Tutorial: exploration of multi-omics data

DUBii 2020

Jacques van Helden

2021-03-17

Contents

Goals of this tutorial

This tutorial aims at

1. Learn to load files from remote locations, either directly or by downloading them to a local folder.
2. Apply some of the methods taught in the previous courses in order to explore a data set.
3. Show some convenient ways of combinig R code and markdown elements within an R markdown document in order to obtain a well-formatted scientific report.
   - configuring the parameters in the yaml header of the R markdown file
   - organising the code in R chunks
   - writing, documenting and using an R function
   - generating an ugly figure, improving it and controlling its incorporation in the report (size, legend)

Study case : mouse kidney

As study case for this tutorial, we will use multi-omics data from a study published by Pavkovic et al. (2019), which combines transctiptomics (RNA-seq) and proteomics approaches to understand the molecular mechanisms underlying the kidney fibrosis pathology.

The authors applied two commonly used treatments to induce kidney fibrosis in mouse:

- a reversible chemical-induced injury model, denoted as **FA** for **folic acid** induced nephropathy;
- an irreversible surgically-induced fibrosis model, denoted as **UUO** for **unilateral uretral obstruction**.

References and data sources

- **Reference**: Pavkovic, M., Pantano, L., Gerlach, C.V. et al. Multi omics analysis of fibrotic kidneys in two mouse models. Sci Data 6, 92 (2019) https://doi.org/10.1038/s41597-019-0095-5
- **Mouse fibrotic kidney browser:** http://hbcreports.med.harvard.edu/fmm/
- Data on Zenodo: https://zenodo.org/record/2592516

Data preparation

A description of the study case can be found in the *Mus musculus section* of the the DUBii study cases repository repository.

We also provide there a detailed explanation of the data preparation steps:

- downloading the data from its original source repository,
- exploring the datasets it with various graphical representtions,
- computing some descriptive statistics on the different samples,
- pre-processing,
- storing the results in a memory image.

Data file naming

We prepared the data from Pavkovic as a text file with tab-separated values (**tsv** files).

All the files are available on github: - https://github.com/DU-Bii/module-3-Stat-R/tree/master/stat-R_2021/data/pavkovic_2019

The files are named according to the following convention:

- the prefix indicate the data type
  - **fa**: folic acid induced nephropathy (reversible), transcriptome data
  - **pfa**: folic acid induced nephropathy, proteome data
  - **uuo**: unilateral uretral obstruction (irreversible), transcriptome data
  - **puuo**: unilateral uretral obstruction, proteome data
- The suffix indicates the data normalisation
  - **raw**: transcriptome counts provided by the authors (note: not integer, because their raw data was already somehow normalized)
  - **normalized¨**: transcriptome counts standardized to achieve the same third quartile for each sample
  - **log2**: log2 transformation for proteome data

- the **metadata** files contain a short description of each sample (one row per sample). Note that the last column contains a sample-specific color specification in hexadecimal web color code to facilitate the drawings. Don't hesitate to chose other colors according to your personal taste.

R style

The R code belows follows the tidyverse styling guide (https://style.tidyverse.org/).

Approach for this tutorial

This tutorial will consist of exercises that can be realised in a stepwise way, with alternance of working sessions and live demos of the solutions, in order to make sure that all the trainees acquire each step.

Data exploration

Before computing any descriptive parameter on a dataset, I generallyi attempt to get a picture of the whole distribution.

Finding a data file on github

We will provide here a magic recipe to download the data from the github repository to your local folder, and to load it in R.

```
## specify the base URL from which data files can be downloaded
url_base <- "https://github.com/DU-Bii/module-3-Stat-R/raw/master/stat-R_2021/data/pavkovic_2019"


## Choose a specific data file
data_prefix <- "pfa" ## proteome data of folic-acid treated mouse
data_suffix <- "model" ## no normalization
file_name <- paste0(data_prefix, "_", data_suffix, "_counts.tsv.gz")

## Compose the URL to download the file from github
url <- file.path(url_base, file_name)

message("URL: ",  url)
```

