats **9**
4.1 General workflow for DDA . . . 9
4.2 Suggested workflow with Skyline output for DDA . . . 29
4.3 Suggested workflow with MaxQuant output for DDA . . . 32
4.4 Suggested workflow with Progenesis output for DDA . . . 34

Reference **36**

---

1. Statistical relative protein quantification: SRM, DDA and DIA experiments

`MSstats` is an open-source R-based package for statistical relative quantification of peptides and proteins in mass spectrometry-based proteomic experiments. This document describes `MSstats`, the most recent version of the package, and its use through the command line.

1.1 Applicability

`MSstats` version 3.0 and above is applicable to multiple types of sample preparation, including label-free workflows, workflows that use stable isotope labeled reference proteins and peptides, and workflows that use fractionation. It is applicable to targeted Selected Reaction Monitoring (SRM), Data-Dependent Acquisition (DDA or shotgun), and Data-Independent Acquisition (DIA or SWATH-MS). It is applicable to experiments that make arbitrary complex comparisons of experimental conditions or times.

`MSstats` is currently not applicable to experiments that compare multiple metabolically labeled endogenous samples within a same run. It is not applicable to experiments with iTRAQ labeling. These experiments will be supported in the future.

1.2 Statistical functionalities

`MSstats` version 3.0 and above performs three analysis steps. The first step, *data processing, visualization, and run-level summarization*, transforms, normalizes and summarizes the intensities of the peaks per MS run and per protein, and generates workflow-specific and customizable numeric summaries for data visualization and quality control.

The second step, *statistical modeling and inference*, automatically detects the experimental design (e.g. group comparison, paired design or time course, presence of labeled reference peptides or proteins) from the data. It then reflects the experimental design and the type of spectral acquisition strategy, and fits an appropriate linear mixed model by means of `lm` and `lmer` functionalities in R. The model is used to detect differentially abundant proteins or peptides, or to summarize the protein or peptide abundance in a single biological replicate or condition (that can be used, e.g. as input to clustering or classification).

The third step, *statistical experimental design*, views the dataset being analyzed as a pilot study of a future experiment, utilizes the variance components of the current dataset, and calculates the minimal number of replicates necessary in the future experiment to achieve a pre-specified statistical power.

1.3 Interoperability with existing computational tools

`MSstats` takes as input data in a tabular .csv format, which can be generated by any spectral processing tool such as Skyline (MacLean et al. 2010), MaxQuant (Jürgen Cox and Mann 2008), Progenesis QI(Nonlinear dynamics/Waters), Proteome Discoverer (Thermo Scientific) MultiQuant(Applied Biosystems), OpenMS (Sturm et al. 2008), SuperHirn (Mueller et al. 2007), OpenSWATH (Röst et al. 2014) or Spectronaut(Biognosys). The functions to convert the required format from several processing tools are available from `MSstats` v3.6. Details are in the section below.

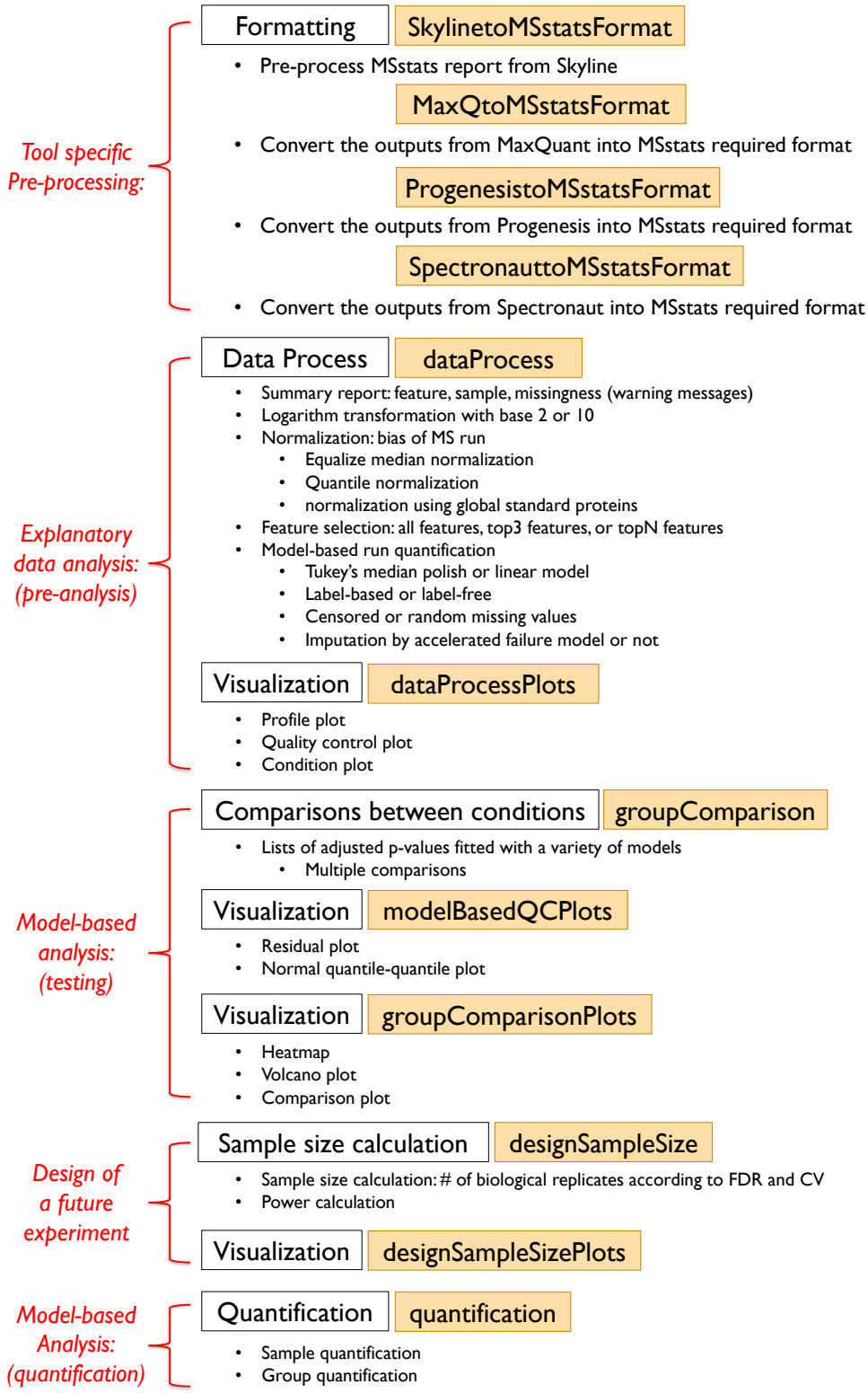
For statistics experts, `MSstats` 3.0 and above satisfies the interoperability requirements of Bioconductor, and takes as input data in the `MSnSet` format (Gatto and Lilley 2012). The command line-based workflow is partitioned into a series of independent steps, that facilitate the development and testing of alternative statistical approaches. It complies with the maintenance and documentation requirements of Bioconductor.

Finally, `MSstats` 3.0 and above is available as an external tool within Skyline. The external tool support within Skyline manages `MSstats` installation, point-and-click execution, parameter collection in Windows forms and output display. Skyline manages the annotations of the experimental design, and the processing of raw data. It outputs a custom report, that is fed as a single stream input into `MSstats`. This design buffers proteomics users from the details of the R implementation, while enabling rigorous statistical modeling.

1.4 Availability

`MSstats` is available under the Artistic-2.0 license at msstats.org. `MSstats` as an external tool for Skyline is available at http://proteome.gs.washington.edu/software/Skyline/tools.html. `MSstats` is now also available in Bioconductor. The most recent version of the package is available at msstats.org or MSstats GitHub. We suggest to use that if possible. The versioning of the main package is updated several times a year, to synchronise with the Bioconductor release.

1.5 Overview of the functionalities



1.6 Troubleshooting

To help troubleshoot potential problems with installation or functionalities of `MSstats`, a progress report is generated in a separate log file *msstats.log*. The file includes information on the R session (R version, loaded software libraries), options selected by the user, checks of successful completion of intermediate analysis steps, and warning messages. If the analysis produces an error, the file contains suggestions for possible reasons for the errors. If a file with this name already exists in working directory, a suffix with a number will be appended to the file name. In this way a record of all the analyses is kept. Please see the file `KnownIssues-Skyline-MSstatsV3.6.pdf` on the "Installation" section of "MSstats" page in `msstats.org` for a list of known issues and possible solutions for installation problem of MSstats external tool in Skyline

---

2. Allowable data formats

2.1 SRM with stable isotope labeled reference peptides

2.1.1 10-column format

`MSstats` performs statistical analysis steps, that follow peak identification and quantitation. Therefore, input to MSstats is the output of other software tools (such as Skyline or MultiQuant) that read raw spectral files and identify and quantify spectral peaks. The preferred structure of data for use in `MSstats` is a .csv file in a "long" format with 10 columns representing the following variables: `ProteinName`, `PeptideSequence`, `PrecursorCharge`, `FragmentIon`, `ProductCharge`, `IsotopeLabelType`, `Condition`, `BioReplicate`, `Run`, `Intensity`. The variable names are fixed, but are case-insensitive.

(a) `ProteinName`: This column needs information about Protein id. Statistical analysis will be done separately for each unique label in this column. For peptide-level modeling and analysis, use peptide id in this column.

(b)-(e) `PeptideSequence, PrecursorCharge, FragmentIon, ProductCharge`: The combination of these 4 columns defines a *feature* of a protein (in SRM experiments, it is a transition that is identified and quantified across runs). If the information for one or several of these columns is not available, please do not discard these columns but use a single fixed value across the entire dataset. For example, if the original raw data does not contain the information of `ProductCharge`, assign the value 0 to the entries in the column `ProductCharge` for the entire dataset. If the peptide sequences should be distinguished based on post-translational modifications, this column can be renamed to `PeptideModifiedSequence`. For example, this allows us to use the `PeptideModifiedSequence` column from the Skyline report.

(f) `IsotopeLabelType`: This column indicates whether this measurement is based on the endogenous peptides (use "L") or labeled reference peptides (use "H").

(g) `Condition`: For group comparison experiments, this column indicates groups of interest (such as "Disease" or "Control"). For time-course experiments, this column indicates time points (such as "T1", "T2", etc). If the experimental design contains both distinct groups of subjects and multiple time points per subject, this column should indicate a combination of these values (such as "Disease_T1", "Disease_T2", "Control_T1", "Control_T2", etc.).

(h) `BioReplicate`: This column should contain a unique identifier for each biological replicate in the experiment. For example, in a clinical proteomic investigation this should be a unique patient id. Patients from distinct groups should have distinct ids. `MSstats` does not require the presence of technical replicates in the experiment. If the technical replicates are present, all samples or runs from a same biological replicate should have a same id. `MSstats` automatically detects the presence of technical replicates and accounts for them in the model-based analysis.

(i) `Run`: This column contains the identifier of a mass spectrometry run. Each mass spectrometry run should have a unique identifier, regardless of the origin of the biological sample. In SRM experiments, if all the transitions of a biological or a technical replicate are split into multiple "methods" due to the technical limitations, each method should have a separate identifier. When processed by Skyline, distinct values of runs correspond to distinct input file names. It is possible to use the actual input file names as values in the column `Run`.

(j) `Intensity`: This column should contain the quantified signal of a feature in a run without any transformation (in particular, no logarithm transform). The signals can be quantified as the peak height or the peak of area under curve. Any other quantitative representation of abundance can also be used.

An example of an acceptable input dataset is shown below. This example dataset is from an SRM experiment with stable isotope labeled reference peptides. The dataset is stored in a .csv file in a "long" format. Each row corresponds to a single intensity. More details on assigning the values of `Condition, BioReplicate` and `Run`, depending on the structure of the experimental design, are given below.

	A	B	C	D	E	F	G	H	I	J
1	ProteinName	PeptideSequence	PrecursorCharge	FragmentIon	ProductCharge	IsotopeLabelType	Condition	BioReplicate	Run	Intensity
2	ACEA	EILGHEIFFDWELP	3	y3	0	H	1	RepIA	1	66472.3847
3	ACEA	EILGHEIFFDWELP	3	y3	0	L	1	RepIA	1	5764.16228
4	ACEA	EILGHEIFFDWELP	3	y4	0	H	1	RepIA	1	101005.166
5	ACEA	EILGHEIFFDWELP	3	y4	0	L	1	RepIA	1	61.65238
6	ACEA	EILGHEIFFDWELP	3	y5	0	H	1	RepIA	1	90055.4993
7	ACEA	EILGHEIFFDWELP	3	y5	0	L	1	RepIA	1	472.691803
8	ACEA	TDSEAATLISSTID	2	y10	0	H	1	RepIA	1	43506.5425
9	ACEA	TDSEAATLISSTID	2	y10	0	L	1	RepIA	1	217.203553
10	ACEA	TDSEAATLISSTID	2	y7	0	H	1	RepIA	1	68023.0377
11	ACEA	TDSEAATLISSTID	2	y7	0	L	1	RepIA	1	725.284308
12	ACEA	TDSEAATLISSTID	2	y8	0	H	1	RepIA	1	68276.0489
13	ACEA	TDSEAATLISSTID	2	y8	0	L	1	RepIA	1	243.658527

2.1.2 Assigning the values of `Condition, BioReplicate` and `Run`

The values of `Condition, BioReplicate, Run` depend on the design of the specific experiment.

