Using spatialLIBD with 10x Genomics public datasets

Abby Spangler [*1], **Brenda Pardo** [†1,2], **and Leonardo Collado-Torres** [‡1,3]

[1]Lieber Institute for Brain Development, Johns Hopkins Medical Campus
[2]Licenciatura de Ciencias Genómicas, Escuela Nacional de Estudios Superiores Unidad Juriquilla, Universidad Nacional Autónoma de México
[3]Center for Computational Biology, Johns Hopkins University

[*]aspangle@gmail.com [†]bpardo@lcgej.unam.mx [‡]lcolladotor@gmail.com

30 March 2022

Package

spatialLIBD 1.7.14

Contents

1 Basics

1.1 Install `spatialLIBD`

`R` is an open-source statistical environment which can be easily modified to enhance its functionality via packages. *spatialLIBD* (Pardo, Spangler, Weber, Hicks, Jaffe, Martinowich, Maynard, and Collado-Torres, 2021) is an `R` package available via the Bioconductor repository for packages. `R` can be installed on any operating system from CRAN after which you can install *spatialLIBD* by using the following commands in your `R` session:

```r
if (!requireNamespace("BiocManager", quietly = TRUE)) {
    install.packages("BiocManager")
}

BiocManager::install("spatialLIBD")

## Check that you have a valid Bioconductor installation
BiocManager::valid()
```

To run all the code in this vignette, you might need to install other R/Bioconductor packages, which you can do with:

```r
BiocManager::install("spatialLIBD", dependencies = TRUE, force = TRUE)
```

If you want to use the development version of `spatialLIBD`, you will need to use the R version corresponding to the current Bioconductor-devel branch as described in more detail on the Bioconductor website. Then you can install `spatialLIBD` from GitHub using the following command.

```r
BiocManager::install("LieberInstitute/spatialLIBD")
```

1.2 Required knowledge

Please first check the *Introduction to spatialLIBD* vignette available through GitHub or Bioconductor.

1.3 Citing `spatialLIBD`

We hope that *spatialLIBD* will be useful for your research. Please use the following information to cite the package and the overall approach. Thank you!

```r
## Citation info
citation("spatialLIBD")
#>
#> Pardo B, Spangler A, Weber LM, Hicks SC, Jaffe AE, Martinowich K,
#> Maynard KR, Collado-Torres L (2021). "spatialLIBD: an R/Bioconductor
#> package to visualize spatially-resolved transcriptomics data."
#> _bioRxiv_. doi: 10.1101/2021.04.29.440149 (URL:
#> https://doi.org/10.1101/2021.04.29.440149), <URL:
#> https://www.biorxiv.org/content/10.1101/2021.04.29.440149v1>.
#>
```

```
#> Maynard KR, Collado-Torres L, Weber LM, Uytingco C, Barry BK, Williams
#> SR, II JLC, Tran MN, Besich Z, Tippani M, Chew J, Yin Y, Kleinman JE,
#> Hyde TM, Rao N, Hicks SC, Martinowich K, Jaffe AE (2021).
#> "Transcriptome-scale spatial gene expression in the human dorsolateral
#> prefrontal cortex." _Nature Neuroscience_. doi:
#> 10.1038/s41593-020-00787-0 (URL:
#> https://doi.org/10.1038/s41593-020-00787-0), <URL:
#> https://www.nature.com/articles/s41593-020-00787-0>.
#>
#> To see these entries in BibTeX format, use 'print(<citation>,
#> bibtex=TRUE)', 'toBibtex(.)', or set
#> 'options(citation.bibtex.max=999)'.
```

2 Download data from 10x Genomics

In this vignette we'll show you how you can use *spatialLIBD* (Pardo, Spangler, Weber, et al., 2021) for exploring spatially resolved transcriptomics data from the Visium platform by 10x Genomics. That is, you will learn how to use *spatialLIBD* for data beyond the one it was initially developed for (Maynard, Collado-Torres, Weber, Uytingco, Barry, Williams, II, Tran, Besich, Tippani, Chew, Yin, Kleinman, Hyde, Rao, Hicks, Martinowich, and Jaffe, 2021). To illustrate these steps, we will use data that 10x Genomics made publicly available at https://support.10xgenomics.com/spatial-gene-expression/datasets. We will use files from the human lymph node example publicly available at https://support.10xgenomics.com/spatial-gene-expression/datasets/1.1.0/V1_Human_Lymph_Node.

2.1 Load packages

To get started, lets load the different packages we'll need for this vignette. Here's a brief summary of why we need these packages:

- *BiocFileCache*: for downloading and saving a local cache of the data
- *SpatialExperiment*: for reading the `spaceranger` files provided by 10x Genomics
- *rtracklayer*: for importing a gene annotation GTF file
- *lobstr*: for checking how much memory our object is using
- *spatialLIBD*: for launching an interactive website to explore the data

```
## Load packages in the order we'll use them next
library("BiocFileCache")
library("SpatialExperiment")
library("rtracklayer")
library("lobstr")
library("spatialLIBD")
```

2.2 Download spaceranger output files

Next we download data from 10x Genomics available from the human lymph node example, available at https://support.10xgenomics.com/spatial-gene-expression/datasets/1.1.0/V1_Human_Lymph_Node. We don't need to download all the files listed there since *Spa-*

tialExperiment doesn't need all of them for importing the data into R. These files are part of the output that gets generated by `spaceranger` which is the processing pipeline provided by 10x Genomics for Visium data.

We'll use *BiocFileCache* to keep the data in a local cache in case we want to run this example again and don't want to re-download the data from the web.

```r
## Download and save a local cache of the data provided by 10x Genomics
bfc <- BiocFileCache::BiocFileCache()
lymph.url <-
    paste0(
        "https://cf.10xgenomics.com/samples/spatial-exp/",
        "1.1.0/V1_Human_Lymph_Node/",
        c(
            "V1_Human_Lymph_Node_filtered_feature_bc_matrix.tar.gz",
            "V1_Human_Lymph_Node_spatial.tar.gz",
            "V1_Human_Lymph_Node_analysis.tar.gz"
        )
    )
lymph.data <- sapply(lymph.url, BiocFileCache::bfcrpath, x = bfc)
```

