2019 Targeted Quantitative Proteomics Course Tutorial: MSstats for statistical analysis

Meena Choi

2019-07-24

Contents

License

Chapter 1

Prerequisites

MSstats is an R package and therefore requires a previously installed version of R (>=3.6.0). If you haven't previously installed MSstats through Skyline then follow the instructions below. Go to The R CRAN website and download R (vesion 3.6.0 or higher for MSstats 3.16.0) for your OS, install R and open an R console. MSstats itself depends on a number of other R packages, which need to be installed as well. These packages can be installed from CRAN and Bioconductor package repositories. Try executing the following commands in the R console. Click YES when you are asked to create a personal library and type a if you are asked to update all/some/none packages.

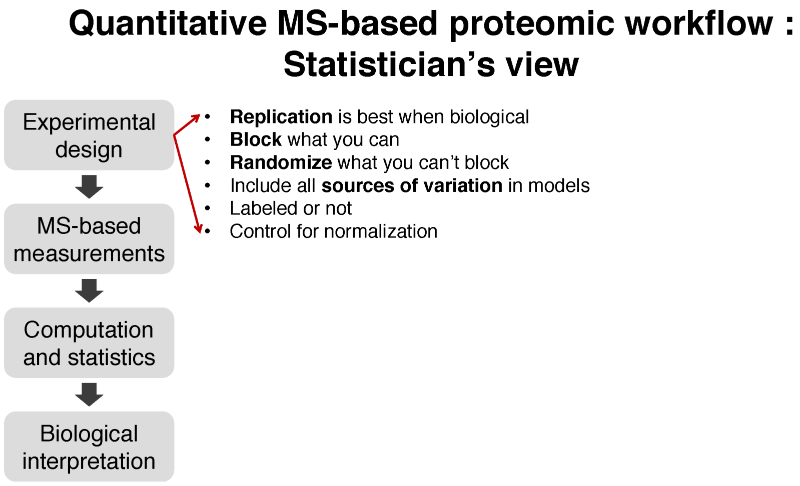
```
install.packages(c("gplots","lme4","ggplot2","ggrepel","reshape","reshape2", "data.table","Rcpp",
```

Now install all the packages we need for MSstats from the Bioconductor repository using biocLite(): ! If you are prompted to "Update all/some/none?" Press "a"

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("MSstats")
```

Chapter 2

Workflow in MSstats



Quantitative MS-based proteomic workflow :
Statistician's view

Experimental design

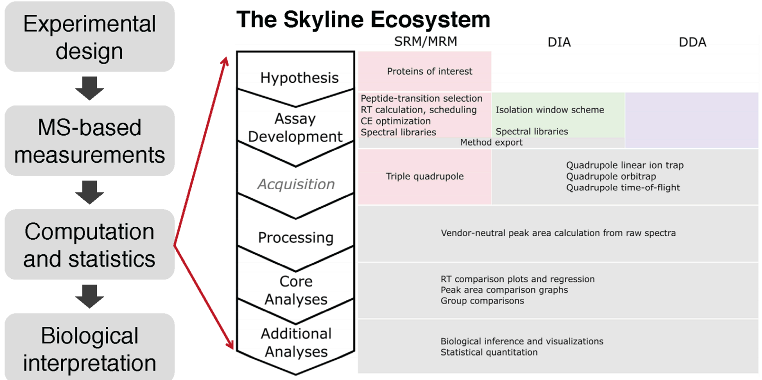
MS-based measurements

Computation and statistics

Biological interpretation

- **Replication** is best when biological
- **Block** what you can
- **Randomize** what you can't block
- Include all **sources of variation** in models
- Labeled or not
- Control for normalization

Quantitative MS-based proteomic workflow : Statistician's view

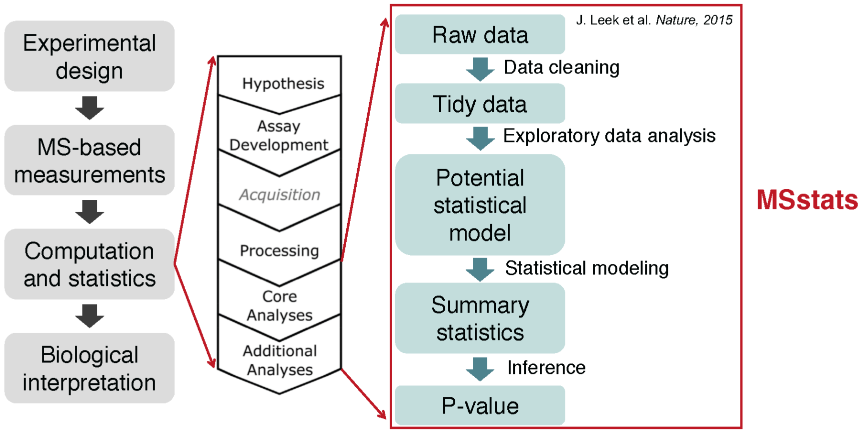


The Skyline Ecosystem

	SRM/MRM	DIA	DDA
Hypothesis	Proteins of interest		
Assay Development	Peptide-transition selection RT calculation, scheduling CE optimization Spectral libraries	Isolation window scheme Spectral libraries	
	Method export		
Acquisition	Triple quadrupole	Quadrupole linear ion trap Quadrupole orbitrap Quadrupole time-of-flight	
Processing	Vendor-neutral peak area calculation from raw spectra		
Core Analyses	RT comparison plots and regression Peak area comparison graphs Group comparisons		
Additional Analyses	Biological inference and visualizations Statistical quantitation		

L. Pino et al. *MassSpectrometryReviews, 2016*

Quantitative MS-based proteomic workflow : Statistician's view



J. Leek et al. *Nature, 2015*

Chapter 3

MSstats

3.1 Load MSstats

Once you have the package installed, load MSstats into an R session and have a look at the documentation by using the question mark ?. Note that in order to use MSstats, the package needs to be loaded every time you restart R.

Load MSstats first. Then you are ready to start MSstats.

```
library(MSstats)
?MSstats
```

3.2 Allowable data formats

`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`, `MaxQuant` and so on) that read raw spectral files , identify and quantify spectral peaks. The preferred structure of data for use in MSstats is a .csv file in a *long* format with at least 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.

```
##   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
```

3.3 Preparing the data for MSstats input

In skyline, go to **File** > **Export** > **Report...** > select **MSstats Input** > click **Export** > choose folder and save the file as **MSstats_Input.csv** > click **Save**

Create an RStudio project in the folder where you saved the file exported from Skyline. From the menu, select **File** > **New Project...**, then select **Existing Directory** and choose the directory where you downloaded this script and the example datasets for this tutorial. All the output files we'll be creating in this tutorial will be saved in the 'working directory' that now has been set by RStudio.

note : add captue

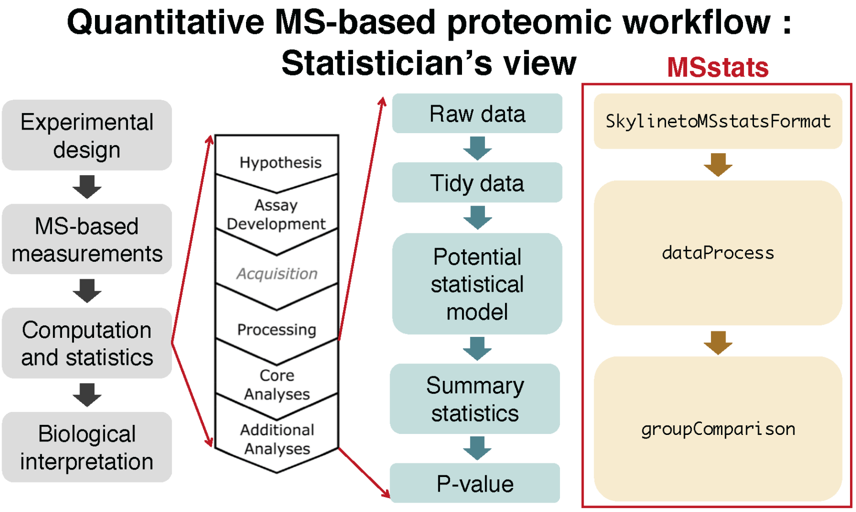
Check where you are using **getwd()**

```
getwd()
```

```
## [1] "/Users/meenachoi/Dropbox/visits/2019/07_UW/MSstats-handson"
```

3.4 Convert to MSstats required format (Data cleaning)

Let's start preprocessing steps to make required input format for MSstats from output from diverse output of spectral processing tools.



- Data input support for various data acquisition methods : DDA, DIA, SRM
- Interoperability with existing computational tools : - Converter functions for 7 data processin tools : `SkylinetoMSstatsFormat`, `MaxQtoMSstatsFormat`, `OpenMStoMSstatsFormat`, `ProgenesistoMSstatsFormat`, `PDtoMSstatsFormat`, `SpectronauttoMSstatsFormat`, `OpenSWATHtoMSstatsFormat`, `DIAUmpiretoMSstatsFormat` - Consistent data cleaning steps across converter functions : filter multiple measurements, shared peptides, etc. - Generate the same format of data from diverse type of data format.

3.4.1 Read data

Then, read in data as it comes out of Skyline.

```
# Read output from skyline
raw <- read.csv('data/MSstats_Input.csv')
```

Use head() function to display the first few (6) rows of the data, and colnames() to see what the columns are. They should match the columns we previewed in Skyline.

```
head(raw)
```

```
##   Protein.Name Peptide.Modified.Sequence Precursor.Charge Fragment.Ion
## 1    NP_036629       C[+57]SLPRPWALTFSYGR                2          y10
## 2    NP_036629       C[+57]SLPRPWALTFSYGR                2          y10
## 3    NP_036629       C[+57]SLPRPWALTFSYGR                2          y10
## 4    NP_036629       C[+57]SLPRPWALTFSYGR                2          y10
## 5    NP_036629       C[+57]SLPRPWALTFSYGR                2          y10
## 6    NP_036629      C[+57]SLPRPWALTFSYGR                2          y10
##   Product.Charge Isotope.Label.Type Condition BioReplicate      File.Name
## 1              1              light  Diseased          102 D_102_REP1.raw
## 2              1              light  Diseased          102 D_102_REP2.raw
## 3              1              light  Diseased          102 D_102_REP3.raw
## 4              1              light  Diseased          103 D_103_REP1.raw
## 5              1              light  Diseased          103 D_103_REP2.raw
## 6              1              light  Diseased          103 D_103_REP3.raw
##    Area Standard.Type Truncated
## 1 14516                   False
## 2  9607                   False
## 3  7480                   False
## 4  5692                   False
## 5  5953                   False
## 6   646                    True
```

```
colnames(raw)
```

```
##  [1] "Protein.Name"              "Peptide.Modified.Sequence"
##  [3] "Precursor.Charge"          "Fragment.Ion"
##  [5] "Product.Charge"            "Isotope.Label.Type"
##  [7] "Condition"                 "BioReplicate"
##  [9] "File.Name"                 "Area"
## [11] "Standard.Type"             "Truncated"
```

Use another useful function str(), to display a summary of each column in our raw dataframe.

```
str(raw)
```

```
## 'data.frame':    30282 obs. of  12 variables:
##  $ Protein.Name             : Factor w/ 48 levels "NP_001007697",..: 11 11 11 11 11
##  $ Peptide.Modified.Sequence: Factor w/ 125 levels "AAPITQYLK","AAVFNHFISDGVK",..:
##  $ Precursor.Charge         : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ Fragment.Ion             : Factor w/ 11 levels "y10","y11","y12",..: 1 1 1 1 1 1
##  $ Product.Charge           : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ Isotope.Label.Type       : Factor w/ 1 level "light": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Condition                : Factor w/ 2 levels "Diseased","Healthy": 1 1 1 1 1 1
##  $ BioReplicate             : int  102 102 102 103 103 103 108 108 108 138 ...
##  $ File.Name                : Factor w/ 42 levels "D_102_REP1.raw",..: 1 2 3 4 5 6
##  $ Area                     : Factor w/ 26510 levels "#N/A","0","1",..: 4281 26072
##  $ Standard.Type            : Factor w/ 2 levels "","Global Standard": 1 1 1 1 1 1
##  $ Truncated                : Factor w/ 3 levels "","False","True": 2 2 2 2 2 3 2 3
```

Notice that some rows are type “Factor” and some rows are type “int”. Although we will avoid any object type conflicts in this tutorial, you can learn more about object types through the R Documentation ?typeof(). For now, it’s enough to just acknowledge that they exist!

There are some column named differently than required input. The information for `Condition` and `BioReplicate` is missing. Let's do preliminary check for this input.