1) Group comparison

In a group comparison design, the conditions (e.g., disease states) are profiled across **non-overlapping sets of biological replicates** (i.e., subjects). In this example there are 2 conditions, Disease and Control (in general the number of conditions can vary). There are 3 subjects (i.e., biological replicates) per condition (in general an equal number of replicates per condition is not required). Each subject has 2 technical replicate runs (in general technical replicates are not required, and their number per sample may vary). Overall, in this example there are $2 \times 3 \times 2 = 12$ mass spectrometry runs.

Table below shows the values of the columns `Condition, BioReplicate` and `Run` for this situation. It is important to note two things. First, the order of subjects and conditions in the experiment should be randomized, and run id does not need to represent the order of spectral acquisition. Second, the values of the columns are repeated for every quantified transition. For example, if in each run the experiment quantifies 50 endogenous transitions and 50 labeled reference counterparts, then the input file has $12 \times 50 \times 2 = 1200$ lines. When a feature intensity is missing in a run, the data structure should contain a separate row for each missing value. The rows should include all the information (from `ProteinName` to `Run`), and indicate missing intensities with `NA`.

`Condition`	`BioReplicate`	`Run`
Disease	Subject1	1
Disease	Subject1	2
Disease	Subject2	3

Condition	BioReplicate	Run
Disease	Subject2	4
Disease	Subject3	5
Disease	Subject3	6
Control	Subject4	7
Control	Subject4	8
Control	Subject5	9
Control	Subject5	10
Control	Subject6	11
Control	Subject6	12

2) Time course

The important feature of a time course experimental design is that **a same subject (i.e., biological replicate) is repetitively measured across multiple time points**. In this example there are 2 time points, Time1 and Time2 (in general the number of times can vary). There are 4 subjects (i.e., biological replicates) measured across times (in general an equal number of times per replicate is not required). There are no technical replicates (in general the number of technical replicates per sample may vary). Overall, in this example there are $2 \times 4 \times 1 = 8$ mass spectrometry runs.

Table below shows the values of the columns `Condition, BioReplicate` and `Run` for this situation. Comments on the order of the runs, on the number of lines in the input data structure, and on the handling of missing peak intensities are as in the group comparison design.

Condition	BioReplicate	Run
Time1	Subject1	1
Time2	Subject1	2
Time1	Subject2	3
Time2	Subject2	4
Time1	Subject3	5
Time2	Subject3	6
Time1	Subject4	7
Time2	Subject4	8

3) Paired design

Another frequently used experimental design is a *paired design*, where measurements from multiple conditions (such as healthy biopsy and disease biopsy) are taken from a same subject. The statistical model for this experimental design is the same as in the time course experiment, however the values in the columns of the input data may have a different appearence. In this example there are 2 subjects, PatientA and PatientB (in general the number of patients can vary). There are two conditions per subject, BiopsyHealthy and BiopsyTumor (in general the number of conditions per subject can exceed two). In this example there are 3 technical replicates of each type (in this example, the technical replicates are biopsies; in general these can also be replicate sample preparations or replicate mass spectrometry runs). Overall, in this example there are $2 \times 2 \times 3 = 12$ mass spectrometry runs.

Table below shows the values of the columns `Condition, BioReplicate` and `Run` for this situation. Comments on the order of the runs, on the number of lines in the input data structure, and on the handling of missing peak intensities are as in the group comparison design.

Condition	BioReplicate	Run
BiopsyHealthy	PatientA	1

Condition	BioReplicate	Run
BiopsyHealthy	PatientA	2
BiopsyHealthy	PatientA	3
BiopsyTumor	PatientA	4
BiopsyTumor	PatientA	5
BiopsyTumor	PatientA	6
BiopsyHealthy	PatientB	7
BiopsyHealthy	PatientB	8
BiopsyHealthy	PatientB	9
BiopsyTumor	PatientB	10
BiopsyTumor	PatientB	11
BiopsyTumor	PatientB	12

2.1.3 `MSnSet` format

`MSstats` also allows data to be in the format of `MSnSet`, which is consistent with the requirements of Bioconductor. The `MSnSet` format has several components, of which the most commonly accessed are `assayData`, `phenoData`, and `featureData`. `assayData` is a matrix of intensities, where each row corresponds a transition, and the columns correspond to sample ids. `phenoData` contains columns that describe the biological samples, conditions in the experiment. `featureData` contains columns describing the peptide features, such as the name or id of the underlying protein and information of features.

If the data are stored in the format `expressionSet`, information for group labels is required. If more than one variable is listed in the argument `group`, then a concatenated variable is created based on all of the specified `group` variables. The remaining information (peptide feature ids, biological replicate ids, and abundance) can be extracted from the rows and columns of `featureData` and `phenoData`, or assigned by the users based on the experimental design.

2.2 Label-free DDA

For label-free DDA experiments the required input is the 10-column format, the same as described in section 2.1 for SRM experiments. In DDA experiments spectral features are defined as peptide ions, which are identified and quantified across runs. Since for label-free DDA experiments some of the columns `PeptideSequence`, `PrecursorCharge`, `FragmentIon`, and `ProductCharge` are not relevant, these columns will have a constant fixed value (such as `NA`) across the entire dataset. Furthermore, the column `IsotopeLabelType` will be set to "L" for the entire dataset.

ProteinName	PeptideSequence	PrecursorCharge	FragmentIon	ProductCharge	IsotopeLabelType	Condition	BioReplicate	Run	Intensity
bovine	S.PVDIDTK	5	NA	NA	L	C1	1	1	2636791.5
bovine	S.PVDIDTK	5	NA	NA	L	C1	1	2	1992418.5
bovine	S.PVDIDTK	5	NA	NA	L	C1	1	3	1982146.38
bovine	S.PVDIDTK	5	NA	NA	L	C2	1	4	5019594
bovine	S.PVDIDTK	5	NA	NA	L	C2	1	5	4560467.5
bovine	S.PVDIDTK	5	NA	NA	L	C2	1	6	3627848.75
bovine	S.PVDIDTK	5	NA	NA	L	C5	1	13	145511.83
bovine	S.PVDIDTK	5	NA	NA	L	C5	1	14	291829.69
bovine	S.PVDIDTK	5	NA	NA	L	C6	1	16	786667.38
bovine	S.PVDIDTK	5	NA	NA	L	C6	1	17	705295.31
bovine	S.PVDIDTK	5	NA	NA	L	C6	1	18	453448.78
bovine	S.PVDIDTK	5	NA	NA	L	C3	1	7	NA

2.2 Label-free DIA

For label-free DIA experiments, the required input is the 10-column format, the same as described in section 2.1 for SRM experiments. The values of the required columns can be extracted from the output of signal processing software such as Skyline or OpenSWATH. By default, the combination of the values in the columns `PeptideSequence, PrecursorCharge, FragmentIon, ProductCharge` uniquely identifies each spectral feature (i.e., a fragment ion identified and quantified across multiple runs). If the signal processing software does not provide the information on some of these columns but provides a unique feature identifier, it is possible to use this unique identifier instead of one of these columns. Furthermore, the column `IsotopeLabelType` is set to "L" for the entire dataset.

An example dataset is shown below. In this example, feature id generated by OpenSWATH is used instead of `ProductCharge` to uniquely characterize each feature.

ProteinName	PeptideSequence	PrecursorCharge	FragmentIon	ProductCharge	IsotopeLabelType	Condition	BioReplicate	Run	Intensity
350748	TPPAAVLLK	2	y7	109401	L	2	1	3	257486
350748	TPPAAVLLK	2	y7	109401	L	2	2	4	141159
350748	TPPAAVLLK	2	y7	109401	L	1	1	1	452908
350748	TPPAAVLLK	2	y7	109401	L	1	2	2	348222
515084	NIC[160]VNAIAPGFIESDMTGVLPEK	3	y3	7717	L	2	1	3	12753
515084	NIC[160]VNAIAPGFIESDMTGVLPEK	3	y3	7717	L	2	2	4	12857
515084	NIC[160]VNAIAPGFIESDMTGVLPEK	3	y3	7717	L	1	1	1	89652
515084	NIC[160]VNAIAPGFIESDMTGVLPEK	3	y3	7717	L	1	2	2	76724
515084	MVNEAIESLGSIDVLVNNAGITNDK	3	y9	57971	L	2	1	3	2052
515084	MVNEAIESLGSIDVLVNNAGITNDK	3	y9	57971	L	2	2	4	1050
515084	MVNEAIESLGSIDVLVNNAGITNDK	3	y9	57971	L	1	1	1	10772
515084	MVNEAIESLGSIDVLVNNAGITNDK	3	y9	57971	L	1	2	2	10516

---

3. Prerequisites and setting for MSstats analysis

`MSstats` is an R-based package. It is assumed that you already have R installed. You can install `MSstats` from Bioconductor:

```
source("https://bioconductor.org/biocLite.R")
biocLite("MSstats")
```

Alternatively, you can downlaod the package from the MSstats installation page and install it as follows:

```
install.packages(pkgs = 'MSstats_3.6.3.tar.gz', repos = NULL, type = 'source')
```

Once you have the package installed, load `MSstats` into an R session and verify that you have the correct version (3.6). Note that in order to use `MSstats`, the package needs to be loaded every time you restart R.

```
library('MSstats', warn.conflicts = F, quietly = T, verbose = F)
?MSstats
```

Finally, set the working directory to where you saved files. Note that you may have a different path on your computer from the example.

```
setwd('/Users/meenachoi/Dropbox/MSstats_GitHub_document/MSstats_v3.6')
```

You can check your working directory by:

```
getwd()
```

```
## [1] "/Users/meenachoi/Dropbox/MSstats_GitHub_document/MSstats_v3.6"
```

---

4. DDA analysis with MSstats

4.1 General workflow for DDA

This section describes a typical workflow for DDA analysis with `MSstats`. Controlled mixture DDA data will be used for demonstration. This dataset is available as an example data(`DDARawData`) in MSstats. Also the csv file for the same dataset, RawData.DDA.csv, is available in MSstats material GitHub in the folder named 'example dataset/DDA_controlledMixture2009'. It is processed by Superhirn. (original reference link)

4.1.1 Preparing the data for MSstats input

The first step in using the `MSstats` is to format the data as described in Section 2. `DDARawData` is already formatted for `MSstats` input.

```
# Check the first 6 rows in DDARawData
head(DDARawData)
```

```
##   ProteinName PeptideSequence PrecursorCharge FragmentIon ProductCharge
## 1      bovine     S.PVDIDTK_5               5          NA            NA
## 2      bovine     S.PVDIDTK_5               5          NA            NA
## 3      bovine     S.PVDIDTK_5               5          NA            NA
## 4      bovine     S.PVDIDTK_5               5          NA            NA
## 5      bovine     S.PVDIDTK_5               5          NA            NA
## 6      bovine     S.PVDIDTK_5               5          NA            NA
##   IsotopeLabelType Condition BioReplicate Run Intensity
## 1                L        C1            1   1   2636792
## 2                L        C1            1   2   1992418
## 3                L        C1            1   3   1982146
## 4                L        C2            1   4   5019594
## 5                L        C2            1   5   4560468
## 6                L        C2            1   6   3627849
```

4.1.2 Processing the data

Normalizing and summarizing data with dataProcess

After reading the datasets, `MSstats` performs 1) logarithm transformation of `Intensity` column, 2) normalization, 3) feature selection, (all features vs subset of features), 4) imputation for censored missing value, which are below the cutoff and undetectable, 5) run-level summarization.

To get started with this function, visit the help section of `dataProcess` first:

```
?dataProcess
```

Default normalization and summarization options

`dataProcess` provides a variety of options in consideration of different experimental protocols. Default values for all options are our suggestion for general cases. However, the default options may not be appropriate for all possible scenarios. It is important to understand their underlying assumption to avoid misuse. Below is the additional explanation for main options.

- **logTrans** : logarithm transformation with base 2(default) of `Intensity` column.
- **Normalization** :

  - **'equalizeMedians'** : The default option for normalization is `equalizeMedians`, where all the intensities in a run are shifted by a constant, to equalize the median of intensities across runs for label-free experiment. This normalization method is appropriate when we can assume that the majority of proteins do not change across runs. *Be cautious when using the `equalizeMedians` option for a label-free DDA dataset with only a small number of proteins.* For label based experiment, `equalizeMedians` equalizes the median of reference intensities across runs and is generally proper even for a dataset with a small number of proteins.
  - **'globalStandards'** : Instead, if you have a spiked in standard, you may set this to `globalStandards` and define the standard with `nameStandards` option.
  - **'quantile'** : The distribution of all the intensities in each run will become the same across runs for label-free experiment. For label-based experiment, the distribution of all the reference intensities will be become the same across runs and all the endogenous intensities are shifted by a constant corresponding to reference intensities.
  - **FALSE** : No normalization is performed. If you had your own normalization before MSstats, you should use `Normalization=FALSE`.

- **nameStandards** : Only for `Normalization='globalStandards'`, global standard peptide or Protein names, which you can assume that they have the same abundance across MS runs, should be assigned in the vector for this option.

- **featureSubset** :

  - **'all'** : Use all features in the dataset.
  - **'top3'** : Use top 3 features which have highest average of log2(intensity) across runs.
  - **'topN'** : Use top N features which have highest average of log2(intensity) across runs. It needs the input for `n_top_feature`option (ex. `n_top_feature=5` for top 5 features).

- **summaryMethod** : Method for run-level summarization.

  - **'TMP'** : Default. Tukey's median polish (`medpolish` function in stats). Robust parameter estimation method with median across rows and columns.
  - **'linear'** : Linear model (`lm` function). Average-based summarization.

