# A complete analysis of peptide microarray binding data using the pepStat framework

Greg Imholte[*], Renan Sauteraud[†], Mike Jiang[‡] and Raphael Gottardo[§]

October 26, 2021

This document present a full analysis, from reading the data to displaying the results that makes use of all the packages we developped for peptide microarray.

## Contents



[*]gimholte@uw.edu
[†]rsautera@fhcrc.org
[‡]wjiang2@fhcrc.org
[§]rgottard@fhcrc.org

# 1 Introduction

The pepStat package offers a complete analytical framework for the analysis of peptide microarray data. It includes a novel normalization method to remove non-specific peptide binding activity of antibodies, a data smoothing reducing step to reduce background noise, and subject-specific positivity calls.

## 1.1 Requirements

The pepStat package requires GSL, an open source scientific computing library. This library is freely available at http://www.gnu.org/software/gsl/.

In this vignette, we make use of the samples and examples available in the data package pepDat.

# 2 Generating a peptideSet

```
library(pepDat)
library(pepStat)
```

## 2.1 Reading in .gpr files

The reading function, makePeptideSet, takes a path as its argument and parses all the *.gpr* files in the given directory. Alternatively, one may specify a character vector of paths to individual *.gpr* files.

By default channels F635 Median and B635 Median are collected, and the 'normexp' method of the backgroundCorrect function in the limma package corrects probe intensities for background fluorescence. Other methods may be selected, see documentation.

```
mapFile <- system.file("extdata/mapping.csv", package = "pepDat")
dirToParse <- system.file("extdata/gpr_samples", package = "pepDat")
pSet <- makePeptideSet(files = NULL, path = dirToParse,
                       mapping.file = mapFile, log=TRUE)
```

While optional, it is strongly recommended to provide a mapping.file giving annotations data for each slide, such as treatment status or patient information. If provided, the mapping.file should be a .csv file. It must include columns labeled filename, ptid, and visit. Elements in column filename must correspond to the filenames of slides to be read in, without the .gpr extension. Column ptid is a subject or slide identifier. Column visit indicates a case or control condition, such as pre/post vaccination, pre/post infection, or healthy/infected status. Control conditions must be labelled *pre*, while case conditions must be labelled *post*. Alternatively, one may input a data.frame satisfying the same requirements.

This minimal information is required by pepStat's functions further in the analysis. Any additional information (column) will be retained and can be used as a grouping variable.

If no mapping file is included, the information will have to be added later on to the peptideSet object.

For our example, we use a toy dataset of 8 samples from 4 patients and we are interested in comparing the antibody binding in placebo versus vaccinated subjects.

```
read.csv(mapFile)

##   filename ptid visit treatment
## 1     f1_1    1   Pre   PLACEBO
## 2     f1_2    1  Post   PLACEBO
## 3     f2_1    2   Pre   PLACEBO
## 4     f2_2    2  Post   PLACEBO
## 5     f3_1    3   Pre   VACCINE
## 6     f3_2    3  Post   VACCINE
## 7     f4_1    4   Pre   VACCINE
## 8     f4_2    4  Post   VACCINE
```