```
# total number of unique protein name
length(unique(raw$Protein))
```

```
## [1] 48
```

```
# several isotopic peaks for peptide charge
unique(raw$FragmentIon)
```

```
## NULL
```

```
# unique FileName, which is MS run.
unique(raw$FileName)
```

```
## NULL
```

```
# 'Truncated' column
unique(raw$Truncated)
```

```
## [1] False True
## Levels:  False True
```

```
# count table for 'Truncated' column
xtabs(~Truncated, raw)
```

```
## Truncated
##       False  True
##   251 29306   725
```

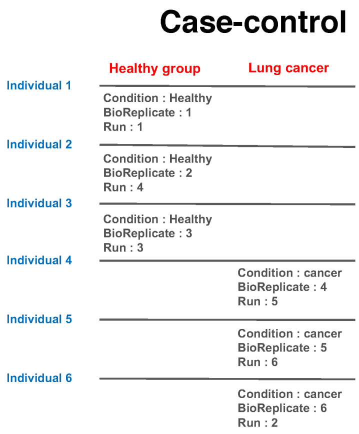
```
# count which 'Truncated' is 'True'
sum(raw$Truncated == 'True')
```

```
## [1] 725
```

3.4.2 Common mistake for annotation file : Incorrect `BioReplicate` information

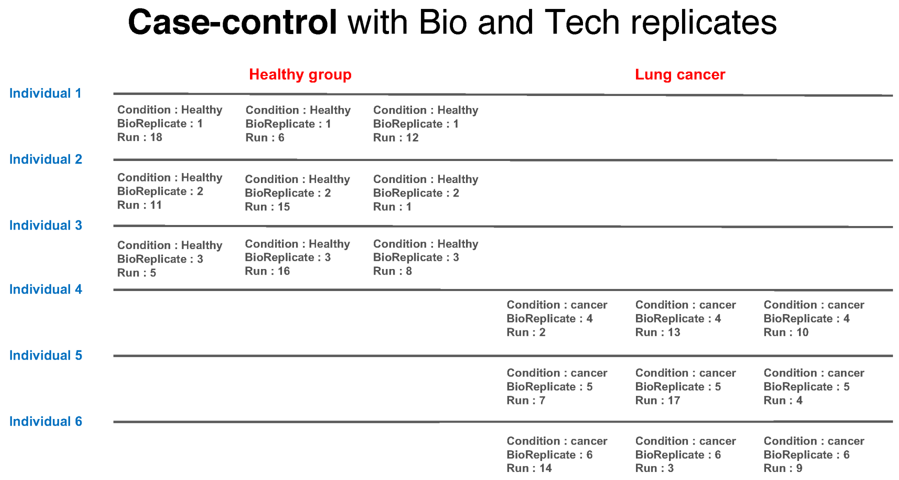
`MSstats` distinguish the design of experiment as group comparison, time course, paired design, with the combination of `Condition`, `BioReplicate`, and `Run`.

3.4.2.1 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). Overall, in this example there are 2 × 3 = 6 mass spectrometry runs.

The most important is that 1) subject IDs for disease group are completely different thatn subject IDs for control group 2) `Run` is not order of spectral acquisition, but just unique MS run ID.

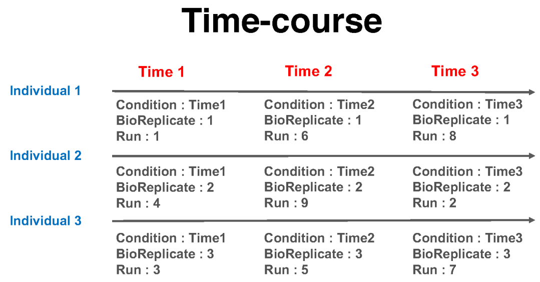


In addition, if each subject has e technical replicate runs (in general technical replicates are not required, and their number per sample may vary). there are

$2 \times 3 \times 3 = 18$ mass spectrometry runs.

3.4.2.2 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 3 time points, Time1, Time2, and Time3 (in general the number of times can vary). There are 3 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 $3 \times 3 = 9$ mass spectrometry runs.

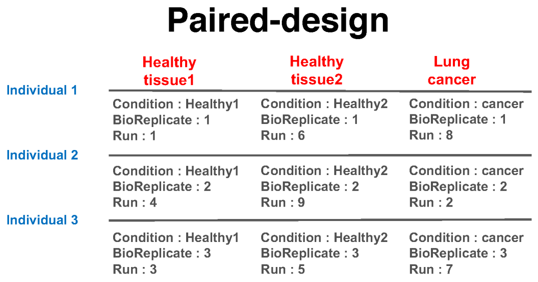


Time-course

	Time 1	Time 2	Time 3
Individual 1	Condition : Time1 BioReplicate : 1 Run : 1	Condition : Time2 BioReplicate : 1 Run : 6	Condition : Time3 BioReplicate : 1 Run : 8
Individual 2	Condition : Time1 BioReplicate : 2 Run : 4	Condition : Time2 BioReplicate : 2 Run : 9	Condition : Time3 BioReplicate : 2 Run : 2
Individual 3	Condition : Time1 BioReplicate : 3 Run : 3	Condition : Time2 BioReplicate : 3 Run : 5	Condition : Time3 BioReplicate : 3 Run : 7

3.4.2.3 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 3 subjects (in general the number of patients can vary). There are three conditions per subject (in general the number of conditions per subject can exceed two). Overall, in this example there are $2 \times 3 = 6$ mass spectrometry runs. `BioReplicate` should indicate each individual ID.



Paired-design

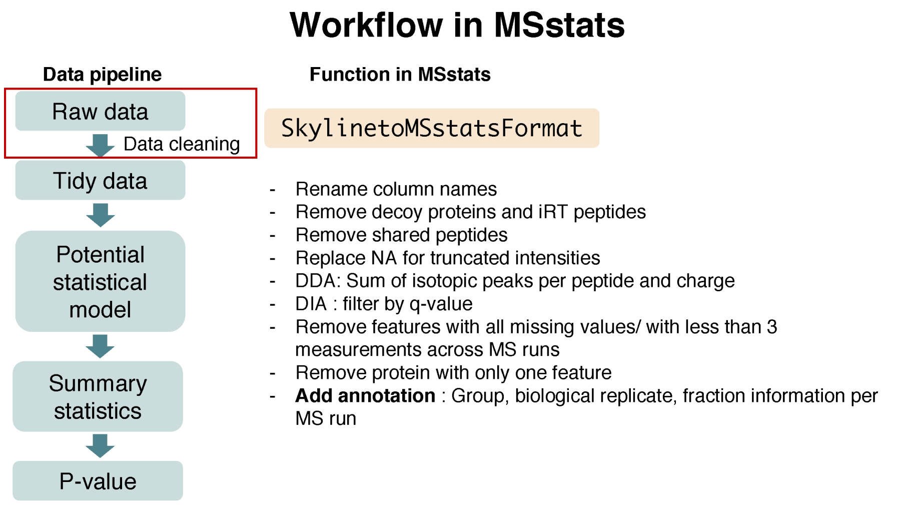
	Healthy tissue1	Healthy tissue2	Lung cancer
Individual 1	Condition : Healthy1 BioReplicate : 1 Run : 1	Condition : Healthy2 BioReplicate : 1 Run : 6	Condition : cancer BioReplicate : 1 Run : 8
Individual 2	Condition : Healthy1 BioReplicate : 2 Run : 4	Condition : Healthy2 BioReplicate : 2 Run : 9	Condition : cancer BioReplicate : 2 Run : 2
Individual 3	Condition : Healthy1 BioReplicate : 3 Run : 3	Condition : Healthy2 BioReplicate : 3 Run : 5	Condition : cancer BioReplicate : 3 Run : 7

3.4.3 Preprocessing with `SkylinetoMSstatsFormat`

Now we'll adapt the column scheme of the dataset so that it fits MSstats input format. The SkylinetoMSstatsFormat() function helps pre-processing for making the Skyline export play nicely with MSstats functions. For example, it removes iRT peptides, renames some column name, and replaces truncated peak intensities with NA.

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. `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).



Workflow in MSstats

Data pipeline

Raw data

Data cleaning

Tidy data

Potential statistical model

Summary statistics

P-value

Function in MSstats

SkylinetoMSstatsFormat

- Rename column names
- Remove decoy proteins and iRT peptides
- Remove shared peptides
- Replace NA for truncated intensities
- DDA: Sum of isotopic peaks per peptide and charge
- DIA : filter by q-value
- Remove features with all missing values/ with less than 3 measurements across MS runs
- Remove protein with only one feature
- **Add annotation** : Group, biological replicate, fraction information per MS run

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

```
?SkylinetoMSstatsFormat
```

```
# reformating and pre-processing for Skyline output.
raw_msstats <- SkylinetoMSstatsFormat(raw, filter_with_Qvalue = FALSE)
```

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

## Warning in SkylinetoMSstatsFormat(raw, filter_with_Qvalue = FALSE): NAs
## introduced by coercion

## ** Truncated peaks are replaced with NA.

## ** 0 features have all NAs or zero intensity values and are removed.
```

You may see a warning message telling you that “NAs introduced by coercion”. This is okay! Let’s check what changed in our processed dataframe using the str() command.

```
str(raw_msstats)
```

```
## 'data.frame':    30282 obs. of  12 variables:
##  $ ProteinName     : Factor w/ 48 levels "NP_001007697",..: 11 11 11 11 11 11 11 11 11 11 ...
##  $ PeptideSequence : Factor w/ 125 levels "AAPITQYLK","AAVFNHFISDGVK",..: 21 21 21 21 21 21 21
##  $ PrecursorCharge : int  2 2 2 2 2 2 2 2 2 2 ...
##  $ FragmentIon     : Factor w/ 11 levels "y10","y11","y12",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ ProductCharge   : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ IsotopeLabelType: Factor w/ 1 level "light": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Condition       : Factor w/ 2 levels "Diseased","Healthy": 1 1 1 1 1 1 1 1 1 1 ...
##  $ BioReplicate    : int  102 102 102 103 103 103 108 108 108 138 ...
##  $ Run             : Factor w/ 42 levels "D_102_REP1.raw",..: 1 2 3 4 5 6 7 8 9 10 ...
##  $ Intensity       : num  14516 9607 7480 5692 5953 ...
##  $ StandardType    : Factor w/ 2 levels "","Global Standard": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Truncated       : Factor w/ 3 levels "","False","True": 2 2 2 2 2 3 2 3 2 2 ...
```

Notice some columns were renamed, and also one column (previously “Area”, now renamed “Intensity”), changed object type! Remember I said we would avoid type conflicts in this tutorial? It’s enough just to acknowledge that this changed, but if you continue using R for your own data, you will likely run into object conflicts in the future and hopefully knowing this helps you troubleshoot!

3.4.4 Preliminary check

So far, we’ve only looked at the processed dataframe raw_msstats at a high level. Let’s pull out just one column from the dataframe, the ProteinName column, and ask R how many unique proteins are in our data.

```
length(unique(raw_msstats$ProteinName))
```

```
## [1] 48
```

```
sum(is.na(raw_msstats$Intensity))
```

```
## [1] 976
```

```
sum(!is.na(raw_msstats$Intensity) & raw_msstats$Intensity==0)
```

```
## [1] 18
```

You should have 48 levels listed, which is exactly what we expected from our Skyline document! Let's practice with one more, how many unique peptides are in our dataframe?

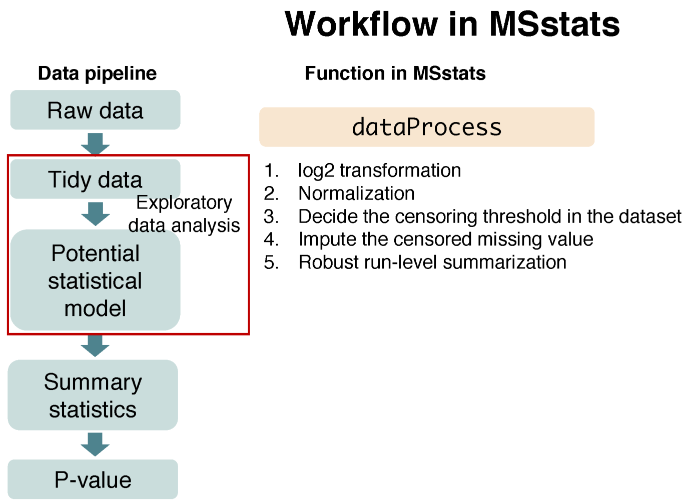
```
unique(raw_msstats$PeptideSequence)
```

```
##   [1] C[+57]SLPRPWALTFSYGR      LGGEEVSVAC[+57]K
##   [3] VGQPGDAGAAGPVAPLC[+57]PGR GSYNLQDLLAQAK
##   [5] TSDQIHFFFAK               LQPLDFK
##   [7] SQLPGIIAEGR               DFATVYVDAVK
##   [9] DYVSQFESSTLGK             TGTNLMDFLSR
##  [11] LMSPEEKPAPAAK             GTITSIAALDDPK
##  [13] TQTPVQGC[+57]HLEGVTGHK    IFPENNIK
##  [15] SLLNSLEEAK                ASGIIDTLFQDR
##  [17] C[+57]IVDGDDR             YLMFFAC[+57]TILVPK
##  [19] WVLTVAHC[+57]FEGR         HTNNGMIC[+57]LTSLLR
##  [21] YDC[+57]VAMNHHGVIR        VFWIEVALFWR
##  [23] SDFQVPC[+57]QYSQQLK       SFSC[+57]EVEILEGDK
##  [25] VGPAVELALAR               ENSSNILDNLLSR
##  [27] TQEQAQALR                 YHGVTGLVVMDK
##  [29] YQIFANTGHFK               C[+57]WAQDPTERPDFGQIK
##  [31] IGVHTGPVC[+57]AGVVGLK     TAC[+57]VLPAPAGPSQGK
##  [33] ALIHC[+57]LHMS            LTHGDFTWTTK
##  [35] WWGQEITELAQGPGR           LTTDHTPER
##  [37] AGDQILAINEINVK            AGVVAEYPAEK
##  [39] YGLDLGSLLVR               ISAEWGEFIK
##  [41] AGSWQITMK                 FAEDHFAHEATK
##  [43] DVNEAIQWMEEK              HQAFENEVNGR
##  [45] HQLLEAEMLAR               AAPITQYLK
##  [47] TLNSINIAVFSK              TVEHPFSVEEFVLPK
##  [49] DLTGFPQGADQR              AIAYLNTGYQR
##  [51] ATIDQNLEDLR               LNHQMEGLAFQMK
##  [53] NLAPLVEDVQSK              ELEEQLGPVAEETR
##  [55] LGPLVEQGR                 LQAEIFQAR
##  [57] LGVIVSAYMHYSK             HFLIETGPK
##  [59] WTNPDGTTSK                HEEEVERPAVEK
##  [61] SLVIQKPSEENAPK            ALYSEYTDGTFTK
##  [63] ENEGTYYGPDGR              ETFTYEWTVPK
##  [65] TYIWQIPER                 ENLSPPLGEC[+57]LLER
##  [67] ELLDSYIDGR                 YGFYTHVFR
##  [69] AFMDC[+57]C[+57]NYITK      AAVFNHFISDGVK
##  [71] ATETQGVNLLFSSR             EEPSADALLPIDC[+57]R
##  [73] AIEDYVNEFSAR               GLIDEANQDFTNR
##  [75] MSPVPDLVPGSFK              SQLQEGPPEWK
##  [77] MHPELGSFYDSR               FGSLTSNFK
##  [79] DNC[+57]C[+57]ILDER        TFYQIGDSWEK
##  [81] ATGVFTTLQPLR               WLPSTSPVTGYR
##  [83] IAWESPQGQVSR               DYFISC[+57]PGR
##  [85] C[+57]NADPGLSALLSDHR       GATYAFSGSHYWR
##  [87] SGYTFQLLR                  EENGDFASFR
##  [89] LIC[+57]EATNFSPK           SGFTTEPVTVEAK
##  [91] SNSMVTLGC[+57]LVK          VTSAAFPSPIEK
##  [93] LALDNGGLAR                 DQGPDVLLAK
##  [95] TLFSVLPGLK                 YNAELESGNQFVLYR
##  [97] DGAETLYSFK                 C[+57]SSLLWAGAAWLR
##  [99] NLGVVVAPHALR               AEQGAYLGPLPYK
## [101] TVSWAVTPK                  AISYLISGYQR
## [103] LADLPGNYITK                VNTLPLNFDK
## [105] SPQGLGASTAEISAR            TDEDVPSGPPR
## [107] YANVIAYDHSR                LFDELVVDK
## [109] ETGLMAFTNLK                VLIVEPEGIK
## [111] MLSGFIPLKPTVK              LQTEGDGIYTLNSEK
## [113] SVVDIGLIK                  IAELFSDLEER
## [115] FSISTDYSLK                 EVLPELGIK
## [117] ALYQAEAFVADFK              IAELFSELDER
## [119] IFSQQADLSR                 IITGNALFIDK
## [121] IQELVSGLK                  DVFSQQADLSR
## [123] HLNGFSVPR                  VVLSGSDATLAYSAFK
## [125] AFGLSSPR
## 125 Levels: AAPITQYLK AAVFNHFISDGVK AEQGAYLGPLPYK ... YQIFANTGHFK
```

