ProteoSign 2.0: A differential expression analysis tool for Proteomics data

by

Evangelos Theodorakis

Master in Bioinformatics
Faculty of Medicine - University of Crete
Heraklion, Crete, June of 2019

Abstract

Bottom-up proteomics analyses have been proved to be a powerful tool in the characterization of the proteome. Through differential proteomic analysis one can shed light on groups of proteins or individual proteins that give rise to various conditions -normal or pathological. However the tools for the analysis of such data sets are either hard-to-use with steep learning curves although powerful, or easy to use, but lack the analytical power of others. In order to cope with these challenges, we present ProteoSign 2.0, the upgrade of ProteoSign, a tool for differential proteomic analysis for proteomics datasets coming from MaxQuant or Proteome Discoverer™.

Introduction

The use of mass spectrometry (MS)-based quantitative proteomics, has proved its value as a robust tool for probing the vast proteome and its functional dynamics regardless of the level of biological organization (cell, tissue or organism)[1]. Advances in bioanalytical chemistry, mass spectrometry and bioinformatics, allow the detection, the relative quantification and the functional annotation of more that 4.000 proteins in a single experiment, within an hour, utilizing methods from the so-called bottom-up Proteomics approach[2]. Proteomics methods can be classified into two distinct groups, depending on their technical approach: the bottom-up proteomics and the top-down proteomics. In the , still developing, top-down proteomics, intact proteins are analyzed using MS techniques, while in , the more mature, bottom-up proteomics, proteins are proteolytically digested into peptides, then separated and analyzed by nano-flow liquid chromatography, electrospray ionization and tandem mass spectrometry (nLC–ESI-MS/MS). In the case that bigger proteomes, such as the human proteome, are to be analyzed, the protein mixtures are fractionated on protein and/or peptide level using gel-electrophoresis or chromatography, prior to proteolysis and nLC–ESI-MS/MS analysis. Following that, peptides are measured with MS

approaches in order to detect their accurate mass, their relative or absolute abundance and their amino acid sequence. Having this information, each peptide is assigned to its "parent" protein and thus the protein abundance is calculated. Despite the fact that, top-down proteomics can natively detect different protein isoforms and post-translational modifications (PTMs), bottom-up proteomics offer better peptides separation, both on nLC and MS level, as well as higher sensitivity and accuracy, establishing the bottom-up proteomics as the most commonly used approach for high throughput proteomics today[3].

While, mass spectrometers can detect extremely low concentrations in complex mixtures, MS-based proteomics are not quantitative by nature. To overcome this restriction, several approaches have been proposed for measuring the intensity and distinguishing the origin of each peptide. These approaches can be grouped into Label-Free Quantification (LFQ) and Labelling methods, each one with its strengths and weaknesses. Labeling methods can be further divided into three subcategories such as metabolic labeling, isotopic labeling, and isobaric labeling. In metabolic labeling, cultures are treated with different growth mediums, each one containing amino acids of labeled with unique molecular weight isotopes. Then, the mass spectrometer takes advantage of the mass shift of each peptide during the first MS stage, making it possible to distinguish the culture from which each peptide came from. The most well-known metabolic labeling method is the stable isotope labeling by amino acids in cell culture (SILAC), which also exists in the form of various altered protocols such as the pulsed SILAC (pSILAC) and the NeuCode SILAC. Metabolic labeling methods offer the advantages of reduced bias and short analysis duration, as the cultures can be mixed and analyzed in the same MS run. In addition, due to the fact than many proteins end up attaining the introduced labels, SILAC is the most well-established metabolic labeling method in bottom-up proteomics [4]. In isotopic labeling, a method similar to metabolic labeling, chemical probes of predetermined weight are attached to the biological samples of interest. Its key difference compared to the metabolic labeling protocols is that this method offers the opportunity to label and analyze samples that do not contain live cells, such as tissue samples and biological fluids. On the same principle, peptides or proteins coming from the same sample are distinguished during the first MS phase , as in metabolic labeling methods, based on their mass shift. While this method is fast, and more robust in terms of sample type, it is not as accurate as

metabolic labeling, due to the fact that newly added tags can be easily separated from the proteins of interest during the LC elution phase. There are several protocols for isotopic labeling as isotope-coded affinity tags (ICAT), terminal amine isotopic labeling of substrates (TAILS) and global internal standard technology (GIST). The last subcategory of labeling methods, is the isobaric labeling. In isobaric labeling peptides are labeled with tags of identical weight and chemical properties. This allows different tags to co-elute during the LC phase without suffering from the isotopic labeling detachment. Furthermore, isobaric labeling possesses greater multiplexing power against isotopic methods as the number of different samples that can enter the first MS phase is greater than in isotopic labeling and offering higher overall throughput. Typical isobaric protocols are the tandem mass tags (TMT) and the isotopic tags for relative and absolute quantification (iTRAQ)[5]. Despite the fact that labeling methods are less biased compared to the label-free methods, these methods are time consuming and costly compared to the label-free methods[6]. Label-free proteomics were proposed as a fast and cheap alternative were each sample is essentially a different MS run. Protein abundance in each sample can be calculated either by measuring the ion peak intensity or the spectral counting of the peptides. However, this method lacks the analytical power and the elasticity of labeling methods, and it is not ideal when samples have small differences in protein expression levels, as well as having each sample run separately on the mass spectrometer increases the technical bias[7]. In all the previously discussed methods, a typical step of the MS/MS run is the fractionation of the protein mixtures in several fractions, which leads to the improvement of the analytical depth of the mass spectrometer, as it reduces the protein mixture complexity. Providing that the proper experimental method was chosen, researchers have to pick a suitable tool for the downstream analysis. A typical pipeline for a proteomics analysis consists from the following steps: (1) data import of the quantified peptides, (2) filtering of the contaminants/unwanted peptides, (3) normalization of peptides intensities, (4) imputation of missing values, (5) aggregation of the peptides to calculate the abundance of their “parent” proteins, (6) statistical analysis and (7) the resulting plots alongside the produced results in a well-structured format. The statistical analysis step could contain several types of analysis such as differential expression analysis, exploratory analysis, pathway/GO enrichment analysis and much more, depending on the biological

question. Nevertheless, the sequence of the steps, as well as their presence is not fixed. Depending on the approach and the experiment some steps as the filtering and the imputation are not present in some tools, while others follow alternative sequences of steps or even major differences in the type of input files, such as peptide input files or protein input files. Depending on the degrees of freedom that the user ask for, the are tools with modular structure where each step is constructed by the user and others that follow a more conservative approach were the user can only tweak only a few parameters of analysis, desktop or web applications, tools that are dedicated to specific experimental types and others which can deal with a variety of experimental types and setups. The reason for such differences between tools is the constant effort of researchers to fill the gap between specificity and user-friendliness.
Notwithstanding, that this plethora of tools could satisfy the needs of researchers, it is not uncommon for a piece of software not to be maintained or upgraded after a certain period of time. In the shifting landscape of Proteomics, where new pipelines proposed regularly, the need for well-polished techniques and up-to-date tools and methods is crucial. In this paper, we will present the new major improvements on ProteoSign, a web application tool for differential expression analysis for bottom-up Proteomics. ProteoSign 2.0 takes into account the collective knowledge of major existing tools regarding their advantages and disadvantages, and aims to increase the accuracy of the previous version through an enhanced pipeline, offer more plot options, while it significantly decreases the amount of time for a full analysis.

Methods

Designers say that “The joy of an early release lasts but a short time. The bitterness of an unusable system lasts for years”[8]. While new tools are being created everyday, many of them consider their scalability as a trivial detail. This makes promiscuous projects difficult to maintain, and eventually they get abandoned due to the lack of funding, server space and software incompatibility. ProteoSign 2.0 is implemented in R, but with a completely different architecture, focusing on scalability, reproducibility, low memory footprint, high analytical power and speed. It lays the foundations for a better

approach on bioinformatics software design by utilizing the terms code scaffolding and dependencies management from the software design world, making its core architecture one of its key striking features.