## 2.2 Additional arguments

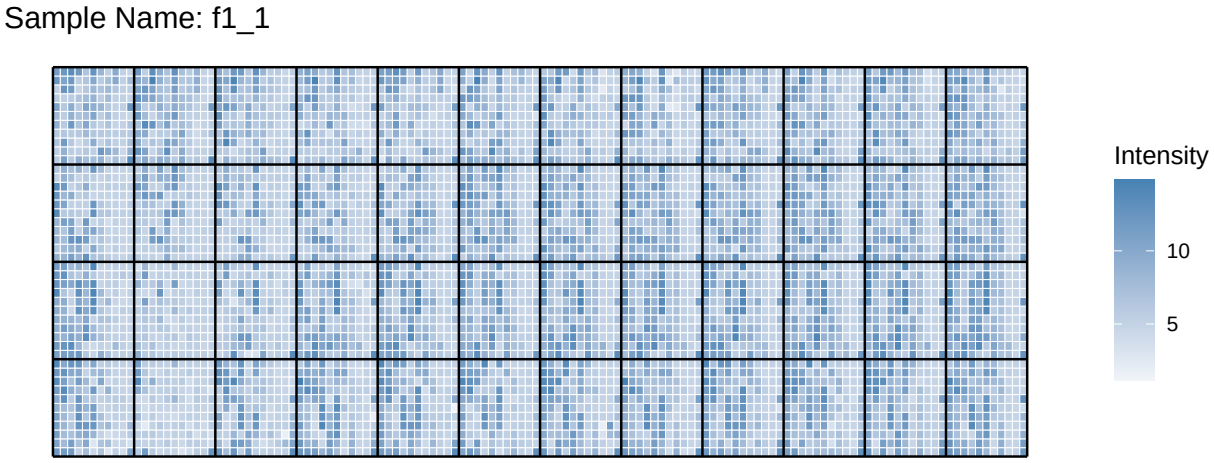
The empty spots should be listed in order to background correct the intensities. It is also useful to remove the controls when reading the data. Here we have the JPT controls, human Ig (A, E and M) and dye controls.

```
pSetNoCtrl <- makePeptideSet(files = NULL, path = dirToParse,
                        mapping.file = mapFile, log = TRUE,
                        rm.control.list = c("JPT-control", "Ig", "Cy3"),
                        empty.control.list= c("empty", "blank control"))
```

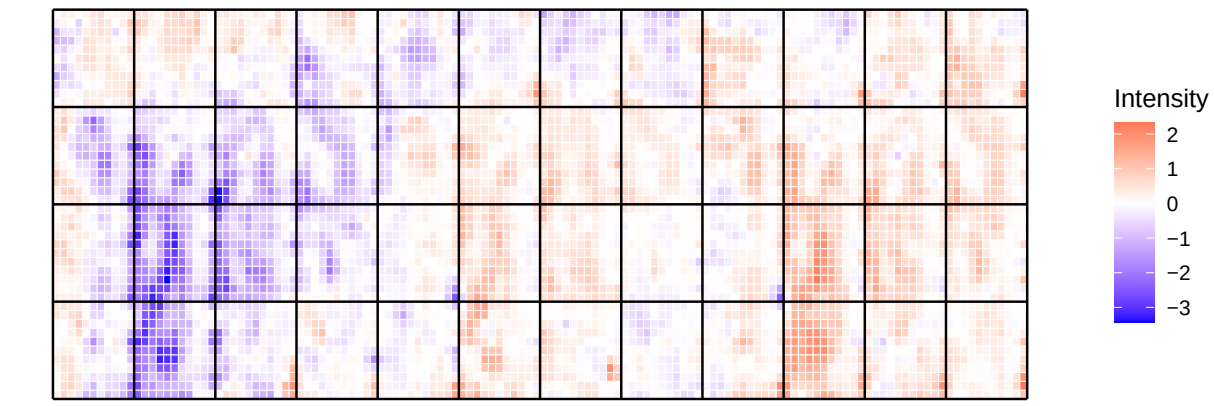
## 2.3 Visualize slides

We include two plotting functions to detect possible spatial slide artifacts. Since the full plate is needed for this visualization, the functions will work best with rm.contol.list and empty.control.list set to NULL in makePeptideSet.

```
plotArrayImage(pSet, array.index = 1)
```




```
plotArrayResiduals(pSet, array.index = 1, smooth = TRUE)
```




## 3 Adding peptide informations

At this point, the peptideSet contain only the peptide sequences and the associated background corrected intensities. To continue with the analysis, we need to add the position information, as well as physicochemical properties of the peptides summarized by their z-scales.

The slides used in this example are the enveloppe of HIV-1 and peptide collections are available for this in our pepDat package (please refere to the vignette and `?pep_hxb2` for more information). However, we will pretend that this is not the case to show an example of how to build a custom peptide collection.

### 3.1 Creating a custom peptide collection

Here, we load a data.frame that contains the peptides used on the array as well as their start and end coordinates, and clade information.

```
peps <- read.csv(system.file("extdata/pep_info.csv", package = "pepDat"))
head(peps)

##   start end         peptide clade
## 1     1  16 MRVKETQMNWPNLWK CRF01
## 2     1  16 MRVMGIQKNYPLLWR CRF02
## 3     1  16 MRVMGIQRNCQHLWR     A
## 4     1  16 MRVKGIRKNYQHLWR     B
## 5     1  16 MRVRGILRNWQQWWI     C
## 6     1  16 MRVRGIERNYQHLWR     D
```

Then we call the constructor that will create the appropriate collection.

```
pep_custom <- create_db(peps)
```

pep_custom is a `GRanges` object with the required "peptide" metadata column and the physiochemical properties of each peptide sequence summarized by z-scores.