What else can you summarize about the dataframe?

3.5 Data processing - Normalization and run summarization

Let's start processing steps. It includes log transformation of intensities, normalization and run-level summarization.


Workflow in MSstats
Data pipeline
Raw data
Tidy data
Exploratory data analysis
Potential statistical model
Summary statistics
P-value
Function in MSstats
dataProcess
1. log2 transformation
2. Normalization
3. Decide the censoring threshold in the dataset
4. Impute the censored missing value
5. Robust run-level summarization

3.5.1 Normalizing and summarizing data with dataProcess

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

```
?dataProcess
```

`dataProcess` perform (1) normalization first. The default option for normalization is `equalizeMedians`. 'equalizeMedians' fits for label-based SRM experiments, which we can use reference signals. There are three more options for normalization. Depending on the suitable assumption for your experiment, you can choose one of them.

Then, (2) run level summarization will be performed including missing value imputation by accerelated failure model and robust parameter estimation by TMP (Tukey's median polish).

Below show the default for all options in dataProcess except `censoredInt`. `censoredInt='0'` should be used for Skyline output.

Note: do pay attention to the default options, which may not be appropriate in some situations and need to be changed. For example, the default option for normalization is equalizeMedians. If you have a spiked in standard, you may set this to globalStandards and define the standard with nameStandards.

```
quant_tmp <- dataProcess(raw = raw_msstats,
                         normalization="globalStandards",
                         nameStandards="VVLSGSDATLAYSAFK",
                         censoredInt = '0')
```

```
##
```

```
##   Summary of Features :
##                           count
## # of Protein                48
## # of Peptides/Protein      1-5
## # of Transitions/Peptide   3-7
##
##   Summary of Samples :
##                             Diseased Healthy
## # of MS runs                     21      21
## # of Biological Replicates        7       7
## # of Technical Replicates         3       3
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   2%
  |
  |===                                                              |   4%
  |
  |====                                                             |   6%
  |
  |=====                                                            |   8%
  |
  |=======                                                          |  10%
  |
  |========                                                         |  12%
  |
  |=========                                                        |  15%
  |
  |===========                                                      |  17%
  |
  |============                                                     |  19%
  |
  |==============                                                   |  21%
  |
  |===============                                                  |  23%
  |
  |================                                                 |  25%
  |
  |==================                                               |  27%
  |
  |===================                                              |  29%
  |
  |====================                                             |  31%
  |
  |======================                                           |  33%
```

```
|
|=======================                                          |  35%
|
|========================                                         |  38%
|
|==========================                                       |  40%
|
|===========================                                      |  42%
|
|============================                                     |  44%
|
|==============================                                   |  46%
|
|===============================                                  |  48%
|
|================================                                 |  50%
|
|==================================                               |  52%
|
|===================================                              |  54%
|
|=====================================                            |  56%
|
|======================================                           |  58%
|
|=======================================                          |  60%
|
|=========================================                        |  62%
|
|==========================================                       |  65%
|
|===========================================                      |  67%
|
|=============================================                    |  69%
|
|==============================================                   |  71%
|
|===============================================                  |  73%
|
|=================================================                |  75%
|
|==================================================               |  77%
|
|===================================================              |  79%
|
|=====================================================            |  81%
```

```
  |
  |======================================================           |  83%
  |
  |========================================================         |  85%
  |
  |=========================================================        |  88%
  |
  |==========================================================       |  90%
  |
  |============================================================     |  92%
  |
  |=============================================================    |  94%
  |
  |==============================================================   |  96%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

Let's check output from `dataProcess`.

```
# show the name of outputs
names(quant_tmp)
```

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

```
# show reformatted and normalized data.
# 'ABUNDANCE' column has normalized log2 transformed intensities.
head(quant_tmp$ProcessedData)
```

```
##            PROTEIN              PEPTIDE TRANSITION
## 23899 NP_001007697 C[+57]SSLLWAGAAWLR_2       y3_1
## 23857 NP_001007697 C[+57]SSLLWAGAAWLR_2       y4_1
## 23815 NP_001007697 C[+57]SSLLWAGAAWLR_2       y5_1
## 23773 NP_001007697 C[+57]SSLLWAGAAWLR_2       y6_1
## 23731 NP_001007697 C[+57]SSLLWAGAAWLR_2       y7_1
## 23689 NP_001007697 C[+57]SSLLWAGAAWLR_2       y8_1
##                         FEATURE LABEL GROUP_ORIGINAL SUBJECT_ORIGINAL RUN
## 23899 C[+57]SSLLWAGAAWLR_2_y3_1     L       Diseased              102   1
## 23857 C[+57]SSLLWAGAAWLR_2_y4_1     L       Diseased              102   1
## 23815 C[+57]SSLLWAGAAWLR_2_y5_1     L       Diseased              102   1
## 23773 C[+57]SSLLWAGAAWLR_2_y6_1     L       Diseased              102   1
## 23731 C[+57]SSLLWAGAAWLR_2_y7_1     L       Diseased              102   1
## 23689 C[+57]SSLLWAGAAWLR_2_y8_1     L       Diseased              102   1
##       GROUP SUBJECT INTENSITY SUBJECT_NESTED  ABUNDANCE FRACTION
## 23899     1       1        24            1.1  4.1151229        1
## 23857     1       1       182            1.1  7.0379550        1
## 23815     1       1       782            1.1  9.1411852        1
## 23773     1       1      1580            1.1 10.1558692        1
## 23731     1       1         1            1.1  0.0000000        1
## 23689     1       1         2            1.1  0.5301604        1
##          originalRUN censored
## 23899 D_102_REP1.raw     TRUE
## 23857 D_102_REP1.raw    FALSE
## 23815 D_102_REP1.raw    FALSE
## 23773 D_102_REP1.raw    FALSE
## 23731 D_102_REP1.raw     TRUE
## 23689 D_102_REP1.raw     TRUE
```

```
# This table includes run-level summarized log2 intensities. (column : LogIntensities)
# Now one summarized log2 intensities per Protein and Run.
# NumMeasuredFeature : show how many features are used for run-level summarization.
#         If there is no missing value, it should be the number of features in certain
# MissingPercentage : the number of missing features / the number of features in certa
head(quant_tmp$RunlevelData)
```

```
##   RUN      Protein LogIntensities NumMeasuredFeature MissingPercentage
## 1   1 NP_001007697       12.84670                 12                 0
## 2   2 NP_001007697       13.52696                 12                 0
## 3   3 NP_001007697       13.47479                 12                 0
## 4   4 NP_001007697       11.92295                 12                 0
## 5   5 NP_001007697       11.87338                 12                 0
## 6   6 NP_001007697       11.91218                 12                 0
##   more50missing NumImputedFeature    originalRUN GROUP GROUP_ORIGINAL
## 1         FALSE                 0 D_102_REP1.raw     1       Diseased
## 2         FALSE                 0 D_102_REP2.raw     1       Diseased
## 3         FALSE                 0 D_102_REP3.raw     1       Diseased
## 4         FALSE                 0 D_103_REP1.raw     1       Diseased
## 5         FALSE                 0 D_103_REP2.raw     1       Diseased
## 6         FALSE                 0 D_103_REP3.raw     1       Diseased
##   SUBJECT_ORIGINAL SUBJECT_NESTED SUBJECT
## 1              102            1.1       1
## 2              102            1.1       1
## 3              102            1.1       1
## 4              103            1.2       2
## 5              103            1.2       2
## 6              103            1.2       2
```

```
# show which summarization method is used.
head(quant_tmp$SummaryMethod)
```

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

Note that the above command, although we only specify two parameters, raw and censoredInt, is running with all the other parameters, they're just automatically set to default. So, for example, the line above is the same as the line below. Note: censoredInt=‘NA’ for the input from other spectral tools.

```
quant_tmp <- dataProcess(raw = raw_msstats,
                        logTrans = 2,
                        normalization = "equalizeMedians",
                        fillIncompleteRows = TRUE,
                        featureSubset = "all",
                        remove_uninformative_feature_outlier = FALSE,
                        summaryMethod = "TMP",
                        censoredInt = "0",
                        cutoffCensored = "minFeature",
                        MBimpute = TRUE)
```

```
##
##   Summary of Features :
##                          count
## # of Protein                48
## # of Peptides/Protein      1-5
## # of Transitions/Peptide   3-7
##
##   Summary of Samples :
##                            Diseased Healthy
## # of MS runs                     21      21
## # of Biological Replicates        7       7
## # of Technical Replicates         3       3
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   2%
  |
  |===                                                              |   4%
  |
  |====                                                             |   6%
  |
  |=====                                                            |   8%
  |
  |=======                                                          |  10%
  |
  |========                                                         |  12%
  |
  |=========                                                        |  15%
  |
  |===========                                                      |  17%
  |
  |============                                                     |  19%
  |
  |==============                                                   |  21%
  |
  |===============                                                  |  23%
  |
  |================                                                 |  25%
  |
  |==================                                               |  27%
  |
  |===================                                              |  29%
  |
  |====================                                             |  31%
  |
  |======================                                           |  33%
  |
  |=======================                                          |  35%
  |
  |========================                                         |  38%
  |
  |==========================                                       |  40%
  |
  |===========================                                      |  42%
  |
  |============================                                     |  44%
  |
  |==============================                                   |  46%
  |
  |===============================                                  |  48%
  |
  |================================                                 |  50%
  |
  |==================================                               |  52%
  |
  |===================================                              |  54%
  |
  |=====================================                            |  56%
  |
  |======================================                           |  58%
  |
  |=======================================                          |  60%
  |
  |=========================================                        |  62%
  |
  |==========================================                       |  65%
  |
  |===========================================                      |  67%
  |
  |=============================================                    |  69%
  |
  |==============================================                   |  71%
  |
  |===============================================                  |  73%
  |
  |=================================================                |  75%
  |
  |==================================================               |  77%
  |
  |===================================================              |  79%
  |
  |=====================================================            |  81%
  |
  |======================================================           |  83%
  |
  |========================================================         |  85%
  |
  |=========================================================        |  88%
  |
  |==========================================================       |  90%
  |
  |============================================================     |  92%
  |
  |=============================================================    |  94%
  |
  |==============================================================   |  96%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

Output of the dataProcess() function contains the processed and run-level summarized data as well as relevant information for the summarization step. Let's unpack the output from dataProcess(), which we named quant_tmp above, so that we can be more familiar with what we just did to our data.

