Expanding the Perseus Software for Omics Data Analysis With Custom Plugins

Sung-Huan Yu,[1,5] Daniela Ferretti,[1,5] Julia P. Schessner,[2,5] Jan Daniel Rudolph,[1,3] Georg H. H. Borner,[2] and Jürgen Cox[1,4,6]

[1]Computational Systems Biochemistry Research Group, Max-Planck Institute of Biochemistry, Martinsried, Germany
[2]Systems Biology of Membrane Trafficking Research Group, Max-Planck Institute of Biochemistry, Martinsried, Germany
[3]Bosch Center for Artificial Intelligence, Robert-Bosch-Campus 1, Renningen, Germany
[4]Department of Biological and Medical Psychology, University of Bergen, Bergen, Norway
[5]These authors contributed equally to this work.
[6]Corresponding author: *cox@biochem.mpg.de*

The Perseus software provides a comprehensive framework for the statistical analysis of large-scale quantitative proteomics data, also in combination with other omics dimensions. Rapid developments in proteomics technology and the ever-growing diversity of biological studies increasingly require the flexibility to incorporate computational methods designed by the user. Here, we present the new functionality of Perseus to integrate self-made plugins written in C#, R, or Python. The user-written codes will be fully integrated into the Perseus data analysis workflow as custom activities. This also makes language-specific R and Python libraries from CRAN (cran.r-project.org), Bioconductor (bioconductor.org), PyPI (pypi.org), and Anaconda (anaconda.org) accessible in Perseus. The different available approaches are explained in detail in this article. To facilitate the distribution of user-developed plugins among users, we have created a plugin repository for community sharing and filled it with the examples provided in this article and a collection of already existing and more extensive plugins. © 2020 The Authors.

Keywords: MaxQuant • omics data analysis • Perseus • plugin development • quantitative proteomics

Current Protocols in Bioinformatics e105, Volume 71
Published in Wiley Online Library (wileyonlinelibrary.com).
doi: 10.1002/cpbi.105

How to cite this article:
Yu, S.-H., Ferretti, D., Schessner, J. P., Rudolph, J. D., Borner, G. H. H., & Cox, J. (2020). Expanding the perseus software for omics data analysis with custom plugins. *Current Protocols in Bioinformatics*, *71,* e105. doi: 10.1002/cpbi.105

INTRODUCTION

The complex downstream analysis of proteomic data requires the integration of bioinformatics, statistics, network analysis, and, frequently, machine learning. This has led to the development of the Perseus software (Tyanova et al., 2016) as a comprehensive multipurpose tool and framework for such analyses. The user-friendly interface facilitates a variety of data transformations and visualizations and provides gapless documentation, a storable analysis workspace, and a visual representation of the analysis workflow. The options for multidimensional omics data analysis include normalization, pattern recognition, time-series analysis, cross-omics comparisons, and controlled multiple-hypothesis testing.

The core data structure is a matrix, containing one row per entry in the dataset, usually a protein or protein group. The columns can contain variable information of different data types. Perseus distinguishes between “Numerical” columns containing single number values, “Multi-numerical” columns that can contain more than one value in one cell (split by semi-colons), “Categorical” columns that can contain binary flags, grouping information, or biological annotation of individual entries (which can be added through Perseus), and, finally “Text” columns for anything that is neither a number nor a category. Perseus additionally distinguishes the columns containing the “Main” data for each entry, e.g., the quantitative expression values of each protein group in different samples. These types can be specified on data import and changed throughout the analysis. In addition to the annotation columns, annotation rows can be defined to specify column grouping parameters such as biological conditions and technical replicates. This structure makes Perseus very flexible, allowing statistical analysis for a considerable variety of experimental designs and thereby facilitating hypothesis generation (Rudolph & Cox, 2019).

Despite this broad applicability there will always be a specific case study or new technology that requires additional functionalities so far not provided by Perseus. To expand Perseus’ functionality with plugins is neither difficult nor complex: the core data structure is propagated to external plugins and back to the Perseus framework through well-defined interfaces. Newly implemented functionalities can be directly incorporated into the Perseus interface, making them indistinguishable from the core functions. This had already been possible for plugins written in C#, since it is Perseus’s native language, but it is now also possible for plugins written in other languages including R and Python. This facilitates the interoperability of Perseus with external scripting languages, and allows developers to use a language they are already comfortable with. The backend providing this new functionality is called PluginInterop: it runs the external plugin with a specified executable and facilitates passing additional arguments to the plugin (Rudolph & Cox, 2019). Since R and Python are widely used in data science, two companion libraries are provided for these two languages to be used alongside PluginInterop: PerseusR and perseuspy. They provide the other half of the interface for seamless transfer of matrices and annotations from Perseus to R/Python data frame objects and vice versa.

In this article, we provide extensive explanations for all the steps and details of creating custom plugins for Perseus in all three languages—starting from the basic installation steps and the use of the different interfaces and proceeding all the way to advanced analysis plugins. Together with the protocols, we provide a GitHub repository (*https://github.com/JurgenCox/perseus-plugin-programming*) where the given examples are available for download, as well as a list of already existing plugins of varying complexity and where to find them online (Table 1).

STRATEGIC PLANNING

Before starting to develop a new Perseus plugin, a few things need to be considered, for instance, which language should be employed, who is going to use the plugin, and how many additional arguments are needed for the plugin to work. These are relevant questions, since there are two ways of integrating non-C# plugins into Perseus. They can be incorporated with a command line interface or with a C# wrapper for R/Python plugins. The command line interface lets you select the plugin script file and provides a single input line for arguments to be passed to the script. On the other hand, the C# wrapper generates a small graphical user interface to ask for parameter values and adds a separate entry to a selectable interface menu in Perseus, thereby avoiding the manual selection of the script file before every run (see Figs. 1-4). Thus, a C# wrapper for R/Python plugins is not required if the plugin is meant to be used only by the developer or users comfortable with the command line interface. Conversely, the use of a C# wrapper is highly recommended if a broad user base is expected or a larger number of arguments needs to be supplied to the plugin. A third alternative is to write the plugin entirely in C#, in which case the integration is direct, and no wrapping code is necessary. For all these variants, detailed protocols are provided. If an R or Python plugin is being developed, it is always possible to initially use the command line interface, and then add a C# wrapper before it is released to other users, as long as some conventions are followed. All the software development tools required for Perseus plugin development are freely available.

BASIC STEPS FOR R PLUGINS

BASIC PROTOCOL 1

R is one of the most widely used programming languages for bioinformatics. Numerous packages for statistical data analysis and visualization have been created by R developers. In order to make Perseus more powerful by making these functions available from within the software, a package for integrating R scripts into Perseus was developed—PerseusR (Rudolph & Cox, 2019). With this, all custom tools originally scripted in R can now be used within Perseus. In this basic protocol, a simple example of an R-only plugin, extracting the head (top rows) of a matrix, will be presented to illustrate how the data transfer between Perseus and R functions. This example will be run through the command line style interface. The code of this example is available at: *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/scripts/head.R*.

Necessary Resources

Hardware

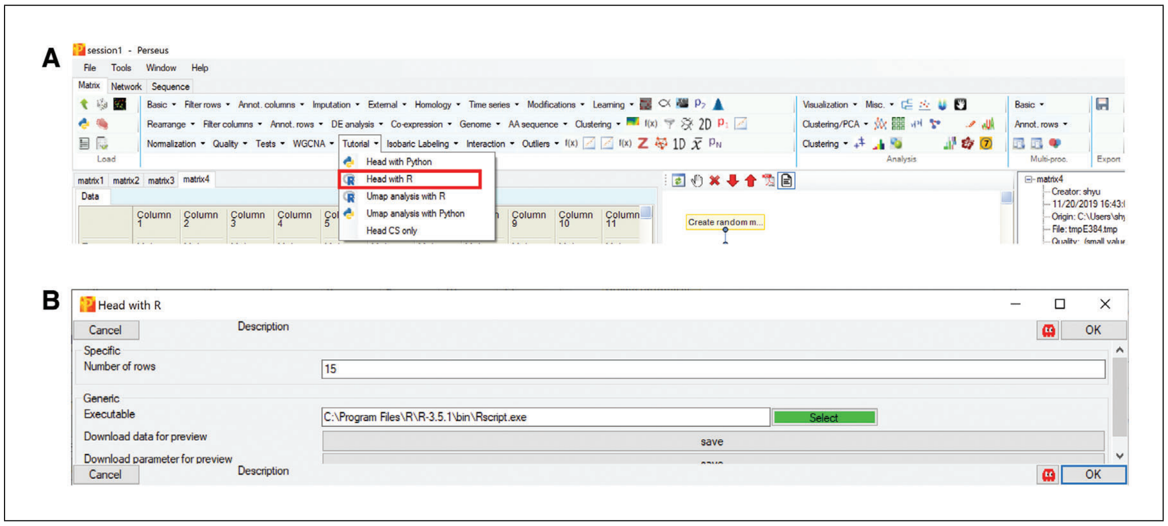
A computer running Windows 8 (64 bit) or higher, or Windows Server 2008 or higher
4 GB RAM minimum
At least a quad core processor is recommended

Software

Perseus 1.6.13 or higher:–can be downloaded from *https://maxquant.org/perseus*
R: Please use a version $\geq$ 3.5.0. The Rscript executable has to be listed in the PATH environment variable of the operating system. Please refer to the

Table 1 Examples of Perseus Plugins

Plugins	Usage	Languages	Reference/link
DualQualityFilter	Dual-quality matrix filter; intended to be used for filtering SILAC data based on MS/MS count and variability.	R	*https://github.com/JuliaS92/PerseusR-DualQualityFilter*
ProfileCorrelation	For each row, calculates pairwise correlations between profiles defined by categorical annotation rows	Python	*https://github.com/JuliaS92/PerseusPy-ReplicateCorrelation*
DE analysis	Differential expression analysis for Omics data. The algorithms include DESeq2 (Love, Huber, & Anders, 2014), EdgeR (Robinson, McCarthy, & Smyth, 2010) and Limma (Ritchie et al., 2015).	R + C#	DESeq2, EdgeR, and Limma
Clustering	Dimensionality-reduction methods. The newly added algorithms are UMAP (Becht et al., 2019; McInnes et al., 2018) and tSNE (Maaten, van der, van der Maaten, & Hinton, 2008).	R/Python + C#	UMAP and tSNE
imputeLCMD	The collection of imputation methods for proteomics data. The software of imputeLCMD (Johnson, Li, & Rabinovic, 2007) is from sva (Leek, Johnson, Parker, Jaffe, & Storey, 2012).	R + C#	sva
Quantile normalization	Making the distributions identical in statistical properties. The software is from Limma (Ritchie et al., 2015).	R + C#	Limma
Remove batch effect (proteinGroup)	Remove the batch effect in protein group level. The algorithms contain Limma (Ritchie et al., 2015) and ComBat (Johnson et al., 2007).	R + C#	Limma and ComBat
PHOTON	Elucidation of Signaling Pathways from Large-Scale Phosphoproteomic Data Using Protein Interaction Networks (Rudolph & Cox, 2019)	Python + C#	*https://github.com/jdrudolph/photon*
WGCNA	Weighted correlation coefficient network analysis (Langfelder & Horvath, 2008)	R + C#	WGCNA
Proteomics ruler	Proteomics normalization without spike-in standard (Wiśniewski, Hein, Cox, & Mann, 2014).	C#	*https://maxquant.org/perseus_plugins*