Code scaffolding is the automatic code generation, either a priori or on runtime, of predefined code templates, on which the software can be build upon, on a more scalable, powerful and reproducible way. While code scaffolding is a well-established and mature technique used by software developers and computers scientists, bioinformaticians tend to underestimate its importance. This assumption can be easily confirmed by the lack of such tools for bioinformatics software development in R for big projects. For this purpose, we developed the project-initilizeR, an R package aiming to fill this gap[9]. Project-initializeR sets up the project folder which consists from the following files and folders: (1) the data-input folder, (2) the data-output folder (3) the src folder (4) the info folder and (5) the reports folder. The data input folder contains all the input data that will be used by the analysis and are not generated by the analysis. Data-output folder contains all the output data generated by the analysis. The info folder acts as a generic folder for all the other data not affiliated with the analysis itself such as PDFs, PowerPoints, TODOs, etc and acts as a laboratory notebook. Finally the src folder contains all the needed scripts for the analysis. The src folder contains the following scripts: (1) main.R which sets the current working directory and calls the initialize.R, load_data.R, pull_data_from_DB.R, build.R analyze.R scripts, (2) functions.R which contains all the needed functions and can be further splitted into functions_XXX.R, (3) explore.R the which is used for data exploration and the testing of code chunks, (4) the initialize.R script which loads all the needed packages, libraries and data regarding the workspace, loads the functions.R script and sets the global variables of the project, (5) load_data.R which loads all the data-input files (csv/xlsx/txt/RDS etc) needed for the analysis and displays the loaded files, (6) pull_data_from_DB.R which is used for explicitly data input using a database, (7) build.R which is the script where all the data wrangling happens in order to prepare the data, (8) analyze.R which contains the all the analysis steps. In addition, there are scripts such as build_ppt.R, prepare_shiny.R, prepare_markdown.R, markdown_report.Rmd, shiny_report.Rmd for the preparation and the constructions of meaningful reports based on the analysis outcomes. All the scripts, apart from the

explore.R, which is allowed to be messy, should follow the Google's R Style in terms of code style[10]. In **Figure 1** you can see the structure of project-initializeR.

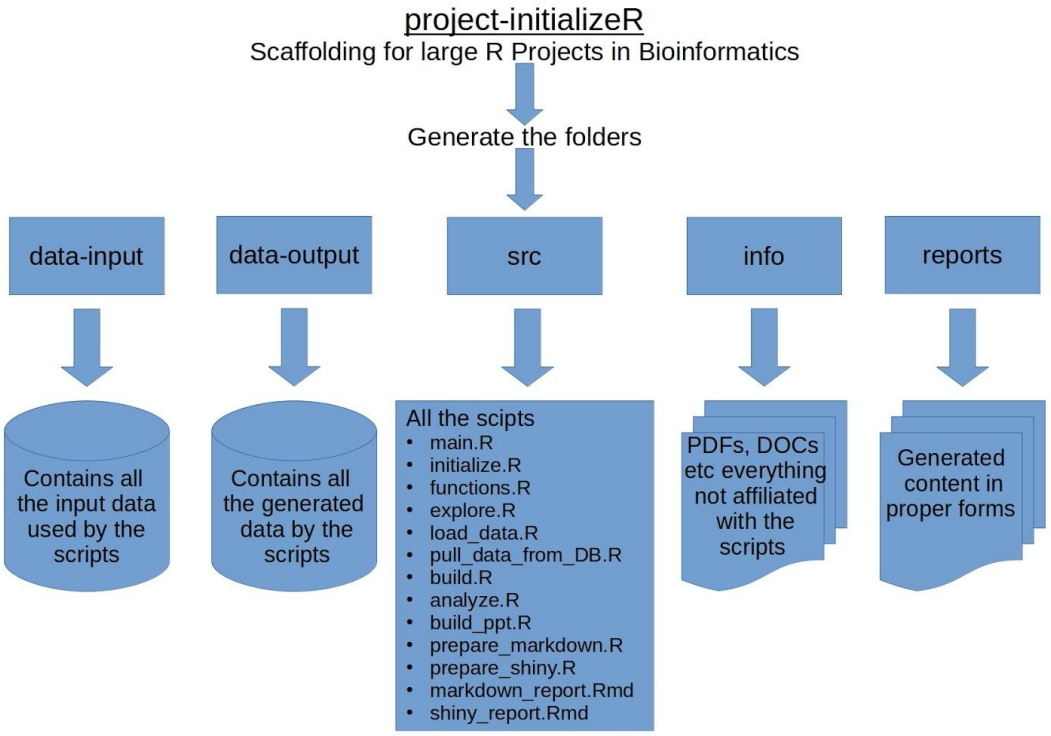

Figure 1 shows the basic architecture of project-initializeR. The user starts by calling the project-initializer.R script, which in turn generates all the needed folders and files, and fills each file with the appropriate content.

The second term borrowed from the software design world is the dependency management. Building data analysis pipelines, using multiple libraries is a challenging - and usually frustrating – task. The developer often struggles to make his pipeline functional due to missing packages, conflicts and unsuitable versions. And even if he or she manages to make the analysis run, isolation, portability and reproducibility are not guaranteed, making dependency management in a non-trivial task during the first stages of software development. In order to overcome this laborious procedure, ProteoSign 2.0 utilizes the Packrat dependency system for R[11]. Packrat is responsible for storing all the needed packages locally in our project directory. During a new R session, R will be able to access only the packages in our private library without affecting our globally installed packages. With Packrat, we ensure that our

projects are bound with specific packages and versions, without worrying that new packages or updates of previously used packages may break our software, as the packages that accompany each project are installed locally. This offers effortless sharing of our projects as the only thing needed in order to ship our project to a different machine, is to take a snapshot of our project and then restore the snapshot on the target machine. Following the aforementioned methods will guarantee the scalability of ProteoSign 2.0, on its future releases.

Apart from the engineering part of our software, clear understanding of the biological question could help us keep things simple and stupid (KISS principle), by not repeating ourselves (DRY principle). Likewise ProteoSign 1.0, ProteoSign 2.0 reads input from either, the proteinGroups.txt and evidence.txt output from the MaxQuant software[12,13], or the PSM file (multiconsensus.txt or psm.txt) output file provided by the Proteome Discoverer™ software[14] (versions 1.4+). Besides the same input support, ProteoSign 2.0 supports multiple biological replicates, technical replicates and fractionation of the samples. Regardless of the experiment type, the ProteoSign 2.0 reads the analysis parameters from the analysis-metadata.csv file, a comma separated file consisting from two columns, the parameter column with multiple parameters, and the value column, with the values of each parameter. The analysis-metadata.csv file contains the following parameters, without quotes: 1) analysis.name, a string variable for the name of the folder which will contain the results of the analysis e.g. PXDXXXX, 2) conditions.to.compare, a list with the 2 conditions to compare with e.g. M, H, 3) replicate.multiplexing.is.used, a boolean variable in case of replicate multiplexing e.g. FALSE, 4) dataset.origin, a string variable stating the origin of the data set e.g. MaxQuant or Proteome-Discoverer 5) is.label.free, a boolean variable stating that the experiment type is Label-Free e.g. FALSE 6) is.isobaric, a boolean variable stating that the experiment type is Isobaric e.g. FALSE, 7) timestamp.to.keep, a string variable for subsetting of the data set in the case of multiple timestamps e.g. 290212, 8) subset.to.keep, a string variable for subsetting of the data set in the case of the simultaneous analysis of different subproteomes e.g. membrane, 9) raw.files.to.remove, a list of strings with raw.files IDs to be removed from the data set e.g. X, 10) raw.files.to.replace, a list of string to overwrite the previously inserted raw.files.to.remove 11) minimum.peptides.detections, an integer variable for the minimum number of detections for a peptide, in order to

be considered as a valid detection e.g. 1 (default is 1), 12) minimum.peptides.per.protein, an integer variable setting the minimum number of peptides belonging to a protein in order to be considered as a valid protein (default is 2),13) min.valid.values.percentance, the percentage of valid values for a protein to XXXX (default is 50),14) plots.format, an integer indicating the format (eps, ps, tex, pdf, jpeg, tiff, png, bmp, svg) of the generated plots (default is 5), 15) error.correction.method, a string variable indicating the error correction method (BH for Benjamini-Hochberg or B for Bonferroni) of the topTreat function during the analysis step (default is BH),16) fold.change.cut.off, a float number indicating the minimum absolute log2-fold-change for a protein in order to be considered as significant and 17) FDR, a float number indicating the minimum cut-off of the adjusted p-value in order to be considered significant. The experimental setup is supported from a universal comma separated file regardless of the analysis, the experimental-structure.csv file, which contains four columns, the raw.file, the biological.replicate, the technical.replicate and the fraction column. The raw.file column contains the raw.files' names as strings, and the rest of the columns are represented as integers. In the special case of a Label-Free experiment, an additional comma separated file, the raw-files-to-conditions.csv, is used. This file contains two columns the raw.file and the condition column, both in string format, where each file is corresponds to a condition. On the other hand, in the case of an isobaric experiment, the tags-to-conditions.csv file consist of two columns, the tag column and the condition column. The tag column contains the ID of the tag and the condition column contains the condition of each tag. The tags-to-conditions.csv and the raw-files-to-conditions.csv follow an identical format.

ProteoSign 2.0 starts with the script main.R bootstrapping the whole analysis. During this early stage, main.R will prepare the ground by cleaning the R environment from any forgotten objects, returning the freed memory to the OS and setting up the Packrat project directory. Following that, main.R will install the package here[15] which makes the browsing between folders easier, load the package here and then it will call the scripts initialize.R, load_data.R, build.R and analyze.R. If by any reason, an exception is raised, the error is catched and the analysis execution is terminated with the appropriate message. During the development stages, an additional package called TODOr was used.