3.5.2 No normalization

```
quant_nonorm <- dataProcess(raw = raw_msstats,
                            normalization=FALSE,
                            censoredInt = '0')
```

```
##
##   Summary of Features :
##                          count
## # of Protein                48
## # of Peptides/Protein      1-5
## # of Transitions/Peptide   3-7
##
##   Summary of Samples :
##                            Diseased Healthy
## # of MS runs                     21      21
## # of Biological Replicates        7       7
## # of Technical Replicates         3       3
##
  |
  |                                                                 |   0%
  |
  |=                                                                |   2%
  |
  |===                                                              |   4%
  |
  |====                                                             |   6%
  |
  |=====                                                            |   8%
  |
  |=======                                                          |  10%
  |
  |========                                                         |  12%
  |
  |=========                                                        |  15%
  |
  |===========                                                      |  17%
  |
  |============                                                     |  19%
  |
  |==============                                                   |  21%
  |
  |===============                                                  |  23%
  |
  |================                                                 |  25%
  |
  |==================                                               |  27%
  |
  |===================                                              |  29%
  |
  |====================                                             |  31%
  |
|======================                                           |  33%
|
|=======================                                          |  35%
|
|========================                                         |  38%
|
|==========================                                       |  40%
|
|===========================                                      |  42%
|
|============================                                     |  44%
|
|==============================                                   |  46%
|
|===============================                                  |  48%
|
|================================                                 |  50%
|
|==================================                               |  52%
|
|===================================                              |  54%
|
|=====================================                            |  56%
|
|======================================                           |  58%
|
|=======================================                          |  60%
|
|=========================================                        |  62%
|
|==========================================                       |  65%
|
|===========================================                      |  67%
|
|=============================================                    |  69%
|
|==============================================                   |  71%
|
|===============================================                  |  73%
|
|=================================================                |  75%
|
|==================================================               |  77%
|
|===================================================              |  79%
|
  |=====================================================            |  81%
  |
  |======================================================           |  83%
  |
  |========================================================         |  85%
  |
  |=========================================================        |  88%
  |
  |==========================================================       |  90%
  |
  |============================================================     |  92%
  |
  |=============================================================    |  94%
  |
  |==============================================================   |  96%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

What's the difference between two normalization methods? With different methods for normalization, the summarized values are different. Let's pull out the RunlevelData from each object, the quant_tmp object that used global standard peptide for normalization and the quant_nonorm object that did not apply any normalization. Specifically, look at the column named LogIntensities.

```
head(quant_tmp$RunlevelData)
```

```
##   RUN      Protein LogIntensities NumMeasuredFeature MissingPercentage
## 1   1 NP_001007697       12.80623                 12                 0
## 2   2 NP_001007697       13.39071                 12                 0
## 3   3 NP_001007697       13.11670                 12                 0
## 4   4 NP_001007697       12.18845                 12                 0
## 5   5 NP_001007697       12.33439                 12                 0
## 6   6 NP_001007697       12.24504                 12                 0
##   more50missing NumImputedFeature    originalRUN GROUP GROUP_ORIGINAL
## 1         FALSE                 0 D_102_REP1.raw     1       Diseased
## 2         FALSE                 0 D_102_REP2.raw     1       Diseased
## 3         FALSE                 0 D_102_REP3.raw     1       Diseased
## 4         FALSE                 0 D_103_REP1.raw     1       Diseased
## 5         FALSE                 0 D_103_REP2.raw     1       Diseased
## 6         FALSE                 0 D_103_REP3.raw     1       Diseased
##   SUBJECT_ORIGINAL SUBJECT_NESTED SUBJECT
## 1              102            1.1       1
## 2              102            1.1       1
## 3              102            1.1       1
## 4              103            1.2       2
## 5              103            1.2       2
## 6              103            1.2       2
```

```
head(quant_nonorm$RunlevelData)
```

```
##   RUN      Protein LogIntensities NumMeasuredFeature MissingPercentage
## 1   1 NP_001007697       13.27635                 12                 0
## 2   2 NP_001007697       13.68839                 12                 0
## 3   3 NP_001007697       13.26387                 12                 0
## 4   4 NP_001007697       12.23041                 12                 0
## 5   5 NP_001007697       12.06794                 12                 0
## 6   6 NP_001007697       11.82927                 12                 0
##   more50missing NumImputedFeature    originalRUN GROUP GROUP_ORIGINAL
## 1         FALSE                 0 D_102_REP1.raw     1       Diseased
## 2         FALSE                 0 D_102_REP2.raw     1       Diseased
## 3         FALSE                 0 D_102_REP3.raw     1       Diseased
## 4         FALSE                 0 D_103_REP1.raw     1       Diseased
## 5         FALSE                 0 D_103_REP2.raw     1       Diseased
## 6         FALSE                 0 D_103_REP3.raw     1       Diseased
##   SUBJECT_ORIGINAL SUBJECT_NESTED SUBJECT
## 1              102            1.1       1
## 2              102            1.1       1
## 3              102            1.1       1
## 4              103            1.2       2
## 5              103            1.2       2
## 6              103            1.2       2
```

3.5.3 Visualization of processed data

Next we'll be using the dataProcessPlots function to visualize our data. To get the documentation for this function, we can again use the ? symbol.

```
?dataProcessPlots
```

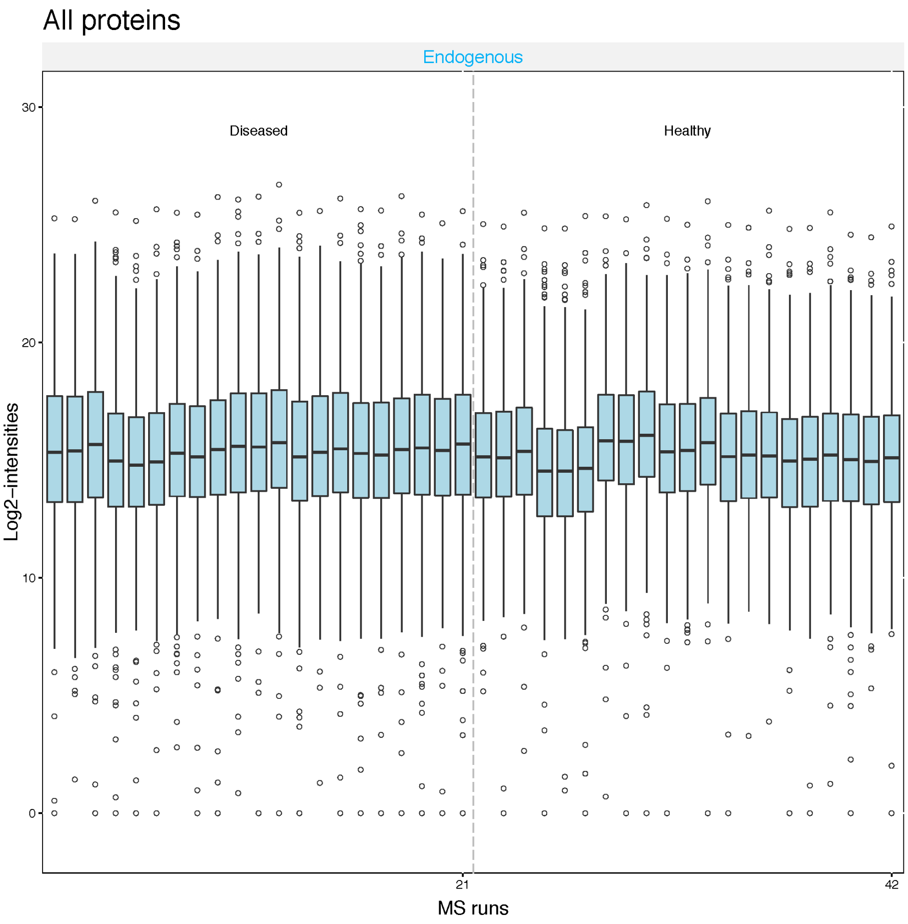
3.5.3.1 Quality control plots

Now let's look at what the equalize medians procedure did to our data. The QCplot type shows boxplots of peak intensities (on log scale) in all runs, where the bottom and top of a box represent the first and third quartiles of the log-intensities and the band inside the box is the median. It provides a quick way to examine and compare distributions between runs, and to detect systematic bias. Also, it is good visualization to check normalization. However, not good to see individual intensities.

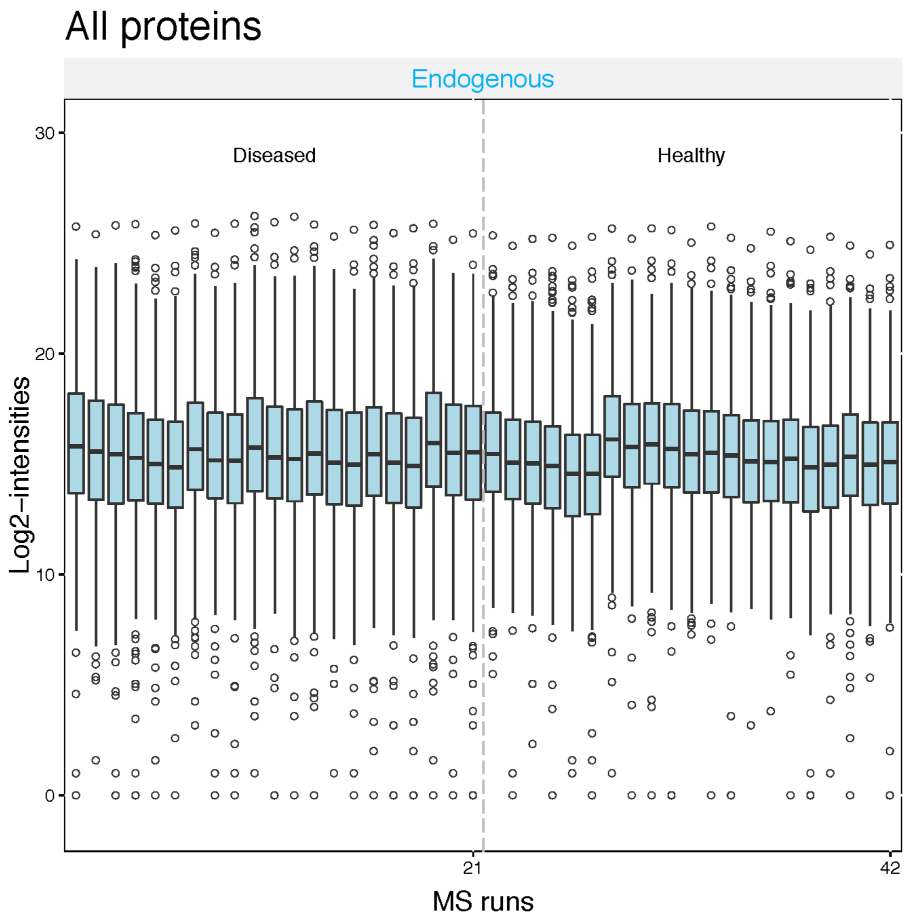
```
dataProcessPlots(data = quant_tmp, type = "QCplot", address = 'MSstats_')
```

By running the above command, TMP_QCPlot.pdf is generated in the working directory. If you can't find the plot, remember you can use getwd() to see where your working directory is.

Here's what the QC plot would look if we did normalize with global standard.



Here's what the QC plot would look if we didn't normalize.

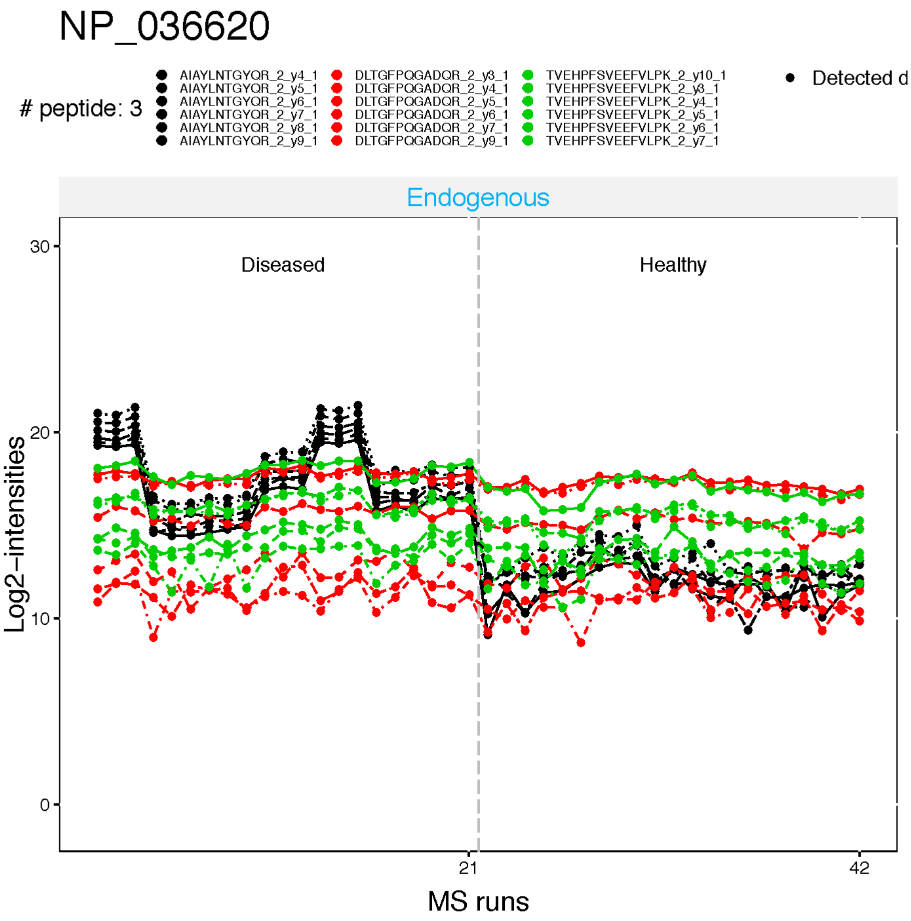


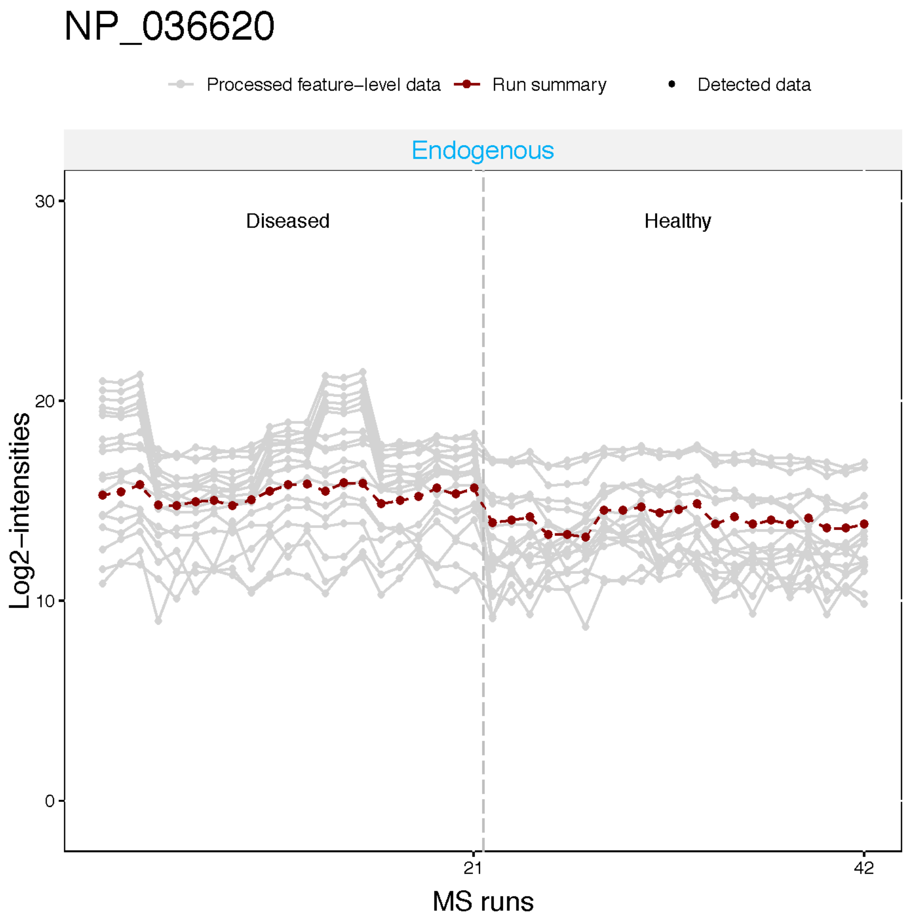
3.5.3.2 Profile plots

Profile plot shows individual observations for each protein. It is useful to examine the consistency of measurements in feature, run and condition, and to detect potential source of variation and missingness in the data. Each dot represents one feature intensity.