Head with R

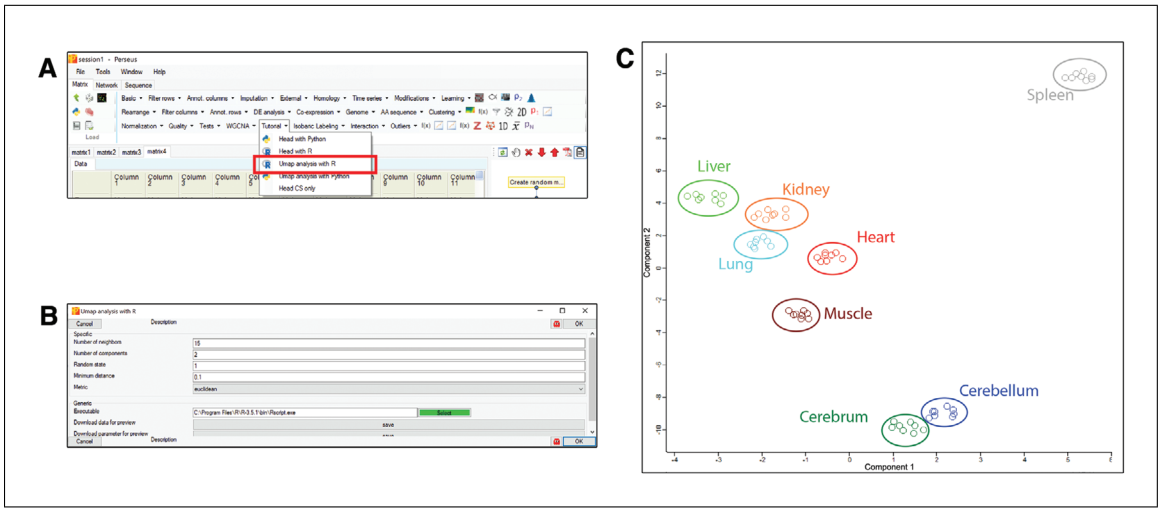
Specific

Number of rows	15

Generic

Executable	C:\Program Files\R\R-3.5.1\bin\Rscript.exe
Download data for preview	save
Download parameter for preview	

Figure 1 C# + R plugin in Perseus. (**A**) Highlights the menu item for running C# + R plugin. The pop-up window containing the parameters of the plugin is shown in (**B**).



Umap analysis with R

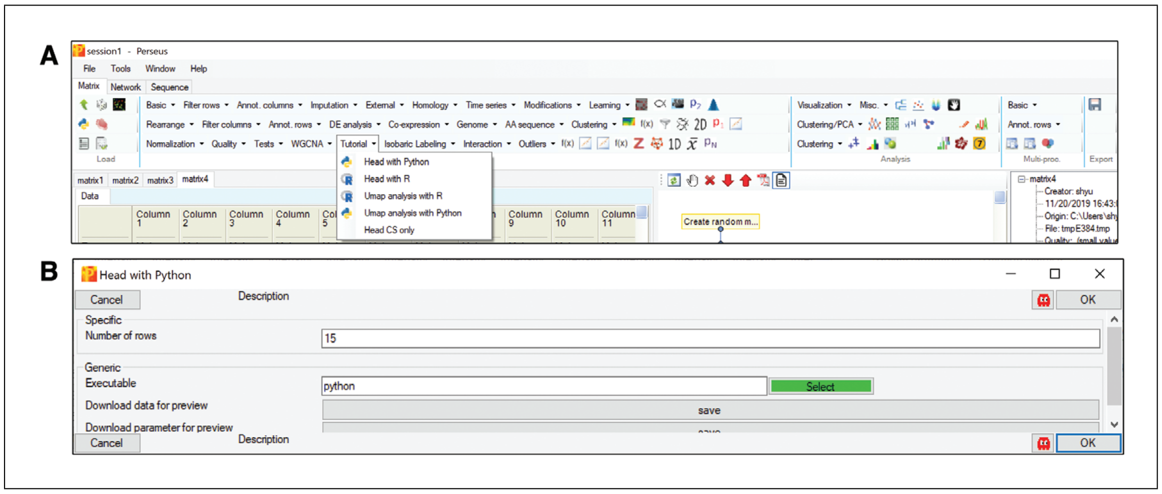
Specific

Number of neighbors	15
Number of components	2
Random state	1
Minimum distance	0.1
Metric	euclidean

Generic

Executable	C:\Program Files\R\R-3.5.1\bin\Rscript.exe
Download data for preview	save
Download parameter for preview	

Figure 2 C# + R plugin for running UMAP analysis. (**A**) Menu item for running UMAP plugin. The pop-up window with the arguments of the plugin is presented in (**B**). (**C**) Shows the results of UMAP analysis in a scatter plot in Perseus.



Head with Python

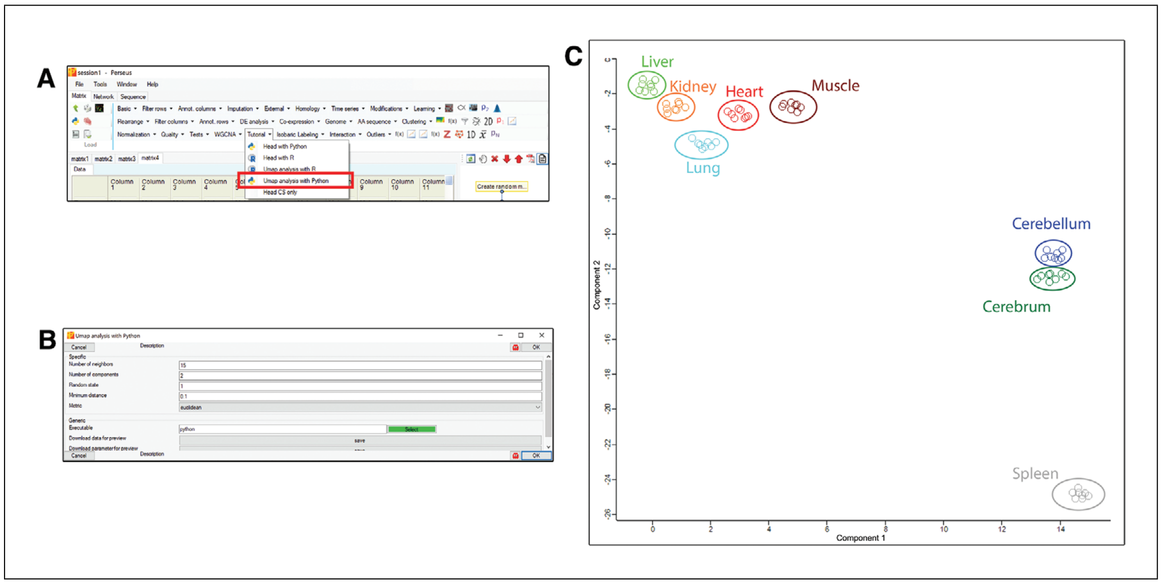
Specific

Number of rows	15

Generic

Executable	python
Download data for preview	save
Download parameter for preview	

Figure 3 C# + Python plugin in Perseus. (**A**) Menu item for running C# + Python plugin. The pop-up window containing the parameters of the plugin is shown in (**B**).



Specific
Number of neighbors: 15
Number of components: 2
Random state: 1
Minimum distance: 0.1
Metric: euclidean
Generic
Executable: python
Download data for preview: save
Download parameter for preview: save

Figure 4 C# + Python plugin for running UMAP analysis. (**A**) Highlights the menu item for running the UMAP plugin. The pop-up window including the arguments of the plugin is presented in (**B**). (**C**) Shows the results of UMAP analysis in a scatter plot.

"Troubleshooting" if Perseus cannot find your R installation, which is indicated in the command line style interface.

PerseusR: available at *https://github.com/cox-labs/PerseusR*, where installation instructions are provided

install.packages("devtools")

library(devtools)

install_github("cox-labs/PerseusR")

R-supported editor like Visual Studio, RStudio or Notepad++

Input files

This protocol requires no extra input files. The outlined plugin works with a randomly generated matrix, which can be done using the dice button in Perseus.

1. Parse command line arguments from Perseus.

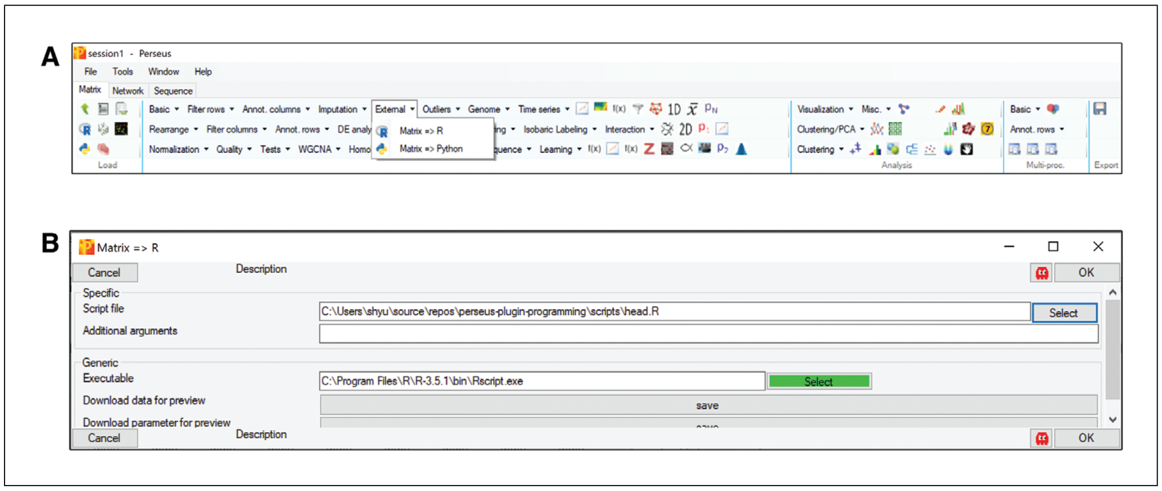
The communication between Perseus and R works through temporary files containing the data, their location being specified by fixed index command line arguments. Therefore, these command line arguments sent from Perseus need to be parsed first.

```
args = commandArgs(trailingOnly=TRUE)
if (length(args) != 2) {
  stop("Do not provide additional arguments!",
  call.=FALSE)
}
inFile <- args[1]
outFile <- args[2]
```

Since the arguments from Perseus are input file and output file, the length of arguments should be 2. The order of the arguments is fixed: the first in this case is for input file and the last one is for the output file.

2. Use PerseusR to read the data matrix written by Perseus.

```
library(PerseusR)
mdata <- read.perseus(inFile)
```



Specific
Script file: C:\Users\shyu\source\repos\perseus-plugin-programming\scripts\head.R
Additional arguments:
Generic
Executable: C:\Program Files\R\R-3.5.1\bin\Rscript.exe
Download data for preview
Download parameter for preview

Figure 5 R plugin in Perseus. (**A**) The labeled menu item is for running an R plugin from an external script. (**B**) Popup window for specifying the path of the script and additional parameters of the plugin.

PerseusR is the package that bridges between Perseus and R. It needs to be imported first. Afterwards, the matrix from Perseus can be read into a special matrixData object by read.perseus.