Loading a data file directly from github

Now we defined the URL, we can easily load the file directly from github to a data frame in our R environement.

```
## this requires to load a specific package
if (!require("data.table")) {
  install.packages("data.table")
}
library(data.table)

pfa <- fread(url, header = TRUE, sep = "\t")
dim(pfa)
names(pfa)
kable(head(pfa))
```

Downloading a data file and storing it locally once forever

We can now download the data file to a local folder, but we would like to do this only once.

```
## Specify the path relative to your home directory (~)
local_folder <- "~/DUBii-m3_data/pavkovic_2019"
local_file <- file.path(local_folder, file_name)

## Create the local data folder if it does not exist
dir.create(local_folder, showWarnings = FALSE, recursive = TRUE)

## Download the file ONLY if it is not already there
if (!file.exists(local_file)) {
  message("Downloading file from github to local file\n\t",
          local_file)
  download.file(url = url, destfile = local_file)
} else {
  message("Local file already exists, no need to download\n\t",
          local_file)
}
```

Loading the local copy of your data file

We will now load the proteome file, with the following parameters

- the first row contains the column headers
- the first columnt contains the protein IDs, and we would like to have them as row.names for the loaded data frame. This is a bit tricky because some protein names are diplicated. We use the **very** convenient function `make.names(x, unique = TRUE)`.

```
## Load the data from the local file
pfa <- read.delim(file = local_file, header = TRUE, sep = "\t")

kable(head(pfa), caption = "Data frame just after loading")
```

Table 1: Data frame just after loading

id	normal_1	normal_2	day1_1	day1_2	day2_1	day2_2	day7_1	day7_2	day14_1	day14_2
ENSMUSG00000037686	680	651.7200	335.5910	334.8460	197.1740	307.194	123.2060	272.6190	93.7247	196.1590
ENSMUSG00000027836	020	266.3590	175.4090	159.4190	234.8080	256.927	149.9380	315.0590	110.5880	126.3600
ENSMUSG00000020723	23	29.1331	57.7329	45.8475	81.6009	88.870	29.8560	44.5586	13.6292	27.6568
ENSMUSG00000041095	500	4784.0800	4064.4800	3917.2900	4599.0300	5957.030	2806.5200	6792.0900	2022.5100	3226.7500
ENSMUSG00000079327	790	1065.3900	914.2870	928.2760	1000.1000	1264.270	738.3520	1362.0700	466.4440	714.5870
ENSMUSG00000082225	225	89.8871	57.9041	76.3510	84.8474	105.245	72.8696	138.9810	40.8101	59.2117

```
## Convert the first colum to row names
row.names(pfa) <- make.names(as.vector(pfa$id), unique = TRUE)
pfa <- pfa[, -1] ## Suppress the ID colimn
kable(head(pfa), caption = "Data frame with row names")
```

Table 2: Data frame with row names

	normal_1	normal_2	day1_1	day1_2	day2_1	day2_2	day7_1	day7_2	day14_1	day14_2
ENSMUSG00000037686	380.2680	651.7200	335.5910	334.8460	197.1740	307.194	123.2060	272.6190	93.7247	196.1590
ENSMUSG00000027830	278.1020	266.3590	175.4090	159.4190	234.8080	256.927	149.9380	315.0590	110.5880	126.3600
ENSMUSG00000020717	29.2077	29.1331	57.7329	45.8475	81.6009	88.870	29.8560	44.5586	13.6292	27.6568
ENSMUSG00000031095	4169.0500	4784.0800	4064.4800	3917.2900	4599.0300	5957.030	2806.5200	6792.0900	2022.5100	3226.7500
ENSMUSG00000079317	879.2790	1065.3900	914.2870	928.2760	1000.1000	1264.270	738.3520	1362.0700	466.4440	714.5870
ENSMUSG00000068225	82.2225	89.8871	57.9041	76.3510	84.8474	105.245	72.8696	138.9810	40.8101	59.2117

Writing a function to download a file only once

Write a functions that will download a file from a remote location to a local folder, but do this only if the local file is not yet present there.