```
dataProcessPlots(data = quant_tmp, type="Profileplot",
                 width = 7, height = 7, address = "MSstats_")
```

By running the above command, two files **MSstats_ProfilePlot.pdf** and **MSstats_ProfilePlot_wSummarization.pdf** are generated in the current directory.


NP_036620
# peptide: 3
● AIAYLNTGYQR_2_y4_1
● AIAYLNTGYQR_2_y5_1
● AIAYLNTGYQR_2_y6_1
● AIAYLNTGYQR_2_y7_1
● AIAYLNTGYQR_2_y8_1
● AIAYLNTGYQR_2_y9_1
● DLTGFPQGADQR_2_y3_1
● DLTGFPQGADQR_2_y4_1
● DLTGFPQGADQR_2_y5_1
● DLTGFPQGADQR_2_y6_1
● DLTGFPQGADQR_2_y7_1
● DLTGFPQGADQR_2_y9_1
● TVEHPFSVEEFVLPK_2_y10_1
● TVEHPFSVEEFVLPK_2_y3_1
● TVEHPFSVEEFVLPK_2_y4_1
● TVEHPFSVEEFVLPK_2_y5_1
● TVEHPFSVEEFVLPK_2_y6_1
● TVEHPFSVEEFVLPK_2_y7_1
● Detected d
Endogenous
Diseased
Healthy
Log2−intensities
30
20
10
0
21
42
MS runs



The dots are connected by a line per feature. If a line is disconnected, it means there is no value (missing value). Colors represent different peptides and charge states.

Parallel profiles on log scale correspond to consistent peak area percentage, from which we gain confidence in the integration of the peptide. When any inconsistency is observed, we should look into the data before conducting subsequent analysis.

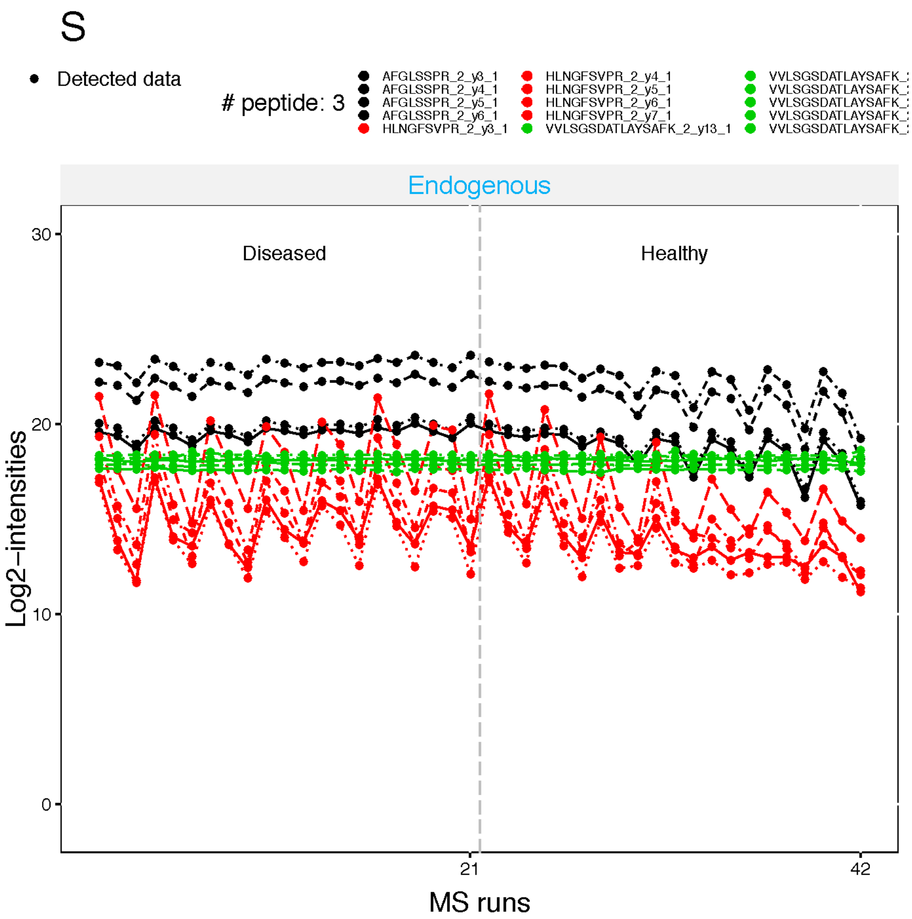
Let's compare these results from global standard normalization to the results we get when we use no normalization. To do this, we can simply run the same command, but replace the data = quant_tmp parameter setting with the linear model dataframe quant_nonorm:

```r
dataProcessPlots(data = quant_nonorm, type="Profileplot",
                 width = 7, height = 7, address = "MSstats_nonorm_")
```

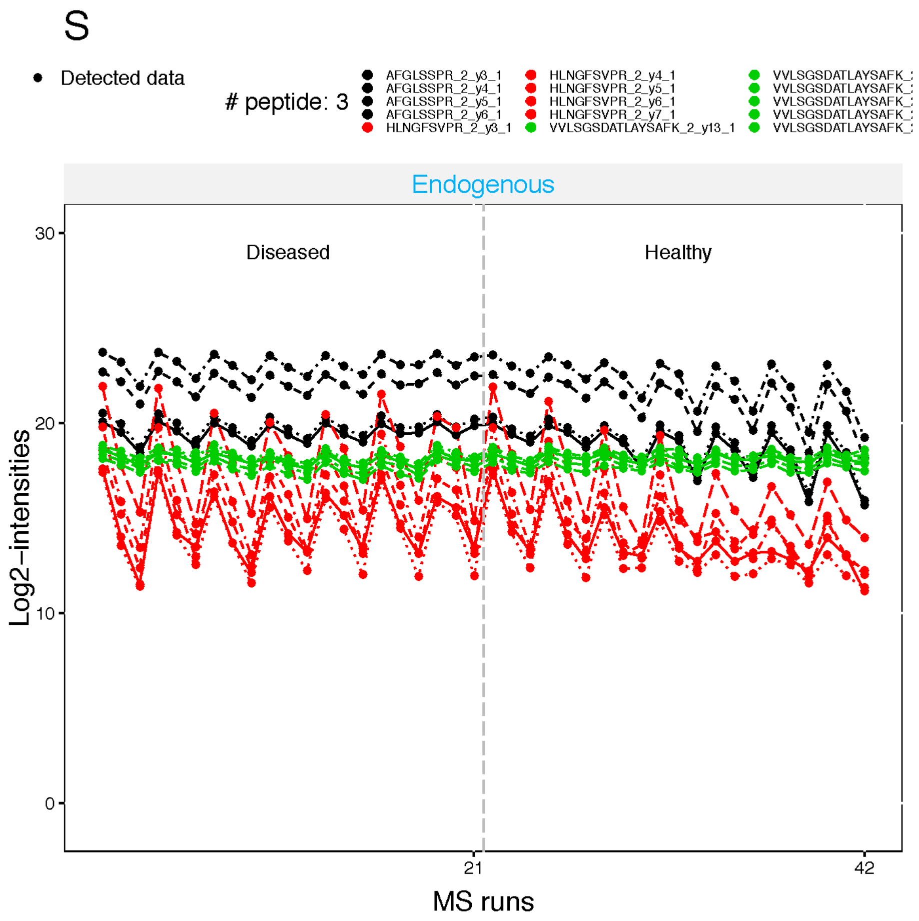
Check the generated plots in your working directory and see how each method summarizes the data in different ways.

We can generate these for all proteins but also for single proteins at a time. Suppose we just want to generate a plot for S. We can specify a particular protein using the **which.Protein** parameter, and setting that parameter to the protein. (Tip: to see all the unique proteins in quant_tmp, you can use unique(quant_tmp*ProcessedData*PROTEIN).)

```r
dataProcessPlots(data = quant_tmp, type="Profileplot",
                 originalPlot = TRUE, summaryPlot = FALSE,
                 which.Protein = 'S',
                 width = 7, height = 7, address = FALSE)
```



```r
dataProcessPlots(data = quant_nonorm, type="Profileplot",
                originalPlot = TRUE, summaryPlot = FALSE,
                which.Protein = 'S',
                 width = 7, height = 7, address = FALSE)
```

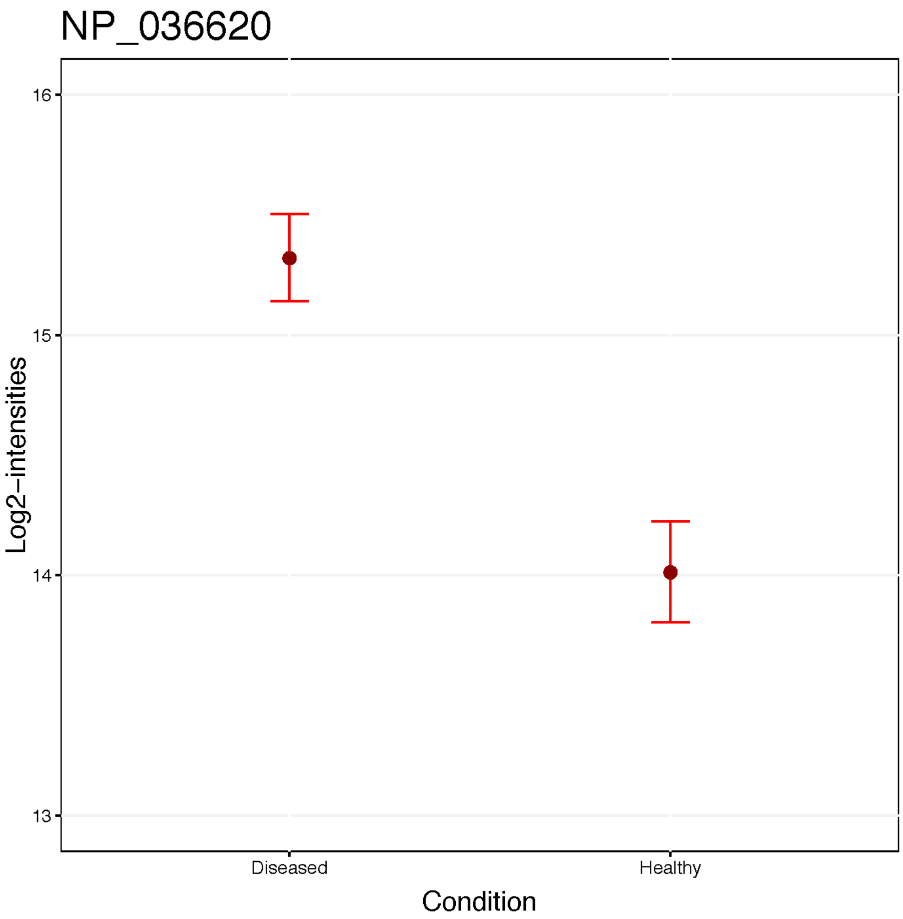


3.5.3.3 Condition plots

The Conditionplot type shows the mean of log-intensity and the 95% confidence interval for each condition. Although this visualization is helpful to get a feel for the data, if we want to say whether a protein is differentially abundant between conditions, this plot is not sufficient and group comparison analysis needs to be conducted! For now, these plots simply provide some helpful summary visualizations of the data.