TODOr is an R addin which helps find notes and flags in the code such as FIXME, TODO, CHANGED, BUG etc, available on GitHub[16]. In the development stage TODOr is called alongside the here package any any other package that is used by the development stage is conditionally called or loaded when needed. However, if the analysis is not called for the first time, which means that the packrat environment is already set, there is no need for new installations and the packages are loaded straight away.

The next step on our analysis is to load all the needed packages and functions regarding the analysis. At the very start of every step, we clear the environment from any objects that are no longer necessary, and return the freed memory back to the OS. After that, we continue with the loading of all packages.

The packages used by ProteoSign 2.0 are the:

1. data.table as the main data structure [17]. data.table is an enhanced data.frame, aiming in a fast and memory efficient way of data manipulation. data.table offers a complete range of functions for data read, write, subsetting, updating and reshaping. It is the 10th more starred package on Github and the fastest package of data manipulation across multiple data science packages such as (spark,(py)datatable, dplyr, pandas, dask,Dataframes.jl) as seen on **Figure 2**
2. splitstackshape. The package splitstackshape provides splitting of concatenated data. We utilize the cSplit function for fast column unfolding into multiple lines or columns based on a specific separator[18].
3. vsn for the normalization of peptides' intensities[19].
4. imputeLCMD for the imputation of the missing values[20]
5. limma for the statistical analysis
6. VennDiagram for Venn diagram generation [21]
7. ggplot2 for plot manipulation and genetation[22]. ggplot2 is the most well-established package for plot generation and is a part of tidyverse, a collection of R packages designed for data science, sharing the same philosophy, grammar and data structures

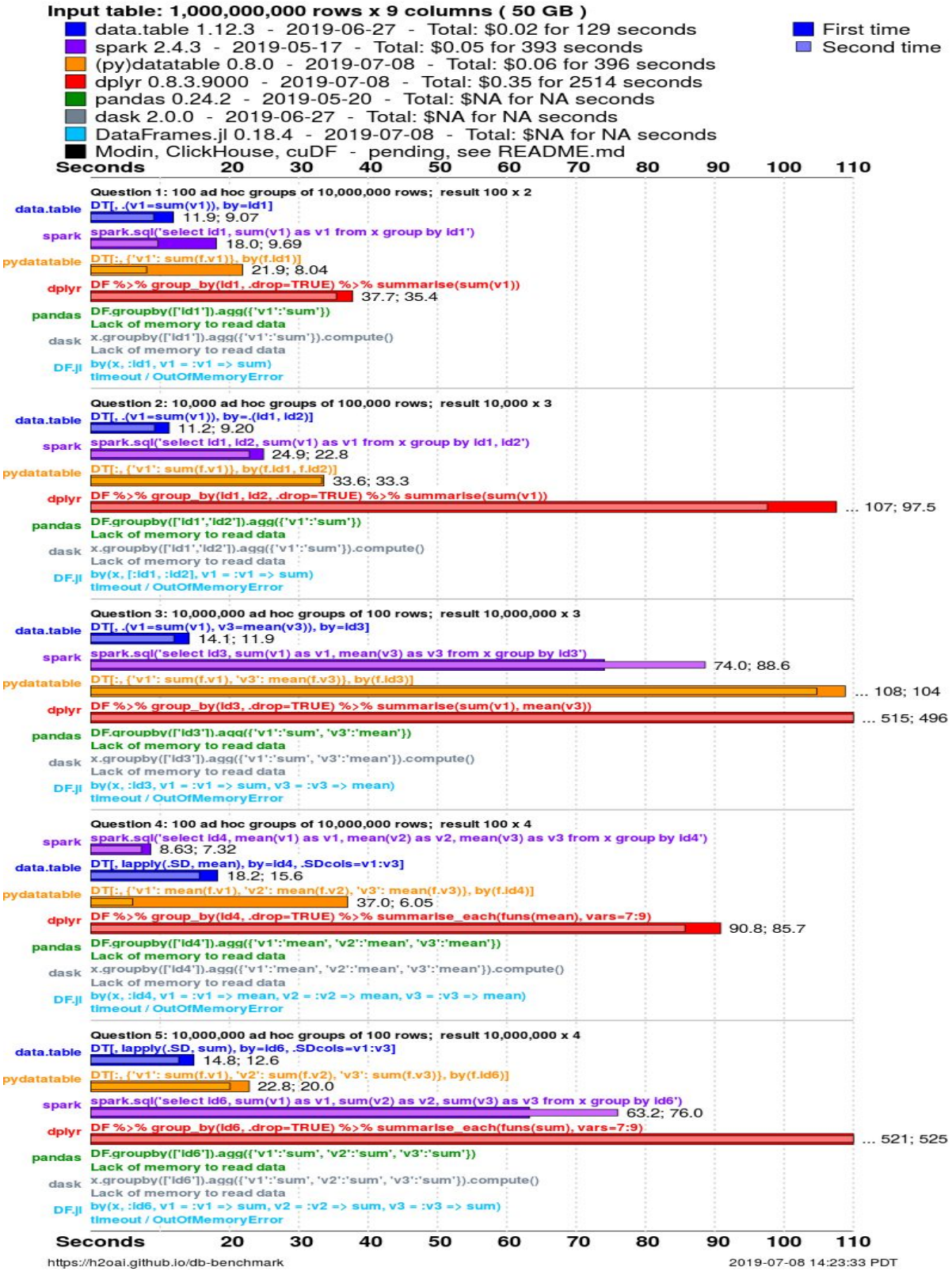

Figure 2 demonstrates the benchmarking results of data.table against the most widely used packages for data manipulation regardless of the technology (Python, R, Julia etc).

During the development stages the package rbenchmark was also used[23]. Rbenchmark was used for the comparison of various methods functions and packages in order to pick the fastest and most memory efficient methods used by ProteoSign 2.0. In **Figure 3** you can see packages used for the construction of ProteoSign 2.0.

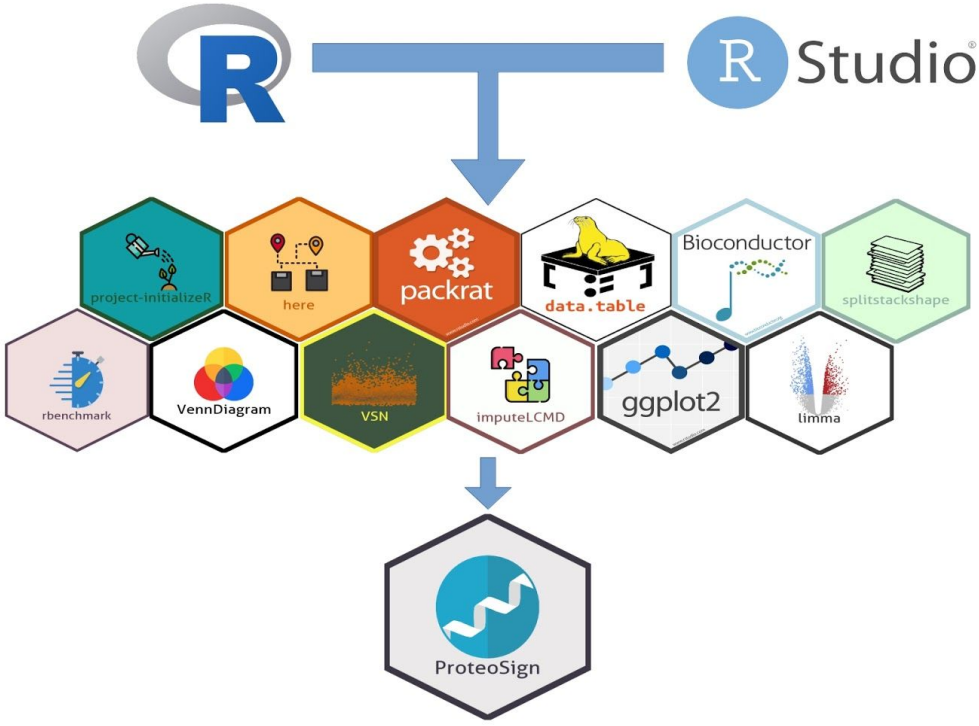

Figure 3 illustrates each package and software that was used for the construction of ProteoSign. The logos were generated using the hexsticker R package.

Finally, we call the functions.R script to load all the functions on our namespace. The functions are split inside the functions_load_data.R, functions_build.R and functions_analyze.R files respectively.