3. Get the main matrix of Perseus for data processing.

The matrix in Perseus is composed of annotation rows/columns and the main data columns. In order to extract the main matrix for analysis, the function —`main()` *needs to be used.*

```
counts <- main(mdata)
```

4. Execute the main custom code for data analysis or modification.

After obtaining the main matrix, the custom analysis steps and modifications can be done. In this protocol, the head of the matrix is extracted ("15 rows" is assigned).

```
mdata2 <- head(counts, n=15)
```

Since the number of rows is reduced for the main matrix, the annotation columns need to be shortened for the output matrix as well. To get the annotation columns, use `annotCols()`*.*

```
aCols <- head(annotCols(mdata), n=15)
```

5. Export the output matrix to Perseus with correct format.

After finishing all data-processing steps, the data needs to be converted back to the Perseus `txt` *format and written to the predefined temporary output file. For generating the final output, a new matrixData object consisting of main matrix (*`main`*), annotation columns (*`annotCols`*). and annotation rows (*`annotRows`*) needs to be created. Similar to* `annotCols()`*, the function* `annotRows()` *extracts the content of Perseus annotation rows. Since the annotation rows are not changed by the plugin, they are reused from the original matrix.*

```
mdata2 <- matrixData(main=mdata2, annotCols=aCols,
  annotRows=annotRows(mdata))
```

To generate the temporary file, `write.perseus()` *is used with the matrixData object and the* `outputfile` *location, which was read in the first step of this protocol.*

```
print(paste('writing to', outFile))
write.perseus(mdata2, outFile)
```

These are all the basic elements of an R-only Perseus plugin. If no further arguments are required, the developers only have to put their custom code in step 4.

6. Apply the plugin in Perseus.

   a. Open Perseus and import the matrix/load a session file.

      A random matrix is used for testing the plugin in this tutorial.

   b. Execute the plugin.

      In the "Processing" block, click "External" –> "Matrix –> R". If the button "select" is green, it means that Perseus recognized your R installation and PerseusR (Figure 5A), otherwise navigate to your `Rscript.exe` *or add it to your systems PATH variable. Afterwards, specify the R script that you want to execute and click OK (Fig. 5B).*

SUPPORT PROTOCOL 1

R PLUGINS WITH ADDITIONAL ARGUMENTS

In order to make a script more flexible and useful, additional parameters are usually required. With the above example of extracting the head of a matrix (Basic Protocol 1), it will be much more convenient if the number of rows can be defined by the users. The following steps will provide the details of how to add parameters to the plugins. The script, including all steps, can also be found at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/scripts/head_add_argument.R*.

Necessary Resources

Same as Basic Protocol 1

1. Install the argparser library (*https://cran.r-project.org/web/packages/argparser/*) with `install_packages("argparser")` and parse command line arguments from Perseus.

```
argv <- commandArgs(trailingOnly=TRUE)
library("argparser")
p <- arg_parser(description = "Head processing")
p <- add_argument(p, 'input', help="path of the
  input file")
p <- add_argument(p, 'output', help="path of the
  output file")
p <- add_argument(p, '--nrow', type="numeric",
  default=15, help="the number of row")
argp <- parse_args(p, argv)
```

   The first two arguments `"input"` *and* `"output"` *are required arguments for storing the input and output files, respectively. An additional optional argument for the number of rows* `"--nrow"` *is added. Its default value is set to 15. Please refer to the argparser manual for more details.*

2. Use PerseusR to read the data in Perseus.

   In the same was as for the scripts without parameters, PerseusR needs to be imported, and read.perseus is used for converting the matrix from Perseus to R format.

```
library(PerseusR)
mdata <- read.perseus(inFile)
```

3. Get the main matrix of Perseus for the data processing.

   Use `main()` *to obtain the main matrix that is needed for the following data processing.*

```
counts <- main(mdata)
```

4. Execute the main part for data analysis or modification.

   The information from the additional parameter was already parsed and stored. The extraction of the head of the matrix can be performed based on the number of rows that the user assigned.

```
mdata2 <- head(counts, n=num)
aCols <- head(annotCols(mdata), n=num)
```

5. Export the output matrix to Perseus with correct format.

   This step is the same as last section. Generate `matrixData()` and write to Perseus by using `write.perseus()`.

   ```
   mdata2 <- matrixData(main=mdata2, annotCols=aCols,
     annotRows=annotRows(mdata))
   print(paste('writing to', outFile))
   write.perseus(mdata2, outFile)
   ```

6. Apply the plugin in Perseus.

   Now the plugin is ready to be executed. The number of extracted rows can be assigned from "Additional arguments" in the pop-up window (Fig. 5B). For instance, writing "`--nrow 10`" in "Additional arguments," only the first 10 rows will remain in the output matrix in Perseus.

BASIC STEPS FOR PYTHON PLUGINS

BASIC PROTOCOL 2

In recent years, many useful Python packages have been developed for computational biology and data visualization. Moreover, an annual conference (SciPy) provides a platform where up-to-date Python tools are released and presented. Perseuspy builds a bridge to integrate Python libraries into Perseus as plugins (Rudolph & Cox, 2019). In this section, we provide the basic steps for generating Python-only plugins through the command line style interface. The code for this example is available at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/scripts/head.py*.

Necessary Resources

Hardware

A computer running Windows 8 (64 bit) or higher or Windows Server 2008 or higher
4 GB RAM minimum
At least a quad core processor is recommended

Software

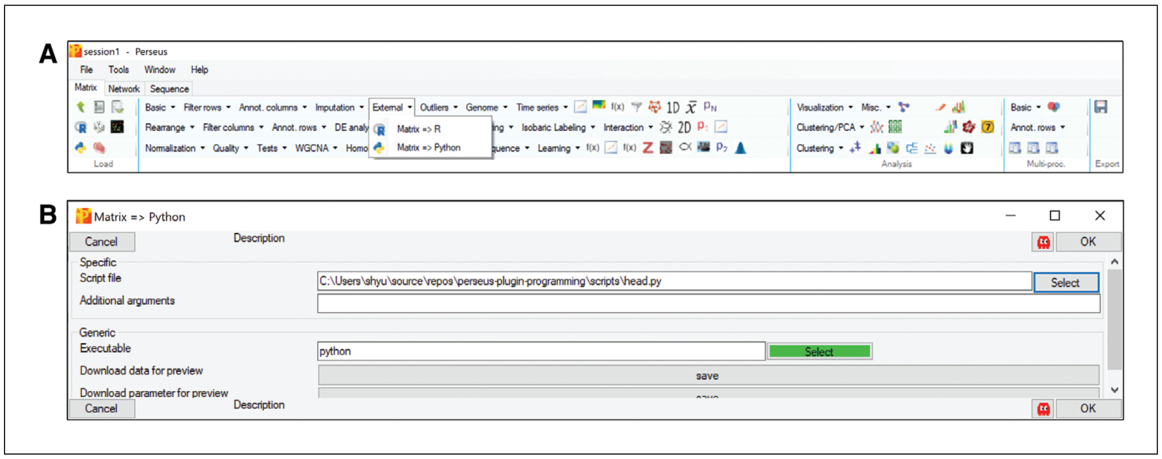
Perseus 1.6.13 or higher: can be downloaded from *https://maxquant.org/perseus*.
Python: use version higher than 3.7.0. The Python executable has to be listed in the PATH environment variable of the operating system. Please refer to Troubleshooting if Perseus cannot find your Python installation, which is indicated in the command line−style interface.
Perseuspy: available at *https://github.com/cox-labs/perseuspy*. An installation guide and required dependencies can be found in the repository. Also available on PyPI (*https://pypi.org/project/perseuspy/*).
Python supported editor like Visual Studio, PyCharm or Notepad++

Input files

This protocol requires no extra input files. The outlined plugin works with a randomly generated matrix, which can be generated using the dice button in Perseus.

1. Import the required packages.

   ```
   import sys
   from perseuspy import pd
   ```



Specific
Script file: C:\Users\shyu\source\repos\perseus-plugin-programming\scripts\head.py
Additional arguments:
Generic
Executable: python
Download data for preview
Download parameter for preview

Figure 6 Python plugin in Perseus. (**A**) The labeled menu item is for running a Python plugin from an external script. (**B**) Popup window for specifying the path of the script and additional parameters of the plugin.

Perseuspy builds on top of the widely-used pandas package to handle matrix processing (Mckinney, 2010). The nested namespace `pd` *contains this extended version of pandas.*

2. Parse command line arguments from Perseus.

   The communication between Perseus and Python works through temporary files containing the data, their location being specified by fixed index command line arguments. Therefore, these command line arguments sent from Perseus need to be parsed first.

   ```
   _, infile, outfile = sys.argv
   ```

   By default, the name of the Python script occupies the first argument of `sys.argv` *and the input and output file are always the last two arguments. Thus, the order of the arguments is: the name of Python script, input file, and output file.*

3. Read the data from Perseus.

   Using the new pandas function `read_perseus` *the data matrix from Perseus can be read from the input file directly into a pandas data frame object.*

   ```
   df = pd.read_perseus(infile)
   ```

4. The main custom code for data analysis or modification.

   This part is for the custom desired data analysis. In this example, the code for extraction of the head of the matrix is placed in this position.

   ```
   df2 = df.head(15)
   ```

   Based on the code, only the first 15 rows will be kept in the matrix.

5. Export the output matrix to Perseus with correct format.

   When all the steps of the data processing are done, the final matrix needs to be exported to Perseus in the correct format. For this, the second new function in pandas, `to_perseus`*, can be used.*

   ```
   df2.to_perseus(outfile)
   ```

6. Apply the plugin in Perseus.

   a. Open Perseus and import the matrix or load a session.

   A random matrix is used for testing the plugin in this tutorial.

   b. Execute the plugin.

   In the "Processing" block, click "External" –> "Matrix –> Python". If the button "select" is green, it means that Perseus recognized your Python installation and perseuspy

(Fig. 6A); otherwise, navigate to your `python.exe` or add it to your systems PATH variable. Afterwards, specify the Python script that you want to execute and click OK (Fig. 6B).

PYTHON PLUGINS WITH ADDITIONAL ARGUMENTS

SUPPORT PROTOCOL 2

For a more elaborate analysis, Python plugins can also be passed additional arguments just like R plugins. The following steps will demonstrate the steps needed for adding parameters to plugins. The example the number of rows to obtain from the top of the matrix can be specified by the user. The script is available at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/scripts/head_add_argument.py*.

Necessary Resources

Same as Basic Protocol 2

Additionally, the package argparse needs to be installed in the Python environment

1. Import the required packages.

```
import argparse
from perseuspy import pd
```

2. Parse command line arguments from Perseus.

```
parser = argparse.ArgumentParser("Head processing")
parser.add_argument("input", help="path of the input
  file")
parser.add_argument("output", help="path of the
  output file")
parser.add_argument("--nrow", type=int, default=15,
  help="the number of row")
arg = parser.parse_args()
```

This part is similar to "Parse command line arguments from Perseus" in Support Protocol 1. The first two arguments `"input"` and `"output"` are positional arguments for storing the input and output files, respectively. The third argument, `"--nrow"` is for the number of rows.

3. Read the data from Perseus.

```
df = pd.read_perseus(arg.input)
```

4. Retrieve the user-define arguments (`--nrow`) to modify the matrix.

The number of rows stored in `"--nrow"`, can now be used for the extraction.

```
df_head = df.head(arg.nrow)
```

5. Export the output matrix to Perseus with correct format.

```
df2.to_perseus(arg.output)
```

6. Apply the plugin in Perseus.

For using the newly added parameter as an input, the assignment needs to be written in "Additional arguments" in the pop-up window (Fig. 6B). If "Additional arguments" is filled in with "`--nrow 10`", the output matrix will be the first 10 rows.