- **MBimpute** : whether model-based imputation will be performed or not. Only for `summaryMethod='TMP'`.

  - **TRUE** : Default. Censored missing values will be imputed by Accelerated Failure Time model. Censored missing values will be determined by other options, `censoredInt` and `maxQuantileforCensored`
  - **FALSE** : No model-based imputation.

- **maxQuantileforCensored** : Maximum quantile for deciding censored missing value. Default is 0.999. If you don't want to apply the threshold of noise intensity in your data, you can use `maxQuantileforCensored=NULL`.

- **censoredInt** : The processing tools report missing values differently. This option is for distinguish which value should be considered as missing, and further whether it is censored or at random.

  - **'NA'** : Default. It assumes that all NAs in `Intensity` column are censored.
  - **'0'** : It assumes that all values between 0 and 1 in `Intensity` column are censored. If there are NAs in `Intensity` with this option, NAs will be considered as random missing.
  - **NULL** : It assumes that all missing values are randomly missing.

- **cutoffCensored** : cutoff value for AFT model. It is only with `censoredInt='NA'` or `censoredInt='0'`. If you have `censoredInt=NULL`, it assumes that there is no censored missing and any imputation will not be performed.

  - **'minFeature'** : cutoff for AFT model will be the minimum value for each feature across runs.
  - **'minRun'** : cutoff for AFT model will be the minimum value for each run across features.
  - **'minFeatureNRun'** : cutoff for AFT model will be the smallest value between minimum value of corresponding feature and minimum value of corresponding run.

A typical label-free DDA dataset may have many missing values and noisy features with outliers. `MSstats` supports several ways to deal with this. The default option for summarization is `TMP` (robust parameter estimation method with median across rows and columns) after imputation by AFT (accelerated failure time model, `MBimpute=TRUE`) based on censored intensity for `NA` (`censoredInt="NA"`) with a cutoff as the minimum value for a feature (`cutoffCensored="minFeature"`).

This process handles missing values through imputation and reduces the influence of the outliers using the `TMP` estimation. Note, however, that those runs with no measurements at all will be removed and not be used for any calculation.

```
# default option
DDA2009.proposed <- dataProcess(raw = DDARawData,
                            normalization = 'equalizeMedians',
                            summaryMethod = 'TMP',
                            censoredInt = "NA", cutoffCensored = "minFeature",
                            MBimpute = TRUE,
                            maxQuantileforCensored=0.999)
```

```
## Log2 intensities under cutoff = 13.456  were considered as censored missing values.

## * Use all features that the dataset origianally has.

##
##   Summary of Features :
##                           count
## # of Protein                 6
## # of Peptides/Protein    11-32
## # of Transitions/Peptide   1-1
##
##   Summary of Samples :
##                             C1 C2 C3 C4 C5 C6
## # of MS runs                3  3  3  3  3  3
## # of Biological Replicates  1  1  1  1  1  1
## # of Technical Replicates   3  3  3  3  3  3

##
##  Summary of Missingness :

##   # transitions are completely missing in one condition: 90

##     -> D.GPLTGTYR_23_23_NA_NA, F.HFHWGSSDDQGSEHTVDR_402_402_NA_NA, G.PLTGTYR_8_8_NA_NA, H.SFNVEYDDSQ

##
##   # run with 75% missing observations: 0

##
##  == Start the summarization per subplot...

## Getting the summarization by Tukey's median polish per subplot for protein  bovine ( 1  of  6 )

## Getting the summarization by Tukey's median polish per subplot for protein  chicken ( 2  of  6 )

## Getting the summarization by Tukey's median polish per subplot for protein  cyc_horse ( 3  of  6 )

## Getting the summarization by Tukey's median polish per subplot for protein  myg_horse ( 4  of  6 )

## Getting the summarization by Tukey's median polish per subplot for protein  rabbit ( 5  of  6 )

## Getting the summarization by Tukey's median polish per subplot for protein  yeast ( 6  of  6 )

##
##  == the summarization per subplot is done.
```

Output of dataProcess

Output of the dataProcess function contains the processed and run-level summarized data as well as relevant information for the summarization step.

```
# output of dataProcess includes several data types.
names(DDA2009.proposed)
```

```
## [1] "ProcessedData"     "RunlevelData"      "SummaryMethod"
## [4] "ModelQC"           "PredictBySurvival"
```

```
# the data after reformatting and normalization
head(DDA2009.proposed$ProcessedData)
```

```
##      PROTEIN                       PEPTIDE TRANSITION
## 55    bovine             D.GPLTGTYR_23_23      NA_NA
## 937   bovine F.HFHWGSSDDQGSEHTVDR_402_402      NA_NA
## 1628  bovine   F.HWGSSDDQGSEHTVDR_229_229      NA_NA
## 19    bovine                G.PLTGTYR_8_8      NA_NA
## 1081  bovine       H.SFNVEYDDSQDK_465_465      NA_NA
## 469   bovine        K.AVVQDPALKPL_156_156      NA_NA
##                              FEATURE LABEL GROUP_ORIGINAL
## 55               D.GPLTGTYR_23_23_NA_NA     L             C1
## 937  F.HFHWGSSDDQGSEHTVDR_402_402_NA_NA     L             C1
## 1628   F.HWGSSDDQGSEHTVDR_229_229_NA_NA     L             C1
## 19                  G.PLTGTYR_8_8_NA_NA     L             C1
## 1081       H.SFNVEYDDSQDK_465_465_NA_NA     L             C1
## 469         K.AVVQDPALKPL_156_156_NA_NA     L             C1
##      SUBJECT_ORIGINAL RUN GROUP SUBJECT SUBJECT_NESTED INTENSITY ABUNDANCE
## 55                  1   1     1       1            1.1  757400.1  19.83052
## 937                 1   1     1       1            1.1 2087125.8  21.29291
## 1628                1   1     1       1            1.1 1485145.8  20.80200
## 19                  1   1     1       1            1.1 4986404.0  22.54939
## 1081                1   1     1       1            1.1 2488141.2  21.54646
## 469                 1   1     1       1            1.1 7519322.0  23.14200
##      METHOD originalRUN censored
## 55        1           1    FALSE
## 937       1           1    FALSE
## 1628      1           1    FALSE
## 19        1           1    FALSE
## 1081      1           1    FALSE
## 469       1           1    FALSE
```

DDA2009.TMP$ProcessedData has the data after normalization and deciding the data-specific threshold for censored missing value. There are several new columns in the datasets. Also dataset is reformated. Intensity column includes original intensities values in the input of dataProcess. ABUNDANCE column contains the *log2 transformed and normalized intensities and it will used for run-level summarization.* censored column has the decision about censored missing or not, based on censoredInt and maxQuantileforCensored options. ABUNDANCE with TRUE value in censored column will be considered as censored missing and imputed with MBimpute=TRUE option. Censored missing will be distinguished in Profile plot from dataProcessPlots.

```
# run-level summarized data
head(DDA2009.proposed$RunlevelData)
```

```
##   RUN Protein LogIntensities NumMeasuredFeature MissingPercentage
## 1   1  bovine       21.28437                 14        0.00000000
## 2   2  bovine       20.85653                 14        0.00000000
```

```
## 3   3  bovine       20.67521                13        0.07142857
## 4   4  bovine       21.60443                13        0.07142857
## 5   5  bovine       21.82186                14        0.00000000
## 6   6  bovine       21.20445                13        0.07142857
##   more50missing NumImputedFeature originalRUN GROUP GROUP_ORIGINAL
## 1         FALSE                 0           1     1             C1
## 2         FALSE                 0           2     1             C1
## 3         FALSE                 1           3     1             C1
## 4         FALSE                 1           4     2             C2
## 5         FALSE                 0           5     2             C2
## 6         FALSE                 1           6     2             C2
##   SUBJECT_ORIGINAL SUBJECT_NESTED SUBJECT
## 1                1            1.1       1
## 2                1            1.1       1
## 3                1            1.1       1
## 4                1            2.1       1
## 5                1            2.1       1
## 6                1            2.1       1
```

`DDA2009.TMP$RunlevelData` includes run-level summarized data based on `DDA2009.TMP$ProcessedData`. `LogIntensities` is run-level summarized data and will be used for `groupComparison` function in next step. It will also used for summarized profile plot (`summaryPlot=TRUE` for `dataProcessPlots` function with `type='ProfilePlot'`). `NumMeasuredFeature` shows how many features were used for summarization of the corresponding run and protein. `MissingPercentage` means the percentage of random and censored missing in the corresponding run and protein out of the total number of feature in the corresponding protein. `more50missing` means whether `MissingPercentage` is greater than 50% or not. `NumImputedFeature` show how many features were imputed in the corresponding run and protein.

```
# here 'TMP' : It shows which summaryMethod is used for run-level summarization.
head(DDA2009.proposed$SummaryMethod)
```

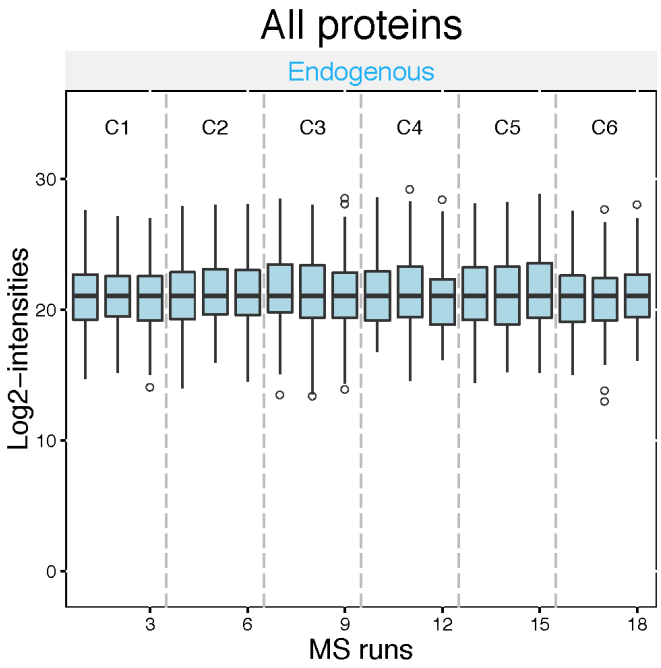
```
## [1] "TMP"
```

4.1.3 Visualization of processed data

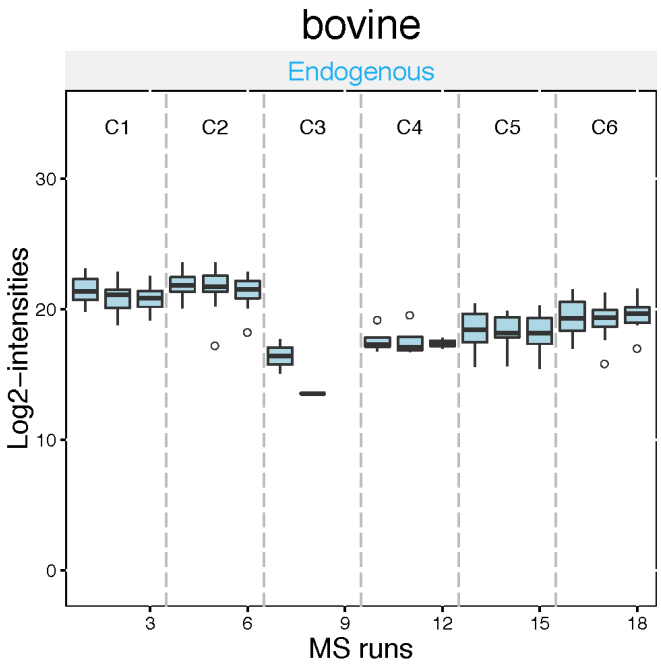
Quality control and normalization effects

QC plot visualizes potential systematic biases between mass spectrometry runs. Also it can be used to assess the effects of the normalization step. After constant normalization, the median intensities of reference transitions across all proteins should be equal between runs. After quantile normalization, the distribution of reference intensities across all proteins should be equal between runs. This step generates two types of QC plots: one for all the proteins combined, and the other separately for each protein (produced in a separate pdf file). These plots can be generated for either all proteins at once or each protein individually if we have a large dataset. The example below shows both options.

```
# use type="QCplot" with all proteins
# change the upper limit of y-axis=35
# set up the size of pdf
dataProcessPlots(data = DDA2009.proposed, type="QCplot", ylimUp=35,
                 width=5, height=5)
```




```
# use type="QCplot" for 1st protein only
# change the upper limit of y-axis=35
# set up the size of pdf
dataProcessPlots(data = DDA2009.proposed, type="QCplot", which.Protein=1,
                 ylimUp=35, width=5, height=5)
```



NOTE Don't worry about warning messages as below. It means `NA` values are not included in the plot, which is a proper way for this case.