Now that the required packages and functions have loaded on our namespace, we can start with our data. The first file loaded on the memory is the analysis-metadata.csv file with the analysis parameters. The analysis parameters are saved on a global object called “global.variables”in order to be containerized and to be globally available. If the input data set corresponds to a Label-Free experiment, we additionally load the raw-files-to-conditions.csv file, where each raw file is assigned to a condition. On the other hand, if the analysis is tailored for an isobaric experiment, the tag-to-conditions.csv is also read. If

neither of the aforementioned files is loaded, the analysis corresponds to an isotopic experiment. Following that, the next file loaded is the experimental-structure.csv file, which contains the information regarding the experimental setup, matching each raw file to a biological replicate, technical replicate and/or fraction. Next off, the evidence file with the peptides' intensities is loaded. Before we are able to use the evidence file, we have to ensure that the file does not contain any trailing whitespaces. To clean the evidence file fast and efficiently, we utilize the tr Unix command. The tr (translate) command reads a byte stream from a standard input, translates it depending on the two sets provided as input, and writes the result on the standard output. This byte reading behavior gives us great speed and efficiency, as it can translate files of 1GB in a split second. Now that we have cleaned the file, we can read it swiftly and turn it into a data table using the fread function of the data.table package. With our draft evidence data ready, we start standardizing the data by trimming the column names of whitespaces and lowercasing them. After that, regardless of the experiment type, we search through the evidence data column names to find the intensity columns and to lowercase them. Here, we exploit the fact that switch statements are faster than if statements in R. To utilize this information, we produce an experiment ID using the following formula :

$$experimental.type.id = (is.label.free * 1) + (is.isobaric * 2) + 1,$$

where code 1 is for Isotopic experiments, 2 for Label-Free experiments and 3 for Isobaric experiments. Now that we have the intensity columns we can start subsetting the evidence data, keeping only the needed columns. This way the evidence data.table will have smaller memory footprint, making any data transformations execute faster. In the case of a MaxQuant input file, we keep the following columns: 1) proteins, 2) raw.file, 3) protein.ids, 4) protein.names, 5) id, 6) protein.descriptions, 7) peptide.id, 8) unique.sequence.id and 9) the intensity columns. In the case of a Proteome Discoverer input file things are not so trivial. Due to the differences between the software version, column names can vary. In order to deal with that we try to keep the potential columns : 1) id, 2) unique.sequence.id, 3) annotated,sequence, 4) spectrul.file, 5) modifications, 6) protein.group.accessions, 7) protein.accessions, 8) protein.descriptions, 9)

quan.usage, 10) peptide.quan.usage, 11) proteins and 12)protein.groups. In addition, we filter the rest of the column names to get rid of the useless columns. Ideally depending on the experiment type we try to also keep the 1) light/medium/heavy column if the experiment type is Isotopic, 2) the intensity column if the experiment type is Label-Free and 3) the isobaric tag e.g. 126/127/128 etc if we are on an isobaric experiment. This is the most challenging part of the analysis for a dataset coming from Proteome-Discoverer. Having all the needed columns of the experiment data, ProteoSign 2.0 supports further trim of the data.table, offering the option to keep specific time stamps of specific sub-proteomes from our initial evidence data, by searching through the file names and keeping only those that match our criteria. Last but not least, it offers the option to rename specific raw files by reading two sets of file names (old.names/new.names) from the optional arguments found in the experimental_metadata.csv. In the case of a MaxQuant input, we read the additional file proteinGroups.txt, following almost the same pattern as we did with the evidence.txt. We clean the file using the tr command, we store the data into a data.table, trim and lowercase the column names and keep only the following columns: 1) protein.ids, 2) fasta.headers, 3) protein.names, 4) only.identified.by.site, 5) reverse, 6) contaminants, 7) id, 8) peptide.ids and 9) evidence.ids. Whenever each file is read successfully, it is stored to the globally available object “global.variables”.

With the raw data available, we can now start transforming our data into a common format regardless of the analysis type. As previously stated, we start the analysis by removing from the environment any variables or objects apart from the functions loaded, the global variables and the project variables, and return the freed memory to the OS. Then, we read the global variables located in the “global.variables” object and store them locally. In the current version, the tool checks the number of the conditions to be compared, and if the number of conditions to be compared is different than 2, the tool terminates with the appropriate error message. In feature releases, the tool could support the comparison of multiple conditions. In the case of a Label-Free experiment, we order the raw-files-to-conditions.csv matrix by raw.file name and then transform the matrix into a linked list where the outer list contains the conditions and for each condition, the inner list contains the raw files belonging to that condition. On the other hand, if the analysis is targeted for the Isobaric experiment, we

merge the reported intensities for each condition based on the tags provided by the tags-to-conditions.csv and replace the peptides with zero intensity with NAs. After that, we add an extra column to the experimental structure with the condition of each raw file, otherwise if it is not a Label-free experiment, we add the “Labeled Experiment” string as a condition. Next, we order the experimental structure matrix rows by conditions, biological replicates, technical replicates and by fractions. Now, we want to check the experimental structure for its biological validity. The experimental structure ought to have at least two biological replicates for each condition, if the analysis is to be (biologically) valid. The only exception is the case were we utilize the experiment multiplexing approach, where we allow the existence of a single biological replicate per condition. If the number of biological replicates per condition is less than to, the analysis is terminated with the appropriate message. Later on, we want to ensure that the numbering of the biological and technical replicates is correct, as it is typical that the replicates numbering may be mistyped e.g. replicates may be numbered as 1, 2 , 34 instead of 1, 2, 3. To achieve that, we first make list of lists where each element is a condition and each condition contains the biological and technical replicates belonging to the aforementioned condition. Then we check the numbering of the biological and technical replicates for each condition, making a list of list containing booleans. If the biological or technical replicates do not have the correct numbering, the tool will try to correct the numbering, and will also inform the user about the alterations and the fixes proposed by the tool. Finally, the global variables will be updated with the corrected values. Then we want to make a column that describes each line on the experimental.structure. To do that we use the formula:

experimental.setup.id = (1 * biological.replicates.exist) +
(2 * technical.replicates.exist) + (3 * fractions.exist),

This way, depending on the id returned, we know the type of experimental structure. This was the last addition to the experimental.structure matrix and now we can turn in into a data.table for faster manipulation. We also hold the information about the number of biological/technical replicates, fractions for each condition on an object called experiment.metadata. Then, we update the experimental.structure variable on the global.variables object with the one previously made, and with the experiment.metadata as well. Finally, if we are

not on an Isobaric experiment,we replace the values of peptides with zero intensity with NAs. Now, we are able to combine all the data provide into a custom data.table that is common for all the experimental set ups regardless on the analysis type. At this point, the evidence.data object contains 5 columns: 1) the protein column, which contains the proteins that the peptide is supposed to belong, 2) the raw.file column with the name of the sample that the peptide was detected, 3) the intensity column with the intensity of the detected peptide, 4) the id column which contains an incremental unique id for each peptide and 5) the peptide.id column which is an id for each peptide. Firstly, we will unfold the evidence.data data.table. To do that, we start by picking up the correct protein names column. If the data set comes from the MaxQuant, we pick the Proteins column. This column contains all the protein ids that this particular peptide might belong to. There is also the leading proteins columns that contains the protein with the best score that we could pick, but we bypass the MaxQuant scoring system and decide to keep all the information about possible parent proteins. On the other hand, if the data set comes from the Proteome Discoverer software, then there is a variety of possible protein column names such as protein.group.accessions or protein.accessions depending on the version of the software. If the aforementioned column names do not exist, then the software returns an error and stops the analysis. In future releases, the ProteoSign will support more versions of the Proteome-Discoverer. Then, regardless of the origin of the data set, we rename the column with the proteins, as protein.ids. Now that we have picked the protein.ids column. We begin by attempting to find the protein names column. If the column does not exist on our data set, we will utilize the fasta headers column in order to extract the protein names, and we will keep only the first fasta header. After that, we will order the protein.groups.data data.table by the evidence.ids column and remove any lines that does not contain any information on the evidence.ids column. This column contains the peptide ids that we will find in the id column in the evidence.data data.table and essentially, it corresponds to a peptide. Then, we subset the protein.group.data data.table, by keeping only the protein.ids, protein.names and evidence.ids columns. Now with the protein.group.data subset, we unfold the data.table by splitting the evidence.ids entries into multiple lines at the “;” character. To do that fast and efficiently, we use the cSplit method found in the splitstackshape package. Then, we rename the evidence.ids column of the