BASIC STEPS AND CONSTRUCTION OF C# PLUGINS

BASIC PROTOCOL 3

Even better integrated plugins with an automatically generated graphical user interface for the parameters can be generated when using C#, which is the original programming language of Perseus. The architecture of Perseus plugins is systematic and well structured. Numerous C# plugins can be found at *https://github.com/JurgenCox/perseus-plugins*. All the scripts can be recycled and modified by users. The same basic example as in the previous sections will be used to explain how to generate a C# plugin. The script of the example can be seen at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Head_c_sharp.cs*.

Necessary Resources

Hardware

A computer running Windows 8 (64 bit) or higher, or Windows Server 2008 or higher
4 GB RAM minimum
At least a quad core processor is recommended

Software

Perseus 1.6.13 or higher: can be downloaded from *https://maxquant.org/perseus*
For editing C# code, Visual Studio Community Edition is recommended (*https://www.visualstudio.com/downloads/*). Please select the ".Net Desktop Development workflow" in the installer to install everything required. To ensure version compatibility, please use .NET Framework <= 4.7.2 or .NET Core <= 2.1.
.NET packages BaseLibS and PerseusAPI: Both of these can be installed by using "Manage NuGet Packages" in Visual Studio, which is explained in step 2 of the protocol

1. Create a C# project.

   Due to the internal connection between the plugin and Perseus, the project name should use "Plugin" as prefix and the type of project should be "Class Library (.NET framework)." In this demonstration, the Project name was set as "PluginTutorial".

2. Add the dependencies.

   PerseusAPI and BaseLibS need to be installed for generating plugins. For installation of these two packages, right-click on the "PluginTutorial" solution and choose "Manage NuGet Packages...." Afterwards, search for PerseusApi in the "Browse" tab and install it for "PluginTutorial". The PerseusAPI and its dependency BaseLibS will be added.

3. Import packages and define namespace with a class.

   In the `Class1.cs` file, remove all default packages and code generated by Visual Studio, and instead import the packages as shown here:

```
using System.Linq;
using BaseLibS.Graph;
using BaseLibS.Param;
using PerseusApi.Document;
using PerseusApi.Generic;
using PerseusApi.Matrix;
namespace PluginTutorial
{
  public class PluginHead : IMatrixProcessing
  {
  }
}
```

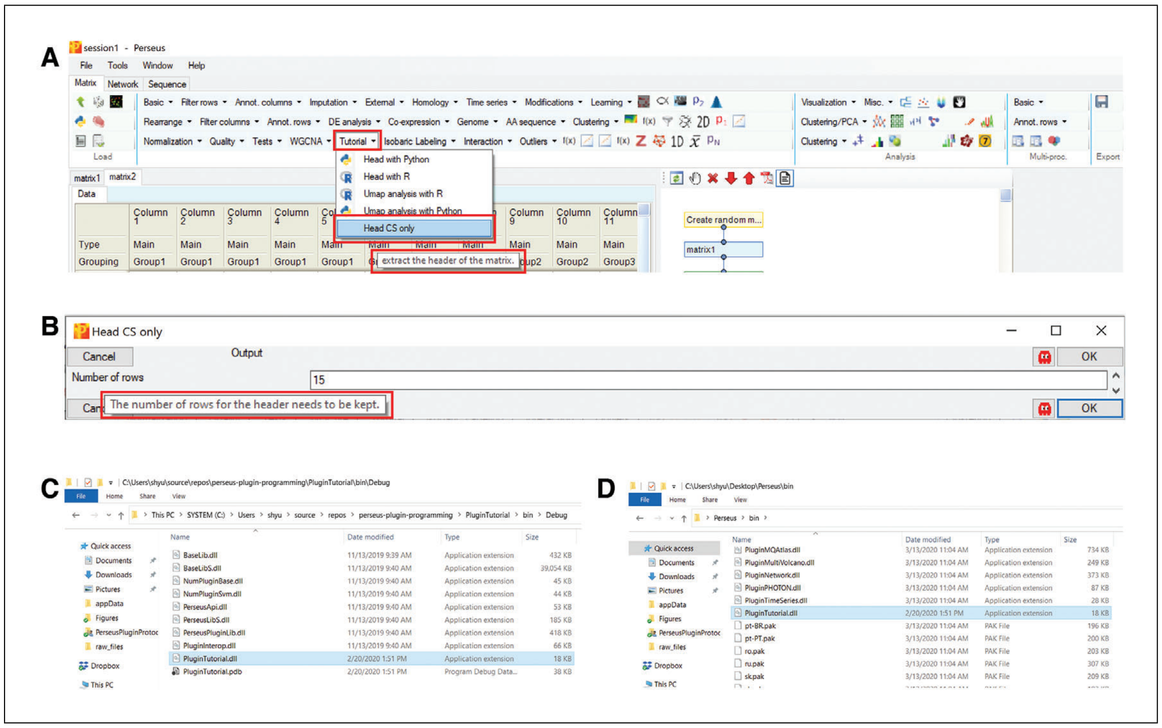
   The project name (`PluginTutorial`) should be placed in `namespace`, and the user-defined class (`PluginHead`) should inherit `IMatrixProcessing`, which is created for processing the matrix from Perseus.

4. Generate basic structure of the C# plugin.

To make `IMatrixProcessing` work, a number of methods and variables need to be implemented.

```
namespace PluginTutorial
{
 public class PluginHead : IMatrixProcessing
  {
  public bool HasButton => false;
  public string Description => "extract the header
  of the matrix.";
  public string HelpOutput => "extract the header of
  the matrix.";
  public string[] HelpSupplTables => new string[0];
  public int NumSupplTables => 0;
  public string Name => "Head CS only";
  public string Heading => "Tutorial";
  public float DisplayRank => 6;
  public string[] HelpDocuments => new string[0];
  public int NumDocuments => 0;
  public string Url => null;
  public Bitmap2 DisplayImage => null;
  public bool IsActive => true;
  public int GetMaxThreads(Parameters parameters)
  {
    return 1;
  }
  public void ProcessData(IMatrixData mdata,
  Parameters param, ref
  IMatrixData[] supplTables,
  ref IDocumentData[] documents, ProcessInfo
  processInfo)
  {
  }
  public Parameters GetParameters(IMatrixData mdata,
  ref string errorString)
  {
 }
 }
}
```

To customize the plugin, the modification of several parameters and methods is required. `Name` defines the name of the plugin as it will appear in Perseus (Fig. 7A). `Heading` defines the name of the drop-down menu the plugin will be added to (Fig. 7A). If the name of the menu does not yet exist, a new one will be created automatically without requiring further changes in other places. Optional parameter changes are the following: the `Description` of the plugin, which will be shown when hovering over the plugin in Perseus (Fig. 7A); the `Url` of the plugin's documentation, which can be opened by clicking the ghost icon besides the "OK" button (Fig. 7B); and `DisplayImage`, which is the icon appearing next to the plugin name in the menu. If more threads need to be used for the plugin, it can be changed in `GetMaxThread`. The code for actual data processing should be placed inside `ProcessData`. The additional input parameters required are defined in `Parameters`. The other parameters defined in the code above are not relevant for custom-generated plugins, but need to be defined for the class to be compiled successfully.



extract the header of the matrix.

Head CS only

Number of rows: 15

The number of rows for the header needs to be kept.

C:\Users\shyu\source\repos\perseus-plugin-programming\PluginTutorial\bin\Debug

Name	Date modified	Type	Size
BaseLib.dll	11/13/2019 9:39 AM	Application extension	432 KB
BaseLibS.dll	11/13/2019 9:40 AM	Application extension	39,054 KB
NumPluginBase.dll	11/13/2019 9:40 AM	Application extension	45 KB
NumPluginSvm.dll	11/13/2019 9:40 AM	Application extension	44 KB
PerseusApi.dll	11/13/2019 9:40 AM	Application extension	53 KB
PerseusLibS.dll	11/13/2019 9:40 AM	Application extension	185 KB
PerseusPluginLib.dll	11/13/2019 9:40 AM	Application extension	418 KB
PluginInterop.dll	11/13/2019 9:40 AM	Application extension	66 KB
PluginTutorial.dll	2/20/2020 1:51 PM	Application extension	18 KB
PluginTutorial.pdb	2/20/2020 1:51 PM	Program Debug Data...	38 KB

C:\Users\shyu\Desktop\Perseus\bin

Name	Date modified	Type	Size
PluginMQAtlas.dll	3/13/2020 11:04 AM	Application extension	734 KB
PluginMultiVolcano.dll	3/13/2020 11:04 AM	Application extension	249 KB
PluginNetwork.dll	3/13/2020 11:04 AM	Application extension	373 KB
PluginPHOTON.dll	3/13/2020 11:04 AM	Application extension	87 KB
PluginTimeSeries.dll	3/13/2020 11:04 AM	Application extension	28 KB
PluginTutorial.dll	2/20/2020 1:51 PM	Application extension	18 KB
pt-BR.pak	3/13/2020 11:04 AM	PAK File	196 KB
pt-PT.pak	3/13/2020 11:04 AM	PAK File	200 KB
ro.pak	3/13/2020 11:04 AM	PAK File	203 KB
ru.pak	3/13/2020 11:04 AM	PAK File	307 KB
sk.pak	3/13/2020 11:04 AM	PAK File	209 KB

Figure 7 C# plugin in Perseus. (**A**) The labeled menu item is for running a C# plugin. The description specified in the C# code is shown when the mouse hovers over it. (**B**) The popup window for specifying the values of parameters. The explanation specified in the script is displayed when hovering with the mouse. For adding plugins in Perseus, the selected files in (**C**) (the folder of $PROJECT_NAME/bin/Debug) need to be copied to the folder in (**D**) (Perseus/bin).

5. Add parameters.

The method `GetParameters` returns a C# object (`Parameters`) containing all the parameters that the user has to provide in the interface.

```
public Parameters GetParameters(IMatrixData mdata,
  ref string errorString)
{
  return new Parameters(new IntParam("Number of
  rows", 15)
  {
    Help = "The number of rows for the header needs
      to be kept."
  });
}
```

In this example, an integer parameter was created (`IntParam`). The name of the parameter is "Number of rows," and the default is 15. A description of the parameter can be added in `Help`. This description will appear when hovering over the parameter text (Fig. 7B). If multiple parameters need to be added, the user just needs to add them one by one in `Parameters`, as shown below. A complete example can be seen at https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Head_c_sharp_two_params.cs.

```
public Parameters GetParameters(IMatrixData mdata,
  ref string errorString)
{
  return new Parameters(new IntParam("Number of
      rows", 15)
       {Help = "The number of rows to retain."},
            new IntParam("Number of columns", 2)
            {Help = "The number of columns to retain."}
        );
    }
```

6. Generate the code for data processing.

   The main data processing is defined in `ProcessData`*. In this example, the first n rows of the matrix are extracted.*

```
public void ProcessData(IMatrixData mdata,
  Parameters param, ref IMatrixData[]
  supplTables, ref IDocumentData[] documents,
  ProcessInfo processInfo)
{
  int lines = param.GetParam<int>("Number of
     rows").Value;
  int[] head = Enumerable.Range(0, lines).ToArray();
  mdata.ExtractRows(head);
}
```

   `param.GetParam<int>("Number of rows").Value` *will get the value stored in the integer parameter called* `"Number of rows"`*, which was created in* `GetParameters` *before. The other lines use this number to extract the head of the matrix.* `mdata` *is the object that stores all the information on the Perseus matrix and* `ExtractRows` *can be used to extract the rows of the matrix based on the list of indices generated from the user input.*

7. Compile the plugin and copy the dll.

   Since C# is a compiled language, the next step is to build the project, which will generate a `.dll` *file. If the build was successful,* `PluginTutorial.dll` *will be saved in the "*`bin/Debug`*" folder of the project directory, which you can open by right-clicking the project in the Solution Explorer –> "Open Folder in File Explorer" (*`PluginTutorial\bin\Debug`*, Fig. 7C). For adding the newly created plugin to Perseus, this dll must be copied to the "*`bin`*" folder of Perseus (*`Perseus/bin`*, Fig. 7D). Afterwards, the plugin can be used in Perseus after a re-start (Fig. 7A and 7B).*

8. Apply the plugin in Perseus.

   a. Open Perseus and import the matrix or open a session.