10x Genomics provides the files in compressed tarballs (`.tar.gz` file extension). Which is why we'll need to use `utils::untar()` to decompress the files. This will create new directories and we will use `list.files()` to see what files these directories contain.

```r
## Extract the files to a temporary location
## (they'll be deleted once you close your R session)
xx <- sapply(lymph.data, utils::untar, exdir = file.path(tempdir(), "outs"))
## The names are the URLs, which are long and thus too wide to be shown here,
## so we shorten them to only show the file name prior to displaying the
## utils::untar() output status
names(xx) <- basename(names(xx))
xx
#> V1_Human_Lymph_Node_filtered_feature_bc_matrix.tar.gz.BFC2
#>                                                         0
#>              V1_Human_Lymph_Node_spatial.tar.gz.BFC3
#>                                                         0
#>              V1_Human_Lymph_Node_analysis.tar.gz.BFC4
#>                                                         0

## List the files we downloaded and extracted
## These files are typically SpaceRanger outputs
lymph.dirs <- file.path(
    tempdir(), "outs",
    c("filtered_feature_bc_matrix", "spatial", "raw_feature_bc_matrix", "analysis")
)
list.files(lymph.dirs)
#>  [1] "aligned_fiducials.jpg"    "barcodes.tsv.gz"
#>  [3] "clustering"               "detected_tissue_image.jpg"
#>  [5] "diffexp"                  "features.tsv.gz"
#>  [7] "matrix.mtx.gz"            "pca"
#>  [9] "scalefactors_json.json"   "tissue_hires_image.png"
#> [11] "tissue_lowres_image.png"  "tissue_positions_list.csv"
```

```
#> [13] "tsne"                    "umap"
```

Now that we have the files that we need, we can import the data into R using `read10xVisium()` from *SpatialExperiment*. We'll import the low resolution histology images produced by `spaceranger` using the `images = "lowres"` and `load = TRUE` arguments. We'll also load the filtered gene expression data using the `data = "filtered"` argument. The count matrix can still be quite large, which is why we'll use the `type = "sparse"` argument to load the data into an R object that is memory-efficient for sparse data.

```r
## Import the data as a SpatialExperiment object
spe <- SpatialExperiment::read10xVisium(
    samples = tempdir(),
    sample_id = "lymph",
    type = "sparse", data = "filtered",
    images = "lowres", load = TRUE
)
## Inspect the R object we just created: class, memory, and how it looks in
## general
class(spe)
#> [1] "SpatialExperiment"
#> attr(,"package")
#> [1] "SpatialExperiment"
lobstr::obj_size(spe) / 1024^2 ## Convert to MB
#> 281.9031 B
spe
#> class: SpatialExperiment
#> dim: 36601 4035
#> metadata(0):
#> assays(1): counts
#> rownames(36601): ENSG00000243485 ENSG00000237613 ... ENSG00000278817
#>   ENSG00000277196
#> rowData names(1): symbol
#> colnames(4035): AAACAAGTATCTCCCA-1 AAACAATCTACTAGCA-1 ...
#>   TTGTTTGTATTACACG-1 TTGTTTGTGTAAATTC-1
#> colData names(4): in_tissue array_row array_col sample_id
#> reducedDimNames(0):
#> mainExpName: NULL
#> altExpNames(0):
#> spatialCoords names(2) : pxl_col_in_fullres pxl_row_in_fullres
#> imgData names(4): sample_id image_id data scaleFactor

## The counts are saved in a sparse matrix R object
class(counts(spe))
#> [1] "dgCMatrix"
#> attr(,"package")
#> [1] "Matrix"
```

3 Modify spe for spatialLIBD

Now that we have an `SpatialExperiment` R object (`spe`) with the data from 10x Genomics for the human lymph node example, we need to add a few features to the R object in order to create the interactive website using `spatialLIBD::run_app()`. These additional elements power features in the interactive website that you might be interested in.

First we start with adding a few variables to the sample information table (`colData()`) of our `spe` object. We add:

- `key`: this labels each spot with a unique identifier. We combine the sample ID with the spot barcode ID to create this unique identifier.
- `sum_umi`: this continuous variable contains the total number of counts for each sample prior to filtering any genes.
- `sum_gene`: this continuous variable contains the number of genes that have at least 1 count.

```
## Add some information used by spatialLIBD
spe <- add_key(spe)
spe$sum_umi <- colSums(counts(spe))
spe$sum_gene <- colSums(counts(spe) > 0)
```

3.1 Add gene annotation information

The files `SpatialExperiment::read10xVisium()` uses to read in the `spaceranger` outputs into R do not include much information about the genes, such as their chromosomes, coordinates, and other gene annotation information. We thus recommend that you read in this information from a gene annotation file: typically a `gtf` file. For a real case scenario, you'll mostly likely have access to the GTF file provided by 10x Genomics. However, we cannot download that file without downloading other files for this example. Thus we'll show you the code you would use if you had access to the GTF file from 10x Genomics and also show a second approach that works for this vignette.

```
## Initially we don't have much information about the genes
rowRanges(spe)
#> GRangesList object of length 36601:
#> $ENSG00000243485
#> GRanges object with 0 ranges and 0 metadata columns:
#>    seqnames    ranges strand
#>       <Rle> <IRanges>  <Rle>
#>    -------
#>    seqinfo: no sequences
#>
#> $ENSG00000237613
#> GRanges object with 0 ranges and 0 metadata columns:
#>    seqnames    ranges strand
#>       <Rle> <IRanges>  <Rle>
#>    -------
#>    seqinfo: no sequences
#>
#> $ENSG00000186092
#> GRanges object with 0 ranges and 0 metadata columns:
```

```
#>    seqnames    ranges strand
#>      <Rle> <IRanges>  <Rle>
#>   -------
#>   seqinfo: no sequences
#>
#> ...
#> <36598 more elements>
```

3.1.1   From 10x

Depending on the version of `spaceranger` you used, you might have used differ-
ent GTF files 10x Genomics has made available at https://support.10xgenomics.
com/single-cell-gene-expression/software/downloads/latest and described at https:
//support.10xgenomics.com/single-cell-gene-expression/software/release-notes/build.
These files are too big though and we won't download them in this example. For instance,
References - 2020-A (July 7, 2020) for *Human reference (GRCh38)* is 11 GB in size and
contains files we do not need for this vignette. If you did have the file locally, you could use
the following code to read in the GTF file prepared by 10x Genomics and add the information
into your `spe` object that `SpatialExperiment::read10xVisium()` does not include.