unfolded protein.groups.subset to id, and if needed, set the column class from string to integer, and we reorder the evidence.data and protein.groups.subset by id. In the case that evidence.data contains the columns protein.names or protein.ids, we remove those columns. With the two data sets, evidence.data and protein.groups.subset sharing common fields, we can now merge them by id, as the union of the two subsets, keeping all the columns and ordering the newly merged superset by id. After that, we order the evidence.data by the protein.ids column, remove any empty lines and update the evidence.column.names variable just in case that the evidence.data file was altered. Next we plan to trim the column with the protein descriptions. This column can contain great volume of information as Taxonomy IDs, Database IDs etc. If our data come from the MaxQuant, the protein.description.column will be the protein.names, otherwise it will be the protein.descriptions column. This step could potentially be used for further data mining. Next we will add to the evidence files a condition column with the condition in which this peptide belongs to. If the experiment type is Isobaric or Isotopic, there will be no column added, as the intensity column will have been added on a previous step. Now, if the conditions provided by the user are more than two, we can easily get rid of the extra columns, and focus on the conditions that we want to compare. Then, depending on the experiment type we will merge the evidence.data data.table with the raw.file and condition columns in the case of a label-free experiment, or with the raw.file column in the case of a Labeled experiment. In both cases we merge the evidende.data with the experimental.structure object using the aforementioned columns as common. Finally, we end up with a common format data.table, containing the protein.ids, the intensity column with the intensities' of a given peptide in a given sample, the condition column with the condition in which each peptide belongs, the protein.ids column, the protein name in which this peptide is believed to belong and the description column, which contains the sample name. Then we count the number of the conditions. If the number of the conditions is 1, then we assume that we have an Isobaric experiment, otherwise is a Label-free or Isotopic/Metabolic labeling experiment. If we are on a Label-free or Isotopic/Metabolic labeling experiment, we transform the data.table by grouping the intensities of each condition for every unique combination of sample, protein.ids and unique.sequence.id. Now instead of the condition and intensity columns, our data.table contains the columns

condition.A and condition.B alongside the columns description, protein.ids and unique.sequence.id. If a unique combination of description, protein.ids and unique.sequence.id does not have a detected intensity (not even a missing value), is marked as NULL and this peptide will be disqualified from any further downstream analysis, as a peptide has to be detected in both conditions for a given combination. After that, we remove any NA protein.ids entries and we store the universal data.table analysis.data in the global.variables object. In the case of an Isobaric experiment, we just find the intensity rows of each column that does not contain any intensity (NA) and remove the union of those rows. Now we have finished with the data.wrangling step and we are ready to proceed with the downstream analysis.

As always, we start our downstream analysis step by cleaning the environment and discarding every object except the global.variables. Then we load all the needed variables for the analysis such as the analysis.data data.table, the minimum.peptides.detections, the minimum.peptides.per.protein, the minimum.valid.values.percentance, the experimental.metadata, the is.label.free boolean variable, the analysis.name, the plots.format, the error.correction.method, the fold.change.cutoff, the FDR and the condition.to.compare variable. Then, we check the analysis.name for its validity. It should not contain any whitespaces or punctuation marks. If there are any, we replace them with dashes. Next, we will make there result folders structure. Each analysis stores its results in a folder named by the analysis.name. Inside of that folder, there are 3 subfolders, the intermediate-data folder, the limma-output folder and the plots folder. The intermediate-data folder contains all the data.table intermediate results in .csv format generated by the downstream analysis at each step (build step,filtering step, normalization step, imputation step, aggregation step, and the data for the VENN diagram), except at the step of the differential analysis using the limma package. The limma-output folder contains the results of the differential analysis of ProteoSign 2.0 in csv format and the plots folder contains all the plots generated by ProteoSign. All the plots are generated in two versions, one with titles and descriptions and one without. The first step on our analysis is to do a VENN diagram of the proteins found in the 2 conditions. The number of the resulted proteins in both conditions will be determined by the restrictions that we introduced before, such as the minimum number of detections for a peptide, the

minimum number of peptides per protein, the number of valid values for a peptide etc. Then we filter our data for contaminants, reverse sequences, and proteins only identified by site. This is done by filtering the analysis.data data.table using the protein.ids column. Following the filtering of the peptides, we will normalize the data and we will plot the normalized and unnormalized intensities of the peptides in each sample. For the normalization step, we will use the VSN (variance stabilizing normalization) method, as it was proposed as the most suitable method for proteomics data normalization as part differential analysis study. A potential drawback of this method ,is that VSN normalization tends to yield underestimated logFC results[24]. We normalize between the conditions we will compare (pairwise-normalization) and if we have multiple detections of a specific peptide inside a sample, we take the median of its detected intensities'. In future upgrades, ProteoSign could offer more normalization methods and/or global normalization. However, the review suggests that the global normalization would not have any impact on AUCs of the analysis. Our next step is the imputation step. At the imputation step we will try to fill the intensities' missing values, using the quantile regression imputation of left-censored data method[25]. This method was proposed as the most suitable method for imputation of left-censored data, which is the type of data that is common in Proteomics experiments [26], and in addition it was shown the superiority of the peptide-level imputation against protein-level imputation[27]. Before the imputation of the values for each condition, we will disqualify those peptides that have more missing values more than our threshold. The default threshold is 40% missing values (min.valid.values.percentage set to 60), but the greater the amount of missing values, the lower the performance of the imputation step. Having our data imputed, we can now proceed to the aggregation step. At this step we will sum the intensities of all peptides that belong to the same protein[28]. Here, the user have control over the strictness of the detected proteins, as he or she can determine the minimum number of peptides belonging to a protein in order for the protein to be accepted in our downstream analysis as truly detected. The default number of minimum peptides per protein is 2 and if a proteins has less peptides detected than the threshold, it is disqualified by further downstream analysis. Following the aggregation of the peptides' we will QQ plot the intensities of the proteins. Finally, the last step of our analysis is the differential

expression analysis using the limma package. We start by constructing the design matrix of our Proteomics experiment. If we do not have technical replicates, the design matrix will be an N x 2 matrix, where the columns correspond to a condition and the rows to a sample. Each cell of this matrix, will contain the value 1 or 0 depending on the fact that this sample belongs to a specific condition. On the other hand, if the experiment contains technical replicates, we construct the limma.block matrix to pass the technical replicates as blocking variables and we also calculate their correlation. Then, we will call the lmFit function of the limma package using the aggregated data and the design matrix as input, and if we have technical replicates, with the limma block and the technical replicates correlation parameter as well, to fit a linear model for each protein of our data set. Next, we call the makeContrasts function of the limma package, to construct the contrasts matrix with the conditions that we will compare and then we call the contrasts.fit calculate the coefficients and the standard errors of our contrasts based on the linear model that we fitted to our data. Now, instead of using the empirical Bayes for the calculation of p-values, we use the treat method from limma package. We prefer the treat method, as we want to examine the null hypothesis that the log2-fold-change of a protein in the conditions we examine, is greater than our user-defined threshold and not just zero, as it is in the eBayes method. In addition, we set the trend argument of treat on TRUE, using the so-called limma-trend method. This will modify the empirical Bayes to also take into account the mean-variance relationship in the modeling of each protein. Apart from the trend method, we utilize the robust Bayes method to protect the empirical Bayes from hyper or hypo variance, a condition that is common in RNA-seq or Proteomics data due to small number of samples. Now that we have all the proteins with their p-values, we apply the error correction method of our choice and we order them by their p-value (lowest to highest) marking the proteins that meet our criteria as differentially expressed. The error correction methods available are the Bonferroni or the Benjamini-Hochberg error correction method. The Bonferroni error correction method, for the control of familywise error rate (FWER), helps us reduce the number of type I errors (false-positives) while the Benjamini-Hochberg is a less stringent control of type I errors that Bonferroni, which controls the proportion of false-positives in the total of our positive results, called false discovery rate (FDR). Finally, we generate some volcano plots, MA plots, value-ordered plots

and fold-change histograms of the results. Now, all the results of the downstream analysis (plots, data, intermediate data) are saved under the data-output folder, inside the folder with the name of the analysis.

Results

ProteoSign 2.0 updated pipeline can be summed up in **Figure 4** while the old pipeline of version 1.0 is demonstrated on **Figure 5**. Apart from its novel pipeline of ProteoSign 2.0, we wanted to test its analytical power compared to the old version of the software. To do that, we decided to test the two versions against the PRIDE PXD001373 data set. The data set belongs to a Label-Free experiment, analysed by MaxQuant that contained a differential expression analysis aiming to shed light on the pathogenicity of Pseudomonas aeruginosa through its ability to form biofilms. A key structure of these biofilms are fibrils that are formed by a functional amyloid system. This system is coded by a 6-gene operon (FapA-F). On this paper, the researchers examine the differences of the biofilm characteristics between a wild-type strain of P. aeruginosa (PAOwt) with low Fap expression and one with overexpression (PAO1pFap).

The data set was analyzed with ProteoSign 1.0, ProteoSign 2.0 , Perseus, InfernoRDN, Prostar and MSqRob. The analysis time was 2 minutes for ProteoSign 2.0 and 11 minutes for ProteoSign 1.0 suggesting a significant decrease on the analysis time for the new version of the software. The analysis with Perseus and InfernoRND took 2 minutes, 5 minutes for Prostar and 35 for MSqRob. **Figure 6** contains the total differentially expressed proteins for each tool. Interestingly, only 121 proteins were marked as differentially expressed by both versions of ProteoSign as we can see on **Figure 7**. MSqRob is absent from the following plots as it failed to detect and proteins as DE (while the ridge regression models of MSqRob yielded significant proteins, the logFC of ther DE proteins was below the 1.5 FC threshold).