```r
dataProcessPlots(data = quant_tmp, type = "conditionplot",
                 width = 7, height = 7,
                 address = "MSstats_")
```

We can draw the condition plot for a protein, `NP_036620` .



Glance through the plots created by the conditionplot type. Are there any proteins that look interesting? Remember, just having the plot doesn't tell us if the differential expression is significant! To make that claim, we need to do the group comparison.

3.5.4 Different normalization option

Let's see the different normalization effect with SRM dataset including two proteins

```
head(SRMRawData)
```

```
##     ProteinName PeptideSequence PrecursorCharge FragmentIon ProductCharge
## 243        IDHC    ATDVIVPEEGELR               2          y7            NA
## 244        IDHC    ATDVIVPEEGELR               2          y7            NA
## 245        IDHC   ATDVIVPEEGELR              2          y8             NA
## 246        IDHC   ATDVIVPEEGELR              2          y8             NA
## 247        IDHC   ATDVIVPEEGELR              2          y9             NA
## 248        IDHC   ATDVIVPEEGELR              2          y9             NA
##     IsotopeLabelType Condition BioReplicate Run   Intensity
## 243                H         1        ReplA   1 84361.08350
## 244                L         1        ReplA   1   215.13526
## 245                H         1        ReplA   1 29778.10188
## 246                L         1        ReplA   1    98.02134
## 247                H         1        ReplA   1 17921.29255
## 248                L         1        ReplA   1    60.47029
```

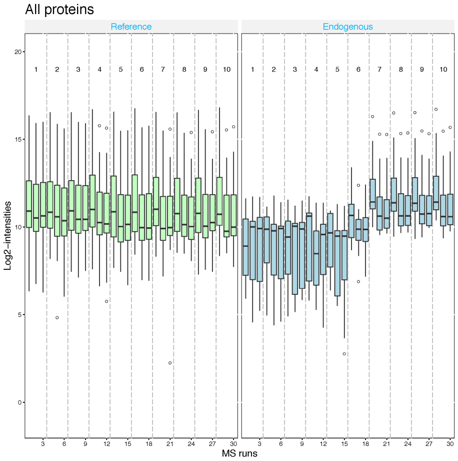
```
unique(SRMRawData$ProteinName)
```

```
## [1] IDHC PMG2
## 45 Levels: ACEA ACH1 ACON ADH1 ADH2 ADH4 ALDH6 ALF CISY1 CISY2 ... SUCB
```

3.5.4.1 No normalization

No normalization is performed. If you had your own normalization before MSstats, you should use like below.

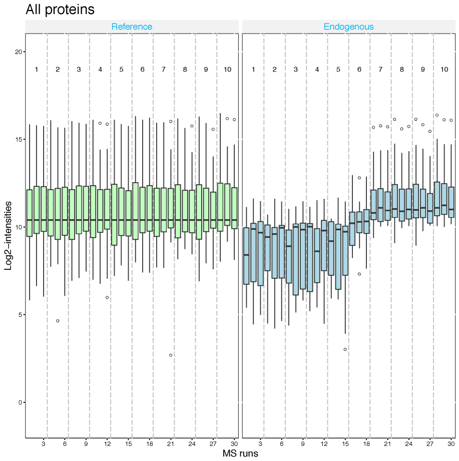
```
srm.nonorm <- dataProcess(SRMRawData, normalization=FALSE)
dataProcessPlots(srm.nonorm, type='QCplot', address='srm_noNorm_')
```



3.5.4.2 Equalize medians normalization

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.

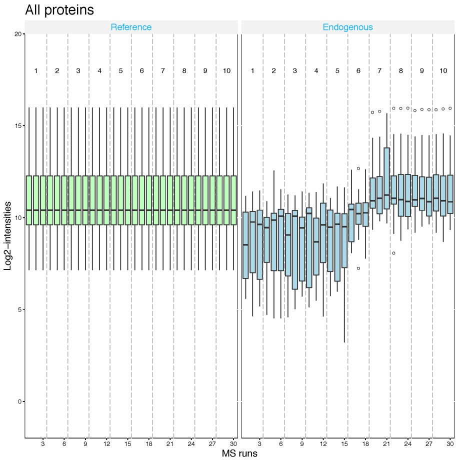
```
srm.equalmed <- dataProcess(SRMRawData, normalization = 'equalizeMedians')
dataProcessPlots(srm.equalmed, type='QCplot', address='srm_equalM_')
```



3.5.4.3 Quantile normalization

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.

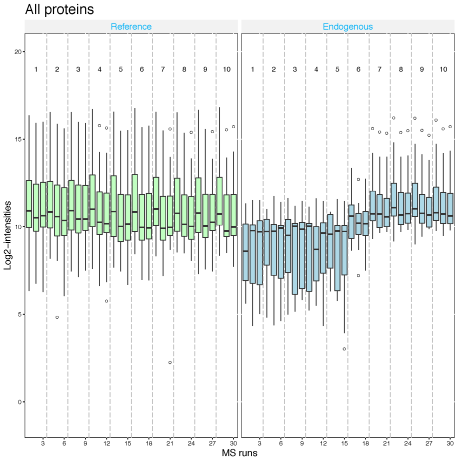
```
srm.quantile <- dataProcess(SRMRawData, normalization='quantile')
dataProcessPlots(srm.quantile, type='QCplot', address='srm_quantile_')
```



#### Global standards normalization : example 1 If you have a spiked in standard across all MS runs, you may set this to `globalStandards` and define the standard with `nameStandards` option. 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.

First, let's assume that `PMG2` proteins is the spike-in protein and shoule be equal amount across MS runs.

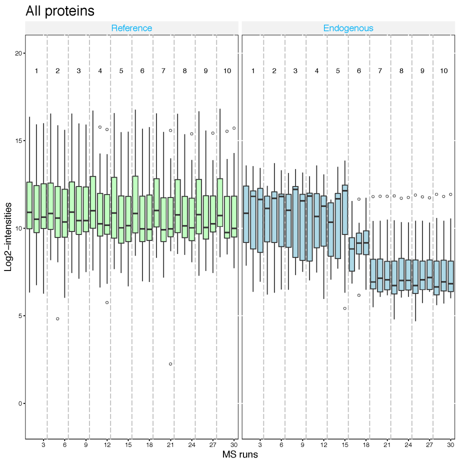
```
srm.global.pmg2 <- dataProcess(SRMRawData, normalization ='globalStandards',
                               nameStandards = 'PMG2')
dataProcessPlots(srm.global.pmg2, type='QCplot', address='srm_global_PMG2_')
```



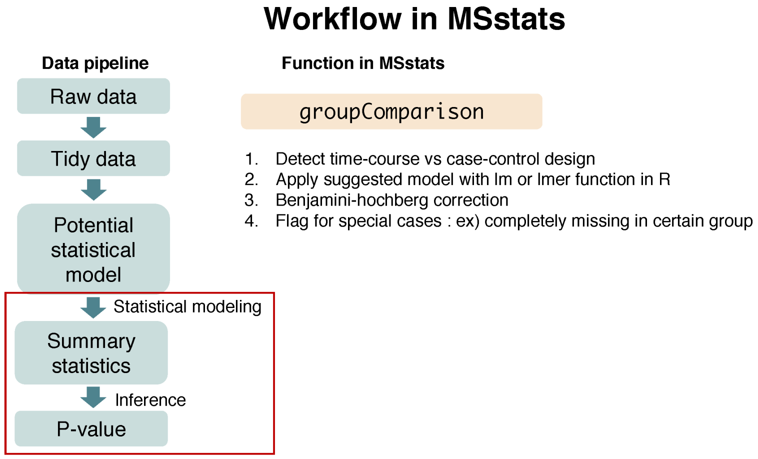
Second, let's assume that `IDHC` proteins is the spike-in protein and shoule be equal amount across MS runs.

3.5.4.4 Global standards normalization : example 2

```
srm.global.idhc <- dataProcess(SRMRawData, normalization ='globalStandards',
                               nameStandards = 'IDHC')
dataProcessPlots(srm.global.idhc, type='QCplot', address='srm_global_IDHC_')
```



3.6 Group comparison to find differentially abundant proteins across conditions



Workflow in MSstats

Data pipeline

Function in MSstats

groupComparison

1. Detect time-course vs case-control design
2. Apply suggested model with lm or lmer function in R
3. Benjamini-hochberg correction
4. Flag for special cases : ex) completely missing in certain group

After we normalized the data and summarized each protein's behaviour across conditions in `dataProcess` step, we are all set to compare protein changes between groups of conditions. Within MSstats we can do this with the `groupComparison` function, which takes as input the output of the `dataProcess` function.

```
?groupComparison
```

3.6.1 Assign contrast matrix

We have to tell `groupComparison` which are the conditions we would like to compare. You can make your `contrast.matrix` in R in a text editor. We define our contrast matrix by adding a column for every condition, **in alphabetical order**. We add a row for every comparison we would like to make between groups of conditions.

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. **-1** is for conditions we would like to put in the denumerator of the ratio or fold-change.

This part is a bit confusing, but it is absolutely critical! If the contrast matrix is set up incorrectly, our data will not be analyzed correctly. Our data has two groups, Diseased and Healthy. It's common to place the experimental group over the control group, so that changes are "upregulated/downregulated" in the experimental group.

```
# check unique conditions and check order of condition information
levels(quant_tmp$ProcessedData$GROUP_ORIGINAL)
```

```
## [1] "Diseased" "Healthy"
```

```
# create a contrast matrix for Diseased vs Healthy
comparison <- matrix(c(1, -1), nrow=1)
row.names(comparison) <- c("Diseased-Healthy")
comparison
```

```
##                  [,1] [,2]
## Diseased-Healthy    1   -1
```

3.6.2 Comparing conditions with groupComparison

`groupComparison` uses the run-level summarized data (`$RunlevelData` from `dataProcess` function) for hypothesis testing. Now that we have our contrast matrix, we can feed it to the contrast.matrix parameter of the groupComparison function.

```
gpcomp_tmp <- groupComparison(contrast.matrix = comparison, data = quant_tmp)
```

```
##
```

```
|
|                                                                |   0%
|
|=                                                               |   2%
|
|===                                                             |   4%
|
|====                                                            |   6%
|
|=====                                                           |   8%
|
|=======                                                         |  10%
|
|========                                                        |  12%
|
|=========                                                       |  15%
|
|===========                                                     |  17%
|
|============                                                    |  19%
|
|==============                                                  |  21%
|
|===============                                                 |  23%
|
|================                                                |  25%
|
|==================                                              |  27%
|
|===================                                             |  29%
|
|====================                                            |  31%
|
|======================                                          |  33%
|
|=======================                                         |  35%
|
|========================                                        |  38%
|
|==========================                                      |  40%
|
|===========================                                     |  42%
|
|============================                                    |  44%
|
|==============================                                  |  46%
```

```
|
|===============================                                  |  48%
|
|================================                                 |  50%
|
|==================================                               |  52%
|
|===================================                              |  54%
|
|=====================================                            |  56%
|
|======================================                           |  58%
|
|=======================================                          |  60%
|
|=========================================                        |  62%
|
|==========================================                       |  65%
|
|===========================================                      |  67%
|
|=============================================                    |  69%
|
|==============================================                   |  71%
|
|===============================================                  |  73%
|
|=================================================                |  75%
|
|==================================================               |  77%
|
|===================================================              |  79%
|
|=====================================================            |  81%
|
|======================================================           |  83%
|
|========================================================         |  85%
|
|=========================================================        |  88%
|
|==========================================================       |  90%
|
|============================================================     |  92%
|
|=============================================================    |  94%
```

```
  |
  |==============================================================   |  96%
  |
  |================================================================ |  98%
  |
  |=================================================================| 100%
```

Let's check the output.