For example, in our computing cluster this GTF file is located at the following path and is
1.4 GB in size:

```
$ cd /dcs04/lieber/lcolladotor/annotationFiles_LIBD001/10x/refdata-gex-GRCh38-2020-A
$ du -sh --apparent-size genes/genes.gtf
1.4G    genes/genes.gtf
```

If you have the GTF file from 10x Genomics, we show next how you can read the information
into R, match it appropriately with the information in the `spe` object and add it back into
the `spe` object.

```
## You could:
## * download the 11 GB file from
## https://cf.10xgenomics.com/supp/cell-exp/refdata-gex-GRCh38-2020-A.tar.gz
## * decompress it

## Read in the gene information from the annotation GTF file provided by 10x
gtf <-
    rtracklayer::import(
        "/path/to/refdata-gex-GRCh38-2020-A/genes/genes.gtf"
    )

## Subject to genes only
gtf <- gtf[gtf$type == "gene"]

## Set the names to be the gene IDs
names(gtf) <- gtf$gene_id

## Match the genes
match_genes <- match(rownames(spe), gtf$gene_id)

## They should all be present if you are using the correct GTF file from 10x
```

```
stopifnot(all(!is.na(match_genes)))

## Keep only some columns from the gtf (you could keep all of them if you want)
mcols(gtf) <-
    mcols(gtf)[, c(
        "source",
        "type",
        "gene_id",
        "gene_version",
        "gene_name",
        "gene_type"
    )]

## Add the gene info to our SPE object
rowRanges(spe) <- gtf[match_genes]

## Inspect the gene annotation data we added
rowRanges(spe)
```

3.1.2   From Gencode

In this vignette, we'll use the GTF file from Gencode v32. That's because the build notes from *References - 2020-A (July 7, 2020)* and *Human reference, GRCh38 (GEN-CODE v32/Ensembl 98)* at https://support.10xgenomics.com/single-cell-gene-expression/software/release-notes/build#GRCh38_2020A show that 10x Genomics used Gencode v32. They also used Ensembl version 98 which is why a few genes we have in our object are going to be missing. We show next how you can read the information into R, match it appropriately with the information in the `spe` object and add it back into the `spe` object.

```
## Download the Gencode v32 GTF file and cache it
gtf_cache <- BiocFileCache::bfcrpath(
    bfc,
    paste0(
        "ftp://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/",
        "release_32/gencode.v32.annotation.gtf.gz"
    )
)

## Show the GTF cache location
gtf_cache
#>                                                                                      L
#> "/Users/leocollado/Library/Caches/org.R-project.R/R/BiocFileCache/425b635f3f8e_gencode.v32.annotation.gtf.

## Import into R (takes ~1 min)
gtf <- rtracklayer::import(gtf_cache)

## Subset to genes only
gtf <- gtf[gtf$type == "gene"]

## Remove the .x part of the gene IDs
gtf$gene_id <- gsub("\\..*", "", gtf$gene_id)
```

```
## Set the names to be the gene IDs
names(gtf) <- gtf$gene_id

## Match the genes
match_genes <- match(rownames(spe), gtf$gene_id)
table(is.na(match_genes))
#>
#> FALSE   TRUE
#> 36572     29

## Drop the few genes for which we don't have information
spe <- spe[!is.na(match_genes), ]
match_genes <- match_genes[!is.na(match_genes)]

## Keep only some columns from the gtf
mcols(gtf) <- mcols(gtf)[, c("source", "type", "gene_id", "gene_name", "gene_type")]

## Add the gene info to our SPE object
rowRanges(spe) <- gtf[match_genes]

## Inspect the gene annotation data we added
rowRanges(spe)
#> GRanges object with 36572 ranges and 5 metadata columns:
#>                   seqnames          ranges strand |    source      type
#>                      <Rle>       <IRanges>  <Rle> | <factor> <factor>
#>   ENSG00000243485     chr1   29554-31109      + |    HAVANA     gene
#>   ENSG00000237613     chr1   34554-36081      - |    HAVANA     gene
#>   ENSG00000186092     chr1   65419-71585      + |    HAVANA     gene
#>   ENSG00000238009     chr1 89295-133723      - |    HAVANA     gene
#>   ENSG00000239945     chr1   89551-91105      - |    HAVANA     gene
#>               ...      ...             ...    ... .      ...      ...
#>   ENSG00000212907     chrM   10470-10766      + |   ENSEMBL     gene
#>   ENSG00000198886     chrM   10760-12137      + |   ENSEMBL     gene
#>   ENSG00000198786     chrM   12337-14148      + |   ENSEMBL     gene
#>   ENSG00000198695     chrM   14149-14673      - |   ENSEMBL     gene
#>   ENSG00000198727     chrM   14747-15887      + |   ENSEMBL     gene
#>                       gene_id   gene_name      gene_type
#>                     <character> <character>    <character>
#>   ENSG00000243485 ENSG00000243485  MIR1302-2HG         lncRNA
#>   ENSG00000237613 ENSG00000237613      FAM138A         lncRNA
#>   ENSG00000186092 ENSG00000186092        OR4F5 protein_coding
#>   ENSG00000238009 ENSG00000238009   AL627309.1         lncRNA
#>   ENSG00000239945 ENSG00000239945   AL627309.3         lncRNA
#>               ...             ...          ...            ...
#>   ENSG00000212907 ENSG00000212907     MT-ND4L protein_coding
#>   ENSG00000198886 ENSG00000198886      MT-ND4 protein_coding
#>   ENSG00000198786 ENSG00000198786      MT-ND5 protein_coding
#>   ENSG00000198695 ENSG00000198695      MT-ND6 protein_coding
#>   ENSG00000198727 ENSG00000198727      MT-CYB protein_coding
#>   -------
```

```
#>   seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

3.1.3 Enable a friendlier gene search

Regardless of which method you used to obtain the gene annotation information, we can now proceed by adding the gene symbol and gene ID information that helps users search for genes in the shiny app produced by `spatialLIBD`. This will enable users to search genes by gene symbol or gene ID. If you didn't do this, users would only be able to search genes by gene ID which makes the web application harder to use.

We also compute the total expression for the mitochondrial chromosome (chrM) as well as the ratio of chrM expression. Both of these continuous variables are interesting to explore and in some situations could be useful for biological interpretations. For instance, in our pilot data (Maynard, Collado-Torres, Weber, et al., 2021), we noticed that the `expr_chrM_ratio` was associated to DLPFC layers. That is, spots with high `expr_chrM_ratio` were not randomly located in our Visium slides.

```r
## Add information used by spatialLIBD
rowData(spe)$gene_search <- paste0(
    rowData(spe)$gene_name, "; ", rowData(spe)$gene_id
)

## Compute chrM expression and chrM expression ratio
is_mito <- which(seqnames(spe) == "chrM")
spe$expr_chrM <- colSums(counts(spe)[is_mito, , drop = FALSE])
spe$expr_chrM_ratio <- spe$expr_chrM / spe$sum_umi
```

3.2 Filter the spe object

We can now continue with some filtering steps since this can help reduce the object size in memory as well as make it ready to use for downstream processing tools such as those from the *scran* and *scuttle* packages. Though these steps are not absolutely necessary.