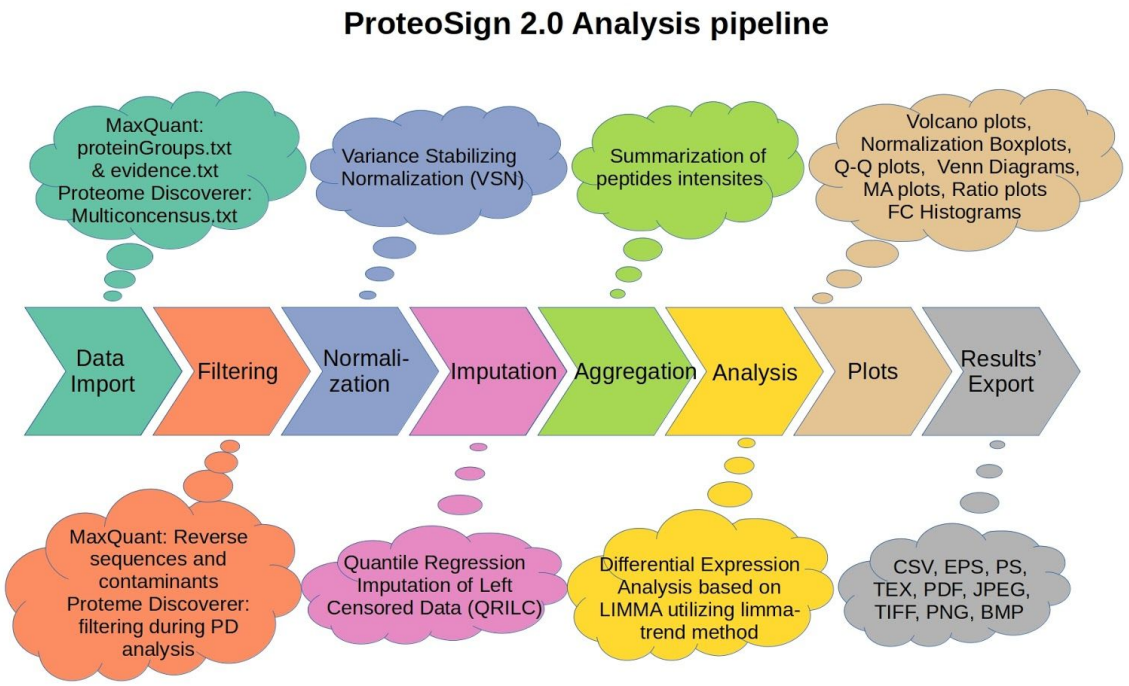

Figure 4 summarizes the pipeline of ProteoSign 2.0. While ProteoSign 1.0 begins with the aggregation of the peptides and then proceeds with the normalization of the protein intensities', ProteoSign 2.0 does the normalization on peptide-level and also incorporates an imputation step.

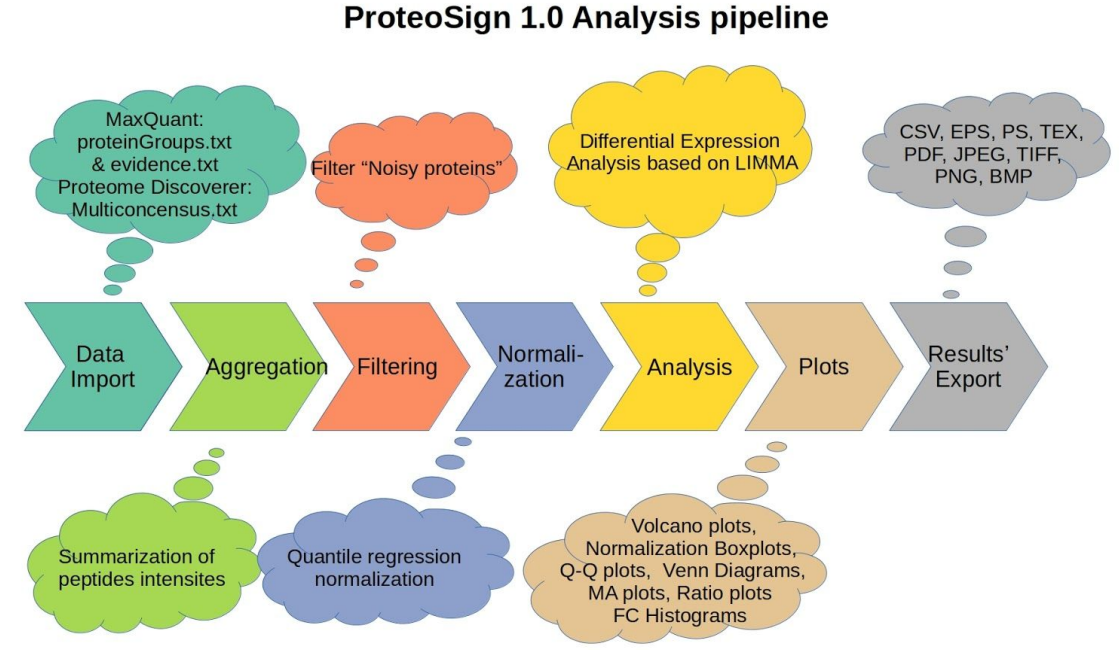

Figure 5 contains the old pipeline of ProteoSign 1.0. In the old pipeline the analysis starts with the aggregation of the peptides, while in the new pipeline it is the last step before the LIMMA model fit.

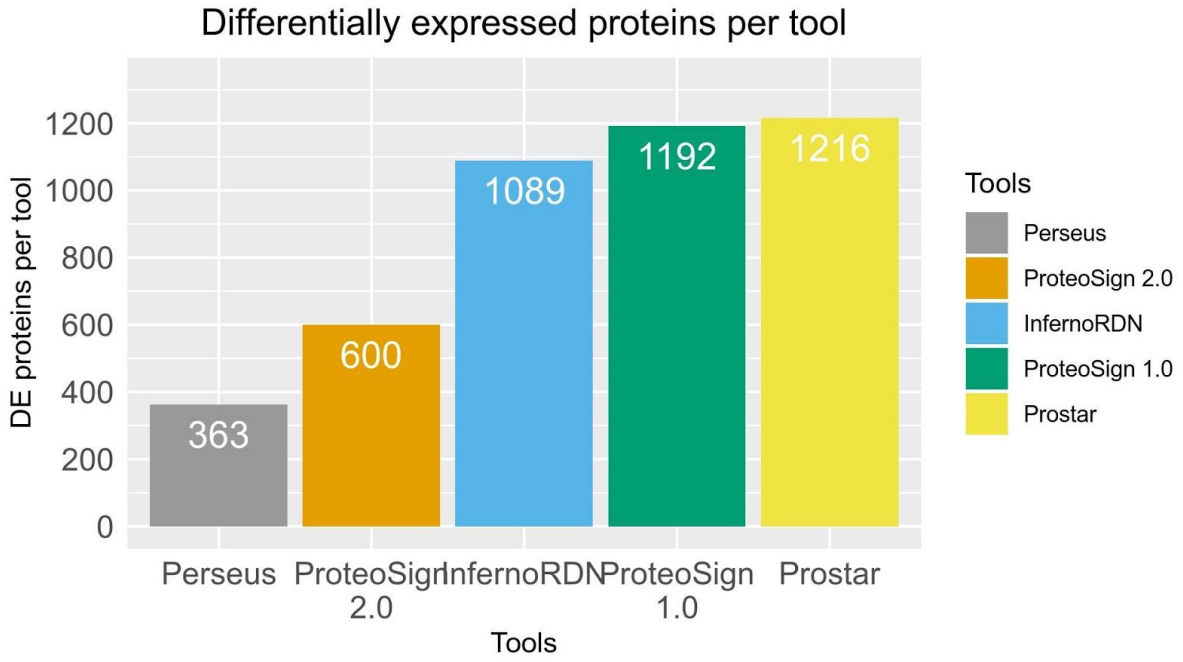

Figure 6 contains a bar plot with the amount of differentially expressed proteins for each tool. While we tried each tool to follow the same analysis pipeline, even minor differences in the methods for each step or even between versions of the same tools can lead to different results.