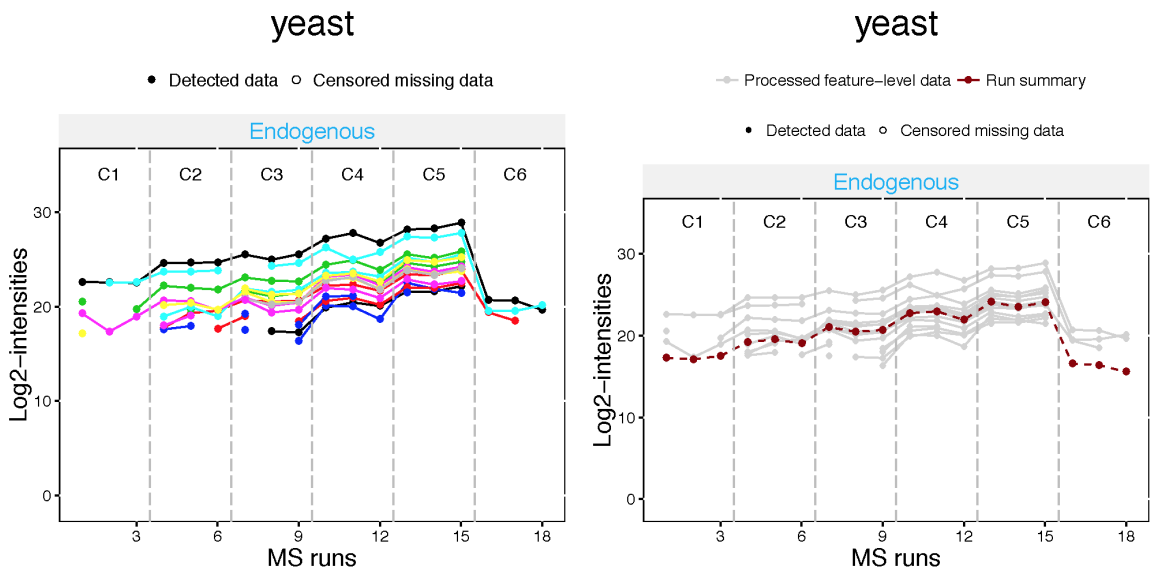
```
Warning messages:
1: Removed 698 rows containing non-finite values
```

(stat_boxplot).
...

Profile plot

Profile plot helps identify potential sources of variation (both variation of interest and nuisance variation) for each protein. Such plots should be done after the normalization. Profile plots with summarization present the effects of the summarization step by showing all individual measurements of a protein and their summarized intensity. With `type="profileplot"`, two pdfs will be generated. The first pdf includes plots (per protein) to show individual measurement for each peptide (peptide for DDA, transition for SRM or DIA) across runs, grouped per condition. Each peptide has a different color/type layout. Disconnected lines show that there are missing value (NA). To ignore these plots, please use the option `originalPlot=FALSE`. The second pdf, which is named with 'wSummarization' suffix, shows run-level summarized data per protein. The same peptides (or transition) in the first plot are presented in grey, with the summarized values (by TMP, in this example) overlaid in red. To ignore these plots with summarization, please use the option `summaryPlot=FALSE`.

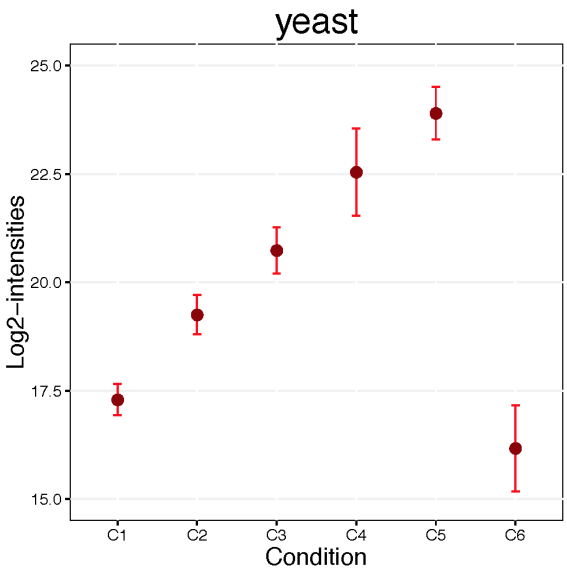
```
dataProcessPlots(data = DDA2009.proposed, type="Profileplot",  ylimUp=35,
                 featureName="NA", width=5, height=5, address="DDA2009_proposed_")
```



Condition plot

Condition plot visualizes potential systematic diffrences in protein inensities between conditions. Dots indicate the mean of log2 intensities for each condition. With the option `interval='CI'`(default), error bars indicate the confidence interval with 0.95 significant level for each condition. With the option `interval='SD'`, error bars indicate the standard deviation among all feature intensities for each condition. **The intervals are for descriptive purposes only, as more refined model-based estimation is obtained as discussed below.** With the option `scale=TRUE`, the levels of conditions are scaled according to their labels. If `scale=FALSE` (default), the conditions on the x-axis are equally spaced.

```
dataProcessPlots(data = DDA2009.proposed, type="Conditionplot",
                 width=5, height=5, address="DDA2009_proposed_")
```



`dataProcessPlots` has a number of layout options, including size and description of axes labels, output file name etc for three types of plots above. The option `address` specifies the name of the folder storing pdf files with the plots. With the option `address=FALSE`, plots will be shown in the graphical window, but not saved in a file. If a file with this name already exists in working directory, a suffix with a number will be appended to the file name. In this way a record of all the analyses is kept.

For more details, visit the help file using the following code.

```
?dataProcessPlots
```

4.1.4 Different imputation options

Here is **the summary of combinations for imputation options** with `summaryMethod='TMP'`.

- `censoredInt=NULL` : It assumes that all intensities are missing at random and there is no action for missing values with `MBimpute=FALSE`. If you have `MBimpute=TRUE` with `censoredInt=NULL`, there will be error message to fix either `MBimpute` or `censoredInt` options.
- `censoredInt='NA'` or `'0'` & `MBimpute=TRUE` : AFT model-based imputation using `cutoffCensored` value in the AFT model.
- `censoredInt='NA'` or `'0'` & `MBimpute=FALSE` : censored intensities (here `NA`'s) will be replaced with the value specified in `cutoffCensored`.

NOTE1 The default option for `cutoffCensored` is `minFeature`, taking the minimum value for the corresponding feature. With this option, those runs with substantial missing measurements will be biased by the cutoff value. In such case, you may remove the runs that have more than 50% missing values from the analysis with the option `remove50missing=TRUE`.

NOTE2 In case that there are completely missing measurements in a run for a protein, any imputation will not be performed. In addition, the condition, which has no measurement at all in a protein, will be not imputed.

Here is the example of `dataProcess` option without imputation, assuming that all missing values are random.

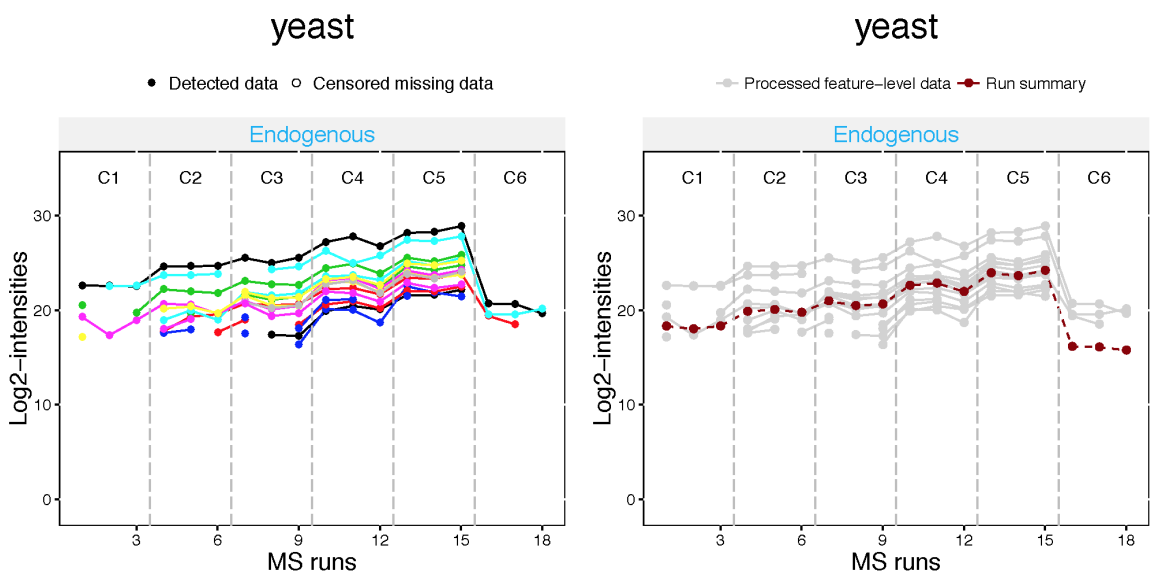
```
# No imputation
DDA2009.TMP <- dataProcess(raw = DDARawData,
                           normalization = 'equalizeMedians',
```

```
                                summaryMethod = 'TMP',
                                censoredInt = NULL, MBimpute=FALSE)
```

```
## * Use all features that the dataset origianally has.
##
##   Summary of Features :
##                          count
## # of Protein                 6
## # of Peptides/Protein    11-32
## # of Transitions/Peptide   1-1
##
##   Summary of Samples :
##                            C1 C2 C3 C4 C5 C6
## # of MS runs                3  3  3  3  3  3
## # of Biological Replicates  1  1  1  1  1  1
## # of Technical Replicates   3  3  3  3  3  3
##
##  Summary of Missingness :
##   # transitions are completely missing in one condition: 90
##     -> D.GPLTGTYR_23_23_NA_NA, F.HFHWGSSDDQGSEHTVDR_402_402_NA_NA, G.PLTGTYR_8_8_NA_NA, H.SFNVEYDDSQ
##
##   # run with 75% missing observations: 0
##
##  == Start the summarization per subplot...
## Getting the summarization by Tukey's median polish per subplot for protein  bovine ( 1  of  6 )
## Getting the summarization by Tukey's median polish per subplot for protein  chicken ( 2  of  6 )
## Getting the summarization by Tukey's median polish per subplot for protein  cyc_horse ( 3  of  6 )
## Getting the summarization by Tukey's median polish per subplot for protein  myg_horse ( 4  of  6 )
## Getting the summarization by Tukey's median polish per subplot for protein  rabbit ( 5  of  6 )
## Getting the summarization by Tukey's median polish per subplot for protein  yeast ( 6  of  6 )
##
##  == the summarization per subplot is done.
```

These plots can be used compare and select among different options for imputation (e.g., `TMP` with or without considering missing values for summarization in `dataProcess`).

```
dataProcessPlots(data = DDA2009.TMP, type="Profileplot",  ylimUp=35,
                 featureName="NA", width=5, height=5, address="DDA2009_TMP_")
```



While original profile plots are the same, summarization plots reveal differences, especially for conditions 'C1' and 'C2' in 'yeast' protein, which have many missing values. Without imputation, summarized values in 'C1' group is higher than with imputation for missing values.

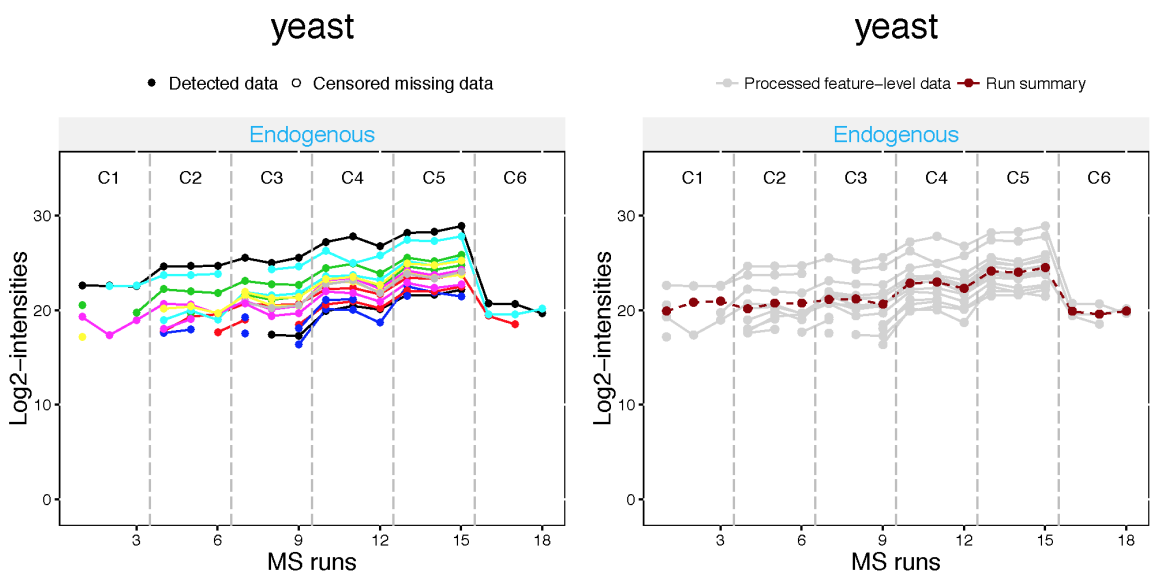
4.1.5 Different summarization options

Besides summarizing observations with the median polish method, `MSstats` also offers a summarization option using linear model with option `summaryMethod="linear"` with `censoredInt=NULL` assumes that all `NA`'s are missing at random and uses `lm` for parameter estimation.

```
# linear model (lm) with run and feature
DDA2009.linear <- dataProcess(raw = DDARawData,
                              normalization = 'equalizeMedians',
                              summaryMethod = 'linear',
                              censoredInt = NULL,
                              MBimpute = FALSE)
```

Profile plots below can be used compare among different options for summarization (e.g., `TMP` with or without imputation vs `linear` for summarization in `dataProcess`).

```
dataProcessPlots(data = DDA2009.linear, type="Profileplot",  ylimUp=35,
                 featureName="NA", width=5, height=5, address="DDA2009_linear_")
```



While original profile plots are the same, summarization plots reveal differences, especially for conditions 'C1', 'C2', and 'C6' in 'yeast' protein, which have many missing values. Summarized values with linear model in these groups are much higher than those with TMP considering missing values or not.