Note that the function will also accept `GRanges` input.

```
pep_custom <- create_db(pep_custom)
```

### 3.2 Summarize the information

The function `summarizePeptides` summarizes within-slide replicates by either their mean or median. Additionaly, with the newly constructed peptide collection, peptides positions and annotations can be passed on to the existing peptideSet. Alternately, the function could be callled directly on the `data.frame` object. Internally, `summarizePeptides` will call `create_db` to make sure the input is formatte appropriately.

```
psSet <- summarizePeptides(pSet, summary = "mean", position = pep_custom)

## Some peptides have no match in the GRanges object rownames and are removed from
the peptideSet!
```

Now that all the required information is available, we can proceed with the analysis.

## 4 Normalization

The primary goal of the data normalization step is to remove non-biological source of bias and increase the comparability of true positive signal intensities across slides. The method developped for this package uses physiochemical properties of individual peptides to model non-specific antibody binding to arrays.

```
pnSet <- normalizeArray(psSet)
```

An object of class `peptideSet` containing the corrected peptides intensities is returned.

## 5 Data smoothing

The optional data smoothing step takes advantage of the overlapping nature of the peptides on the array to remove background noise caused by experimental variation. It is likely that two overlapping peptides will share common binding signal, when present. `pepStat` use a sliding mean technique technique to borrow strength across neighboring peptides and to reduce signal variability. This statistic increases detection of binding *hotspots* that noisy signals might otherwise obscure. Peptides are smoothed according to their sequence alignment position, taken from `position(psSet)`.

From here on, two types of analyses are possible. The peptides can be aggregated by position or split by clade. When aggregating by position, the sliding mean will get information from surrounding peptides as well as peptides located around their coordinates in other clades. This increase the strength of calls but the clade specificity is lost.

It is common to do a first run with aggregated clades to detect binding hotspots and then do a second run to look for clade specificity in the peaks found during the first run.

This is decided by the `split.by.clade` argument. By default it is set to TRUE for a clade specific analysis.

```
psmSet <- slidingMean(pnSet, width = 9)
```

For the aggregated `peptideSet` we set it to FALSE.

```
psmSetAg <- slidingMean(pnSet, width = 9, split.by.clade = FALSE)
```

## 6 Making calls

The final step is to make the positivity calls. The function `makeCalls` automatically uses information provided in the mapping file, accessed via `pData(pSet)`. It detects whether samples are paired or not. If samples are paired, POST intensities are subtracted from PRE intensities, then thresholded. Otherwise, PRE samples are averaged, and then subtracted from POST intensities. These corrected POST intensities are thresholded.

The `freq` argument controls whether we return the percentage of responders against each peptide, or a matrix of subject specific call. When `freq` is `TRUE`, we may supply a `group` variable from `pData(psmSet)` on which we split the frequency calculation.

```
calls <- makeCalls(psmSet, freq = TRUE, group = "treatment",
                   cutoff = .1, method = "FDR", verbose = TRUE)

## You have paired PRE/POST samples

## The selected threshold T is  1.100119
```

The function automatically selected an appropriate FDR threshold.

```
callsAg <- makeCalls(psmSetAg, freq = TRUE, group = "treatment",
                     cutoff = .1, method = "FDR")
```