Note that we use the roxygen2 format to write the documentation of this function. In any programming language, a function should always be documented in order to enable other people to use it, and the doc is also very useful for a developer to reuse her/his own code. The documentation becomes particularly interesting when you start building your own R packages, since it will automatically generate the help pages.

The documentation of a function should include - a description of what it does - the author name and a way to contact her/him - a description of each parameter (argument) of the function - a description of the return value

Roxygen2 provides is a very convenient way of documenting a function, because - the formalism is very simple - the doc comes together with the code of the function (by default, R functions are documented in a separate file)

```
#' @title Download a file only if it is not yet here
#' @author Jacques van Helden email{Jacques.van-Helden@@france-bioinformatique.fr}
#' @param url_base base of the URL, that will be prepended to the file name
#' @param file_name name of the file (should not contain any path)
#' @param local_folder path of a local folder where the file should be stored
#' @return the function returns the path of the local file, built from local_folder and file_name
#' @export
downloadOnlyOnce <- function(url_base,
                             file_name,
                             local_folder) {

  ## Define the source URL
  url <- file.path(url_base, file_name)
  message("Source URL\n\t",  url)

  ## Define the local file
  local_file <- file.path(local_folder, file_name)

  ## Create the local data folder if it does not exist
  dir.create(local_folder, showWarnings = FALSE, recursive = TRUE)

  ## Download the file ONLY if it is not already there
  if (!file.exists(local_file)) {
    message("Downloading file from source URL to local file\n\t",
            local_file)
    download.file(url = url, destfile = local_file)
```

```
  } else {
    message("Local file already exists, no need to download\n\t",
            local_file)
  }

  return(local_file)
}
```

We can now use our new function `downloadOnlyOnce()` to download the files from the folic acid dataset and store them in a local folder. We will download successively :

- transcriptome data (`fa`)
- transcriptome medata
- proteome data (`pfa`)
- proteome medata

```
## Specify the basic parameters
pavkovic_base <- "https://github.com/DU-Bii/module-3-Stat-R/raw/master/stat-R_2021/data/pavkovic_2019"
pavkovic_folder <- "~/DUBii-m3_data/pavkovic_2019"

#### Dowload folic acid data and metadata ####

## Transcriptome data table
local_fa_file <- downloadOnlyOnce(
  url_base = pavkovic_base,
  file_name = "fa_raw_counts.tsv.gz",
  local_folder =  pavkovic_folder
)

## Transcriptome metadata
trans_metadata_file <- downloadOnlyOnce(
  url_base = pavkovic_base,
  file_name = "transcriptome_metadata.tsv",
  local_folder =  pavkovic_folder
)

## Proteome data table
local_pfa_file <- downloadOnlyOnce(
  url_base = pavkovic_base,
  file_name = "pfa_model_counts.tsv.gz",
  local_folder =  pavkovic_folder
)

## Proteome metadata
prot_metadata_file <- downloadOnlyOnce(
  url_base = pavkovic_base,
  file_name = "proteome_metadata.tsv",
  local_folder =  pavkovic_folder
)
```

After having run the chunk of code above, try to re-run it. In principle, you should just receive messages telling you that the files are already there.

Loading the files

We now write a function `load_fix_row_names()` that loads a data file and takes a specified column as row names, whilst automatically fixing potential problems due to duplicate labels in this column.

```
#' @title Load a tab-separated value file and manually set row ames after having forced them to unique v
#' @author Jacques van Helden email{Jacques.van-Helden@@france-bioinformatique.fr}
#' @param file file path
#' @param header=1 Header is set to 1 by default
#' @param sep="\t" Column separator is set to tab by default
#' @param rownames.col=1 Column containing the row names
#' @param ... all other parameters are passed to read.delim()
#' @return a data frame with the loaded data
load_fix_row_names <- function(file,
                       header = 1,
                       sep = "\t",
                       rownames.col = 1,
                       ...) {
  x <- read.delim(file = file, ...)
  rownames(x) <- make.names(x[, rownames.col], unique = TRUE)
  x <- x[, -rownames.col]
  return(x)
}
```