      A random matrix is used for testing the plugin in this tutorial.

   b. Execute the plugin.

      Click "Tutorial" –> "Head CS only" in "Processing" block (Fig. 7A). Then, specify the number of rows for extraction and click OK (Fig. 7B).

Resource for C# Plugins

For more examples and source codes of C# plugins, please check the repository: *https://github.com/JurgenCox/perseus-plugins*.

BASIC STEPS OF CONSTRUCTION AND CONNECTION FOR R PLUGINS WITH C# INTERFACE

BASIC PROTOCOL 4

Although C# can generate a user-friendly interface for the plugins, R and Python packages are still not able to be integrated into Perseus with the native C# interface. To combine the flexibility of R and Python with the graphical user interface generated by C#, the C# package PluginInterop was created (Rudolph & Cox, 2019). Here, the basic methods of PluginInterop needed to create an R plugin with C# interface will be presented step by step. The R script can be found at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Resources/head_c_sharpR.R*, and the C# script at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Head_with_r.cs*.

Necessary Resources

All requirements of Basic Protocols 1 and 3

Additionally, the C# package PluginInterop is required: this can also be installed by using "Manage NuGet Packages" in a Visual Studio project as described in step 2.

1. Create a C# project.

   See step 1 of Basic Protocol 3.

2. Add the dependencies.

   Besides PerseusAPI and BaseLibS, PluginInterop also needs to be installed. The steps are the same as step 2 of Basic Protocol 3. Right-click on you project solution (PluginTutorial) and choose "Manage NuGet Packages....", then search for PluginInterop in the "Browse" tab and install it to your project solution.

3. Import packages and define namespace with a class.

   Since `PluginInterop` is the bridge between R and C#, it needs to be used in the wrapper. Additionally, the R script needs to be placed into the resource folder of the project (will be descripted at step 6). Thus, `PluginTutorial.Properties` needs to be used as well.

```
using BaseLibS.Param;
using PerseusApi.Matrix;
using System.IO;
using PluginInterop;
using System.Text;
using PluginTutorial.Properties;
namespace PluginTutorial
{
  public class HeadR : PluginInterop.R.
  MatrixProcessing
  {
  }
}
```

   Here, the class `HeadR` inherits from `PluginInterop.R.MatrixProcessing`, which is the method for managing the matrix and parameter transfer between C# and the R script.

4. Override methods in the class.

   `PluginInterop.R.MatrixProcessing` is a template for generating a C# plugin that can connect to R. For applying it to a user-specific project, some methods and variables need to be overridden.

```
public class HeadR : PluginInterop.R.
  MatrixProcessing
{
  public override string Heading => "Tutorial";
  public override string Name => "Head with R";
  public override string Description => "extract the
  header of the matrix";
```

```
    protected override bool TryGetCodeFile(Parameters
    param, out string
    codeFile)
    {
       byte[] code = (byte[])Resources.
         ResourceManager.GetObject("head_c_sharpR");
       codeFile = Path.GetTempFileName();
       File.WriteAllText(codeFile, Encoding.UTF8.
         GetString(code));
       return true;
    }
    protected override string GetCommandLineArguments
      (Parameters param)
    {
       var tempFile = Path.GetTempFileName();
       param.ToFile(tempFile);
       return tempFile;
    }
    protected override Parameter[] SpecificParameters
      (IMatrixData mdata, ref string errString)
    {
       if (mdata.ColumnCount < 3)
       {
       errString = "Please add at least 3 main columns
         to the matrix.";
       return null;
      }
      return new Parameter[]
      {
       new IntParam("Number of rows", 15)
       {
       Help = "The number of rows for the header needs
         to be kept."
       }
      };
    }
  }
```

`Heading`, `Name` and `Description` should be changed to match the needs of the project (the details of these methods were mentioned at step 4 of Basic Protocol 3). `TryGetCodeFile()` is for getting the R script from the project resources. The only thing that needs to be edited here is the name of the R script in `(byte[])Resources.ResourceManager.GetObject()`, omitting the file extension (`head_c_sharpR`). `GetCommandLineArguments()` will convert the parameters specified at `SpecificParameters()` and filled in by the user to a temporary file which can be read inside the R script. This method definition does not require further editing. `SpecificParameters()` is similar to `GetParameters()`, which was introduced at step 5 of Basic Protocol 3. It needs to return an array of parameter definitions, which will be used to generate the interface. For definition of several parameters please refer to Support Protocol 4, which shows a more advanced example.

5. Generate R script for data processing.

   At this point, the C# side of the interface has been established. Next, the R code for the actual data processing is added. The code is very similar to the script presented in Support Protocol 1. The only difference is that the additional parameters are transferred as one file from the C# interface, rather than as individual arguments. The order of the command line arguments is now parameter file, input file and output file.

```
args = commandArgs(trailingOnly = TRUE)
paramFile <- args[1]
inFile <- args[2]
outFile <- args[3]
```

Afterwards, the user input has to be extracted by a function called "`parseParameters`*" and several type-specific functions like* `intParamValue()`*, or* `boolParamValue()`*. Table 1 summarizes the most commonly used functions and their C# counterparts. In this demonstration, the type of the parameter is integer and the name is "Number of rows."*

```
parameters <- parseParameters(paramFile)
num_row <- intParamValue(parameters, 'Number of
  rows')
```

The remainder of the R script is the same as in Support Protocol 1.

6. Store R script to resource folder.

   As mentioned in steps 2 and 3, the R script needs to be placed in the resource folder of the C# project. To do so follow these steps:

   a. Right-click "Properties" under the project (PluginTutorial) in Visual Studio.
   b. Click "Resources."
   c. Click "Add Resource" and navigate to the target R script (`head_c_sharpR.R`). Then save it.

7. Build the solution and place the required files to the `bin` folder of Perseus.

   Just as with a C#-only plugin, the code needs to be compiled and moved to the Perseus `bin` *folder. Please refer to step 7 of Basic Protocol 3 for this. The only difference is that the* `.pdb` *file also has to be transferred, as it contains the R script. Thus,* `PluginTutorial.dll` *and* `PluginTutorial.pdb` *need to be copied. The plugins can be used after re-starting Perseus (Fig. 1).*

SUPPORT PROTOCOL 3

ADVANCED EXAMPLE OF R PLUGIN WITH C# INTERFACE: UMAP

UMAP (Becht et al., 2019; McInnes, Healy, & Melville, 2018) is a powerful dimensionality-reduction algorithm that is widely used for many different studies. It will be extremely useful to add UMAP to Perseus. This section will take UMAP as an advanced example to demonstrate how powerful the new Perseus plugin interface is for data analysis. The C# script can be found at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/UmapAnalysis_with_r.cs*, and the R script is saved at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Resources/Umap_R.R*.

Necessary Resources

All of the resources listed in Basic Protocol 4

UMAP: a dimensionality-reduction method. The R version of UMAP can be found at CRAN (*https://cran.r-project.org/web/packages/umap/index.html*)

Input files

The samples for the example of UMAP analysis can be downloaded at PRIDE (PXD003710) (Bailey, McDevitt, Westphall, Pagliarini, & Coon, 2014). Additionally, the MaxQuant (Cox & Mann, 2008; Sinitcyn, Rudolph, & Cox, 2018; Tyanova, Temu, & Cox, 2016; Yu, Kiriakidou, & Cox, 2020) proteinGroup table of this dataset is also provided at *https://github.com/JurgenCox/perseus-plugin-programming/tree/master/dataset*. The values are normalized and transformed by logarithm, and the unreliable protein groups (reversed, only identified by site, contaminant, containing more than 30%

missing values) are all removed from the table. Moreover, the data is well annotated by experimental design. This table can be directly used for the advanced example.

1. Create a C# project.

   Please see step 1 of Basic Protocol 3.

2. Add the dependencies.

   Please see step 2 of Basic Protocol 4.

3. Import packages and define namespace with a class.

   Please see step 3 of Basic Protocol 4.

4. Override methods in the class.

   In general, the practical procedures are the same as in step 4 of Basic Protocol 4. The only requirement is to modify several methods and variables to match this example. First, the Head, Name, and Description need to be changed.

```
public override string Heading => "Tutorial";
public override string Name => "Umap analysis with
  R";
public override string Description => "Applying
  Umap to cluster the data";
```

   Secondly, the R script of UMAP analysis has to be assigned properly.

```
byte[] code = (byte[])Resources.ResourceManager.
  GetObject("Umap_R");
```

   Third, all the parameters should be added to `SpecificParameters()` *one by one with their corresponding data type.*

```
protected override Parameter[] SpecificParameters
  (IMatrixData mdata, ref string errString)
{
  if (mdata.ColumnCount < 3)
  {
    errString = "Please add at least 3 main data
      columns to the matrix.";
    return null;
    }
  return new Parameter[]
  {
    new IntParam("Number of neighbors", 15)
    {
          Help = "The number of neighbors."
    },
    new IntParam("Number of components", 2)
    {
          Help = "The number of components."
    },
    new IntParam("Random state", 1)
    {
          Help = "Set seed for reproducibility."
    },
    new DoubleParam("Minimum distance", 0.1)
    {
```

```
                Help = "Set minimum distance between the
                  data point."
        },
        new SingleChoiceParam("Metric")
        {
                Values= new[] { "euclidean", "manhattan",
                  "cosine", "pearson","pearson2"},
                Help = "The method of metric for doing
                  clustering."
        }
      };
    }
```

The structure of `SingleChoiceParam()` *is different from* `IntParam()` *or* `DoubleParam()`*.* `SingleChoiceParam()` *has to contain an array called* `Values` *to store all the options for the users. The first element of* `Values` *is the default one, which will appear on the dropdown list of the Perseus plugin.*

5. Generate R script for data processing.

   The basic rules for generating an R script are the same as in step 5 of Basic Protocol 4. Hence, the most important thing is to obtain the information of parameters from C#. The other parts of the R script can be done like normal R programming and are not shown here.

```
args = commandArgs(trailingOnly = TRUE)
paramFile <- args[1]
inFile <- args[2]
outFile <- args[3]
parameters <- parseParameters(paramFile)
n_neighbor <- intParamValue(parameters, "Number of
  neighbors")
n_component <- intParamValue(parameters, "Number of
  components")
seed <- intParamValue(parameters, "Random state")
metric <- singleChoiceParamValue(parameters,
  "Metric")
m_dist <- intParamValue(parameters, "Minimum
  distance")
```

   `intParamValue()` *can also be used to receive the value with the double-precision data type because R can handle the conversion at the first assignment of the variables.*

6. Store R script to resource folder.

   Please see step 6 of Basic Protocol 4.