```r
## Remove genes with no data
no_expr <- which(rowSums(counts(spe)) == 0)

## Number of genes with no counts
length(no_expr)
#> [1] 11397

## Compute the percent of genes with no counts
length(no_expr) / nrow(spe) * 100
#> [1] 31.16318
spe <- spe[-no_expr, , drop = FALSE]

## Remove spots without counts
summary(spe$sum_umi)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>      23   15917   20239   20738   25252   54931

## If we had spots with no counts, we would remove them
```

```
if (any(spe$sum_umi == 0)) {
    spots_no_counts <- which(spe$sum_umi == 0)
    ## Number of spots with no counts
    print(length(spots_no_counts))
    ## Percent of spots with no counts
    print(length(spots_no_counts) / ncol(spe) * 100)
    spe <- spe[, -spots_no_counts, drop = FALSE]
}
```

3.3   Check object

Next, we add the `ManualAnnotation` variable to the sample information table (`colData()`) with `"NA"`. That variable is used by the interactive website to store any manual annotations.

```
## Add a variable for saving the manual annotations
spe$ManualAnnotation <- "NA"
```

Finally, we can now check the final object using `spatialLIBD::check_spe()`. This is a helper function that will warn us if some important element is missing in `spe` that we use later for the interactive website. If it all goes well, it will return the original `spe` object.

```
## Check the final dimensions and object size
dim(spe)
#> [1] 25175  4035
lobstr::obj_size(spe) / 1024^2 ## Convert to MB
#> 283.9554 B

## Run check_spe() function
check_spe(spe)
#> class: SpatialExperiment
#> dim: 25175 4035
#> metadata(0):
#> assays(1): counts
#> rownames(25175): ENSG00000238009 ENSG00000241860 ... ENSG00000198695
#>   ENSG00000198727
#> rowData names(6): source type ... gene_type gene_search
#> colnames(4035): AAACAAGTATCTCCCA-1 AAACAATCTACTAGCA-1 ...
#>   TTGTTTGTATTACACG-1 TTGTTTGTGTAAATTC-1
#> colData names(10): in_tissue array_row ... expr_chrM_ratio
#>   ManualAnnotation
#> reducedDimNames(0):
#> mainExpName: NULL
#> altExpNames(0):
#> spatialCoords names(2) : pxl_col_in_fullres pxl_row_in_fullres
#> imgData names(4): sample_id image_id data scaleFactor
```

4   Explore the data

With our complete `spe` object, we can now use *spatialLIBD* for visualizing our data. We can do so using functions such as `vis_gene()` and `vis_clus()` that are described in more detail at the *Introduction to spatialLIBD* vignette available through GitHub or Bioconductor.

```
## Example visualizations. Let's start with a continuous variable.
spatialLIBD::vis_gene(
    spe = spe,
    sampleid = "lymph",
    geneid = "sum_umi",
    assayname = "counts"
)
```
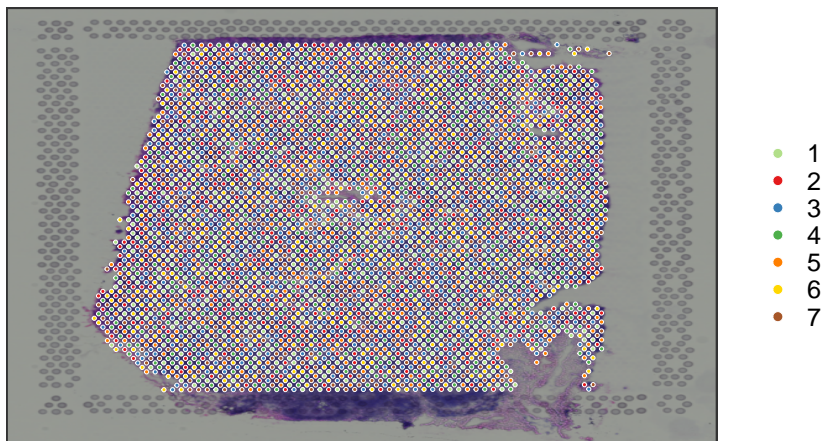
lymph sum_umi
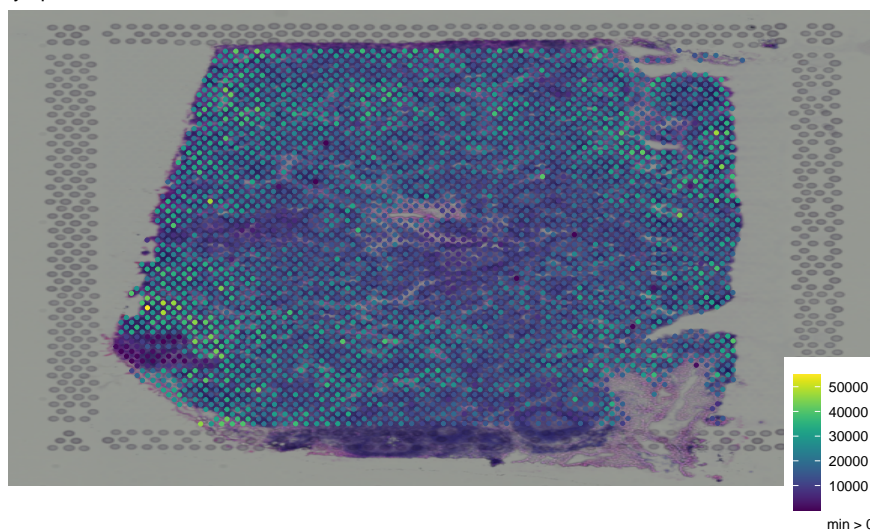

```
## We next create a random cluster label to visualize
set.seed(20210428)
spe$random_cluster <- sample(1:7, ncol(spe), replace = TRUE)

## Next we visualize that random cluster
spatialLIBD::vis_clus(
    spe = spe,
    sampleid = "lymph",
    clustervar = "random_cluster"
)
```

lymph

4.1 Run the interactive website

We are now ready to create our interactive website for the human lymph node data. The interactive website is a *shiny* web application that uses *plotly* to power several of the interactive features. We can create the interactive website using the `spatialLIBD::run_app()` function. The default arguments of that function are customized for the data from our initial study (Maynard, Collado-Torres, Weber, et al., 2021), so we will need to make some adjustments:

- `sce_layer`, `modeling_results` and `sig_genes` will be set to `NULL` since do not have any pseudo-bulk results for this example data.
- `title`: we will use a custom title that reflect our data
- `spe_discreate_vars`: we don't have really any discrete variables to show beyond `ManualAnnotation` which is used for the manual annotations and `random_cluster` that we created in the previous section.
- `spe_continous_vars`: we have computed several continuous variables while adapting our `spe` object for *spatialLIBD*, so we'll list these variables below in order to visually inspect them.
- `default_cluster`: this is used for indicating the default discrete variable and for now we'll set it to our `random_cluster`.