# 7 Results

## 7.1 summary

To get a summary of the analysis, for each peptide, the package provides the function `restab` that combines a `peptideSet` and the result of `makeCalls` into a single `data.frame` with one row per peptide and per clade.

```
summary <- restab(psmSet, calls)
head(summary)
```

```
##                              peptide position start end width clade PLACEBO
## MRVKETQMNWPNLWK_CRF01 MRVKETQMNWPNLWK        8     1  16    16 CRF01       0
## MRVKGIRKNYQHLWR_B     MRVKGIRKNYQHLWR        8     1  16    16     B       0
## MRVMGIQKNYPLLWR_CRF02 MRVMGIQKNYPLLWR        8     1  16    16 CRF02       0
## MRVMGIQRNCQHLWR_A     MRVMGIQRNCQHLWR        8     1  16    16     A       0
## MRVMGIQRNWQHLWR_M     MRVMGIQRNWQHLWR        8     1  16    16     M       0
## MRVRGIERNYQHLWR_D     MRVRGIERNYQHLWR        8     1  16    16     D     100
##                       VACCINE
## MRVKETQMNWPNLWK_CRF01       0
## MRVKGIRKNYQHLWR_B           0
## MRVMGIQKNYPLLWR_CRF02       0
## MRVMGIQRNCQHLWR_A           0
## MRVMGIQRNWQHLWR_M           0
## MRVRGIERNYQHLWR_D           0
```

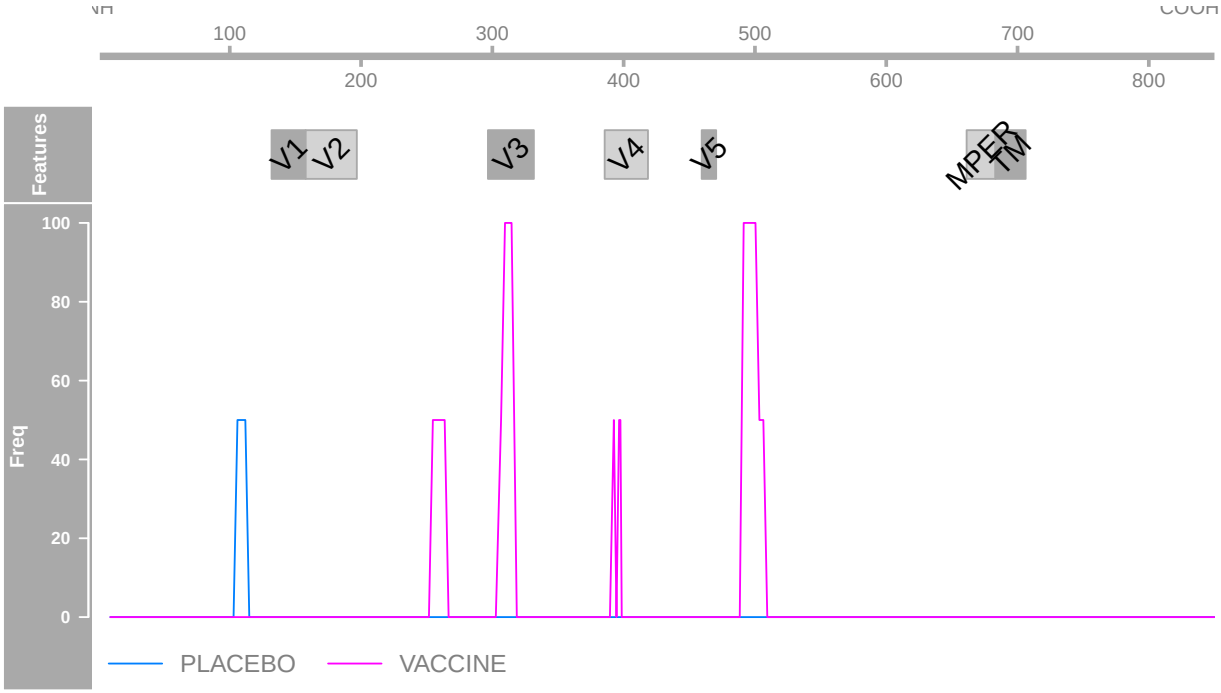
Note that if calls are made with a `peptideSet` that has been normalized with `split.by.clade` set to FALSE, the table will have one row per peptide. Peptides that are identical accross clades will only have one entry.

## 7.2 Plots

As part of the pipeline for the analysis of peptide microarray data, the `Pviz` package includes a track that can use the result of an experiment to generate plots.