7. Build the solution and place the required files to the `bin` folder of Perseus

   Please see step 7 of Basic Protocol 4.

8. Run UMAP and plot the result.

   In this section, the dataset from Bailey, D. J., et al. will be applied to test the newly developed plugin of UMAP (Bailey et al., 2014).

   a. Open Perseus and load `proteinGroup.txt`.

   Since `proteinGroup.txt` *is already pre-processed and grouped, it can be directly loaded into Perseus by clicking the green icon of arrow in the block of "Load."*

   b. Run R plugin of UMAP.

Click "Tutorial" −> "Umap analysis with R" to specify the parameters and run the plugin (Fig. 2A and 2B). After running the plugin of UMAP, the matrix will be transposed and the main values will be changed to components.

c. Plot the result of UMAP.

Use scatter plot (with columns) to view the result of the UMAP analysis. The outcome shows that the data points are clustered based on cell types (Fig. 2C).

BASIC STEPS OF CONSTRUCTION AND CONNECTION FOR PYTHON PLUGINS WITH C# INTERFACE

BASIC PROTOCOL 5

This protocol will continue to demonstrate how to generate Python plugins with C# interface using the same examples as Basic Protocol 4. The C# script of the basic example can be found at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Head_with_py.cs*, and the Python code can be found at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Resources/head_c_sharpPy.py*.

Necessary Resources

All requirements of Basic Protocols 2 and 3
Additionally, the C# package PluginInterop is required: this can also be installed by using "Manage NuGet Packages" in a Visual Studio project as described in step 2

1. Create a C# project.

   Please see step 1 of Basic Protocol 3.

2. Add the dependencies.

   Please see step 2 of Basic Protocol 4.

3. Import packages and define namespace with a class.

   The only difference between this step and step 3 of Basic Protocol 4 is that the inheritance has to be edited for connecting Python instead of R to C#.

```
public class HeadPy : PluginInterop.Python.
  MatrixProcessing
```

   Like `PluginInterop.R.MatrixProcessing`, `PluginInterop.Python.MatrixProcessing` is for managing the matrix of Perseus and the communication between Python and C#. Of course, the name of the class can be changed as well (from `HeadR` to `HeadPy`).

4. Override methods in the class.

   The majority of the code is the same as step 4 of Basic Protocol 4. Only several commands need to be modified to match the needs of integrating the Python script.

```
public override string Heading => "Tutorial";
public override string Name => "Head with Python";
public override string Description => "extract the
  header of the matrix";
protected override bool TryGetCodeFile(Parameters
  param, out string codeFile)
{
  byte[] code = (byte[])Resources.ResourceManager.
   GetObject(
                "head_c_sharpPy");
  codeFile = Path.GetTempFileName();
```

```
    File.WriteAllText(codeFile, Encoding.UTF8.
      GetString(code));
    return true;
  }
```

Based on the above code, `Heading`*,* `Name` *and* `Description` *were changed from R to Python. Additionally, the name of Python script was assigned to* `(byte[]) Resources.ResourceManager.GetObject()`*.*

5. Generate Python script of data processing.

   For the basic principles of generating Python plugins, please refer to Basic Protocol 2. In order to gain the information about the parameters assigned through the C# interface, a variable needs to be created for receiving the file storing the parameter values.

```
_, paramfile, infile, outfile = sys.argv
```

   `parse_parameters()` *is a function for parsing the parameters from the parameter file. Based on different data types, all the values of parameters can be used for the Python script by applying corresponding functions like* `intParam()`*. Table 1 summarizes the most commonly used functions and their C# counterparts.*

```
parameters = parse_parameters(paramfile)
head = intParam(parameters, "Number of rows")
```

6. Store Python script to resource folder.

   Please see step 6 of Basic Protocol 4.

7. Build the solution and place the required files to the bin folder of Perseus

   Please see step 7 of Basic Protocol 4.

   If all the procedures are done, the plugin will be shown in Perseus (Fig. 3)

SUPPORT PROTOCOL 4

ADVANCED EXAMPLE OF PYTHON PLUGIN WITH C# INTERFACE: UMAP

Since UMAP is also available in Python, the same analysis can be used as an advanced example to show the power of Perseus Plugins for data analysis. The C# and Python scripts are listed at *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/UmapAnalysis_with_py.cs*, and *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/PluginTutorial/Resources/Umap_Py.py*, respectively.

Necessary Resources

All of the requirements listed in Basic Protocol 5.
UMAP: a dimensionality-reduction method. The Python version of UMAP can be found at PyPI (*https://pypi.org/project/umap-learn/*).

Input files

Same as Support Protocol 4

1. Create a C# project.

   Please see step 1 of Basic Protocol 3.

2. Add the dependencies.

   Please see step 2 of Basic Protocol 4.

3. Import packages and define namespace with a class.

   Please see step 3 of Basic Protocol 5.

4. Override methods in the class.

   The most important thing is to specify the correct names of plugin and Python script.

```
public override string Heading => "Tutorial";
public override string Name => "Umap analysis with
  Python";
public override string Description => "Applying
  Umap to cluster the data";
protected override bool TryGetCodeFile(Parameters
  param, out string codeFile)
{
  byte[] code = (byte[])Resources.ResourceManager.
  GetObject("Umap_Py");
  codeFile = Path.GetTempFileName();
  File.WriteAllText(codeFile, Encoding.UTF8.
  GetString(code));
  return true;
}
```

5. Generate Python script for data processing.

   Since this part of data processing will be done in Python script, all the parameters need to be transferred to Python variables.

```
n_neighbor = intParam(parameters, "Number of
  neighbors")
n_component = intParam(parameters, "Number of
  components")
seed = intParam(parameters, "Random state")
m_dist = doubleParam(parameters, "Minimum distance")
metric = singleChoiceParam(parameters, "Metric")
```

   As in step 5 of Support Protocol 4, `intParam()` *can be used for handling the double-precision data type because Python can automatically adjust for the conversion. Additionally, the Perseus matrix has to be transposed due to the input format of UMAP. Therefore, the category rows need to be extracted for the modification as well. Similar to* `annotRows` *in PerseusR,* `read_annotation` *can return a matrix containing all category rows.*

```
annotations = read_annotations(infile)
```

   Furthermore, Perseuspy has a function called `main_df()`*, which is similar to* `main()` *in PerseusR, for extracting the main matrix in Perseus. Based on this, only the values of the main matrix will be used for the UMAP analysis, which is not shown here.*

```
newDF1 = main_df(infile, df)
```

6. Store Python script to resource folder.

   Please see step 6 of Basic Protocol 4.

7. Build the solution and place the required files in the `bin` folder of Perseus

   Please see step 6 of Basic Protocol 4.

8. Run UMAP and plot the results.

   Please see step 8 of Support Protocol 4. Figure 4 shows the outcome of the plugin and the results of UMAP with the data points grouped based on cell types.

SUPPORT PROTOCOL 5

A BASIC WORKFLOW FOR THE ANALYSIS OF LABEL-FREE QUANTIFICATION PROTEOMICS DATA USING PERSEUS

Based on the above protocols, Perseus plugins can be generated according to the user's needs. In order to demonstrate the benefits that Perseus can offer for data analysis, a basic workflow for the analysis of label-free quantification (LFQ) will be presented in this section. The UMAP plugin generated via Support Protocols 4 and 5 can be applied to this analysis. The samples are from a part of the dataset in a Proteome Informatics Research Group (iPRG) 2015 Study (Choi et al., 2017). The proteinGroup table used for this example can be downloaded from *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/dataset/proteinGroups_LFQ.txt*.

Necessary Resources

All of the requirements listed in Basic Protocol 5 and Support Protocol 5

Input files

`proteinGroups_LFQ.txt` from *https://github.com/JurgenCox/perseus-plugin-programming/blob/master/dataset/proteinGroups_LFQ.txt*. This dataset contains three samples named as 1, 2, and 3. Moreover, each sample has three technical replicates labeled as A, B, and C.

The workflow, plugins, and settings for the basic analysis are shown in Figure 8. The results of most commonly used statistics methods–differential expression analysis (ANOVA test is used) and dimensionality reduction (UMAP is applied) are presented in Figure 9 and 10. This example only demonstrates a basic workflow. Perseus contains numerous useful plugins and parameters. The user can change the settings based on different requirements.