We can now load the data that from our local folder.

```
## Load transcriptome data
fa <- read.delim(file = local_fa_file, sep = "\t", header = TRUE)

## Check the first lines of the loaded file
dim(fa)
```

```
[1] 46679    19
```

```
kable(head(fa), caption = "Loaded with read.delim()")
```

Table 3: Loaded with read.delim()

rowname	day1__1	day1__2	day1__3	day14_d	day14_d	day14_d	day2__1	day2__2	day2__3	day3__1	day3__2	day3__3	day7__1	day7__2	day7__3	normaln	dr
ENSMUSG	[illegible]																
ENSMUSG	[illegible]																
ENSMUSG	[illegible]																
ENSMUSG	[illegible]																
ENSMUSG	[illegible]																
ENSMUSG	[illegible]																

```
## Load same data with load_fix_row_names
fa <- load_fix_row_names(file = local_fa_file, rownames.col = 1)
dim(fa)
```

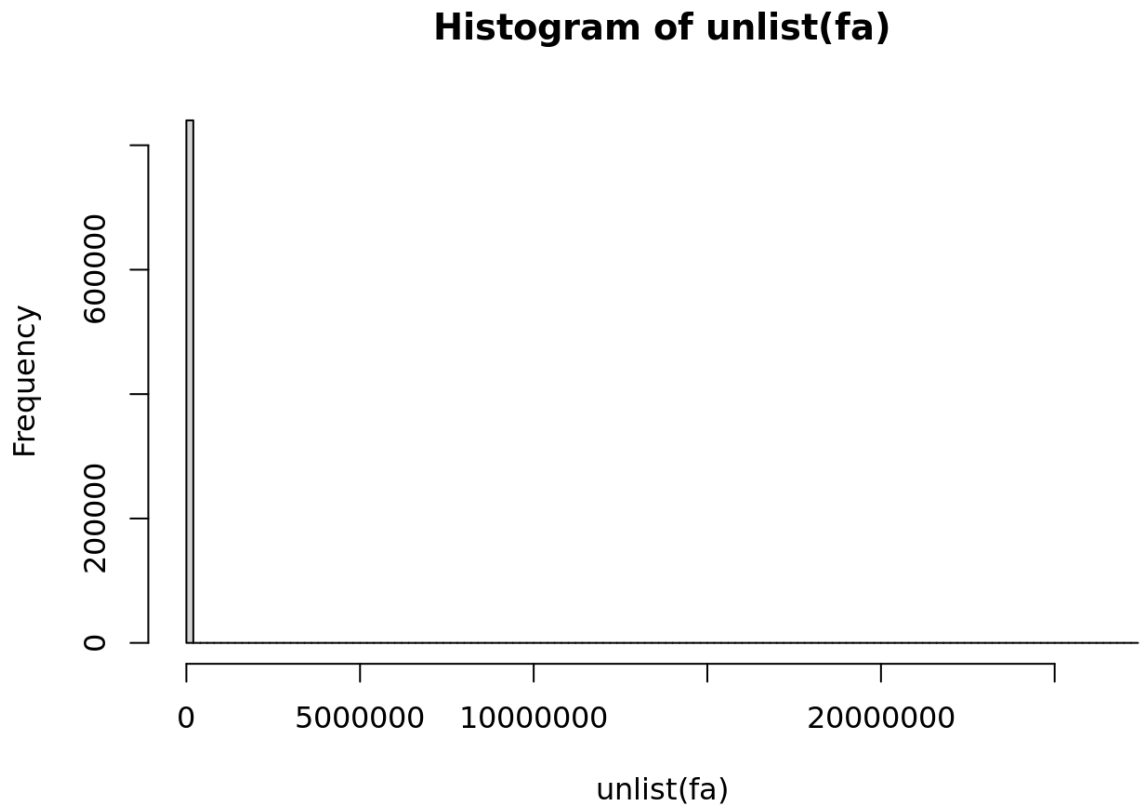
```
[1] 46679    18
```

```
kable(head(fa), caption = "Loaded with myEasyLad()")
```