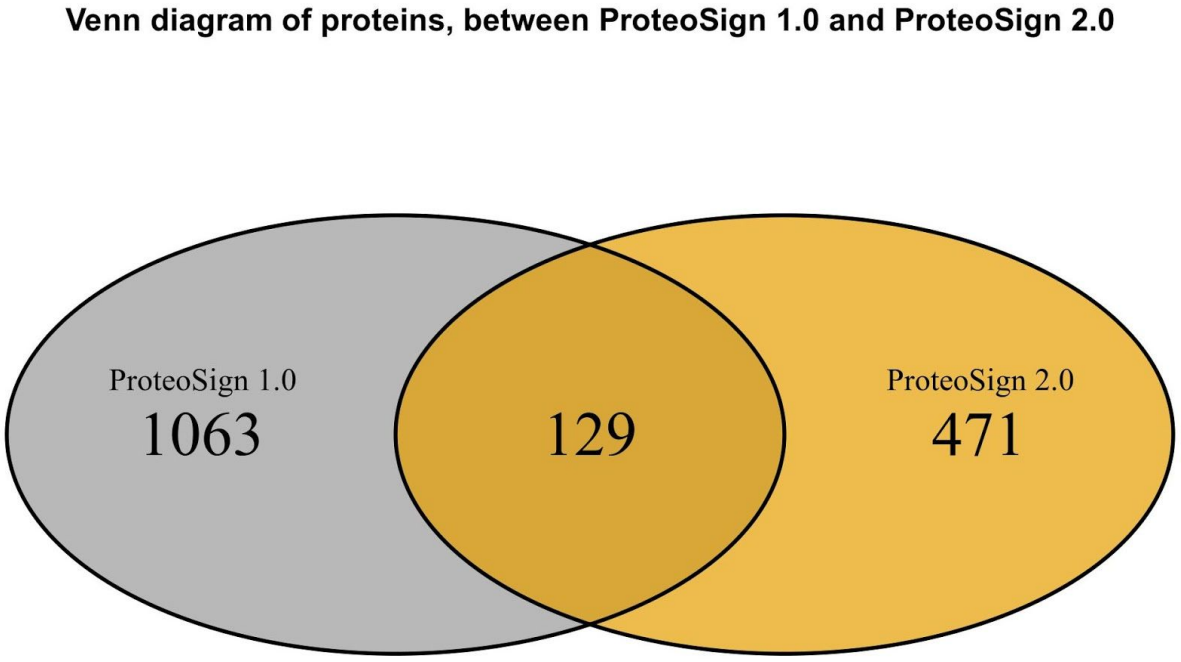

Figure 7 Venn diagram of the detected proteins as differentially expressed between the two versions of ProteoSign. The difference between the amount of the detected proteins can be explained by the difference in their pipelines as well as in the more strict approach of version 2.0.

In the original paper, it was supported that 41 proteins associated with the functional amyloid system were differentially expressed. As we see on **Figure 8**,
there are various differences between the tools.

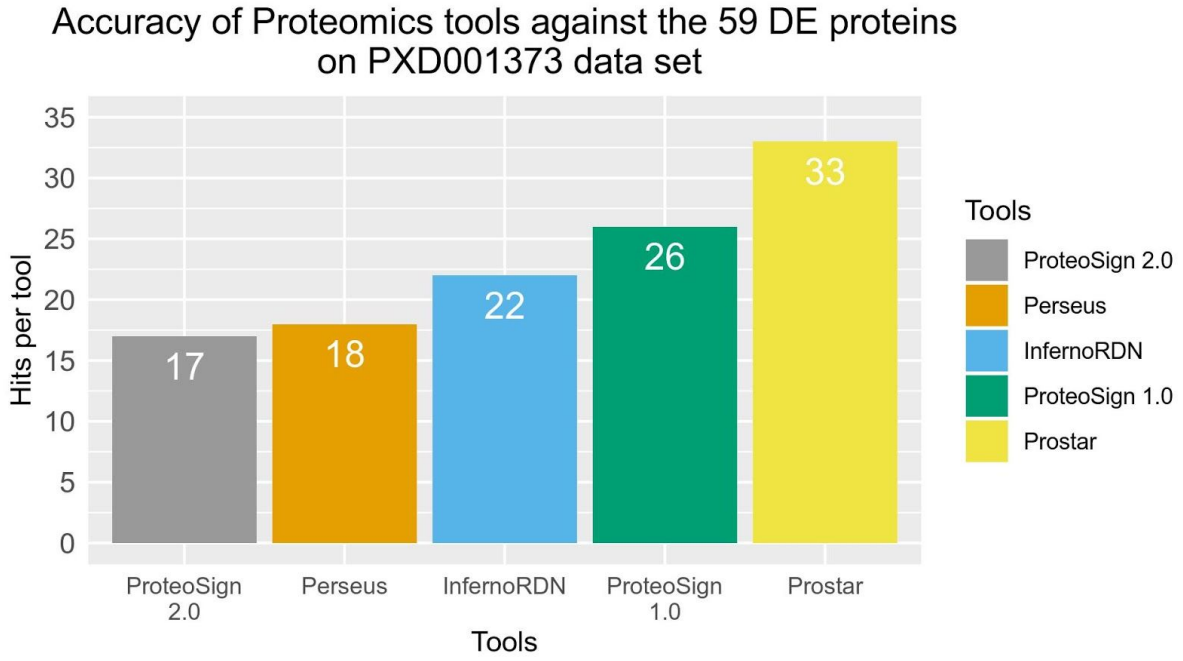

Figure 8 shows the barplots with the number of hits of each tool against the 59 differentially expressed proteins detected by the study for the PXD001373 dataset.

However, to decide which tool had the best performance over our data set, we use the F1 score metric. F1 score offers a harmonic mean between the specificity and the sensitivity of a test. As seen on **Figure 9** ProteoSign 2.0.

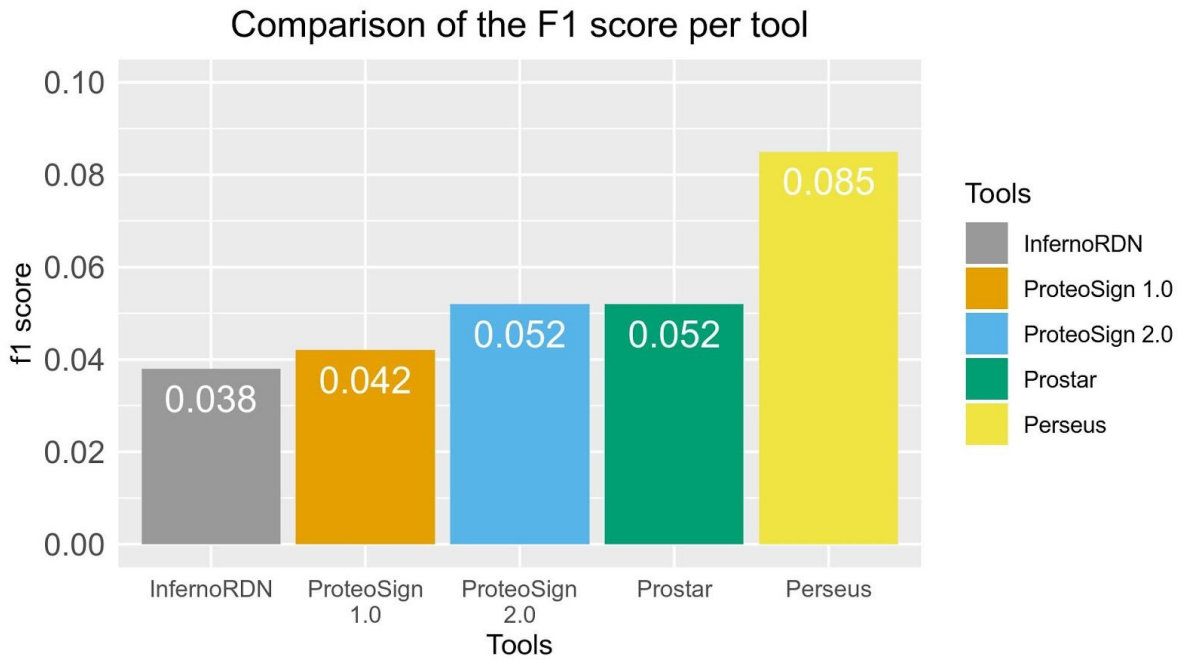

Figure 9 contains demonstrates the F1 score of each tool against 59 DE proteins of the original data set. As we can see ProteoSign 2.0 scores the second best score across the tools, balancing specificity and sensitivity.

Finally in **Figure 10** you can find a collage of the analysis' plots for the PXD001373 dataset.