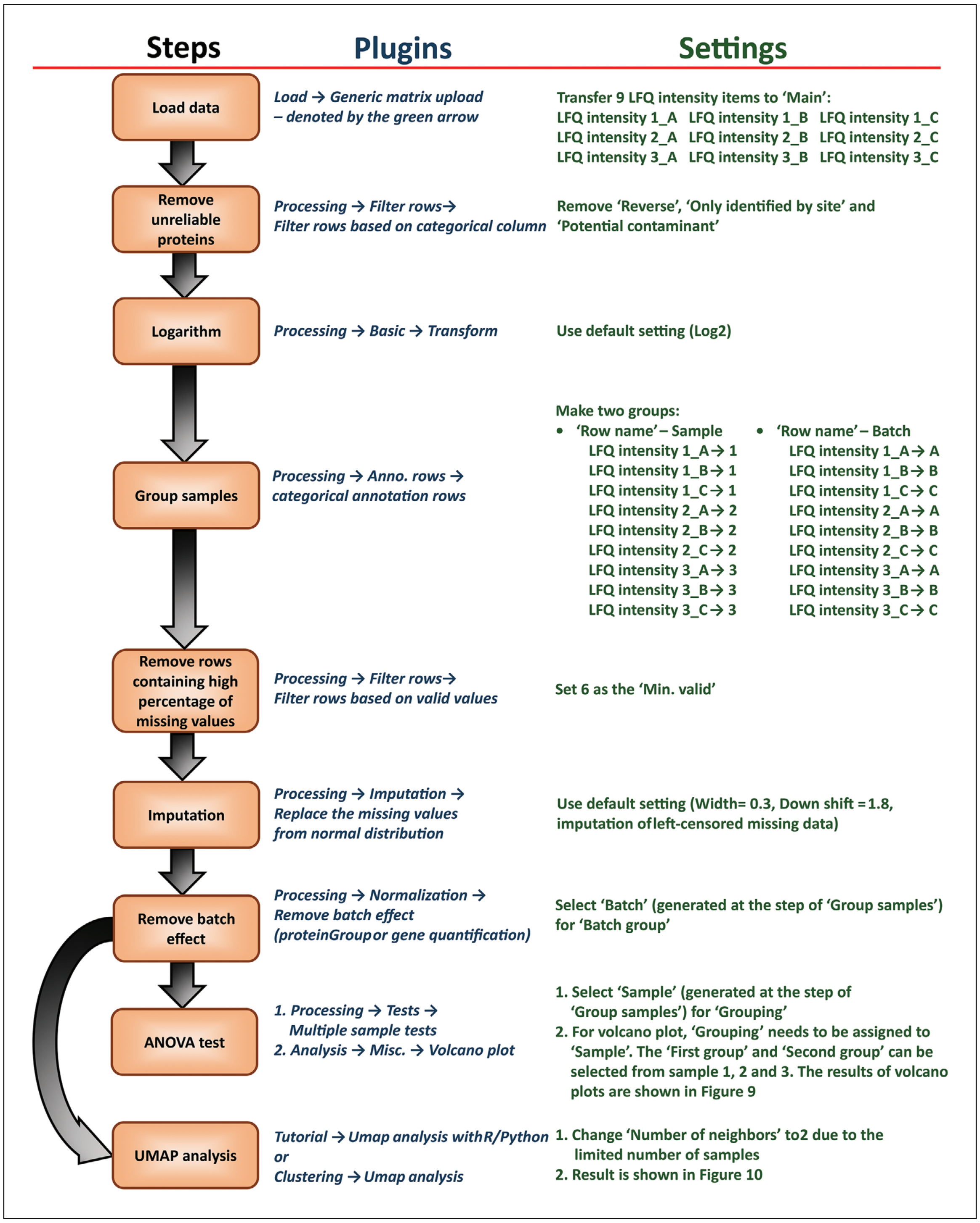
GUIDELINES FOR UNDERSTANDING RESULTS

During the development of new Perseus plugins, execution errors may occur. In this case. a window, "Execution halted," will pop up and show the trace-back of the error. If this happens, refer to the troubleshooting information in Table 2 about common mistakes and how to avoid them. If the error stems from the external plug-in, it is recommended to use the two download options provided in the command line−style interfaces. The first one allows you to download a data preview, which is just the regular temporary file written for the data transfer from Perseus to the plugin. The second download option is for the parameters. This allows the developer to generate test data and parameters for debugging the plugin outside of Perseus, without writing the same temporary files multiple times. If the plugin executes smoothly without any errors, it is still recommended to validate correctness of the results, for instance by writing unit tests. For this, the developer should prepare a minimal test data set that allows validation of the computation results.

Table 2 Summary of the Most Widely Used Parameter Options

C# function parameter specification	R function for parameter value retrieval	Python function for parameter value retrieval
IntParam	intParamValue	intParam
DoubleParam	intParamValue	doubleParam
BoolParam	boolParamValue	boolParam
SingleChoiceParam	singleChoiceParamValue	singleChoiceParam
SingleChoiceWithSubParams	singleChoiceParamValue[a]	singleChoiceWithSubParams
BoolWithSubParams	boolParamValue[a]	boolWithSubParams

[a]For these functions to return the correct value, all sub-parameters need to have unique names across the whole plugin.



Steps	Plugins	Settings
Load data	*Load → Generic matrix upload – denoted by the green arrow*	Transfer 9 LFQ intensity items to 'Main': LFQ intensity 1_A LFQ intensity 1_B LFQ intensity 1_C LFQ intensity 2_A LFQ intensity 2_B LFQ intensity 2_C LFQ intensity 3_A LFQ intensity 3_B LFQ intensity 3_C
Remove unreliable proteins	*Processing → Filter rows→ Filter rows based on categorical column*	Remove 'Reverse', 'Only identified by site' and 'Potential contaminant'
Logarithm	*Processing → Basic → Transform*	Use default setting (Log2)
Group samples	*Processing → Anno. rows → categorical annotation rows*	Make two groups: • 'Row name' – Sample: LFQ intensity 1_A→ 1, LFQ intensity 1_B→ 1, LFQ intensity 1_C→ 1, LFQ intensity 2_A→ 2, LFQ intensity 2_B→ 2, LFQ intensity 2_C→ 2, LFQ intensity 3_A→ 3, LFQ intensity 3_B→ 3, LFQ intensity 3_C→ 3 • 'Row name' – Batch: LFQ intensity 1_A→ A, LFQ intensity 1_B→ B, LFQ intensity 1_C→ C, LFQ intensity 2_A→ A, LFQ intensity 2_B→ B, LFQ intensity 2_C→ C, LFQ intensity 3_A→ A, LFQ intensity 3_B→ B, LFQ intensity 3_C→ C
Remove rows containing high percentage of missing values	*Processing → Filter rows→ Filter rows based on valid values*	Set 6 as the 'Min. valid'
Imputation	*Processing → Imputation → Replace the missing values from normal distribution*	Use default setting (Width= 0.3, Down shift =1.8, imputation of left-censored missing data)
Remove batch effect	*Processing → Normalization → Remove batch effect (proteinGroup or gene quantification)*	Select 'Batch' (generated at the step of 'Group samples') for 'Batch group'
ANOVA test	*1. Processing → Tests → Multiple sample tests* *2. Analysis → Misc. → Volcano plot*	1. Select 'Sample' (generated at the step of 'Group samples') for 'Grouping' 2. For volcano plot, 'Grouping' needs to be assigned to 'Sample'. The 'First group' and 'Second group' can be selected from sample 1, 2 and 3. The results of volcano plots are shown in Figure 9
UMAP analysis	*Tutorial → Umap analysis with R/Python or Clustering → Umap analysis*	1. Change 'Number of neighbors' to 2 due to the limited number of samples 2. Result is shown in Figure 10

Figure 8 Workflow and settings of a basic analysis. The rectangle blocks represent the steps for the analysis. The blue paths are the plugins used for the steps. The green statements are the settings that need to be changed and results of the plugins; the rest of the parameters can remain as the default.

Additionally, the returned data types of all columns should be assessed, to seamlessly integrate the resulting matrix into the overall workflow. If everything is correct, a second test data set that challenges the plugin with common error sources like missing values or false column types should be generated. Once the plugin is fully functional, proper documentation of the plug-in's dependencies and parameters will ensure that it can be successfully applied by Perseus users or other developers. Many useful tools have already been integrated into Perseus and made available to the community (Table 1). With this, we hope to enable many developers to add custom functionality to Perseus; we also

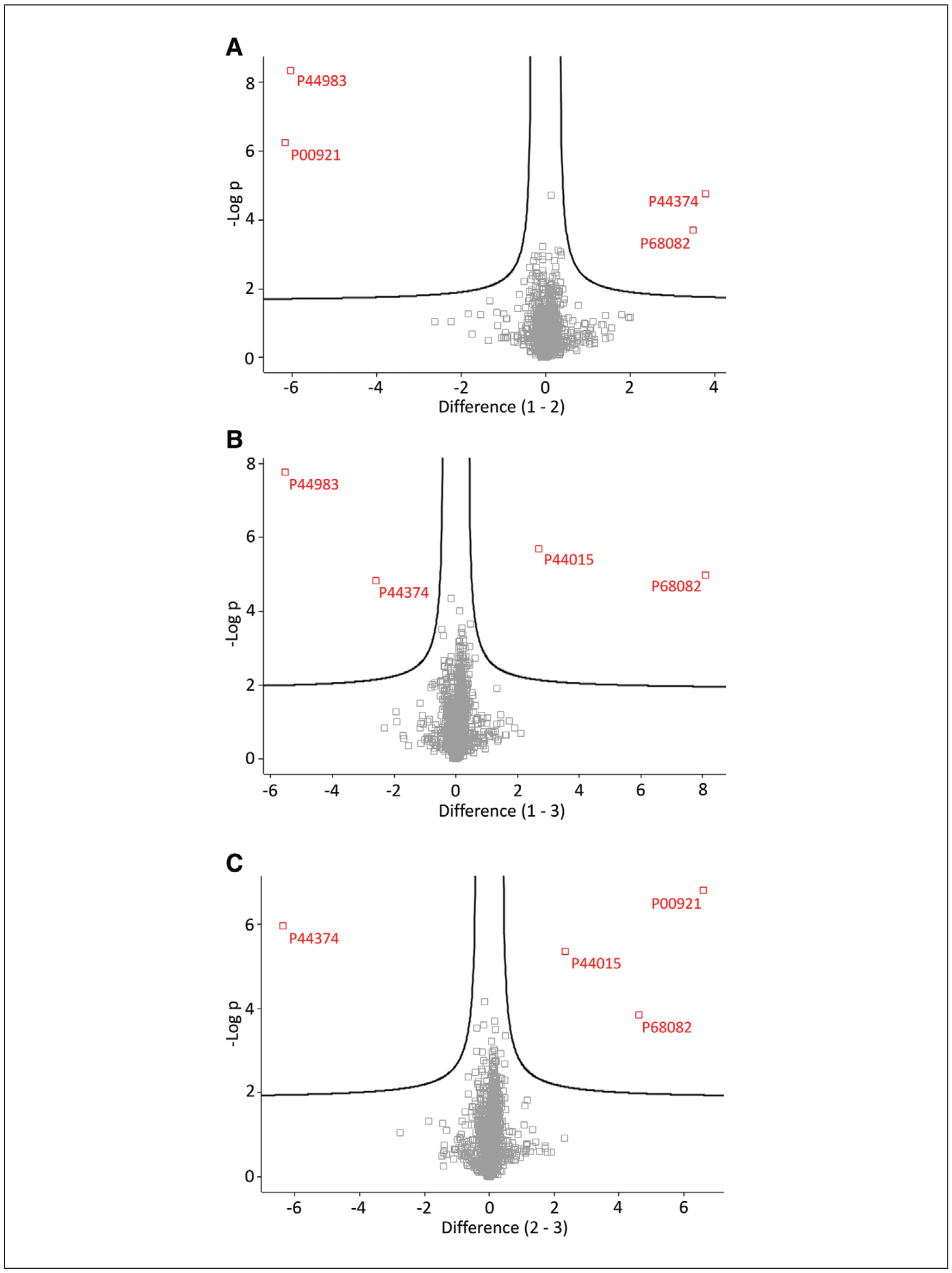


Figure 9 Volcano plots of the example LFQ dataset. (**A, B,** and **C**) show the volcano plots of sample 1 versus 2, sample 1 versus 3, and sample 2 versus 3, respectively. The red squares represent the differential expressed proteins.

hope that users will soon have an even larger collection of plugins available to use in their research.