Table 4: Loaded with myEasyLad()

	day1__1	day1__2	day1__3	day14_1	day14_2	day14_3	day2__1	day2__2	day2__3	day3__1	day3__2	day3__3	day7__1	day7__2	day7__3	normaln
ENSMUSG[illegible]	[illegible]															
ENSMUSG[illegible]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
ENSMUSG[illegible]	[illegible]															
ENSMUSG[illegible]	[illegible]															
ENSMUSG[illegible]	[illegible]															
ENSMUSG[illegible]	[illegible]															

```
hist(unlist(fa), breaks = 100)
```




```
## Load proteome data
pfa <- load_fix_row_names(file = local_pfa_file, rownames.col = 1)
dim(pfa)
```

```
[1] 8044   10
```

```
## Check the first lines of the loaded file
kable(head(pfa))
```

	normal__1	normal__2	day1__1	day1__2	day2__1	day2__2	day7__1	day7__2	day14__1	day14__2
ENSMUSG[illegible]	[illegible]	651.7200	335.5910	334.8460	197.1740	307.194	123.2060	272.6190	93.7247	196.1590
ENSMUSG[illegible]	[illegible]	266.3590	175.4090	159.4190	234.8080	256.927	149.9380	315.0590	110.5880	126.3600

	normal_1	normal_2	day1_1	day1_2	day2_1	day2_2	day7_1	day7_2	day14_1	day14_2
ENSMUSG00000029207	29.2072	29.1331	57.7329	45.8475	81.6009	88.870	29.8560	44.5586	13.6292	27.6568
ENSMUSG00000031695	4163.9500	4784.0800	4064.4800	3917.2900	4599.0300	5957.030	2806.5200	6792.0900	2022.5100	3226.7500
ENSMUSG00000079327	879.2790	1065.3900	914.2870	928.2760	1000.1000	1264.270	738.3520	1362.0700	466.4440	714.5870
ENSMUSG00000038225	68.2225	89.8871	57.9041	76.3510	84.8474	105.245	72.8696	138.9810	40.8101	59.2117

```
## Load proteome metadata
proteome_metadata <- read.delim(file = prot_metadata_file, sep = "\t", header = TRUE)
kable(proteome_metadata, caption = "Metadata for the proteome dataset")
```

Table 6: Metadata for the proteome dataset

dataType	sampleName	condition	sampleNumber	color
transcriptome	normal_1	normal	1	#BBFFBB
transcriptome	normal_2	normal	2	#BBFFBB
transcriptome	day3_1	day3	1	#FFFFDD
transcriptome	day3_2	day3	2	#FFFFDD
transcriptome	day3_3	day3	3	#FFFFDD
transcriptome	day7_1	day7	1	#FFDD88
transcriptome	day7_2	day7	2	#FFDD88
transcriptome	day7_3	day7	3	#FFDD88
transcriptome	day14_1	day14	1	#FF4400
transcriptome	day14_2	day14	2	#FF4400

```
## Load transcriptome metadata
transcriptome_metadata <- read.delim(file = trans_metadata_file, sep = "\t", header = TRUE)
kable(transcriptome_metadata, caption = "Metadata for the transcriptome dataset")
```

Table 7: Metadata for the transcriptome dataset

dataType	sampleName	condition	sampleNumber	color
transcriptome	day14_12	day14	12	#FF4400
transcriptome	day14_13	day14	13	#FF4400
transcriptome	day14_14	day14	14	#FF4400
transcriptome	day14_15	day14	15	#FF4400
transcriptome	day3_4	day3	4	#FFFFDD
transcriptome	day3_5	day3	5	#FFFFDD
transcriptome	day3_6	day3	6	#FFFFDD
transcriptome	day3_7	day3	7	#FFFFDD
transcriptome	day7_10	day7	10	#FFDD88
transcriptome	day7_11	day7	11	#FFDD88
transcriptome	day7_8	day7	8	#FFDD88
transcriptome	day7_9	day7	9	#FFDD88
transcriptome	normal_1	normal	1	#BBFFBB
transcriptome	normal_2	normal	2	#BBFFBB
transcriptome	normal_3	normal	3	#BBFFBB