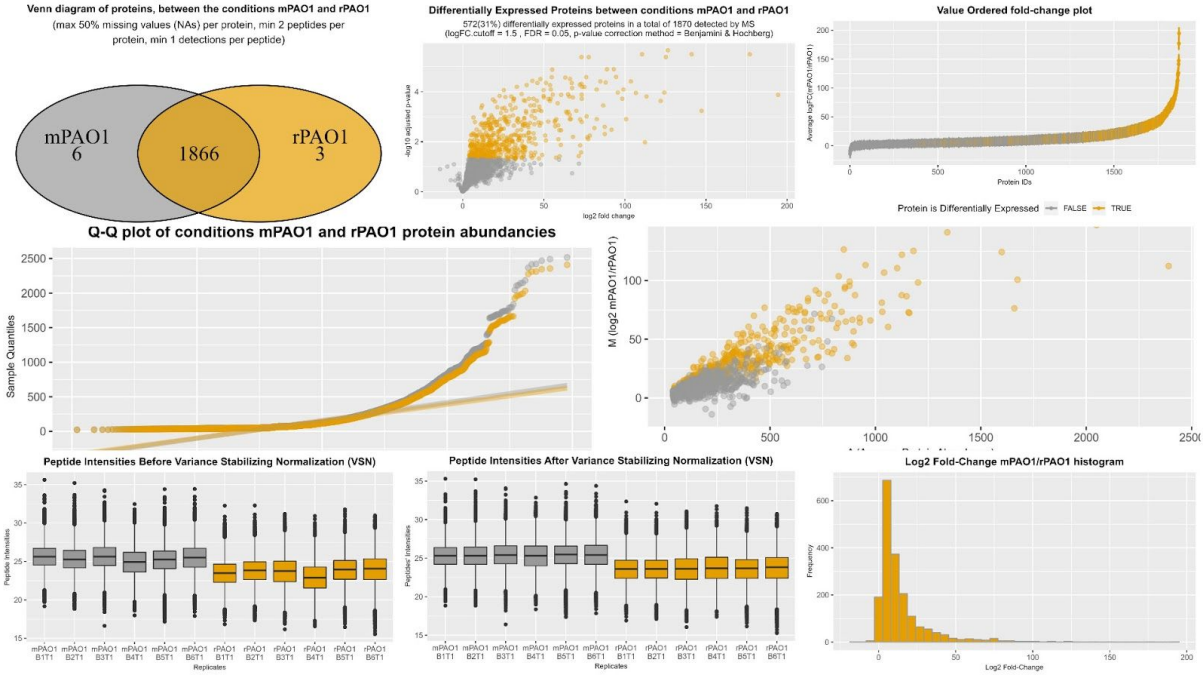

Figure 10 is a collage of all the descriptive plots of the PXD001373 data set analysis, produced by ProteoSign 2.0. The plots from top-to-bottom and from left-to-right are: 1) venn diagram, 2) volcano

plot, 3)value ordered plot , 4) Q-Q plot, 5) MA plot, 6) boxplots of each replicate before the normalization, 7) boxplots of each replicate after the normalization, 8)histogram of the logFC.

Discussion

Differential analysis of Proteomics data can offer us better understanding of the poly-dynamic potential of Proteome. Despite the challenging nature of wet-lab methods and techniques of Proteomics' data generation, both labeling and label-free techniques have made a great impact on the field of Proteomics, offering us accurate results, increased number of proteins successfully detected and shorter analysis' durations. The availability of new dry-lab methods for each individual step in the differential expression analysis pipelines, demands that scientists keep their tools sharp through constant evaluation, re-evaluation and maintenance of existing software tools. ProteoSign 2.0 embraces the idea of natural selection in software design, were the old methods are altered, re-evaluated and enhanced or replaced by new ones, adapting to the constantly shifting landscape of data analysis. It tries to fill the gaps of the previous version by introducing an alternative pipeline, alongside new steps for data normalization and imputation. A challenging part of such software tools, are not only the methods that change day by day, but the lack of standardized protocols and data formats in data input too, which makes the life of both software developers and biologists hard. ProteoSign 2.0 is planned to run under a new more user-friendly web page in the future. The web-page will be built upon the MEAN stack (MongoDB, Express, Angular and NodeJS) and with the MaterializeCSS CSS framework, focusing on UX, and running as a whole on a Docker Container to promote software portability. In addition its functionality could expand further, with GO enrichment analysis and pathway analysis. Last but not least, the project-initializeR project could mature into a complete CRAN or Bioconductor package and become a standard for R-based bioinformatics analysis scaffolding.

References

1. Anderson, N. L. & Anderson, N. G. Proteome and proteomics: New technologies, new concepts, and new words. *Electrophoresis* 19, 1853–1861 (1998).

2. Hebert, A. S. *et al.* The One Hour Yeast Proteome. *Mol. Cell. Proteomics* 13, 339–347 (2014).

3. Coorssen, J. R. & Yergey, A. L. Proteomics Is Analytical Chemistry: Fitness-for-Purpose in the Application of Top-Down and Bottom-Up Analyses. *Proteomes* 3, 440–453 (2015).

4. Schwanhäusser, B., Gossen, M., Dittmar, G. & Selbach, M. Global analysis of cellular protein translation by pulsed SILAC. *Proteomics* 9, 205–209 (2009).

5. Gevaert, K. *et al.* Stable isotopic labeling in proteomics. *Proteomics* 8, 4873–4885 (2008).

6. Patel, V. J. *et al.* A comparison of labelling and label-free mass spectrometry-based proteomics approaches. *J. Proteome Res.* 8, 3752–3759 (2000).

7. Zhu, W., Smith, J. W. & Huang, C.-M. Mass spectrometry-based label-free quantitative proteomics. *J. Biomed. Biotechnol.* 2010, 840518 (2010).

8. Pavelin, K. *et al.* Bioinformatics meets user-centred design: a perspective. *PLoS Comput. Biol.* 8, e1002554 (2012).

9. Vangelis Theodorakis. project-initializeR: Scaffolding for large R Projects. (2018). Available at: https://github.com/VangelisTheodorakis/project-initializeR. (Accessed: 24th June 2019)

10. Google's R Style Guide. Available at: http://web.stanford.edu/class/cs109l/unrestricted/resources/google-style.html. (Accessed: 24th June 2019)

11. Packrat: Reproducible package management for R. Available at: https://rstudio.github.io/packrat/. (Accessed: 24th June 2019)

12. Cox, J. & Mann, M. MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification. *Nat. Biotechnol.* 26, 1367–1372 (2008).

13. Tyanova, S., Temu, T. & Cox, J. The MaxQuant computational platform for mass spectrometry-based shotgun proteomics. *Nat. Protoc.* 11, 2301–2319 (2016).

14. Proteome Discoverer™ Software. Available at: https://www.thermofisher.com/order/catalog/product/OPTON-30795. (Accessed: 24th June 2019)

15. Müller, K. A Simpler Way to Find Your Files [R package here version 0.1].

16. Krzemiński, D. Find All TODO Comments and More [R package todor version 0.0.5].

17. Matt, D. data.table: Extension of 'data.frame'. Available at: https://cran.r-project.org/web/packages/data.table/index.html. (Accessed: 24th June 2019)

18. Ananda, M. splitstackshape: Stack and Reshape Datasets After Splitting Concatenated Values. (2019). Available at:

https://cran.r-project.org/web/packages/splitstackshape/index.html. (Accessed: 24th June 2019)

19. Huber, W., von Heydebreck, A., Sultmann, H., Poustka, A. & Vingron, M. Variance stabilization applied to microarray data calibration and to the quantification of differential expression. *Bioinformatics* 18, S96–S104 (2002).

20. Lazar, C. imputeLCMD: A collection of methods for left-censored missing data imputation version 2.0 from CRAN. *2015* Available at: https://rdrr.io/cran/imputeLCMD/. (Accessed: 30th April 2018)

21. Hanbo, C. VennDiagram: Generate High-Resolution Venn and Euler Plots. Available at: https://cran.r-project.org/web/packages/VennDiagram/index.html. (Accessed: 24th June 2019)

22. Hadley, W. Create Elegant Data Visualisations Using the Grammar of Graphics [R package ggplot2 version 3.2.0].

23. Kusnierczyk, W. Benchmarking routine for R [R package rbenchmark version 1.0.0].

24. Välikangas, T., Suomi, T. & Elo, L. L. A systematic evaluation of normalization methods in quantitative label-free proteomics. *Brief. Bioinform.* 19, bbw095 (2016).

25. Hastie, T., Tibshirani, R., Narasimhan, B. & Chu, G. impute: Imputation for microarray data. Available at: https://bioconductor.riken.jp/packages/3.2/bioc/html/impute.html. (Accessed: 30th April 2018)

26. Wei, R. *et al.* Missing Value Imputation Approach for Mass Spectrometry-based Metabolomics Data. *Sci. Rep.* 8, 663 (2018).

27. Lazar, C., Gatto, L., Ferro, M., Bruley, C. & Burger, T. Accounting for the Multiple Natures of Missing Values in Label-Free Quantitative Proteomics Data Sets to Compare Imputation Strategies. *J. Proteome Res.* 15, 1116–1125 (2016).

28. Carrillo, B., Yanofsky, C., Laboissiere, S., Nadon, R. & Kearney, R. E. Methods for combining peptide intensities to estimate relative protein abundance. *Bioinformatics* 26, 98–103 (2010).