COMMENTARY

Background Information

Perseus was originally developed together with MaxQuant (Cox & Mann, 2008; Tyanova et al., 2016) for quantitative proteomics analysis. MaxQuant is one of the most commonly used software applications for mass-spectrometry-based proteomics data analysis. It can support numerous types of labeling strategies and MS platforms (Cox et al., 2014; Tyanova, Mann, & Cox, 2014; Yu et al., 2020). Moreover, different quantification methods, false-discovery rate control, and

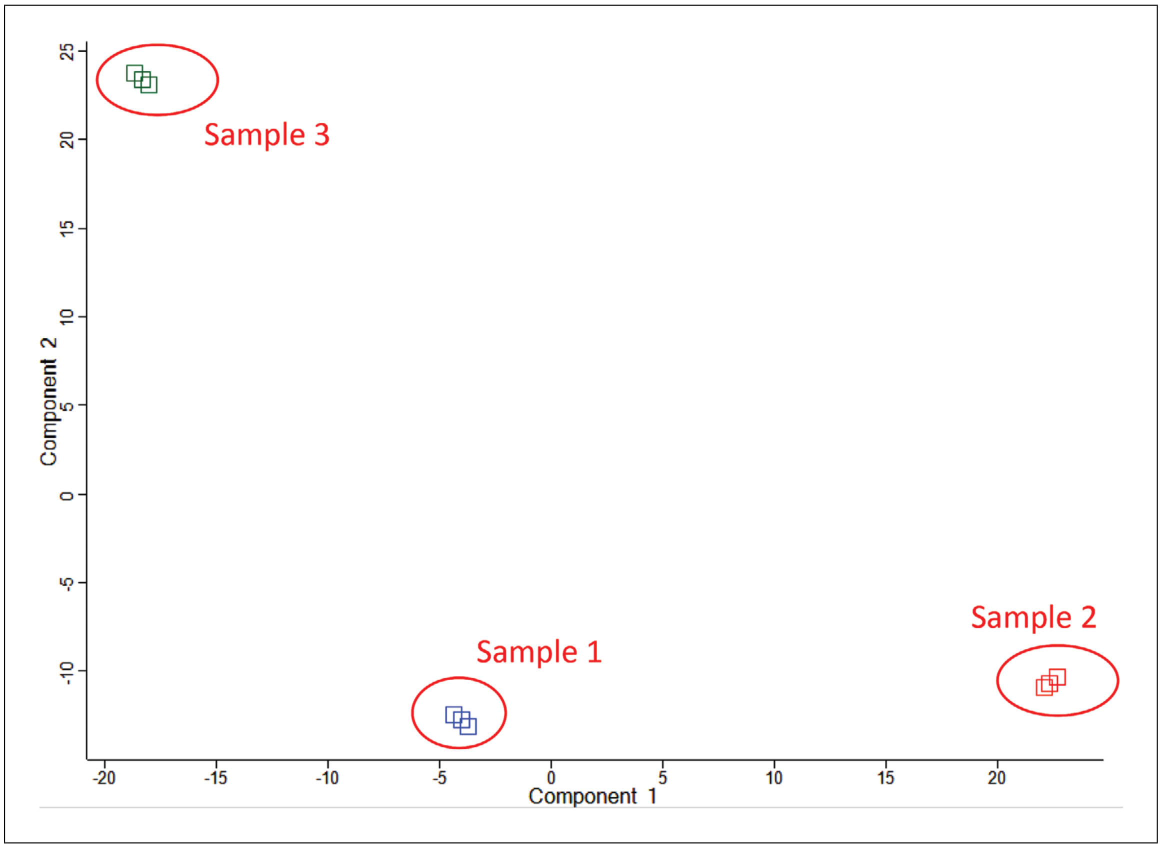


Figure 10 UMAP plot of the example LFQ dataset. The blue, red, and green squares represent the sample 1, 2, and 3, respectively.

Table 3 Troubleshooting Guide for Perseus

Error	Solution
The R or python executable cannot be found by Perseus.	The executables have to be added to the system "Path" environment variable. To do this, open the "Control Panel," go to "System," and then to "Advanced System Settings." In the new window, click on the "Environment variables" configuration. From the list of environment variables, select the "PATH," click "edit," and check that the entry for the R/Python executable is correct. If it is missing, add the directory as a new entry. These instructions are for Windows 10. If you cannot locate your path variable, please refer to one of many instructions that can be found online.
A package (in R) or library (in python) cannot be imported.	If this error occurs, please make sure that your R or Python environment contains the respective library. Perseus cannot install required libraries on the fly.
Perseus cannot recognize the output matrix generated after R or Python execution	Be sure in which kind of format the output from the R/Python script is defined. Perseus takes the matrix generated by a `data.frame`
Identification of Annotation Rows in case of possible groups	In plugins where annotation rows are necessary, be sure that your script recognizes the groups defined in the matrix input, adding the if-case as the argument (!length(annotRows(mdata)), and then call it again at the end of script execution if you want to combine your annotation with the results obtained
PerseusR package (in R) and/or Perseuspy (in python) are not recognized	Before to use the packages, check if the installation is done correctly by the command line In R: `library(PerseusR)` In Python: `import argparse from perseuspy`

visualization are also provided in MaxQuant. The output tables of MaxQuant can be directly imported into Perseus for the downstream bioinformatics and statistical analyses.

In past decades, high-throughput sequencing (HTS) has become a potent method in numerous biological research fields. Perseus also provides the ability to import BAM files and genome annotation files for mRNA quantification (Tyanova et al., 2016). Thus, the bioinformatics and statistical analyses can be applied to HTS datasets as well. This makes Perseus a powerful multi-omics data analysis platform (Poulopoulos et al., 2019).

Critical Parameters

Table 2 lists commonly used parameters for R, Python, and C# plugins.

Troubleshooting

Table 3 provides troubleshooting information.

Acknowledgments

This work was partially funded by the Max Planck Society for the Advancement of Science and the German Research Foundation (DFG/Gottfried Wilhelm Leibniz Prize MA 1764/2-1), and has been made possible in part by grant number 2019-202671 from the Chan Zuckerberg Foundation. Open access funding enabled and organized by Projekt DEAL

Author Contributions

Sung-Huan Yu: Conceptualization; formal analysis; methodology; writing-original draft; writing-review & editing. **Daniela Ferretti:** Formal analysis; software; writing-original draft; writing-review & editing. **Julia P. Schessner:** Formal analysis; writing-original draft; writing-review & editing. **Jan Daniel Rudolph:** Writing-original draft; writing-review & editing. **Georg H. H. Borner:** Writing-original draft; writing-review & editing. **Jürgen Cox:** Conceptualization; formal analysis; funding acquisition; investigation; methodology; project administration; software; supervision; writing-original draft; writing-review & editing.

Literature Cited

Bailey, D. J., McDevitt, M. T., Westphall, M. S., Pagliarini, D. J., & Coon, J. J. (2014). Intelligent data acquisition blends targeted and discovery methods. *Journal of Proteome Research*, *13*(4), 2152–2161. doi: 10.1021/pr401278j.

Becht, E., McInnes, L., Healy, J., Dutertre, C.-A., Kwok, I. W. H., Ng, L. G., … Newell, E. W. (2019). Dimensionality reduction for visualizing single-cell data using UMAP. *Nature Biotechnology*, *37*(1), 38–44. doi: 10.1038/nbt.4314.

Choi, M., Eren-Dogu, Z. F., Colangelo, C., Cottrell, J., Hoopmann, M. R., Kapp, E. A., … Vitek, O. (2017). ABRF Proteome Informatics Research Group (iPRG) 2015 study: Detection of differentially abundant proteins in label-free quantitative LC-MS/MS experiments. *Journal of Proteome Research*, *16*(2), 945–957. doi: 10.1021/acs.jproteome.6b00881.

Cox, J., Hein, M. Y., Luber, C. A., Paron, I., Nagaraj, N., & Mann, M. (2014). Accurate proteome-wide label-free quantification by delayed normalization and maximal peptide ratio extraction, termed MaxLFQ. *Molecular and Cellular Proteomics*, *13*(9), 2513–2526. doi: 10.1074/mcp.M113.031591.

Cox, J., & Mann, M. (2008). MaxQuant enables high peptide identification rates, individualized p.p.b.-range mass accuracies and proteome-wide protein quantification. *Nature Biotechnology*, *26*(12), 1367–1372. doi: 10.1038/nbt.1511.

Johnson, W. E., Li, C., & Rabinovic, A. (2007). Adjusting batch effects in microarray expression data using empirical Bayes methods. *Biostatistics*, *8*(1), 118–127. doi: 10.1093/biostatistics/kxj037.

Langfelder, P., & Horvath, S. (2008). WGCNA: An R package for weighted correlation network analysis. *BMC Bioinformatics*, *9*(1), 559. doi: 10.1186/1471-2105-9-559.

Leek, J. T., Johnson, W. E., Parker, H. S., Jaffe, A. E., & Storey, J. D. (2012). The sva package for removing batch effects and other unwanted variation in high-throughput experiments. *Bioinformatics (Oxford, England)*, *28*(6), 882–883. doi: 10.1093/bioinformatics/bts034.

Love, M. I., Huber, W., & Anders, S. (2014). Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biology*, *15*(12), 550. doi: 10.1186/s13059-014-0550-8.

van der Maaten, L., & Hinton, G. (2008). *Visualizing data using t-SNE*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.457.7213.

McInnes, L., Healy, J., & Melville, J. (2018). *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction*. Retrieved from http://arxiv.org/abs/1802.03426.

Mckinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der W. J. Millman (Ed.), *Proceedings of the 9th Python in Science Conference* (51–56).

Poulopoulos, A., Murphy, A. J., Ozkan, A., Davis, P., Hatch, J., Kirchner, R., & Macklis, J. D. (2019). Subcellular transcriptomes and proteomes of developing axon projections in the cerebral cortex. In *Nature* (Vol. 565, Issue 7739, pp. 356–360). New York: Nature Publishing Group. doi: 10.1038/s41586-018-0847-y.

Ritchie, M. E., Phipson, B., Wu, D., Hu, Y., Law, C. W., Shi, W., & Smyth, G. K. (2015). Limma powers differential expression analyses for RNA-sequencing and microarray studies. *Nucleic Acids Research*, *43*(7), e47–e47. doi: 10.1093/nar/gkv007.

Robinson, M. D., McCarthy, D. J., & Smyth, G. K. (2010). edgeR: A Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics (Oxford, England)*, *26*(1), 139–140. doi: 10.1093/bioinformatics/btp616.

Rudolph, J. D., & Cox, J. (2019). A network module for the perseus software for computational proteomics facilitates proteome interaction graph analysis. *Journal of Proteome Research*, *18*(5), 2052–2064. doi: 10.1021/acs.jproteome.8b00927.

Sinitcyn, P., Rudolph, J. D., & Cox, J. (2018). Computational methods for understanding mass spectrometry–based shotgun proteomics data. *Annual Review of Biomedical Data Science*, *1*(1), 207–234. doi: 10.1146/annurev-biodatasci-080917-013516.

Tyanova, S., Mann, M., & Cox, J. (2014). MaxQuant for in-depth analysis of large SILAC datasets. *Methods in Molecular Biology*, *1188*, 351–364. doi: 10.1007/978-1-4939-1142-4_24.

Tyanova, S., Temu, T., & Cox, J. (2016). The MaxQuant computational platform for mass spectrometry−based shotgun proteomics. *Nature Protocols*, *11*(12), 2301–2319. doi: 10.1038/nprot.2016.136.

Tyanova, S., Temu, T., Sinitcyn, P., Carlson, A., Hein, M. Y., Geiger, T., … Cox, J. (2016). The Perseus computational platform for comprehensive analysis of (prote)omics data. *Nature Methods*, *13*(9), 731–740. doi: 10.1038/nmeth.3901.

Wiśniewski, J. R., Hein, M. Y., Cox, J., & Mann, M. (2014). A “Proteomic Ruler” for protein copy number and concentration estimation without spike-in standards. *Molecular & Cellular Proteomics*, *13*(12), 3497–3506. doi: 10.1074/mcp.M113.037309.

Yu, S.-H., Kiriakidou, P., & Cox, J. (2020). Isobaric matching between runs and novel PSM-level normalization in MaxQuant strongly improve reporter ion-based quantification. *BioRxiv*, 2020.03.30.015487. doi: 10.1101/2020.03.30.015487.

Internet Resources

Plugin repositories

https://github.com/cox-labs/PluginTutorial
Tutorial scripts.

https://github.com/JurgenCox/perseus-plugins
Source code for many plug-ins.

Tutorial videos can be found in MaxQuant Summer School

https://www.youtube.com/watch?v=fYGx4oplCpI&t=3146s
MQSS 2018.

https://www.youtube.com/watch?v=-3oq9e_92lc
MQSS 2019.