```r
## Run our shiny app
if (interactive()) {
    run_app(
        spe,
        sce_layer = NULL,
        modeling_results = NULL,
        sig_genes = NULL,
        title = "spatialLIBD: human lymph node by 10x Genomics",
        spe_discrete_vars = c("random_cluster", "ManualAnnotation"),
        spe_continuous_vars = c("sum_umi", "sum_gene", "expr_chrM", "expr_chrM_ratio"),
        default_cluster = "random_cluster"
    )
}
```

We also recommend creating custom website documentation files as described in the documentation of `spatialLIBD::run_app()`. Those documentation files will help you describe your project to your users in a more personalized way. The easiest way to start is to copy our documentation files to a new location and adapt them. You can locate them at the following path.

```
## Locate our documentation files
docs_path <- system.file("app", "www", package = "spatialLIBD")
docs_path
#> [1] "/Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library/spatialLIBD/app/www"
list.files(docs_path)
#> [1] "documentation_sce_layer.md" "documentation_spe.md"
#> [3] "favicon.ico"                "footer.html"
#> [5] "README.md"
```

5   Wrapper functions

To facilitate reading in the data and preparing it to visualize it interactively using `spatialLIBD::run_app()`, we implemented `read10xVisiumWrapper()` which expands `SpatialExperiment::read10xVisium()` and performs the steps described in this vignette. In this example, we'll load all four images created by SpaceRanger: lowres, hires, detected, and aligned. That way we can toggle between them on the web application.

```
## Import the data as a SpatialExperiment object
spe_wrapper <- read10xVisiumWrapper(
    samples = file.path(tempdir(), "outs"),
    sample_id = "lymph",
    type = "sparse", data = "filtered",
    images = c("lowres", "hires", "detected", "aligned"), load = TRUE,
    reference_gtf = gtf_cache
)
#> 2022-03-30 18:13:02 SpatialExperiment::read10xVisium: reading basic data from SpaceRanger
#> 2022-03-30 18:13:09 read10xVisiumAnalysis: reading analysis output from SpaceRanger
#> 2022-03-30 18:13:10 add10xVisiumAnalysis: adding analysis output from SpaceRanger
#> 2022-03-30 18:13:10 rtracklayer::import: reading the reference GTF file
#> 2022-03-30 18:13:30 adding gene information to the SPE object
#> Warning: Gene IDs did not match. This typically happens when you are not using
#> the same GTF file as the one that was used by SpaceRanger. For example, one file
#> uses GENCODE IDs and the other one ENSEMBL IDs. read10xVisiumWrapper() will try
#> to convert them to ENSEMBL IDs.
#> Warning: Dropping 29 out of 36601 genes for which we don't have information on
#> the reference GTF file. This typically happens when you are not using the same
#> GTF file as the one that was used by SpaceRanger.
#> 2022-03-30 18:13:30 adding information used by spatialLIBD
```

```
## Run our shiny app
if (interactive()) {
    vars <- colnames(colData(spe_wrapper))

    run_app(
```

```
        spe_wrapper,
        sce_layer = NULL,
        modeling_results = NULL,
        sig_genes = NULL,
        title = "spatialLIBD: human lymph node by 10x Genomics (made with wrapper)",
        spe_discrete_vars = c(vars[grep("10x_", vars)], "ManualAnnotation"),
        spe_continuous_vars = c("sum_umi", "sum_gene", "expr_chrM", "expr_chrM_ratio"),
        default_cluster = "10x_graphclust"
    )
}
```

6   Publishing your web application

Now we have have our `spe_wrapper` object and have verified that the web application works, we can share it with others through shinyapps.io by RStudio if our data is small enough. To do, you will need to:

- create a directory `app_dir` which will contain the script for loading the data and running the web application
- save the `spe_wrapper` object into that directory such as `app_dir/spe_wrapper.rds`
- create an `app.R` script and save it at `app_dir/app.R`
- optionally copy the contents from `system.file("app", "www", package = "spatialLIBD")` into `app_dir/www`
- authenticate to shinyapps.io and deploy the app, which we usually do with a `app_dir/deploy.R` script

Below we illustrate these steps

```
## Directory we created to host the data for the web application
## Use a directory of your preference instead of copy-pasting this code
app_dir <- here::here("inst", "spe_wrapper_app")
dir.create(app_dir, showWarnings = FALSE)

## Code we used to save the data
saveRDS(spe_wrapper, file = file.path(app_dir, "spe_wrapper.rds"))

## Copy the contents of system.file("app", "www", package = "spatialLIBD")
file.copy(system.file("app", "www", package = "spatialLIBD"), app_dir, recursive = TRUE)
## Manually edit them to your liking.
```

Next, create an `app_dir/app.R` script with these contents.

```
#> library("spatialLIBD")
#>  library("markdown") ## due to a shinyapps.io bug
#>
#>  ## spatialLIBD uses golem
#>  options("golem.app.prod" = TRUE)
#>
#>  ## You need this to enable shinyapps to install Bioconductor packages
#>  options(repos = BiocManager::repositories())
#>
```

```
#>  ## Load the data (all paths are relative to this script's location)
#>  spe_wrapper <- readRDS("spe_wrapper.rds")
#>  vars <- colnames(colData(spe_wrapper))
#>
#>  ## Deploy the website
#>  run_app(
#>      spe_wrapper,
#>      sce_layer = NULL,
#>      modeling_results = NULL,
#>      sig_genes = NULL,
#>      title = "spatialLIBD: human lymph node by 10x Genomics",
#>      spe_discrete_vars = c(vars[grep("10x_", vars)], "ManualAnnotation"),
#>      spe_continuous_vars = c("sum_umi", "sum_gene", "expr_chrM", "expr_chrM_ratio"),
#>      default_cluster = "10x_graphclust",
#>      docs_path = "www"
#>  )
```

Next create an `app_dir/deploy.R` script with these contents. You will need to login to your shinyapps.io account to get the exact authentication details for your access token.