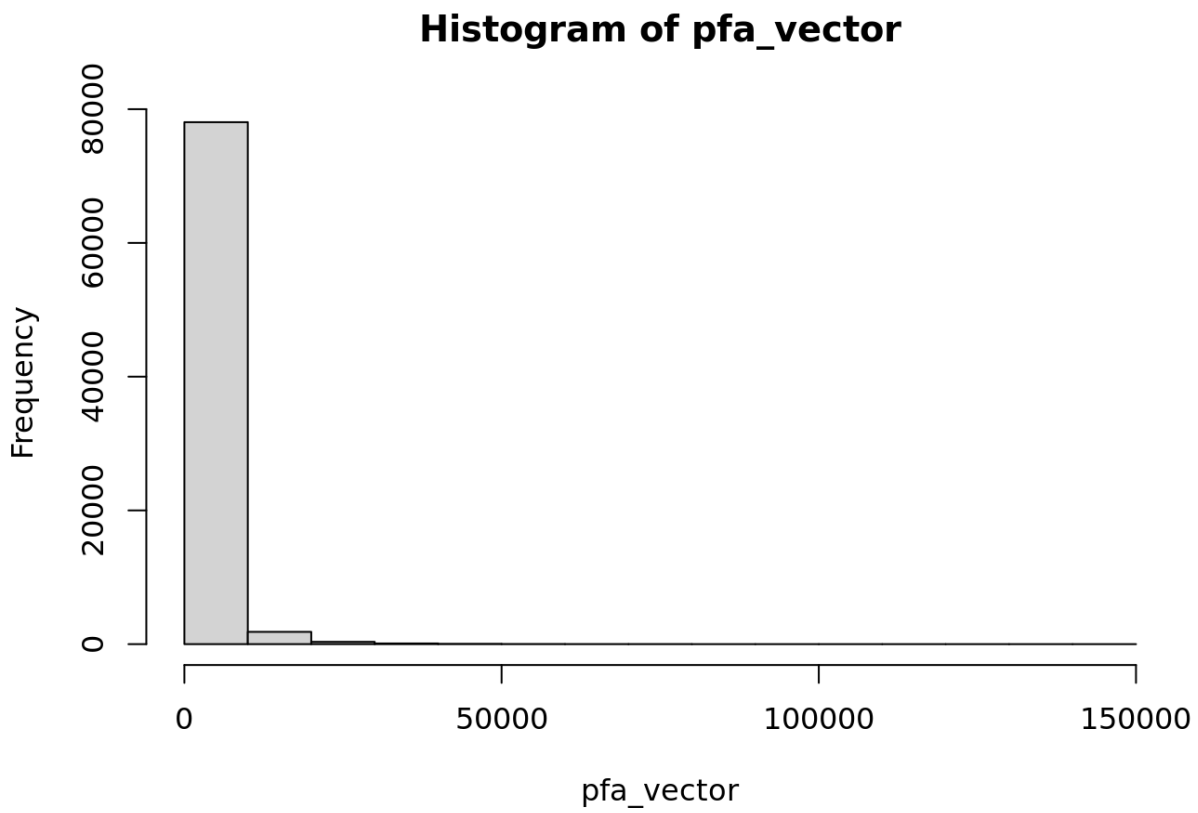
We can now use this function to download and load the different data files.