```
class(gpcomp_tmp)
```

```
## [1] "list"
```

```
names(gpcomp_tmp)
```

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

```
# Show test result
# Label : which comparison is used
# log2FC : estimated log2 fold change between Diseased and Healthy
# adj.pvalue : adjusted p value
# issue : detect whether this protein has any issue for comparison
#    such as, there is measurement in certain group, or no measurement at all.
# MissingPercentage : the number of missing intensities/total number of intensities
#     in conditions your are interested in for comparison
# ImputationPercentage : the number of imputed intensities/total number of intensities
#     in conditions your are interested in for comparison
head(gpcomp_tmp$ComparisonResult)
```

```
##        Protein            Label     log2FC        SE    Tvalue DF
## 1 NP_001007697 Diseased-Healthy -0.8336648 0.1587751 -5.250601 28
## 2 NP_001008724 Diseased-Healthy  0.2536262 0.1374329  1.845454 28
## 3 NP_001010968 Diseased-Healthy -0.2364582 0.1377050 -1.717136 28
## 4 NP_001011908 Diseased-Healthy -0.5805380 0.2071939 -2.801907 28
## 5 NP_001012027 Diseased-Healthy  0.4264091 0.1567340  2.720591 28
## 6 NP_001013967 Diseased-Healthy -0.3841446 0.1812183 -2.119789 28
##         pvalue   adj.pvalue issue MissingPercentage ImputationPercentage
## 1 1.398222e-05 4.194665e-05    NA        0.00000000                    0
## 2 7.557162e-02 8.847410e-02    NA        0.00170068                    0
## 3 9.699793e-02 1.108548e-01    NA        0.20158730                    0
## 4 9.113170e-03 1.325552e-02    NA        0.06107660                    0
## 5 1.107302e-02 1.518585e-02    NA        0.00000000                    0
## 6 4.302086e-02 5.434214e-02    NA        0.00000000                    0
```

```
# After fitting linear model, residuals and fitted values can be shown.
head(gpcomp_tmp$ModelQC)
```

```
##    RUN      PROTEIN ABUNDANCE NumMeasuredFeature MissingPercentage
## 1:   1 NP_001007697  12.80623                 12                 0
## 2:   2 NP_001007697  13.39071                 12                 0
## 3:   3 NP_001007697  13.11670                 12                 0
## 4:   4 NP_001007697  12.18845                 12                 0
## 5:   5 NP_001007697  12.33439                 12                 0
## 6:   6 NP_001007697  12.24504                 12                 0
##    more50missing NumImputedFeature    originalRUN GROUP GROUP_ORIGINAL
## 1:         FALSE                 0 D_102_REP1.raw     1       Diseased
## 2:         FALSE                 0 D_102_REP2.raw     1       Diseased
## 3:         FALSE                 0 D_102_REP3.raw     1       Diseased
## 4:         FALSE                 0 D_103_REP1.raw     1       Diseased
## 5:         FALSE                 0 D_103_REP2.raw     1       Diseased
## 6:         FALSE                 0 D_103_REP3.raw     1       Diseased
##    SUBJECT_ORIGINAL SUBJECT_NESTED SUBJECT   residuals   fitted
## 1:              102            1.1       1 -0.04146609 12.84769
## 2:              102            1.1       1  0.54301574 12.84769
## 3:              102            1.1       1  0.26900406 12.84769
## 4:              103            1.2       2 -0.19024316 12.37869
## 5:              103            1.2       2 -0.04429893 12.37869
## 6:              103            1.2       2 -0.13364853 12.37869
```

```
# Fitted model per protein
head(gpcomp_tmp$fittedmodel)
```

```
## [[1]]
## Linear mixed model fit by REML ['lmerMod']
## Formula: ABUNDANCE ~ GROUP + (1 | SUBJECT)
##    Data: data2
## REML criterion at convergence: 43.9118
## Random effects:
##  Groups   Name        Std.Dev.
##  SUBJECT  (Intercept) 0.2208
##  Residual             0.3441
## Number of obs: 42, groups:  SUBJECT, 14
## Fixed Effects:
## (Intercept)       GROUP2
##     12.5303       0.8337
##
## [[2]]
## Linear mixed model fit by REML ['lmerMod']
## Formula: ABUNDANCE ~ GROUP + (1 | SUBJECT)
##    Data: data2
## REML criterion at convergence: -29.6564
## Random effects:
##  Groups   Name        Std.Dev.
##  SUBJECT  (Intercept) 0.2508
##  Residual             0.0984
## Number of obs: 42, groups:  SUBJECT, 14
## Fixed Effects:
## (Intercept)       GROUP2
##     15.8949      -0.2536
##
## [[3]]
## Linear mixed model fit by REML ['lmerMod']
## Formula: ABUNDANCE ~ GROUP + (1 | SUBJECT)
##    Data: data2
## REML criterion at convergence: 8.1204
## Random effects:
##  Groups   Name        Std.Dev.
##  SUBJECT  (Intercept) 0.2323
##  Residual             0.1930
## Number of obs: 42, groups:  SUBJECT, 14
## Fixed Effects:
## (Intercept)       GROUP2
##     14.5870       0.2365
## convergence code 0; 1 optimizer warnings; 0 lme4 warnings
##
## [[4]]
## Linear mixed model fit by REML ['lmerMod']
## Formula: ABUNDANCE ~ GROUP + (1 | SUBJECT)
##    Data: data2
## REML criterion at convergence: -2.2351
## Random effects:
##  Groups   Name        Std.Dev.
##  SUBJECT  (Intercept) 0.3797
##  Residual             0.1347
## Number of obs: 42, groups:  SUBJECT, 14
## Fixed Effects:
## (Intercept)       GROUP2
##     13.7445       0.5805
##
## [[5]]
## Linear mixed model fit by REML ['lmerMod']
## Formula: ABUNDANCE ~ GROUP + (1 | SUBJECT)
##    Data: data2
## REML criterion at convergence: 2.328
## Random effects:
##  Groups   Name        Std.Dev.
##  SUBJECT  (Intercept) 0.2774
##  Residual             0.1647
## Number of obs: 42, groups:  SUBJECT, 14
## Fixed Effects:
## (Intercept)       GROUP2
##     13.8510      -0.4264
##
## [[6]]
## Linear mixed model fit by REML ['lmerMod']
## Formula: ABUNDANCE ~ GROUP + (1 | SUBJECT)
##    Data: data2
## REML criterion at convergence: -14.8569
## Random effects:
##  Groups   Name        Std.Dev.
##  SUBJECT  (Intercept) 0.3326
##  Residual             0.1138
## Number of obs: 42, groups:  SUBJECT, 14
## Fixed Effects:
## (Intercept)       GROUP2
##     15.2037       0.3841
```

```
gpcomp_tmp$fittedmodel[[1]]
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: ABUNDANCE ~ GROUP + (1 | SUBJECT)
##    Data: data2
## REML criterion at convergence: 43.9118
## Random effects:
##  Groups   Name        Std.Dev.
##  SUBJECT  (Intercept) 0.2208
##  Residual             0.3441
## Number of obs: 42, groups:  SUBJECT, 14
## Fixed Effects:
## (Intercept)       GROUP2
##     12.5303       0.8337
```

Show only the results for significant changes.

```
# pull just the results out of the whole group comparison output
gpcomp_res <- gpcomp_tmp$ComparisonResult

# subset only proteins with adjusted p-value < 0.05 and a FC > 2^2
list_sig <- gpcomp_res[gpcomp_res$adj.pvalue < 0.05 & abs(gpcomp_res$log2FC) > 2 , ]
head(list_sig)
```

```
##      Protein            Label    log2FC        SE     Tvalue DF
## 14 NP_036714 Diseased-Healthy -5.243740 0.2812962 -18.641346 28
## 18 NP_036828 Diseased-Healthy  3.657823 0.5514314   6.633323 28
## 31 NP_150641 Diseased-Healthy -2.568541 0.2657715  -9.664474 28
##          pvalue   adj.pvalue issue MissingPercentage ImputationPercentage
## 14 0.000000e+00 0.000000e+00    NA       0.000000000                    0
## 18 3.389228e-07 1.478936e-06    NA       0.000000000                    0
## 31 2.033405e-10 2.440085e-09    NA       0.009157509                    0
```

```
nrow(list_sig)
```

```
## [1] 3
```

3.6.3 Save the comparison result

Let's save the testing result as rdata and .csv file.

```
save(gpcomp_res, file='gpcomp_res.rda')
write.csv(gpcomp_res, file='testresult_wglobalstandNorm.csv')
```

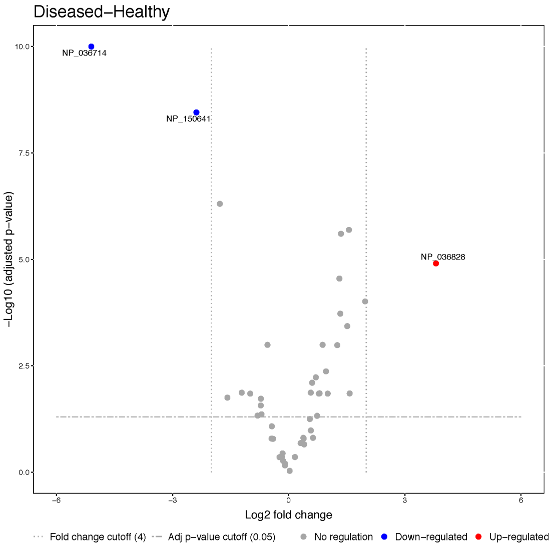
3.7 Visualization of differentially abundant proteins

```
?groupComparisonPlots
```

3.7.1 Volcano plot

Volcano plot summarizes all the proteins with respect to their practical significance (log2 [fold change]) and statistical significance (-log10 [adjusted p-value]). Proteins with greater values on the y axis are more statistically significant. Changes with an adjusted p-value less than a significance level (default of 0.05) are considered as statistically significant. Up-regulated and down-regulated proteins are shown in red and blue, respectively.

```
groupComparisonPlots(data = gpcomp_tmp$ComparisonResult,
                     type = 'VolcanoPlot',
                     sig = 0.05, FCcutoff = 2^2,
                     address = 'MSstats_')
```



3.7.2 Heatmap

Heatmaps are useful to visualize the results of multiple comparisons at once. In this tutorial, we only had one comparison (Diseased vs Healthy), but I'm leaving this code here so that you can recycle it if you do a multi-way comparison in the future!

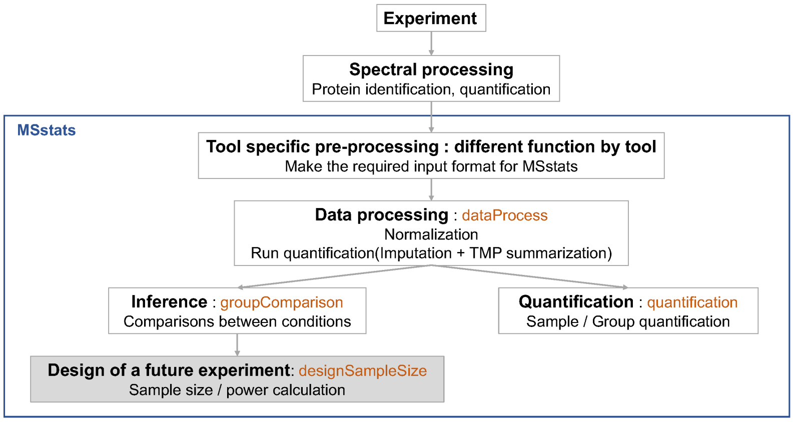
```r
groupComparisonPlots(data = gpcomp_tmp$ComparisonResult,
                     type = 'Heatmap',
                     address = 'MSstats_')
```

3.7.3 Comparison plot

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.

```r
groupComparisonPlots(data = gpcomp_tmp$ComparisonResult,
                     type = 'ComparisonPlot',
                     address = 'MSstats_')
```

3.8 Planning future experimental designs


Experiment
Spectral processing
Protein identification, quantification
MSstats
Tool specific pre-processing : different function by tool
Make the required input format for MSstats
Data processing : dataProcess
Normalization
Run quantification(Imputation + TMP summarization)
Inference : groupComparison
Comparisons between conditions
Quantification : quantification
Sample / Group quantification
Design of a future experiment: designSampleSize
Sample size / power calculation

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`.

Above, when we made our volcano plot, we use 2^2 as the fold-change cut off, but that number is experiment-dependent! Some strongly-powered experiments can achieve fold-change sensitivities like 1.25, but other poorly-powered experiments may not be sensitive to even 5 FC.

```
?designSampleSize
```

3.8.1 Designing sample size for desired fold-change

The designSampleSize function has three major parameters that can be manipulated to explore the range of samples, desired FC, or statistical power. It's most common to fix the power at 0.8 or 0.9 to explore how varying the FC or sample size affects an experiment.

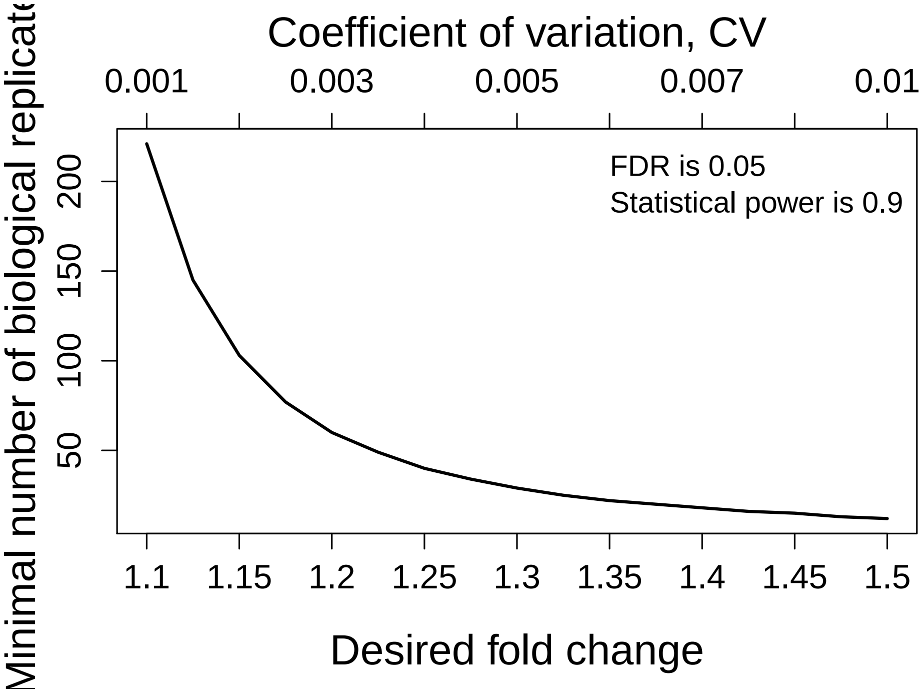
```
# calculate the number of samples to achieve a range of fold changes from 1.1-1.5, at a fixed 90%
design_size <- designSampleSize(data = gpcomp_tmp$fittedmodel,
                                desiredFC = c(1.1, 1.5), FDR = 0.05,
                                power = 0.9,
                                numSample = TRUE)
design_size
```

```
##    desiredFC numSample  FDR power    CV
## 1      1.100       221 0.05   0.9 0.001
## 2      1.125       145 0.05   0.9 0.001
## 3      1.150       103 0.05   0.9 0.002
## 4      1.175        77 0.05   0.9 0.002
## 5      1.200        60 0.05   0.9 0.003
## 6      1.225        49 0.05   0.9 0.003
## 7      1.250        40 0.05   0.9 0.004
## 8      1.275        34 0.05   0.9 0.004
## 9      1.300        29 0.05   0.9 0.005
## 10     1.325        25 0.05   0.9 0.006
## 11     1.350        22 0.05   0.9 0.006
## 12     1.375        20 0.05   0.9 0.007
## 13     1.400        18 0.05   0.9 0.007
## 14     1.425        16 0.05   0.9 0.008
## 15     1.450        15 0.05   0.9 0.008
## 16     1.475        13 0.05   0.9 0.010
## 17     1.500        12 0.05   0.9 0.010
```

3.8.2 Visualize the relationship between desired fold-change and mininum sample size number

The design_size output above can be visualized in a sample size plot for ease of interpretation. Notice the axis labels, and the information contained in this plot.