4.1.6 Finding differentially abundant proteins across conditions

Comparing conditions with groupComparison

With the normalized data and run-level summarized data obtained by applying one of the `dataProcess` summarization methods, it is of general interest to find proteins changing between groups of conditions. Within `MSstats` this can be done by using the `groupComparison` function, which takes the output of the `dataProcess` function as input.

```
?groupComparison
```

In addition to the processed data, the `groupComparison` function requires a contrast matrix to define the comparison to be made. The contrast matrix is created with each condition in column and each comparison in row. Note that the conditions are arranged **in alphabetical order**. The order of condition that `MSstats` recognizes can be shown by using `levels`:

```
levels(DDA2009.TMP$ProcessedData$GROUP_ORIGINAL)
```

```
## [1] "C1" "C2" "C3" "C4" "C5" "C6"
```

Entries in each row of the contrast matrix are filled in with 0, 1, or -1 to specify the comparison, where **0** is for conditions we would like to ignore, **1** is for conditions we would like to put in the numerator of the ratio or fold-change, and **-1** is for conditions we would like to put in the denumerator of the ratio or fold-change.

For example, if you want to compare C2-C1, which means log(C2)-log(C1) and the same as log(C2/C1), set '1' for C2 and '-1' for C1 in the row. Combining multiple groups for comparison is also possible. For example, if you want to compare between average of C2 and C3 and average of C1, (C3+C2)/2-C1 as formula, set '-1' for C1, '0.5' for C2 and '0.5' for C3, and '0' for rest of groups.

```
comparison1 <- matrix(c(-1,1,0,0,0,0),nrow=1)
comparison2 <- matrix(c(0,-1,1,0,0,0),nrow=1)
comparison3 <- matrix(c(0,0,-1,1,0,0),nrow=1)
comparison4 <- matrix(c(0,0,0,-1,1,0),nrow=1)
comparison5 <- matrix(c(0,0,0,0,-1,1),nrow=1)
```

```
comparison6 <- matrix(c(1,0,0,0,0,-1),nrow=1)

comparison<-rbind(comparison1,comparison2,comparison3,comparison4,comparison5,comparison6)
row.names(comparison) <- c("C2-C1","C3-C2","C4-C3","C5-C4","C6-C5","C1-C6")
```

With the contrast matrix specified, group comparison can be performed as follows.

```
DDA2009.comparisons <- groupComparison(contrast.matrix = comparison, data = DDA2009.proposed)
```

Output of the `groupComparison` function contains three data frames:

```
# output from groupComparison function has three data frames
names(DDA2009.comparisons)
```

```
## [1] "ComparisonResult" "ModelQC"          "fittedmodel"
```

Results of the statistical comparison are stored in the data frame named `ComparisonResult`:

```
# name of columns in result data.frame
head(DDA2009.comparisons$ComparisonResult)
```

```
##      Protein Label     log2FC        SE     Tvalue DF       pvalue
## 1     bovine C2-C1  0.6048799 0.4245943   1.424607 11 1.820186e-01
## 7    chicken C2-C1  0.7876884 0.2205455   3.571545 12 3.841470e-03
## 13 cyc_horse C2-C1  1.1294964 0.1955787   5.775149 12 8.809149e-05
## 19 myg_horse C2-C1 -7.9717333 0.2807086 -28.398612 12 2.254641e-12
## 25    rabbit C2-C1  1.0617105 0.2209612   4.804963 12 4.298913e-04
## 31     yeast C2-C1  1.9575344 0.3134408   6.245309 12 4.284464e-05
##      adj.pvalue issue MissingPercentage ImputationPercentage
## 1  1.820186e-01    NA        0.03571429           0.03571429
## 7  4.609764e-03    NA        0.25757576           0.25757576
## 13 1.761830e-04    NA        0.15104167           0.15104167
## 19 1.352785e-11    NA        0.45833333           0.45833333
## 25 6.448369e-04    NA        0.72043011           0.72043011
## 31 1.285339e-04    NA        0.56666667           0.56666667
```

The result of the test for diffrential abundance is a table with columns `Protein`, `Label` (of the comparison), log2 fold change (`log2FC`), standard error of the log2 fold change (`SE`), test statistic of the Student test (`Tvalue`), degree of freedom of the Student test (`DF`), raw p-values (`pvalue`), p-values adjusted among all the proteins in the specific comparison using the approach by Benjamini and Hochberg (`adj.pvalue`). The cutoff of the adjusted p-value corresponds to the cutoff of the False Discovery Rate (Benjamini and Hochberg 1955). The positive values of log2FC for `Label=C2-C1` indicate evidence in favor of C2 > C1 (i.e. proteins upregulated in C2), while the negative values indicate evidence in favor of C2 < C1 (i.e. proteins downregulated in C2), as compared to C1. The same model can be used to perform several comparisons of conditions simultaneously in the same protein.

NOTE `issue` column shows if there is any issue for inference in corresponding protein and comparison, for example, `OneConditionMissing` or `CompleteMissing`. If one of condition for compariosn is completely missing, it would flag with `OneConditionMissiong` with `adj.pvalue=0` and `log2FC=Inf or -Inf` even though `pvalue=NA`. For example, if you want to compare 'Condition A - Condition B', but condition B has complete missing, `log2FC=Inf` and `adj.pvalue=0`. `SE`, `Tvalue`, and `pvalue` will be NA. if you want to compare 'Condition A - Condition B', but condition A has complete missing, then `log2FC=-Inf` and `adj.pvalue=0`. But, please be careful for using this `log2FC` and `adj.pvalue`.

Based on the comparison results and desired significance level, a short list of the differentially abundant proteins can be obtained for further investigation:

```
# get only significant proteins and comparisons among all comparisons
# To simultaneoulsy controll the overall FDR at the level, 0.05
SignificantProteins <- with(DDA2009.comparisons,
                            ComparisonResult[ComparisonResult$adj.pvalue < 0.05, ])
nrow(SignificantProteins)
```

```
## [1] 34
```

4.1.6 Verifying the assumption of the model

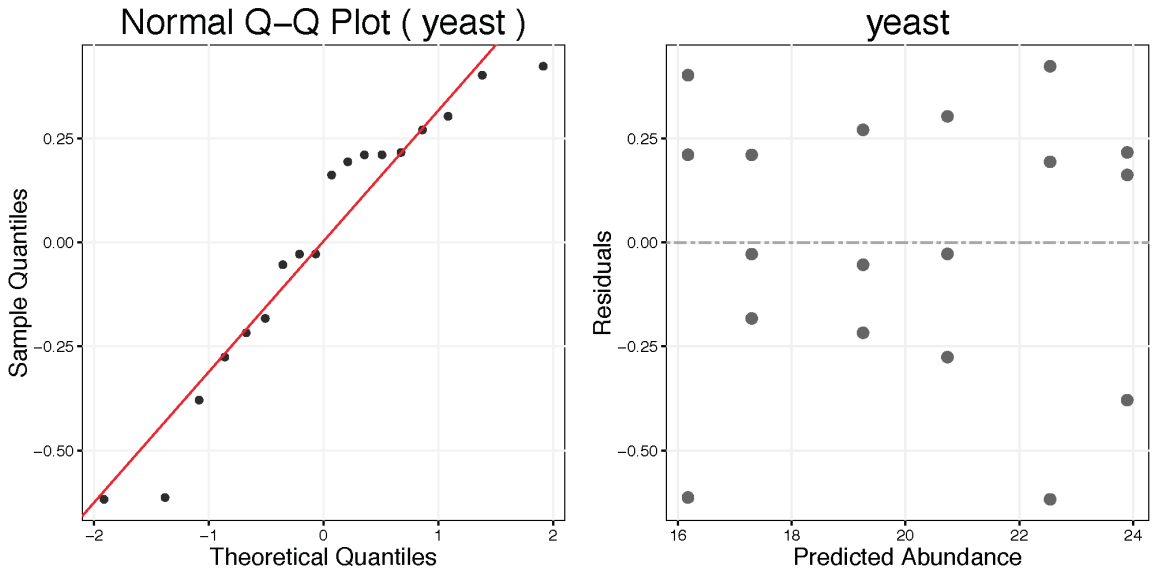
Results based on the statistical models are accurate as long as the assumptions of the models hold. Here we focus on the assumption of the Normal distribution of the measurement errors, and also on the assumption of constant variance of the measurement errors (if this option is specified in the model above). The assumptions can be checked by examining the residuals of the model fit (i.e., the deviations of the observed intensities of the transition from their model-based predictions).

`modelBasedQCPlots` function generates residual plots and Normal quantile-quantile plots for each protein, taking as input the results of model fitting and testing in `groupComparison`. Normal quantile-quantile plot with the option `type='QQPlots'` illustrates that such deviations from constant variance can be mistaken for deviations from Normality. Only large deviations of transition intensities from the straight line are problematic.

Residual plot with the option `type='ResidualPlots'` shows variance of the residuals that is associated with the mean feature intensity. Any specific pattern, such as increasing or decreasing by predicted abundance, is problematic.

```
# normal quantile-quantile plots
modelBasedQCPlots(data=DDA2009.comparisons, type="QQPlots",
                  width=5, height=5, address="DDA2009_proposed_")

# residual plots
modelBasedQCPlots(data=DDA2009.comparisons, type="ResidualPlots",
                  width=5, height=5, address="DDA2009_proposed_")
```



For more details, visit the help file using the following code.

```
?modelBasedQCPlots
```

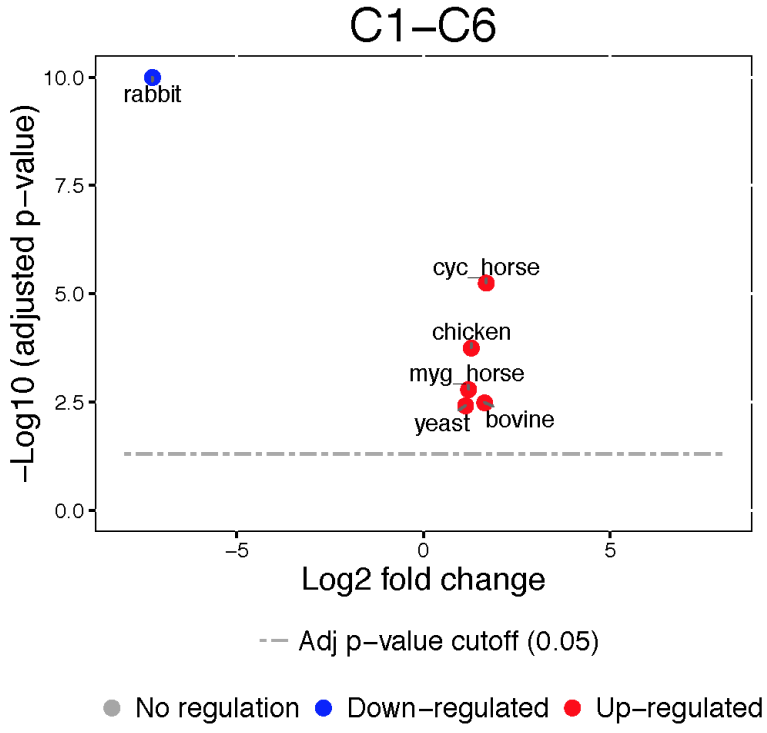
4.1.7 Visualization of differentially abundant proteins

Volcano plots

Volcano plots visualize the outcome of one comparison between conditions for all the proteins, and combine the information on statistical and practical significance. The y-axis displays the FDR-adjusted p-values on the negative log10 scale, representing statistical significance. The horizontal dashed line shows the FDR cutoff. The points above the FDR cutoff line are statistically significant proteins that are differentially abundant across conditions. These points are colored in red and blue for upregulated and downregulated proteins, respectively. The x-axis is the model-based estimate of fold change on log scale (the base of logarithm transform is the same as specified in the `logTrans` option of the `dataProcess` function), and represents practical significance. It is possible to specify a practical significance cutoff based on the estimate of fold change in addition to the statistical significance cutoff. If the fold change cutoff is specified, the points above the horizontal cutoff line but within the vertical cutoff line will be considered as not differentially abundant (and will be colored in black). The practical significance cutoff should only be applied in addition to the statistical significance cutoff (i.e., the fold change alone does not present enough evidence for differential abundance).

```
groupComparisonPlots(data = DDA2009.comparisons$ComparisonResult, type = 'VolcanoPlot',
                     width=5, height=5, address="DDA2009_proposed_")
```

'VolcanoPlot.pdf' will be saved under the folder you assigned. It has the plots per comparison defined in `contrast.matrix`. Please check `?groupComparisonPlots` for detail, such as labelling protein names, size of dots, font sizes, etc. Below is one of volcano plots, for comparison 'C1-C6' including protein name labelling. Protein name will be shown for significant proteins, without overlapping protein names each other.



Heatmap

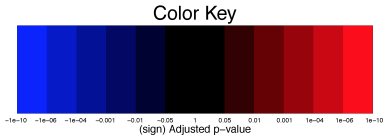
Heatmaps illustrate the patterns of up- and down-regulation of proteins in several comparisons. Columns in the heatmaps are comparison of conditions assigned in `contrast.matrix`, and rows are proteins. The heatmaps display signed FDR-adjusted p-values of the tests, colored in red/blue for significantly up-/down-regulated proteins, while taking into account the specified FDR cutoff and the additional optional fold change cutoff. Brighter colors indicate stronger evidence in favor of differential abundance. Black color represents proteins that are not significantly differentially abundant.