Graphical exploration of the data

Histogram

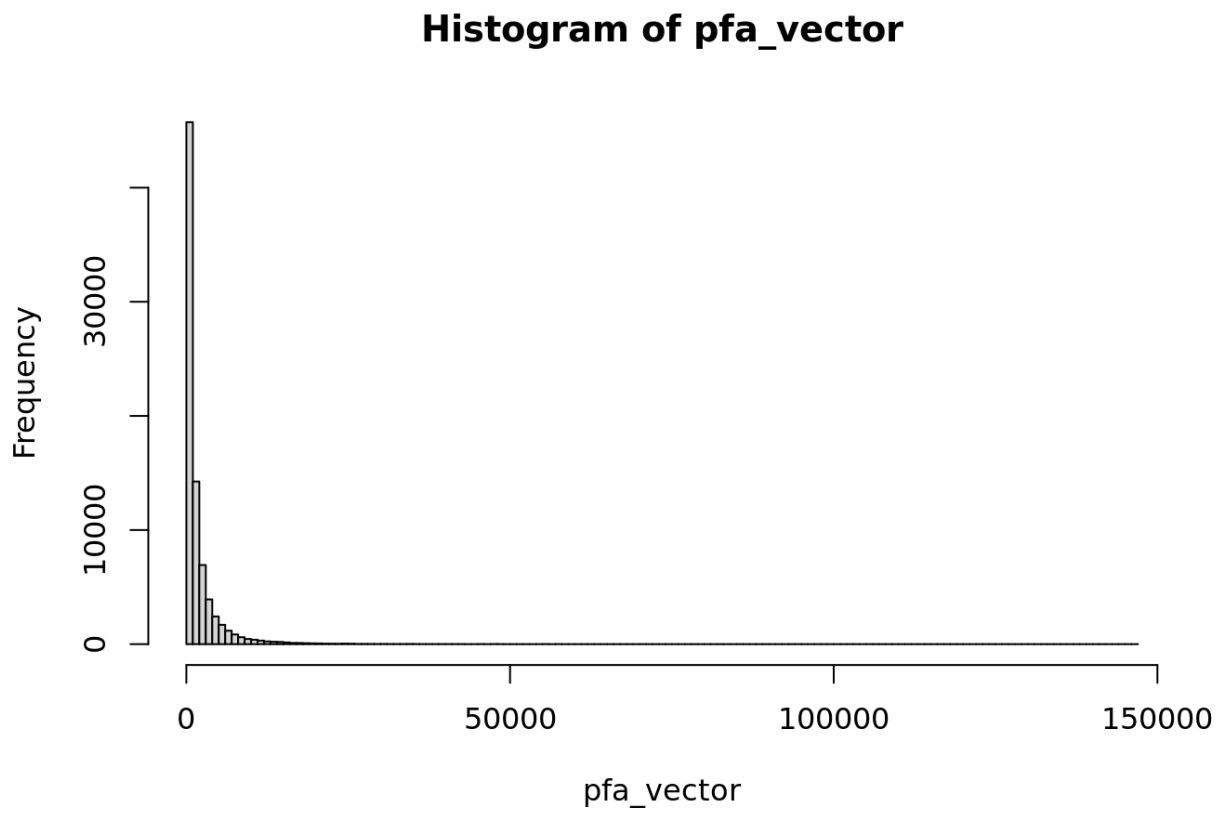
Draw histograms of the transcriptome and proteom data, all samples together.

```
pfa_vector <- unlist(as.vector(pfa))
hist(pfa_vector)
```