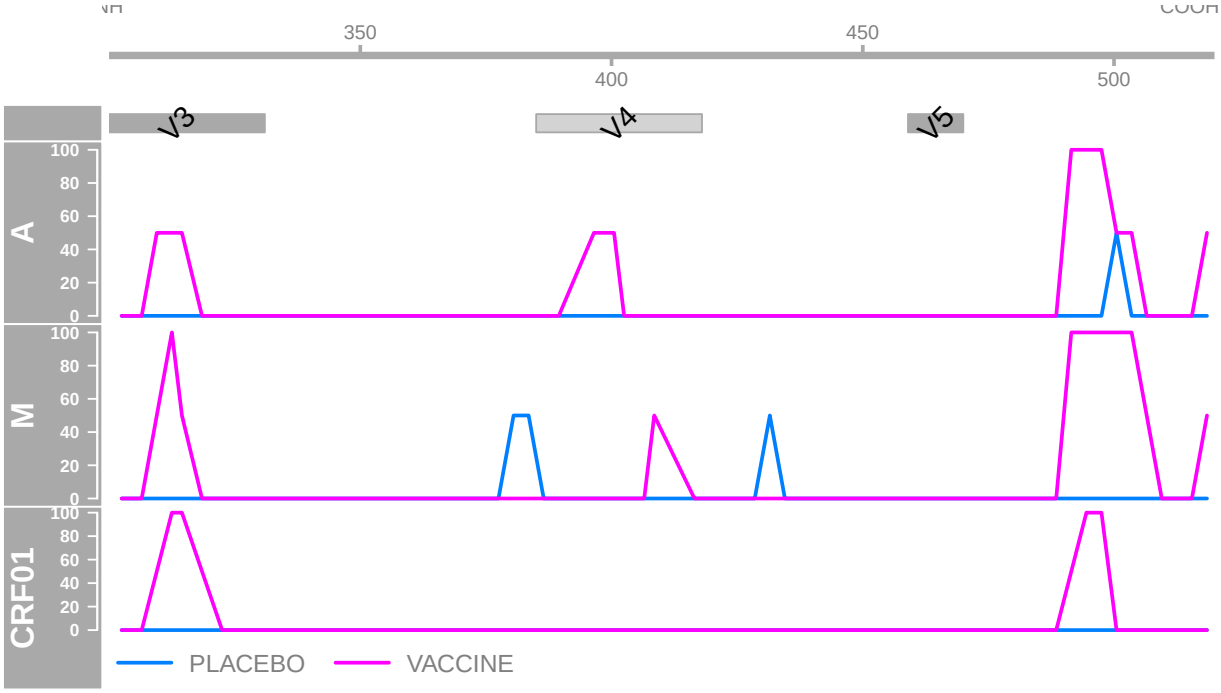
When analysing all clades at once, the `plot_inter` function can be used to easily identify binding peaks. It gives an overview of the differences between the selected groups. In this case, comparing placebo and vaccine.

```
library(Pviz)
summaryAg <- restab(psmSetAg, callsAg)
plot_inter(summaryAg)
```




When clade specific calls have been made, it is more interesting to plot each clade on a separate track.

```
plot_clade(summary, clade=c("A", "M", "CRF01"), from = 300, to = 520)
```



Much more complex plots can be made, custom tracks can be added and every graphical parameter can be tweaked. Refer to the `Pviz` documentation as well as the `Gviz` package for detailed information on all tracks and display paramters.

# 8 shinyApp

As part of the package, a shinyApp provides a user interface for peptide microarray analysis. After making the calls, the results can be downloaded and the app displays plots as shown in the previous sections.

The app can be started from the command line using the `shinyPepStat` function.

```
shinyPepStat()
```

# 9 Quick analysis

Here we showcase a quick analysis of peptide microarray data for HIV-1 gp160. This displays the minimal amount of code required to go from raw data file to antibody binding positivity call.

```
library(pepStat)
library(pepDat)
mapFile <- system.file("extdata/mapping.csv", package = "pepDat")
dirToParse <- system.file("extdata/gpr_samples", package = "pepDat")
ps <- makePeptideSet(files = NULL, path = dirToParse, mapping.file = mapFile)
data(pep_hxb2)
ps <- summarizePeptides(ps, summary = "mean", position = pep_hxb2)
ps <- normalizeArray(ps)
ps <- slidingMean(ps)
calls <- makeCalls(ps, group = "treatment")
summary <- restab(ps, calls)
```