```
#> library("rsconnect")
#>
#>  ## Locate app_dir. Edit as needed
#>  app_dir <- here::here("inst", "spe_wrapper_app")
#>
#>  ## Or you can go to your shinyapps.io account and copy this
#>  ## Here we do this to keep our information hidden.
#>  load(file.path(app_dir, ".deploy_info.Rdata"), verbose = TRUE)
#>
#>  ## Authenticate to shinyapps.io
#>  rsconnect::setAccountInfo(
#>      name = deploy_info$name,
#>      token = deploy_info$token,
#>      secret = deploy_info$secret
#>  )
#>
#>  ## You need this to enable shinyapps to install Bioconductor packages
#>  options(repos = BiocManager::repositories())
#>
#>  ## Deploy the app, that is, upload it to shinyapps.io
#>  ## Note that appFiles has to be relative to app_dir.
#>  ## Drop the www directory if you didn't customize the documentation files and
#>  ## edit app.R accordingly.
#>  rsconnect::deployApp(
#>      appDir = app_dir,
#>      appFiles = c(
#>          "app.R",
#>          "spe_wrapper.rds",
#>          gsub(file.path(app_dir, "www"), "www", dir(file.path(app_dir, "www"), full.names = TRUE))
#>      ),
#>      appName = "spatialLIBD_Human_Lymph_Node_10x",
```

```
#>      account = "libd",
#>      server = "shinyapps.io"
#>  )
```

Next, you should test that `app_dir/app.R` works locally before running `app_dir/deploy.R`.

You'll end up with a website just like this one. In our case, we further configured our website through the shinyapps.io dashboard. We selected the following options:

- *General* `Instance Size`: 3X-Large (8GB)
- *Advanced* `Max Worker Processes`: 1. Setting this to one helps reduce the memory burden.
- *Advanced* `Max Connections`: 15. You don't want `Max Worker Processes` multiplied by `Max Connections` and the object memory to exceed the `Instance Size`. So this this value to `8 / (lobstr::obj_size(spe_wrapper) / 1024^3)` or lower.
- *Advanced* `Startup Timeout`: 300. This one is important otherwise your app might error out before even loading since the default is 5 seconds, which is typically too short.
- *Advanced* `Idle Timeout`: 60. This is useful since you might want to give users as much time for their manual annotations as possible.

7 Limitations

`spatialLIBD::run_app()` has limitations that are inherent to the methods used to implement it, such as:

1. the memory per user required by a server for hosting the web application,
2. response speeds for the interactive views due to the number of spots,
3. the resolution of the images displayed limiting the usefulness to magnify specific spots,
4. and customization of the web application by the end user.

7.1 Memory

Regarding the memory limitation, you can estimate how much memory you need per user by considering the memory required for the `spe` and `sce_layer` objects.

```
lobstr::obj_size(spe) / 1024^2 ## Convert to MB
#> 283.9709 B
```

In our pilot data (Maynard, Collado-Torres, Weber, et al., 2021) our object uses about 2.1 GB of RAM since it contains the data for 12 Visium slides and we considered using about 3 GB of RAM per user. You could filter the genes more aggressively to drop lowly expressed genes or if you have many Visium slides, you could consider making multiple websites for different sets of slides. You could also have multiple mirrors to support several users, though in that case, we recommend linking users to a stable website instead of one that might not be available if you have too many users: for us our stable website is http://research.libd.org/spatialLIBD/ which includes the links to all the mirrors.

Given these memory limitations, we chose to deploy our main web application at http://spatial.libd.org/spatialLIBD/ using an Amazon EC2 instance: an 'r5.4xlarge' EC2 instance with 16 vCPUs, 128 GB DRAM, 10 Gb max network, 1.008 USD/Hour. We also have deployed mirrors at https://www.shinyapps.io/ using the "3X-Large (8 GB)" instances they provide.

7.2 Response speeds

This limitation is mostly due to the number of spots shown under the "clusters (interactive)" section of the interactive website powered by *plotly*. Each spot is shown four times which is about 16 thousand spots for one Visium slide (depending on any filter steps you applied). The response time will depend on your own computer RAM memory, that is, the *client* side. This limitation might be more noticeable if you have a computer with 8GB of RAM or lower, as well as if you have other high-memory software open. Furthermore, if you are running web application locally through `spatialLIBD::run_app()` then you also need to consider the required memory for the R objects. That is, the *server* side memory use.

Thanks to Jesús Vélez Santiago, the app is more responsive as of version 1.3.15 by using `plotly::toWebGL()`.

7.3 Image resolution

When you construct the SpatialExperiment `spe` object with *SpatialExperiment*, you can read in higher resolution images. However, the benefit of loading the raw histology images (500 MB to 20 GB per image) is likely non-existent in this web application. The memory required would likely become prohibitive. Other solutions load these raw histology images in chunks and display the chunks necessary for a given visualization area. We thus recommend using other software if you want to zoom in at the spot and/or cell resolution.

7.4 Customization

While the documentation, title, icon and HTML footer are all customizable at `spatial LIBD::run_app()`, ultimately the panels shown are not unless you fork and adapt the internal code of this package. Thus, the interactive web applications powered by *spatialLIBD* are not as easy to customize as say *iSEE* web applications are. We think of our web application as a good enough prototype that can be useful for initial explorations of 10x Genomics Visium data. We welcome additions to our code, though we recognize that you might want to build your own production-level solution.

8 Reproducibility

The *spatialLIBD* package (Pardo, Spangler, Weber, et al., 2021) was made possible thanks to:

- R (R Core Team, 2022)
- *BiocFileCache* (Shepherd and Morgan, 2022)
- *BiocStyle* (Oleś, 2021)
- *knitr* (Xie, 2022)
- *lobstr* (Wickham, 2019)
- *RefManageR* (McLean, 2017)
- *rmarkdown* (Allaire, Xie, McPherson, Luraschi, Ushey, Atkins, Wickham, Cheng, Chang, and Iannone, 2022)
- *rtracklayer* (Lawrence, Gentleman, and Carey, 2009)
- *sessioninfo* (Wickham, Chang, Flight, Müller, and Hester, 2021)
- *SpatialExperiment* (Righelli, Risso, Crowell, and Weber, 2022)
- *testthat* (Wickham, 2011)

This package was developed using *biocthis*.

Code for creating the vignette

```
## Create the vignette
library("rmarkdown")
system.time(render("TenX_data_download.Rmd", "BiocStyle::html_document"))

## Extract the R code
library("knitr")
knit("TenX_data_download.Rmd", tangle = TRUE)
```

Date the vignette was generated.

```
#> [1] "2022-03-30 18:13:32 CST"
```

Wallclock time spent generating the vignette.

```
#> Time difference of 1.195 mins
```

R session information.