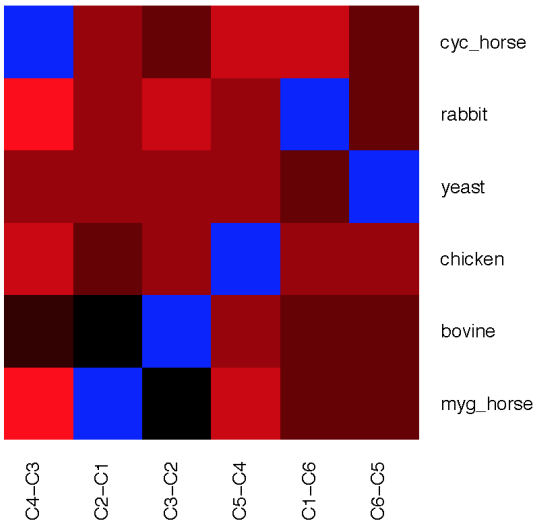
NOTE To draw heatmap, at least two comparisons are needed.

The rows and columns of the heatmaps can be ordered with the option clustering, which performs hierarchical clustering with the Ward method (minimum variance). The option `clustering='protein'` (default) clusters the rows (proteins) in the space of comparisons, based on the values of (sign of comparison) · (-log2(adjusted p-values)). The option `clustering='comparison'` clusters the columns in the space of proteins, based on the values of (sign of comparison) · (-log2(adjusted p-value)). The option `clustering='both` reorders both columns and rows.

```
groupComparisonPlots(data = DDA2009.comparisons$ComparisonResult, type = 'Heatmap')
```

'Heatmap.pdf' will be saved under the folder you assigned. Below is one example, showing the results for several comparisons simultaneously.

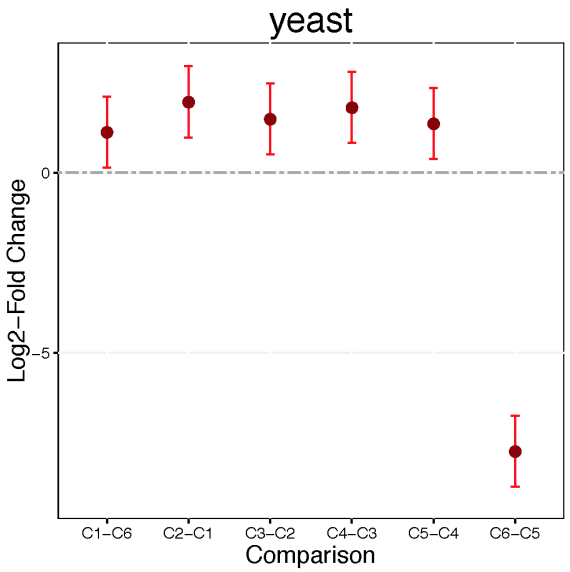




Comparison plots

Comparison plots illustrate model-based estimates of log-fold changes, and the associated uncertainty, in several comparisons of conditions for one protein. X-axis is the comparison of interest. Y-axis is the log fold change. The dots are the model-based estimates of log-fold change, and the error bars are the model-based 95% confidence intervals (the option sig can be used to change the significance level of significance). For simplicity, the confidence intervals are adjusted for multiple comparisons within protein only, using the Bonferroni approach. For proteins with N comparisons, the individual confidence intervals are at the level of 1-sig/N.

```
groupComparisonPlots(data=DDA2009.comparisons$ComparisonResult, type="ComparisonPlot",
                     width=5, height=5, address="DDA2009_proposed_")
```



For further details, such as labelling protein names, size of dots, font sizes, etc., visit the help file using the following code.

```
?groupComparisonPlots
```

4.1.8 Sample size calculation for a future experiment

This last analysis step views the dataset as a pilot study of a future experiment, utilizes its variance components, and calculates the minimal number of replicates required in a future experiment to achieve the desired statistical power. The calculation is performed by the function `designSampleSize`, which takes as input the fitted model in `groupComparison`. Sample size calculation assumes same experimental design (i.e. group comparison, time course or paired design) as in the current dataset, and uses the model fit to estimate the median variance components across all the proteins. Finally, sample size calculation assumes that a large proportion of proteins (specifically, 99%) will not change in abundance in the future experiment. This assumption also provides conservative results. Using the estimated variance components, the function relates the number of biological replicates per condition (`numSample`, rounded to 0 decimal), average statistical power across all the proteins (`power`), minimal fold change that we would like to detect (can be specified as a range, e.g. `desiredFC=c(1.1, 2)`), and the False Discovery Rate (`FDR`). The user should specify all these quantities but one, and the function will solve for the remainder. The quantity to solve for should be set to `= TRUE`.

```
# Minimal number of biological replicates per condition
result.sample <- designSampleSize(data=DDA2009.comparisons$fittedmodel, numSample=TRUE,
                 desiredFC=c(1.25, 3), FDR=0.05, power=0.8)
result.sample
```

```
##    desiredFC numSample  FDR power    CV
## 1      1.250        35 0.05   0.8 0.004
## 2      1.275        30 0.05   0.8 0.005
## 3      1.300        25 0.05   0.8 0.006
## 4      1.325        22 0.05   0.8 0.007
## 5      1.350        19 0.05   0.8 0.007
## 6      1.375        17 0.05   0.8 0.008
## 7      1.400        16 0.05   0.8 0.009
## 8      1.425        14 0.05   0.8 0.010
## 9      1.450        13 0.05   0.8 0.010
## 10     1.475        12 0.05   0.8 0.011
## 11     1.500        11 0.05   0.8 0.012
## 12     1.525        10 0.05   0.8 0.013
## 13     1.550         9 0.05   0.8 0.014
## 14     1.575         9 0.05   0.8 0.014
## 15     1.600         8 0.05   0.8 0.015
## 16     1.625         7 0.05   0.8 0.017
## 17     1.650         7 0.05   0.8 0.017
## 18     1.675         7 0.05   0.8 0.016
## 19     1.700         6 0.05   0.8 0.019
## 20     1.725         6 0.05   0.8 0.018
## 21     1.750         6 0.05   0.8 0.018
## 22     1.775         5 0.05   0.8 0.022
## 23     1.800         5 0.05   0.8 0.021
## 24     1.825         5 0.05   0.8 0.021
## 25     1.850         5 0.05   0.8 0.021
## 26     1.875         4 0.05   0.8 0.026
## 27     1.900         4 0.05   0.8 0.025
## 28     1.925         4 0.05   0.8 0.025
## 29     1.950         4 0.05   0.8 0.025
## 30     1.975         4 0.05   0.8 0.024
## 31     2.000         4 0.05   0.8 0.024
## 32     2.025         4 0.05   0.8 0.024
## 33     2.050         3 0.05   0.8 0.031
## 34     2.075         3 0.05   0.8 0.031
## 35     2.100         3 0.05   0.8 0.030
## 36     2.125         3 0.05   0.8 0.030
## 37     2.150         3 0.05   0.8 0.030
## 38     2.175         3 0.05   0.8 0.029
## 39     2.200         3 0.05   0.8 0.029
## 40     2.225         3 0.05   0.8 0.029
## 41     2.250         3 0.05   0.8 0.028
## 42     2.275         3 0.05   0.8 0.028
## 43     2.300         3 0.05   0.8 0.028
## 44     2.325         2 0.05   0.8 0.041
## 45     2.350         2 0.05   0.8 0.041
## 46     2.375         2 0.05   0.8 0.040
## 47     2.400         2 0.05   0.8 0.040
## 48     2.425         2 0.05   0.8 0.039
## 49     2.450         2 0.05   0.8 0.039
## 50     2.475         2 0.05   0.8 0.039
## 51     2.500         2 0.05   0.8 0.038
```

```
## 52     2.525          2 0.05   0.8 0.038
## 53     2.550          2 0.05   0.8 0.038
## 54     2.575          2 0.05   0.8 0.037
## 55     2.600          2 0.05   0.8 0.037
## 56     2.625          2 0.05   0.8 0.036
## 57     2.650          2 0.05   0.8 0.036
## 58     2.675          2 0.05   0.8 0.036
## 59     2.700          2 0.05   0.8 0.035
## 60     2.725          2 0.05   0.8 0.035
## 61     2.750          2 0.05   0.8 0.035
## 62     2.775          2 0.05   0.8 0.034
## 63     2.800          2 0.05   0.8 0.034
## 64     2.825          2 0.05   0.8 0.034
## 65     2.850          2 0.05   0.8 0.034
## 66     2.875          2 0.05   0.8 0.033
## 67     2.900          2 0.05   0.8 0.033
## 68     2.925          2 0.05   0.8 0.033
## 69     2.950          1 0.05   0.8 0.065
## 70     2.975          1 0.05   0.8 0.064
## 71     3.000          1 0.05   0.8 0.064
```

```r
# Power calculation
result.power <- designSampleSize(data=DDA2009.comparisons$fittedmodel, numSample=3,
                desiredFC=c(1.25, 3), FDR=0.05, power=TRUE)
result.power
```

```
##    desiredFC numSample  FDR power    CV
## 1      1.250         3 0.05  0.01 0.051
## 2      1.275         3 0.05  0.01 0.050
## 3      1.300         3 0.05  0.01 0.049
## 4      1.325         3 0.05  0.01 0.048
## 5      1.350         3 0.05  0.01 0.047
## 6      1.375         3 0.05  0.01 0.046
## 7      1.400         3 0.05  0.01 0.046
## 8      1.425         3 0.05  0.01 0.045
## 9      1.450         3 0.05  0.01 0.044
## 10     1.475         3 0.05  0.01 0.043
## 11     1.500         3 0.05  0.02 0.043
## 12     1.525         3 0.05  0.03 0.042
## 13     1.550         3 0.05  0.04 0.041
## 14     1.575         3 0.05  0.06 0.041
## 15     1.600         3 0.05  0.08 0.040
## 16     1.625         3 0.05  0.10 0.039
## 17     1.650         3 0.05  0.13 0.039
## 18     1.675         3 0.05  0.16 0.038
## 19     1.700         3 0.05  0.20 0.038
## 20     1.725         3 0.05  0.23 0.037
## 21     1.750         3 0.05  0.27 0.036
## 22     1.775         3 0.05  0.31 0.036
## 23     1.800         3 0.05  0.35 0.035
## 24     1.825         3 0.05  0.39 0.035
## 25     1.850         3 0.05  0.43 0.034
## 26     1.875         3 0.05  0.47 0.034
## 27     1.900         3 0.05  0.51 0.034
## 28     1.925         3 0.05  0.55 0.033
```

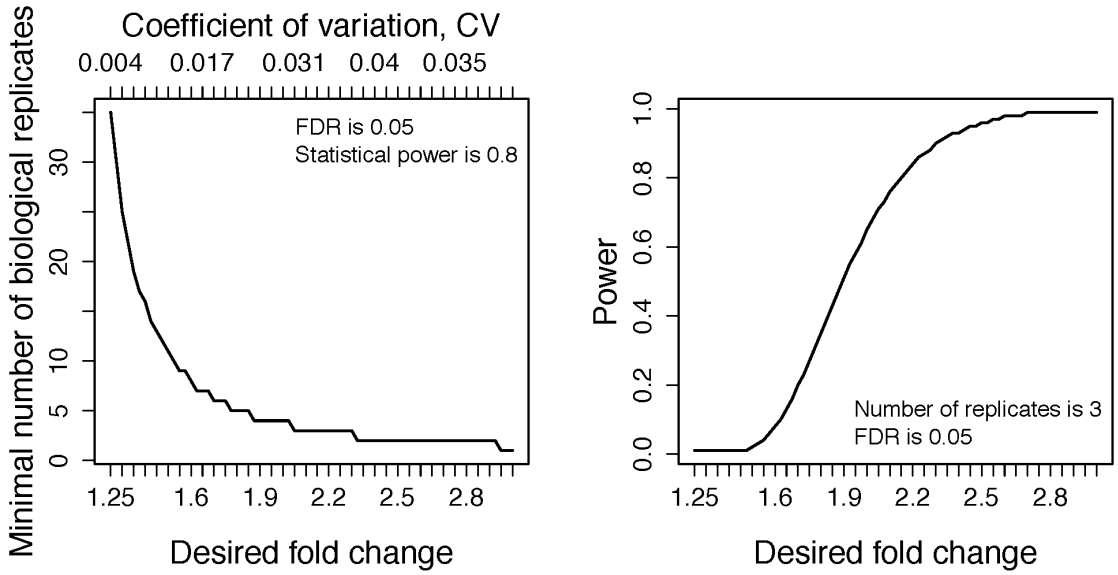
```
## 29     1.950         3 0.05  0.58 0.033
## 30     1.975         3 0.05  0.61 0.032
## 31     2.000         3 0.05  0.65 0.032
## 32     2.025         3 0.05  0.68 0.032
## 33     2.050         3 0.05  0.71 0.031
## 34     2.075         3 0.05  0.73 0.031
## 35     2.100         3 0.05  0.76 0.030
## 36     2.125         3 0.05  0.78 0.030
## 37     2.150         3 0.05  0.80 0.030
## 38     2.175         3 0.05  0.82 0.029
## 39     2.200         3 0.05  0.84 0.029
## 40     2.225         3 0.05  0.86 0.029
## 41     2.250         3 0.05  0.87 0.028
## 42     2.275         3 0.05  0.88 0.028
## 43     2.300         3 0.05  0.90 0.028
## 44     2.325         3 0.05  0.91 0.027
## 45     2.350         3 0.05  0.92 0.027
## 46     2.375         3 0.05  0.93 0.027
## 47     2.400         3 0.05  0.93 0.027
## 48     2.425         3 0.05  0.94 0.026
## 49     2.450         3 0.05  0.95 0.026
## 50     2.475         3 0.05  0.95 0.026
## 51     2.500         3 0.05  0.96 0.026
## 52     2.525         3 0.05  0.96 0.025
## 53     2.550         3 0.05  0.97 0.025
## 54     2.575         3 0.05  0.97 0.025
## 55     2.600         3 0.05  0.98 0.025
## 56     2.625         3 0.05  0.98 0.024
## 57     2.650         3 0.05  0.98 0.024
## 58     2.675         3 0.05  0.98 0.024
## 59     2.700         3 0.05  0.99 0.024
## 60     2.725         3 0.05  0.99 0.023
## 61     2.750         3 0.05  0.99 0.023
## 62     2.775         3 0.05  0.99 0.023
## 63     2.800         3 0.05  0.99 0.023
## 64     2.825         3 0.05  0.99 0.023
## 65     2.850         3 0.05  0.99 0.022
## 66     2.875         3 0.05  0.99 0.022
## 67     2.900         3 0.05  0.99 0.022
## 68     2.925         3 0.05  0.99 0.022
## 69     2.950         3 0.05  0.99 0.022
## 70     2.975         3 0.05  0.99 0.021
## 71     3.000         3 0.05  0.99 0.021
```

For further details, visit the help file using the following code.

```
?designSampleSize
```

Visualization of sample size calculations

The calculated relationship between the number of biological replicates per condition (`numSample`), average statistical power across all the proteins (`power`), minimal fold change that we would like to detect (`desiredFC`), and the False Discovery Rate (`FDR`) can be visualized using the function `designSampleSizePlots`. The function takes as input the output of `designSampleSize`.