```
designSampleSizePlots(data = design_size)
```



3.8.3 Calculating statistical power

Instead of calculating with a fixed statistical power, let's consider an example where all our data for the experiment has been acquired, and we need to calculate what our statistical power is at various fold-change values. In our heart failure experiment, there were 7 Diseased and 7 Healthy animals, so 7 biological replicates.

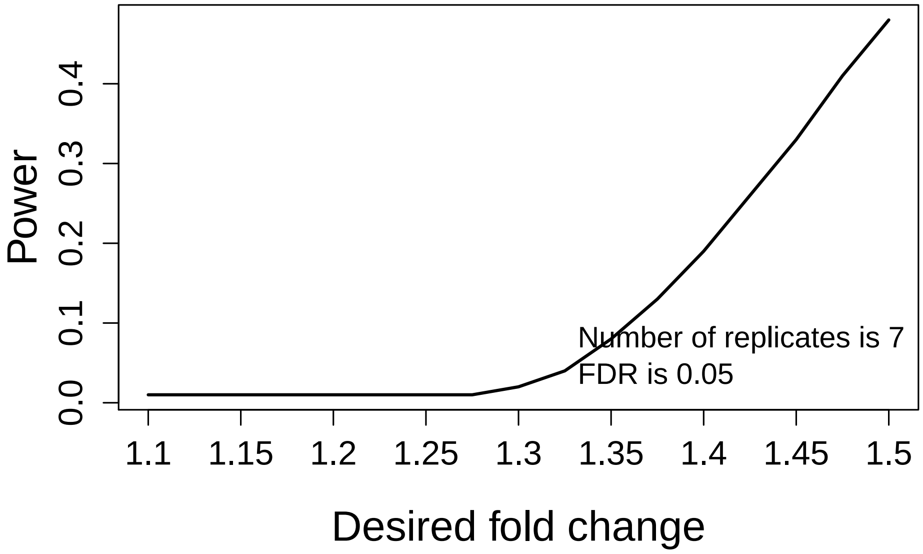
```
# power calculation with 7 replicates
design_power <- designSampleSize(data = gpcomp_tmp$fittedmodel,
                                 desiredFC = c(1.1, 1.5),
                                 FDR = 0.05,
                                 power = TRUE,
                                 numSample = 7)
design_power
```

```
##    desiredFC numSample  FDR power    CV
## 1      1.100         7 0.05  0.01 0.024
## 2      1.125         7 0.05  0.01 0.023
## 3      1.150         7 0.05  0.01 0.023
## 4      1.175         7 0.05  0.01 0.022
## 5      1.200         7 0.05  0.01 0.022
## 6      1.225         7 0.05  0.01 0.021
## 7      1.250         7 0.05  0.01 0.021
## 8      1.275         7 0.05  0.01 0.020
## 9      1.300         7 0.05  0.02 0.020
## 10     1.325         7 0.05  0.04 0.020
## 11     1.350         7 0.05  0.08 0.019
## 12     1.375         7 0.05  0.13 0.019
## 13     1.400         7 0.05  0.19 0.019
## 14     1.425         7 0.05  0.26 0.018
## 15     1.450         7 0.05  0.33 0.018
## 16     1.475         7 0.05  0.41 0.018
## 17     1.500         7 0.05  0.48 0.017
```

3.8.4 Visualize the relationship between desired fold-change and power

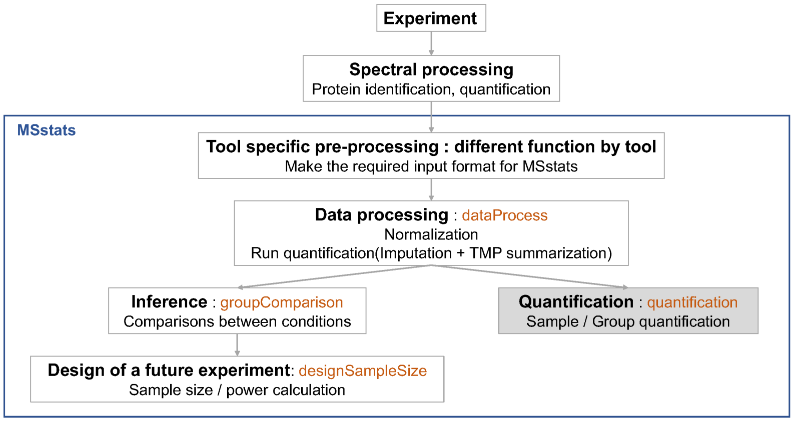
Again, we can plot the matrix of numbers stored in the design_power variable to get a more easily interpretable representation of the data.

```
designSampleSizePlots(data = design_power)
```



3.9 Protein subject quantification

With the summarized protein abundance, you can apply clustering and/or classification techniques to perform downstream analysis. If there is no technical replicate, subject (or sample) quantification should be the same as run-level summarization (quant_tmp$RunlevelData). However, our heart failure experiment used technical triplicate for each of the biological replicates, so we the subject-level summarization (quantification) with run-level summarization will be useful for downstream analysis, such as classification.


Experiment
Spectral processing
Protein identification, quantification
MSstats
Tool specific pre-processing : different function by tool
Make the required input format for MSstats
Data processing : dataProcess
Normalization
Run quantification(Imputation + TMP summarization)
Inference : groupComparison
Comparisons between conditions
Quantification : quantification
Sample / Group quantification
Design of a future experiment: designSampleSize
Sample size / power calculation

```
?quantification
```

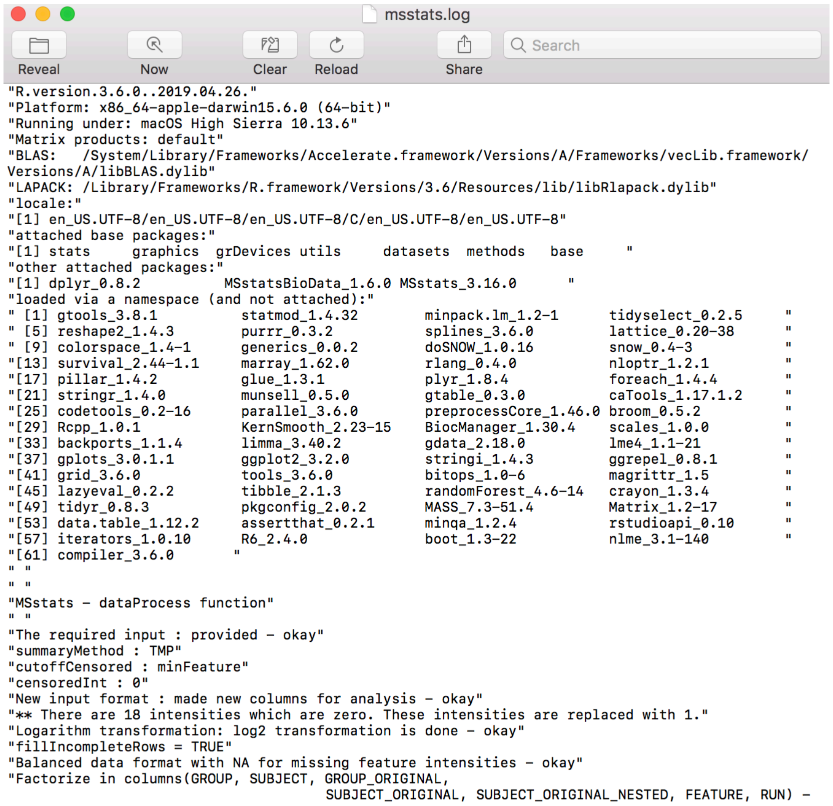
To perform the quantification for each condition, we simply supply our **quant_tmp** variable as the input, and here we'll save the result in a new variable, **sampleQuant**. You can explore the **sampleQuant** data the same way we explored other data above.

```
## sample quantification : estimated protein abundance per biological replicate
sampleQuant <- quantification(quant_tmp)
head(sampleQuant)
```

```
##        Protein Diseased_102 Diseased_103 Diseased_108 Diseased_138
## 1 NP_001007697     13.11670     12.24504     12.62943     12.17727
## 2 NP_001008724     16.12783     15.72596     15.83643     15.64176
## 3 NP_001010968     14.69073     14.51685     14.75055     14.51171
## 4 NP_001011908     12.86150     14.10219     14.00146     13.93043
## 5 NP_001012027     14.36359     13.42129     13.38806     13.97415
## 6 NP_001013967     15.28486     15.18118     15.15666     15.81058
##   Diseased_154 Diseased_172 Diseased_196 Healthy_146 Healthy_147
## 1     12.63568     12.75957     12.65846    13.07605    12.99485
## 2     16.16695     15.84984     16.04258    15.68226    15.47625
## 3     15.30123     14.24985     14.18392    14.96027    14.49220
## 4     14.09568     14.10892     13.10736    14.60360    14.43814
## 5     14.18741     13.70413     13.79411    13.47386    12.98094
## 6     14.90268     15.51735     14.62367    15.70800    14.97041
##   Healthy_148 Healthy_159 Healthy_160 Healthy_161 Healthy_162
## 1    13.19913    14.04278    13.65691    13.48622    13.49249
## 2    16.20094    15.88184    15.48883    15.28579    15.48859
## 3    14.79855    14.94489    14.81125    14.81254    14.97455
## 4    14.07497    14.30102    14.49728    14.21577    14.11472
## 5    13.62825    13.52809    13.26273    13.31545    13.60441
## 6    15.54271    15.46914    15.72900    15.63306    15.88452
```

3.10 Tracking the whole process

When running MSstats, **msstats.log** and **sessionInfo.txt** are automatically generated. These two files are important to keep the records of package versions and options in functions. To help troubleshoot potential problems with installation or functionalities of MSstats, a progress report is generated in a 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.



```
"R.version.3.6.0..2019.04.26."
"Platform: x86_64-apple-darwin15.6.0 (64-bit)"
"Running under: macOS High Sierra 10.13.6"
"Matrix products: default"
"BLAS:   /System/Library/Frameworks/Accelerate.framework/Versions/A/Frameworks/vecLib.framework/
Versions/A/libBLAS.dylib"
"LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib"
"locale:"
"[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8"
"attached base packages:"
"[1] stats     graphics  grDevices utils     datasets  methods   base      "
"other attached packages:"
"[1] dplyr_0.8.2          MSstatsBioData_1.6.0 MSstats_3.16.0       "
"loaded via a namespace (and not attached):"
" [1] gtools_3.8.1          statmod_1.4.32        minpack.lm_1.2-1      tidyselect_0.2.5      "
" [5] reshape2_1.4.3        purrr_0.3.2           splines_3.6.0         lattice_0.20-38       "
" [9] colorspace_1.4-1      generics_0.0.2        doSNOW_1.0.16         snow_0.4-3            "
"[13] survival_2.44-1.1     marray_1.62.0         rlang_0.4.0           nloptr_1.2.1          "
"[17] pillar_1.4.2          glue_1.3.1            plyr_1.8.4            foreach_1.4.4         "
"[21] stringr_1.4.0         munsell_0.5.0         gtable_0.3.0          caTools_1.17.1.2      "
"[25] codetools_0.2-16      parallel_3.6.0        preprocessCore_1.46.0 broom_0.5.2           "
"[29] Rcpp_1.0.1            KernSmooth_2.23-15    BiocManager_1.30.4    scales_1.0.0          "
"[33] backports_1.1.4       limma_3.40.2          gdata_2.18.0          lme4_1.1-21           "
"[37] gplots_3.0.1.1        ggplot2_3.2.0         stringi_1.4.3         ggrepel_0.8.1         "
"[41] grid_3.6.0            tools_3.6.0           bitops_1.0-6          magrittr_1.5          "
"[45] lazyeval_0.2.2        tibble_2.1.3          randomForest_4.6-14   crayon_1.3.4          "
"[49] tidyr_0.8.3           pkgconfig_2.0.2       MASS_7.3-51.4         Matrix_1.2-17         "
"[53] data.table_1.12.2     assertthat_0.2.1      minqa_1.2.4           rstudioapi_0.10       "
"[57] iterators_1.0.10      R6_2.4.0              boot_1.3-22           nlme_3.1-140          "
"[61] compiler_3.6.0       "
" "
" "
"MSstats - dataProcess function"
" "
"The required input : provided - okay"
"summaryMethod : TMP"
"cutoffCensored : minFeature"
"censoredInt : 0"
"New input format : made new columns for analysis - okay"
"** There are 18 intensities which are zero. These intensities are replaced with 1."
"Logarithm transformation: log2 transformation is done - okay"
"fillIncompleteRows = TRUE"
"Balanced data format with NA for missing feature intensities - okay"
"Factorize in columns(GROUP, SUBJECT, GROUP_ORIGINAL,
                                      SUBJECT_ORIGINAL, SUBJECT_ORIGINAL_NESTED, FEATURE, RUN) -
```