```
#> - Session info ----------------------------------------------------------------------------------
#>  setting  value
#>  version  R Under development (unstable) (2022-02-28 r81833)
#>  os       macOS Monterey 12.2.1
#>  system   aarch64, darwin20
#>  ui       X11
#>  language (EN)
#>  collate  en_US.UTF-8
#>  ctype    en_US.UTF-8
#>  tz       America/Mexico_City
#>  date     2022-03-30
#>  pandoc   2.17.1.1 @ /opt/homebrew/bin/ (via rmarkdown)
#>
#> - Packages --------------------------------------------------------------------------------------
#>  package                * version  date (UTC) lib source
#>  AnnotationDbi            1.57.1   2021-10-29 [1] Bioconductor
#>  AnnotationHub            3.3.9    2022-02-28 [1] Bioconductor
#>  assertthat               0.2.1    2019-03-21 [1] CRAN (R 4.2.0)
#>  attempt                  0.3.1    2020-05-03 [1] CRAN (R 4.2.0)
#>  beachmat                 2.11.0   2021-10-26 [1] Bioconductor
#>  beeswarm                 0.4.0    2021-06-01 [1] CRAN (R 4.2.0)
#>  benchmarkme              1.0.7    2021-03-21 [1] CRAN (R 4.2.0)
#>  benchmarkmeData          1.0.4    2020-04-23 [1] CRAN (R 4.2.0)
#>  Biobase                * 2.55.0   2021-10-26 [1] Bioconductor
#>  BiocFileCache          * 2.3.4    2022-01-20 [1] Bioconductor
#>  BiocGenerics           * 0.41.2   2021-11-15 [1] Bioconductor
#>  BiocIO                   1.5.0    2021-10-26 [1] Bioconductor
#>  BiocManager              1.30.16  2021-06-15 [1] CRAN (R 4.2.0)
#>  BiocNeighbors            1.13.0   2021-10-26 [1] Bioconductor
#>  BiocParallel             1.29.18  2022-03-21 [1] Bioconductor
#>  BiocSingular             1.11.0   2021-10-26 [1] Bioconductor
```

```
#>  BiocStyle            * 2.23.1  2021-12-17 [1] Bioconductor
#>  BiocVersion            3.15.2  2022-03-29 [1] Bioconductor
#>  Biostrings             2.63.3  2022-03-29 [1] Bioconductor
#>  bit                    4.0.4   2020-08-04 [1] CRAN (R 4.2.0)
#>  bit64                  4.0.5   2020-08-30 [1] CRAN (R 4.2.0)
#>  bitops                 1.0-7   2021-04-24 [1] CRAN (R 4.2.0)
#>  blob                   1.2.2   2021-07-23 [1] CRAN (R 4.2.0)
#>  bookdown               0.25    2022-03-16 [1] CRAN (R 4.2.0)
#>  brio                   1.1.3   2021-11-30 [1] CRAN (R 4.2.0)
#>  bslib                  0.3.1   2021-10-06 [1] CRAN (R 4.2.0)
#>  cachem                 1.0.6   2021-08-19 [1] CRAN (R 4.2.0)
#>  cli                    3.2.0   2022-02-14 [1] CRAN (R 4.2.0)
#>  codetools              0.2-18  2020-11-04 [1] CRAN (R 4.2.0)
#>  colorout               1.2-2   2022-03-01 [1] Github (jalvesaq/colorout@79931fd)
#>  colorspace             2.0-3   2022-02-21 [1] CRAN (R 4.2.0)
#>  config                 0.3.1   2020-12-17 [1] CRAN (R 4.2.0)
#>  cowplot                1.1.1   2020-12-30 [1] CRAN (R 4.2.0)
#>  crayon                 1.5.1   2022-03-26 [1] CRAN (R 4.2.0)
#>  curl                   4.3.2   2021-06-23 [1] CRAN (R 4.2.0)
#>  data.table             1.14.2  2021-09-27 [1] CRAN (R 4.2.0)
#>  DBI                    1.1.2   2021-12-20 [1] CRAN (R 4.2.0)
#>  dbplyr               * 2.1.1   2021-04-06 [1] CRAN (R 4.2.0)
#>  DelayedArray           0.21.2  2021-11-16 [1] Bioconductor
#>  DelayedMatrixStats     1.17.0  2021-10-26 [1] Bioconductor
#>  desc                   1.4.1   2022-03-06 [1] CRAN (R 4.2.0)
#>  digest                 0.6.29  2021-12-01 [1] CRAN (R 4.2.0)
#>  doParallel             1.0.17  2022-02-07 [1] CRAN (R 4.2.0)
#>  dotCall64              1.0-1   2021-02-11 [1] CRAN (R 4.2.0)
#>  dplyr                  1.0.8   2022-02-08 [1] CRAN (R 4.2.0)
#>  dqrng                  0.3.0   2021-05-01 [1] CRAN (R 4.2.0)
#>  DropletUtils           1.15.2  2021-11-08 [1] Bioconductor
#>  DT                     0.22    2022-03-28 [1] CRAN (R 4.2.0)
#>  edgeR                  3.37.1  2022-03-16 [1] Bioconductor
#>  ellipsis               0.3.2   2021-04-29 [1] CRAN (R 4.2.0)
#>  evaluate               0.15    2022-02-18 [1] CRAN (R 4.2.0)
#>  ExperimentHub          2.3.5   2022-01-20 [1] Bioconductor
#>  fansi                  1.0.3   2022-03-24 [1] CRAN (R 4.2.0)
#>  farver                 2.1.0   2021-02-28 [1] CRAN (R 4.2.0)
#>  fastmap                1.1.0   2021-01-25 [1] CRAN (R 4.2.0)
#>  fields                 13.3    2021-10-30 [1] CRAN (R 4.2.0)
#>  filelock               1.0.2   2018-10-05 [1] CRAN (R 4.2.0)
#>  foreach                1.5.2   2022-02-02 [1] CRAN (R 4.2.0)
#>  fs                     1.5.2   2021-12-08 [1] CRAN (R 4.2.0)
#>  generics               0.1.2   2022-01-31 [1] CRAN (R 4.2.0)
#>  GenomeInfoDb         * 1.31.6  2022-03-16 [1] Bioconductor
#>  GenomeInfoDbData       1.2.7   2022-03-02 [1] Bioconductor
#>  GenomicAlignments      1.31.2  2021-11-05 [1] Bioconductor
#>  GenomicRanges        * 1.47.6  2022-01-12 [1] Bioconductor
#>  ggbeeswarm             0.6.0   2017-08-07 [1] CRAN (R 4.2.0)
#>  ggplot2                3.3.5   2021-06-25 [1] CRAN (R 4.2.0)
#>  ggrepel                0.9.1   2021-01-15 [1] CRAN (R 4.2.0)
```