For further details, visit the help file using the following code.

```
?designSampleSizePlots
```

4.1.9 Quantification of protein abundance in individual samples or conditions

Many downstream analysis steps (such as clustering or classification of individual samples in the space of their protein profiles) require summary values of protein abundance in each biological replicate or in each condition, on a relative scale that is comparable between runs.

`dataProcess` function performs model-based run-level summarization. `quantification` function enables subject-level summarization or group-level summarization with the run-level summarization from `dataProcess`.

The option, `type='sample'`(default), performs sample quantification, i.e. it outputs the estimates of relative protein abundance in each biological replicate. If there are technical replicates for biological replicates, sample quantification will be the median among technical replicates. If there is no technical replicate for biological replicate (sample), sample quantification will be the same as run-level summarization. In presence of completely missing values in biological replicate, the estimates will be zero.

The option `type='group'` performs group quantification, i.e. it outputs the estimates of relative protein abundance in each condition, summarized over the biological replicates (median among sample quantification). In presence of completely missing values in a condition, the estimates will be zero.

MSstats supports two output formats. The option `format='matrix'` (default) outputs an array where rows are `proteins`, and columns are `conditions` (for group quantification), or combinations of biological replicate and condition ids (for sample quantification). The option `format='long'` produces an array where each row corresponding to relative protein abundances, and columns are `Protein`, `Condition`, `LogIntensities` (and BioReplicate in the case of sample quantification).

```
subQuant <- quantification(DDA2009.proposed)
head(subQuant)
```

```
##     Protein     C1_1     C2_1     C3_1     C4_1     C5_1     C6_1
## 1    bovine 20.85653 21.60443 14.32690 16.10441 17.63141 19.27802
## 2   chicken 18.48792 19.43204 20.41274 22.42284 15.92462 17.09803
## 3 cyc_horse 20.25927 21.33967 22.22028 15.85252 17.62720 18.45536
## 4 myg_horse 22.66495 14.73701 14.99667 18.61740 20.26392 21.52022
```

```
## 5    rabbit 14.89507 15.88492 17.43767 20.19014 21.27964 22.07550
## 6     yeast 17.26792 19.19987 20.71073 22.73666 24.06156 16.38660
```

```
groupQuant <- quantification(DDA2009.proposed, type='group')
head(groupQuant)
```

```
##     Protein       C1       C2       C3       C4       C5       C6
## 1    bovine 20.85653 21.60443 14.32690 16.10441 17.63141 19.27802
## 2   chicken 18.48792 19.43204 20.41274 22.42284 15.92462 17.09803
## 3 cyc_horse 20.25927 21.33967 22.22028 15.85252 17.62720 18.45536
## 4 myg_horse 22.66495 14.73701 14.99667 18.61740 20.26392 21.52022
## 5    rabbit 14.89507 15.88492 17.43767 20.19014 21.27964 22.07550
## 6     yeast 17.26792 19.19987 20.71073 22.73666 24.06156 16.38660
```

For further details, visit the help file using the following code.

```
?quantification
```

4.2 Suggested workflow with Skyline output for DDA

This section describes steps and considerations to properly format data processed by Skyline, prior to the `MSstats` analysis. In the following example, the raw files for Dynamic benchmark dataset (J. Cox et al. 2014) are used,searched by Andromeda in MaxQuant, but quantified by Skyline.

4.2.1 Load Skyline output

This required input data is generated automatically when using `MSstats` report format in Skyline. We first load and access the dataset processed by Skyline. The name of saved file from Skyline using `MSstats report format` is 'Cox.Skyline.csv'.

```
# Read output from skyline : Cox.skyline.csv
raw <- read.csv("Cox.skyline.csv")
```

We can read csv file. Here we will load R data file which is the exactly same data in Cox.skyline.csv file.

```
# Load R data, which is convered from csv file, output from skyline : Cox.skyline.csv
load("Cox.skyline.RData")
raw <- Cox.skyline
```

Annotation information is required to fill in `Condition` and `BioReplicate` for corresponding `Run` information. Users have to prepare as csv or txt file like 'Cox_skyline_annotation.csv', which includes `Run`, `Condition`, and `BioReplicate` information, and load it in R.

```
annot <- read.csv("Cox_skyline_annotation.csv", header=TRUE)
annot
```

```
##                                 Run Condition BioReplicate
## 1 20130510_EXQ1_IgPa_QC_UPS1_01.raw      UPS1            1
## 2 20130510_EXQ1_IgPa_QC_UPS1_02.raw      UPS1            2
## 3 20130510_EXQ1_IgPa_QC_UPS1_03.raw      UPS1            3
## 4 20130510_EXQ1_IgPa_QC_UPS1_04.raw      UPS1            4
## 5 20130510_EXQ1_IgPa_QC_UPS2_01.raw      UPS2            5
## 6 20130510_EXQ1_IgPa_QC_UPS2_02.raw      UPS2            6
## 7 20130510_EXQ1_IgPa_QC_UPS2_03.raw      UPS2            7
## 8 20130510_EXQ1_IgPa_QC_UPS2_04.raw      UPS2            8
```

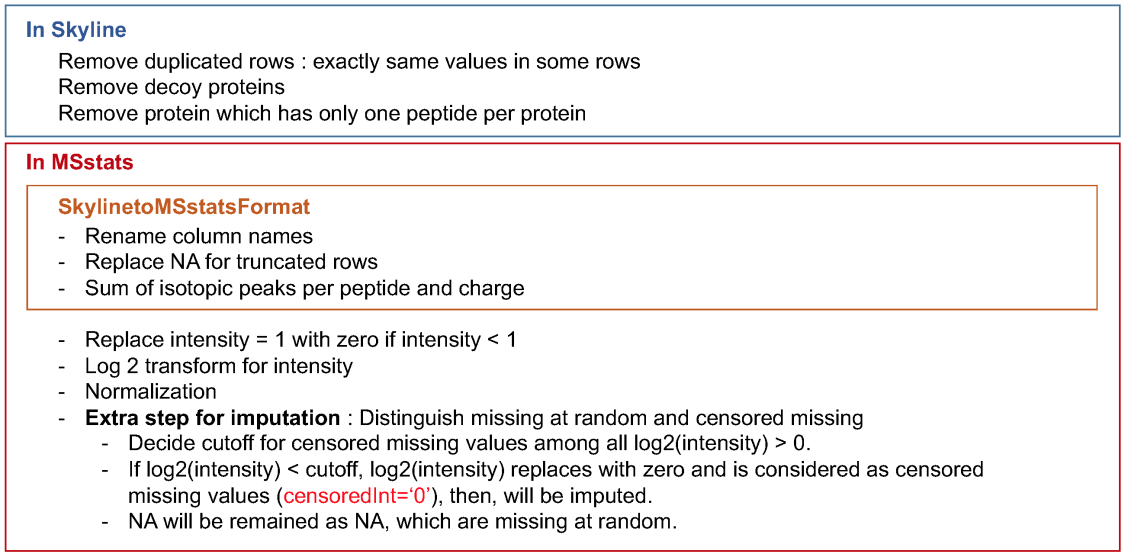
4.2.2 Preprocessing with DDA experiment from Skyline output

The input data for `MSstats` is required to contain variables of `ProteinName`, `PeptideSequence`, `PrecursorCharge`, `FragmentIon`, `ProductCharge`, `IsotopeLabelType`, `Condition`, `BioReplicate`, `Run`, `Intensity`. These variable names should be fixed. `MSstats` input from Skyline adapts the column scheme of the dataset so that it fits `MSstats` input format. However there are several extra column names and also some of them need to be changed.

```
##             ProteinName PeptideSequence PeptideModifiedSequence
## 73 sp|P02768|ALBU_HUMAN        DLGEENFK                DLGEENFK
## 74 sp|P02768|ALBU_HUMAN        DLGEENFK                DLGEENFK
## 75 sp|P02768|ALBU_HUMAN        DLGEENFK                DLGEENFK
## 76 sp|P02768|ALBU_HUMAN        DLGEENFK                DLGEENFK
## 77 sp|P02768|ALBU_HUMAN        DLGEENFK                DLGEENFK
## 78 sp|P02768|ALBU_HUMAN        DLGEENFK                DLGEENFK
##    PrecursorCharge PrecursorMz FragmentIon ProductCharge ProductMz
## 73               2    476.2245   precursor             2  476.2245
## 74               2    476.2245   precursor             2  476.2245
## 75               2    476.2245   precursor             2  476.2245
## 76               2    476.2245   precursor             2  476.2245
## 77               2    476.2245   precursor             2  476.2245
## 78               2    476.2245   precursor             2  476.2245
##    IsotopeLabelType Condition BioReplicate
## 73            light        NA           NA
## 74            light        NA           NA
## 75            light        NA           NA
## 76            light        NA           NA
## 77            light        NA           NA
## 78            light        NA           NA
##                             FileName       Area StandardType Truncated
## 73 20130510_EXQ1_IgPa_QC_UPS1_01.raw 1222621440           NA     False
## 74 20130510_EXQ1_IgPa_QC_UPS1_02.raw 1641793792           NA     False
## 75 20130510_EXQ1_IgPa_QC_UPS1_03.raw 1225490048           NA     False
## 76 20130510_EXQ1_IgPa_QC_UPS1_04.raw 1777631616           NA     False
## 77 20130510_EXQ1_IgPa_QC_UPS2_01.raw 7395562496           NA     False
## 78 20130510_EXQ1_IgPa_QC_UPS2_02.raw 6193937408           NA     False
##    DetectionQValue
## 73            #N/A
## 74            #N/A
## 75            #N/A
## 76            #N/A
## 77            #N/A
## 78            #N/A
```

`SkylinetoMSstatsFormat` function helps pre-processing for making right format of MSstats input from Skyline output. For example, it renames some column name, and replace truncated peak intensities with NA. Another important step is to handle isotopic peaks before using `dataProcess` The output from Skyline for DDA experiment has several measurements of peak area from the monoisotopic, M+1 and M+2 peaks. To get a robust measure of peptide intensity, we can sum over isotopic peaks per peptide or use the highest peak. Here we take a summation per peptide ion.

Here is the summary of pre-processing steps in `SkylinetoMSstatsFormat` function (in orange box below).




```
# reformating and pre-processing for Skyline output.
quant <- SkylinetoMSstatsFormat(raw, annotation=annot)
```

```
## Warning in SkylinetoMSstatsFormat(raw, annotation = annot): NAs introduced
## by coercion
```

```
## ** Truncated peaks are replaced with NA.
```

```
## ** For DDA datasets, three isotopic peaks per feature and run are summed.
```

For further details, visit the help file using the following code.

```
?SkylinetoMSstatsFormat
```

4.2.3 Different options for Skyline in dataProcess

The difference between output from Skyline and other spectral processing tool is that Skyline distinguishes random missing (NA) by technical issues and low noisy intensity due to less than limit of etection. The output from Skyline can have both NA (expect small number of NAs or none of them) and very small intensity close to zero (less than 1 in intensity) and those should be treated different types of missing. In `dataProcess`, users need to use `censoredInt='0'` for Skyline output, which means to distinguish between `NA` as random missing and `0` as censored missing.

```
cox.skyline.proposed <- dataProcess(quant,
                                    normalization='equalizeMedian',
                                    summaryMethod="TMP",
                                    cutoffCensored="minFeature", censoredInt="0",
                                    MBimpute=TRUE,
                                    maxQuantileforCensored=0.999)
```

Further steps is the same as in general workflow for DDA.