# 10 sessionInfo

```
sessionInfo()

## R version 4.1.1 (2021-08-10)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows Server x64 (build 17763)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=C
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] grid      stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] Pviz_1.28.0          Gviz_1.38.0          GenomicRanges_1.46.0
##  [4] GenomeInfoDb_1.30.0  pepStat_1.28.0       IRanges_2.28.0
##  [7] S4Vectors_0.32.0     Biobase_2.54.0       BiocGenerics_0.40.0
## [10] pepDat_1.13.0        knitr_1.36
##
## loaded via a namespace (and not attached):
##   [1] colorspace_2.0-2           rjson_0.2.20
##   [3] ellipsis_0.3.2             biovizBase_1.42.0
##   [5] htmlTable_2.3.0            XVector_0.34.0
##   [7] base64enc_0.1-3            dichromat_2.0-0
##   [9] rstudioapi_0.13            farver_2.1.0
##  [11] bit64_4.0.5                AnnotationDbi_1.56.0
##  [13] fansi_0.5.0                xml2_1.3.2
##  [15] splines_4.1.1              cachem_1.0.6
##  [17] spam_2.7-0                 Formula_1.2-4
##  [19] Rsamtools_2.10.0           cluster_2.1.2
##  [21] dbplyr_2.1.1               png_0.1-7
##  [23] compiler_4.1.1             httr_1.4.2
##  [25] backports_1.2.1            lazyeval_0.2.2
##  [27] assertthat_0.2.1           Matrix_1.3-4
##  [29] fastmap_1.1.0              limma_3.50.0
##  [31] htmltools_0.5.2            prettyunits_1.1.1
##  [33] tools_4.1.1                dotCall64_1.0-1
```

```
##  [35] gtable_0.3.0                glue_1.4.2
##  [37] GenomeInfoDbData_1.2.7      dplyr_1.0.7
##  [39] maps_3.4.0                  rappdirs_0.3.3
##  [41] Rcpp_1.0.7                  vctrs_0.3.8
##  [43] Biostrings_2.62.0           rtracklayer_1.54.0
##  [45] xfun_0.27                   stringr_1.4.0
##  [47] lifecycle_1.0.1             ensembldb_2.18.0
##  [49] restfulr_0.0.13             XML_3.99-0.8
##  [51] zlibbioc_1.40.0             scales_1.1.1
##  [53] BSgenome_1.62.0             VariantAnnotation_1.40.0
##  [55] ProtGenerics_1.26.0         hms_1.1.1
##  [57] MatrixGenerics_1.6.0        parallel_4.1.1
##  [59] SummarizedExperiment_1.24.0 AnnotationFilter_1.18.0
##  [61] RColorBrewer_1.1-2          fields_12.5
##  [63] yaml_2.2.1                  curl_4.3.2
##  [65] memoise_2.0.0               gridExtra_2.3
##  [67] ggplot2_3.3.5               biomaRt_2.50.0
##  [69] rpart_4.1-15                latticeExtra_0.6-29
##  [71] stringi_1.7.5               RSQLite_2.2.8
##  [73] highr_0.9                   BiocIO_1.4.0
##  [75] checkmate_2.0.0             GenomicFeatures_1.46.0
##  [77] filelock_1.0.2              BiocParallel_1.28.0
##  [79] rlang_0.4.12                pkgconfig_2.0.3
##  [81] matrixStats_0.61.0          bitops_1.0-7
##  [83] evaluate_0.14               lattice_0.20-45
##  [85] purrr_0.3.4                 GenomicAlignments_1.30.0
##  [87] htmlwidgets_1.5.4           labeling_0.4.2
##  [89] bit_4.0.4                   tidyselect_1.1.1
##  [91] plyr_1.8.6                  magrittr_2.0.1
##  [93] R6_2.5.1                    generics_0.1.1
##  [95] Hmisc_4.6-0                 DelayedArray_0.20.0
##  [97] DBI_1.1.1                   pillar_1.6.4
##  [99] foreign_0.8-81              survival_3.2-13
## [101] KEGGREST_1.34.0             RCurl_1.98-1.5
## [103] nnet_7.3-16                 tibble_3.1.5
## [105] crayon_1.4.1                utf8_1.2.2
## [107] BiocFileCache_2.2.0         viridis_0.6.2
## [109] jpeg_0.1-9                  progress_1.2.2
## [111] data.table_1.14.2           blob_1.2.2
## [113] digest_0.6.28               munsell_0.5.0
## [115] viridisLite_0.4.0
```