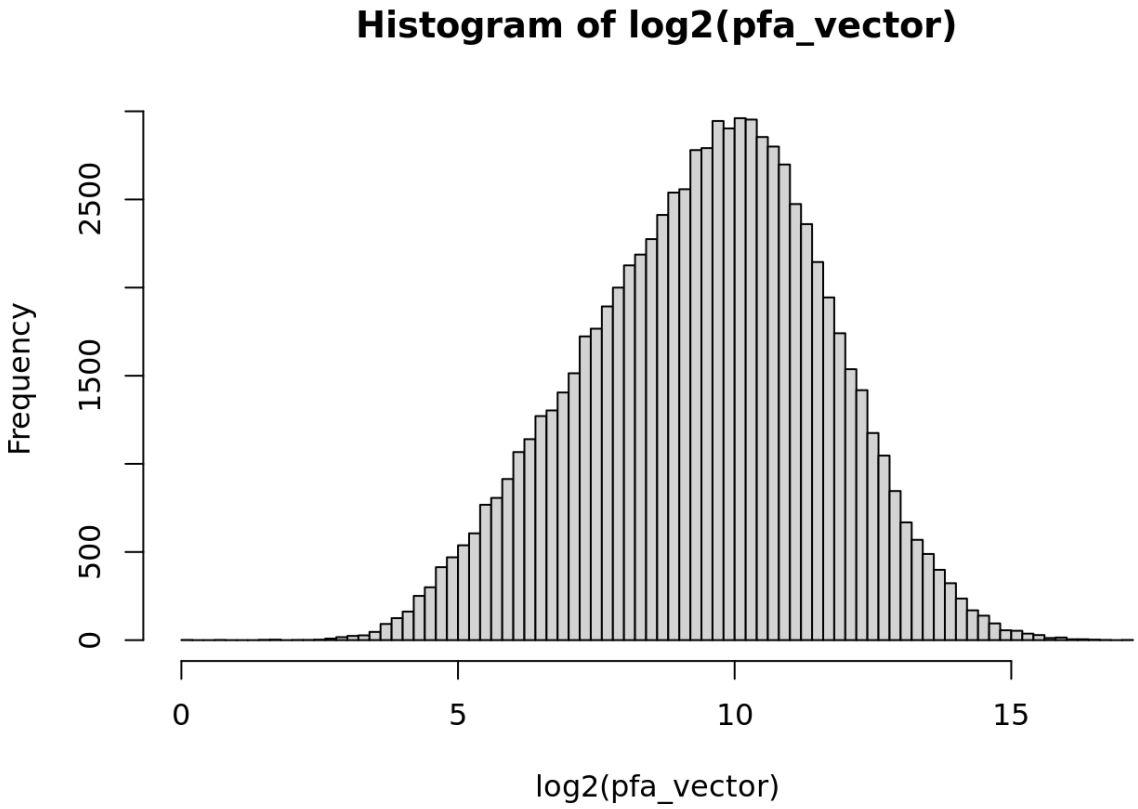


Let us now improve the histogram in order to get an intuition of our data.

```
hist(pfa_vector, breaks = 200)
```




```
hist(log2(pfa_vector), breaks = 100)
```



Box plot

Scatter plot

Save a memory image of your session

Use the finction `save.image()` to store an image of your session, in a file `pavkovic_memory_image.Rdata` in the local folder specified above. This file will contain all the data that you loaded during this session, and enable you to reload everything without having to re-execute all the steps.

```
## Define the path to the memory image file
memory_image_file <- file.path(local_folder, "pavkovic_memory_image.Rdata")

## Save the memory image
save.image(file = memory_image_file)

message("Memory image saved in file\n\t", memory_image_file)
```

For a future session, you will be able to reload all the data with a single command:

`load([path_to_your_memory_image])`

You will need to replace `[path_to_your_memory_image]` by your actual path. In my case, the command becomes:

```
load("~/DUBii-m3_data/pavkovic_2019/pavkovic_memory_image.Rdata")
```

Save your session info

For the sake of traceability, store the specifications of your R environment in the report, with the command `sessionInfo()`. This will indicate the version of R as wella sof all the libraries used in this notebook.

```
sessionInfo()
```

```
R version 4.0.2 (2020-06-22)
Platform: x86_64-conda_cos6-linux-gnu (64-bit)
Running under: CentOS Linux 7 (Core)

Matrix products: default
BLAS/LAPACK: /shared/ifbstor1/software/miniconda/envs/r-4.0.2/lib/libopenblasp-r0.3.10.so

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C               LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.U
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods   base

other attached packages:
[1] knitr_1.30

loaded via a namespace (and not attached):
 [1] compiler_4.0.2    magrittr_2.0.1    tools_4.0.2       htmltools_0.5.1.1 yaml_2.2.1        stringi_
```