4.3 Suggested workflow with MaxQuant output for DDA

The following R code chunks show steps to format a MaxQuant output for analysis by `MSstats`. Here a controlled mixture dataset with dynamic range benchmark (J. Cox et al. 2014) is used for demonstration. This dataset is available in MSstats material GitHub under the folder named 'example dataset/DDA_controlledMixture20014'.

4.3.1 Load MaxQuant outputs

Three files should be prepared before `MSstats`. Two files, 'proteinGroups.txt' and 'evidence.txt' are outputs from MaxQuant.

```
## First, get protein ID information
proteinGroups <- read.table("Cox_maxquant_proteinGroups.txt", sep = "\t", header = TRUE)
```

```
## Read in MaxQuant file: evidence.txt
infile <- read.table("evidence.txt", sep = "\t", header = TRUE)
```

One file is for annotation information, required to fill in `Condition` and `BioReplicate` for corresponding `Run` information. Users have to prepare as csv or txt file like 'Cox_maxquant_annotation.csv', which includes `Run`, `Condition`, and `BioReplicate` information, and load it in R.

```
## Read in annotation including condition and biological replicates: annotation.csv
annot <- read.csv("Cox_maxquant_annotation.csv", header = TRUE)
annot
```

```
##                        Raw.file Condition BioReplicate Experiment
## 1 20130510_EXQ1_IgPa_QC_UPS1_01      UPS1            1    UPS1_01
## 2 20130510_EXQ1_IgPa_QC_UPS1_02      UPS1            1    UPS1_02
## 3 20130510_EXQ1_IgPa_QC_UPS1_03      UPS1            1    UPS1_03
## 4 20130510_EXQ1_IgPa_QC_UPS1_04      UPS1            1    UPS1_04
## 5 20130510_EXQ1_IgPa_QC_UPS2_01      UPS2            2    UPS2_01
## 6 20130510_EXQ1_IgPa_QC_UPS2_02      UPS2            2    UPS2_02
## 7 20130510_EXQ1_IgPa_QC_UPS2_03      UPS2            2    UPS2_03
## 8 20130510_EXQ1_IgPa_QC_UPS2_04      UPS2            2    UPS2_04
##   IsotopeLabelType
## 1                L
## 2                L
## 3                L
## 4                L
## 5                L
## 6                L
## 7                L
## 8                L
```

4.3.2 Preprocessing with DDA experiment from MaxQuant output

`MaxQtoMSstatsFormat` function helps pre-processing for making right format of MSstats input from MaxQuant output. Basically, this function gets peptide ion intensity from 'evidence.txt' file. In addition, there are several steps to filter out or to modify the data in order to get required information.

Here is the summary of pre-processing steps in `MaxQtoMSstatsFormat` function (in orange box below).

In MSstats

MaxQtoMSstatsFormat

- Remove 'Contaminant', 'Potential.contaminant', 'Reverse', or 'Only.identified.by.site' protein ID.
- Match protein group ID with 'Proteins' column in evidence.txt
- Remove not unique peptides (assigned for more than one protein)
- Use Maximum(highest) intensity if there are multiple measurements for feature and run.
- Remove features that have 1 or 2 measurements across runs.
- Remove peptides including 'M' sequence.
- Remove proteins which have one peptide and charge in a protein.

- Replace intensity = 1 with zero if intensity < 1
- Log 2 transform for intensity
- Normalization
- **Extra step for imputation**
  - Decide cutoff for censored missing values among all log2(intensity).
  - If log2(intensity) < cutoff, replaces with zero and is considered as censored missing values (censoredInt='NA'), then, will be imputed.

```
## check options for converting format
?MaxQtoMSstatsFormat
```

```
quant <- MaxQtoMSstatsFormat(evidence=infile, annotation=annot, proteinGroups=proteinGroups)
```

```
## + Contaminant, + Reverse, + Only.identified.by.site, proteins are removed.

## Peptides including M in the sequence are removed.

## Peptides, that are used in more than one proteins, are removed.

## Peptide and charge, that have 1 or 2 measurements across runs, are removed.
```

```
## now 'quant' is ready for MSstats
head(quant)
```

```
##   ProteinName        PeptideSequence PrecursorCharge FragmentIon
## 1      A5A614            QVAESTPDIPK               2          NA
## 2   O00762ups            DPAATSVAAAR               2          NA
## 3   O00762ups FLTPCYHPNVDTQGNICLDILK               2          NA
## 4   O00762ups FLTPCYHPNVDTQGNICLDILK               3          NA
## 5   O00762ups             GAEPSGGAAR               2          NA
## 6   O00762ups          GISAFPESDNLFK               2          NA
##   ProductCharge IsotopeLabelType Condition BioReplicate
## 1            NA                L      UPS1            1
## 2            NA                L      UPS1            1
## 3            NA                L      UPS1            1
## 4            NA                L      UPS1            1
## 5            NA                L      UPS1            1
## 6            NA                L      UPS1            1
##                             Run  Intensity
## 1 20130510_EXQ1_IgPa_QC_UPS1_01         NA
## 2 20130510_EXQ1_IgPa_QC_UPS1_01 1144800000
## 3 20130510_EXQ1_IgPa_QC_UPS1_01   32793000
## 4 20130510_EXQ1_IgPa_QC_UPS1_01  566960000
## 5 20130510_EXQ1_IgPa_QC_UPS1_01   58709000
```

```
## 6 20130510_EXQ1_IgPa_QC_UPS1_01  861090000
```

4.3.3 Different options for MaxQuant in dataProcess

MaxQuant has certain or fixed threshold for intensity value internally as an parameter. Intensities less than the threshold are reported as NA. All missing values are NA in output from MaxQuant. In `dataProcess`, users need to use `censoredInt='NA'`. Users can used the same choice for other options.

```
cox.maxquant.proposed <- dataProcess(quant,
                                     normalization='equalizeMedian',
                                     summaryMethod="TMP",
                                     cutoffCensored="minFeature", censoredInt="NA",
                                     MBimpute=TRUE,
                                     maxQuantileforCensored=0.999)
```

Further steps is the same as in general workflow for DDA.

4.4 Suggested workflow with Progenesis output for DDA

This section describes steps and considerations to properly format data processed by Progenesis, prior to the `MSstats` analysis. In the following example, the same raw dataset from the previous section is used, but it is processed by Progenesis.

4.4.1 Load progenesis output

The output from Progenesis includes peptide ion-level quantification for each MS runs.

```
## First, get protein ID information
raw <- read.csv("Cox_progenesis.csv")
```

One file is for annotation information, required to fill in `Condition` and `BioReplicate` for corresponding `Run` information. Users have to prepare as csv or txt file like 'Cox_progenesis_annotation.csv', which includes `Run`, `Condition`, and `BioReplicate` information, and load it in R.

```
## Read in annotation including condition and biological replicates: annotation.csv
annot <- read.csv("Cox_progenesis_annotation.csv", header = TRUE)
annot
```

```
##                              Run Condition BioReplicate
## 1 X20130510_EXQ1_IgPa_QC_UPS1_01      UPS1            1
## 2 X20130510_EXQ1_IgPa_QC_UPS1_02      UPS1            2
## 3 X20130510_EXQ1_IgPa_QC_UPS1_03      UPS1            3
## 4 X20130510_EXQ1_IgPa_QC_UPS1_04      UPS1            4
## 5 X20130510_EXQ1_IgPa_QC_UPS2_01      UPS2            5
## 6 X20130510_EXQ1_IgPa_QC_UPS2_02      UPS2            6
## 7 X20130510_EXQ1_IgPa_QC_UPS2_03      UPS2            7
## 8 X20130510_EXQ1_IgPa_QC_UPS2_04      UPS2            8
```

4.4.2 Preprocessing with DDA experiment from progenesis output

`ProgenesistoMSstatsFormat` function helps pre-processing for making right format of MSstats input from Progenesis output. Basically, this function reformats wide format to long format. In addition, there are several steps to filter out or to modify the data in order to get required information.

Here is the summary of pre-processing steps in `ProgenesistoMSstatsFormat` function (in orange box below).

In MSstats

ProgenesistoMSstatsFormat

- Remove intensities with 'Use.in.quantitation' = FALSE.
- Remove not unique peptides (assigned for more than one protein)
- Use Maximum(highest) intensity if there are multiple measurements for feature and run.
- Remove features that have 1 or 2 measurements across runs.
- Remove proteins which have one peptide and charge in a protein.
- Missing values are left with zero value.

- Replace intensity = 1 with zero if intensity < 1
- Log 2 transform for intensity
- Normalization
- **Extra step for imputation**
  - Decide cutoff for censored missing values among all log2(intensity) > 0
  - If log2(intensity) < cutoff, log2(intensity) replaces with zero and is considered as censored missing values (censoredInt='0'), then, will be imputed.

```
## check options for converting format
?ProgenesistoMSstatsFormat
```

```
quant <- ProgenesistoMSstatsFormat(raw, annotation=annot)
```

```
## ** Peptides, that are used in more than one proteins, are removed.
## ** Multiple measurements in a feature and a run are summarized by summaryforMultipleRows.
```

```
## now 'quant' is ready for MSstats
head(quant)
```

```
##                 ProteinName
## 1 O00762ups|UBE2C_HUMAN_UPS
## 2 O00762ups|UBE2C_HUMAN_UPS
## 3 O00762ups|UBE2C_HUMAN_UPS
## 4 O00762ups|UBE2C_HUMAN_UPS
## 5 O00762ups|UBE2C_HUMAN_UPS
## 6 O00762ups|UBE2C_HUMAN_UPS
##                                     PeptideSequence PrecursorCharge
## 1 FLTPCYHPNVDTQGNICLDILK[5] C+57.0215|[17] C+57.0215               2
## 2 FLTPCYHPNVDTQGNICLDILK[5] C+57.0215|[17] C+57.0215               3
## 3 FLTPCYHPNVDTQGNICLDILK[5] C+57.0215|[17] C+57.0215               4
## 4                                   GISAFPESDNLFK               2
## 5                                WVGTIHGAAGTVYEDLR               2
## 6                                WVGTIHGAAGTVYEDLR               3
##   FragmentIon ProductCharge IsotopeLabelType Condition BioReplicate
## 1          NA            NA                L      UPS1            1
## 2          NA            NA                L      UPS1            1
## 3          NA            NA                L      UPS1            1
## 4          NA            NA                L      UPS1            1
## 5          NA            NA                L      UPS1            1
## 6          NA            NA                L      UPS1            1
```

```
##                               Run  Intensity
## 1 X20130510_EXQ1_IgPa_QC_UPS1_01  3790142.3
## 2 X20130510_EXQ1_IgPa_QC_UPS1_01 63386703.5
## 3 X20130510_EXQ1_IgPa_QC_UPS1_01   165145.8
## 4 X20130510_EXQ1_IgPa_QC_UPS1_01 98738397.0
## 5 X20130510_EXQ1_IgPa_QC_UPS1_01  9624505.6
## 6 X20130510_EXQ1_IgPa_QC_UPS1_01 12633723.8
```

4.4.3 Different options for Progenesis in dataProcess

Progenesis reports 0(zero) for missing values and does not have NA. Therefore,in `dataProcess`, users need to use `censoredInt='0'`. Users can used the same choice for other options.

```
cox.progenesis.proposed <- dataProcess(quant,
                                       normalization='equalizeMedian',
                                       summaryMethod="TMP",
                                       cutoffCensored="minFeature", censoredInt="0",
                                       MBimpute=TRUE,
                                       maxQuantileforCensored=0.999)
```

Further steps is the same as in general workflow for DDA.

Reference

Benjamini, Y., and Y. Hochberg. 1955. "Controlling the false discovery rate: a practical and powerful approach to multiple testing." *J.R. Statist. Soc. B* 57 (1): 289–300.

Cox, J., M. Y. Hein, C. A. Luber, I. Paron, N. Nagaraj, and M. Mann. 2014. "Accurate proteome-wide label-free quantification by delayed normalization and maximal peptide ratio extraction, termed MaxLFQ." *Mol. Cell. Proteomics* 13 (9): 2513–26.

Cox, Jürgen, and Matthias Mann. 2008. "MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification." *Nature Biotechnology* 26 (12): 1367–72.

Gatto, L., and K. S. Lilley. 2012. "MSnbase-an R/Bioconductor package for isobaric tagged mass spectrometry data visualization, processing and quantitation." *Bioinformatics* 28: 288–89.

MacLean, B., D. M. Tomazela, N. Shulman, M. Chambers, G. Finney, B. Frewen, R. Kern, D. L Tabb, D. C. Liebler, and M. J. MacCoss. 2010. "Skyline: An open source document editor for creating and analyzing targeted proteomics experiments." *Bioinformatics* 26–27: 966.

Mueller, L. N., O. Rinner, A. Schmidt, S. Letarte, B. Bodenmiller, M.-Y. Brusniak, O. Vitek, R. Aebersold, and M. Müller. 2007. "SuperHirn - a novel tool for high resolution LC-MS-based peptide/protein profiling." *Proteomics* 7: 3470–80.

Röst, H. L., G. Rosenberger, P. Navarro, L. Gillet, S. M. Miladinović, O. T. Schubert, W. Wolski, et al. 2014. "OpenSWATH enables automated, targeted analysis of data- independent acquisition MS data." *Nat. Biotechnol.* 32 (3): 219–23.

Sturm, Marc, Andreas Bertsch, Clemens Gröpl, Andreas Hildebrandt, Rene Hussong, Eva Lange, Nico Pfeifer, et al. 2008. "OpenMS – An open-source software framework for mass spectrometry." *BMC Bioinformatics* 9 (1): 163.