```
#>   glue                  1.6.2    2022-02-24 [1] CRAN (R 4.2.0)
#>   golem                 0.3.2    2022-03-04 [1] CRAN (R 4.2.0)
#>   gridExtra             2.3      2017-09-09 [1] CRAN (R 4.2.0)
#>   gtable                0.3.0    2019-03-25 [1] CRAN (R 4.2.0)
#>   HDF5Array             1.23.2   2021-11-15 [1] Bioconductor
#>   htmltools             0.5.2    2021-08-25 [1] CRAN (R 4.2.0)
#>   htmlwidgets           1.5.4    2021-09-08 [1] CRAN (R 4.2.0)
#>   httpuv                1.6.5    2022-01-05 [1] CRAN (R 4.2.0)
#>   httr                  1.4.2    2020-07-20 [1] CRAN (R 4.2.0)
#>   interactiveDisplayBase 1.33.0  2021-10-26 [1] Bioconductor
#>   IRanges             * 2.29.1   2021-11-16 [1] Bioconductor
#>   irlba                 2.3.5    2021-12-06 [1] CRAN (R 4.2.0)
#>   iterators             1.0.14   2022-02-05 [1] CRAN (R 4.2.0)
#>   jquerylib             0.1.4    2021-04-26 [1] CRAN (R 4.2.0)
#>   jsonlite              1.8.0    2022-02-22 [1] CRAN (R 4.2.0)
#>   KEGGREST              1.35.0   2021-10-26 [1] Bioconductor
#>   knitr                 1.38     2022-03-25 [1] CRAN (R 4.2.0)
#>   labeling              0.4.2    2020-10-20 [1] CRAN (R 4.2.0)
#>   later                 1.3.0    2021-08-18 [1] CRAN (R 4.2.0)
#>   lattice               0.20-45  2021-09-22 [1] CRAN (R 4.2.0)
#>   lazyeval              0.2.2    2019-03-15 [1] CRAN (R 4.2.0)
#>   lifecycle             1.0.1    2021-09-24 [1] CRAN (R 4.2.0)
#>   limma                 3.51.5   2022-02-17 [1] Bioconductor
#>   lobstr              * 1.1.1    2019-07-02 [1] CRAN (R 4.2.0)
#>   locfit                1.5-9.5  2022-03-03 [1] CRAN (R 4.2.0)
#>   lubridate             1.8.0    2021-10-07 [1] CRAN (R 4.2.0)
#>   magick                2.7.3    2021-08-18 [1] CRAN (R 4.2.0)
#>   magrittr              2.0.3    2022-03-30 [1] CRAN (R 4.2.0)
#>   maps                  3.4.0    2021-09-25 [1] CRAN (R 4.2.0)
#>   Matrix                1.4-1    2022-03-23 [1] CRAN (R 4.2.0)
#>   MatrixGenerics      * 1.7.0    2021-10-26 [1] Bioconductor
#>   matrixStats         * 0.61.0   2021-09-17 [1] CRAN (R 4.2.0)
#>   memoise               2.0.1    2021-11-26 [1] CRAN (R 4.2.0)
#>   mime                  0.12     2021-09-28 [1] CRAN (R 4.2.0)
#>   munsell               0.5.0    2018-06-12 [1] CRAN (R 4.2.0)
#>   pillar                1.7.0    2022-02-01 [1] CRAN (R 4.2.0)
#>   pkgconfig             2.0.3    2019-09-22 [1] CRAN (R 4.2.0)
#>   pkgload               1.2.4    2021-11-30 [1] CRAN (R 4.2.0)
#>   plotly                4.10.0   2021-10-09 [1] CRAN (R 4.2.0)
#>   plyr                  1.8.7    2022-03-24 [1] CRAN (R 4.2.0)
#>   png                   0.1-7    2013-12-03 [1] CRAN (R 4.2.0)
#>   Polychrome            1.3.1    2021-07-16 [1] CRAN (R 4.2.0)
#>   promises              1.2.0.1  2021-02-11 [1] CRAN (R 4.2.0)
#>   purrr                 0.3.4    2020-04-17 [1] CRAN (R 4.2.0)
#>   R.methodsS3           1.8.1    2020-08-26 [1] CRAN (R 4.2.0)
#>   R.oo                  1.24.0   2020-08-26 [1] CRAN (R 4.2.0)
#>   R.utils               2.11.0   2021-09-26 [1] CRAN (R 4.2.0)
#>   R6                    2.5.1    2021-08-19 [1] CRAN (R 4.2.0)
#>   rappdirs              0.3.3    2021-01-31 [1] CRAN (R 4.2.0)
#>   RColorBrewer          1.1-2    2014-12-07 [1] CRAN (R 4.2.0)
#>   Rcpp                  1.0.8.3  2022-03-17 [1] CRAN (R 4.2.0)
```

```
#>   RCurl                1.98-1.6 2022-02-08 [1] CRAN (R 4.2.0)
#>   RefManageR         * 1.3.0    2020-11-13 [1] CRAN (R 4.2.0)
#>   restfulr             0.0.13   2017-08-06 [1] CRAN (R 4.2.0)
#>   rhdf5                2.39.6   2022-03-09 [1] Bioconductor
#>   rhdf5filters         1.7.0    2021-10-26 [1] Bioconductor
#>   Rhdf5lib             1.17.3   2022-01-31 [1] Bioconductor
#>   rjson                0.2.21   2022-01-09 [1] CRAN (R 4.2.0)
#>   rlang                1.0.2    2022-03-04 [1] CRAN (R 4.2.0)
#>   rmarkdown            2.13     2022-03-10 [1] CRAN (R 4.2.0)
#>   roxygen2             7.1.2    2021-09-08 [1] CRAN (R 4.2.0)
#>   rprojroot            2.0.2    2020-11-15 [1] CRAN (R 4.2.0)
#>   Rsamtools            2.11.0   2021-10-27 [1] Bioconductor
#>   RSQLite              2.2.11   2022-03-23 [1] CRAN (R 4.2.0)
#>   rstudioapi           0.13     2020-11-12 [1] CRAN (R 4.2.0)
#>   rsvd                 1.0.5    2021-04-16 [1] CRAN (R 4.2.0)
#>   rtracklayer        * 1.55.4   2022-03-23 [1] Bioconductor
#>   S4Vectors          * 0.33.15  2022-03-28 [1] Bioconductor
#>   sass                 0.4.1    2022-03-23 [1] CRAN (R 4.2.0)
#>   ScaledMatrix         1.3.0    2021-10-26 [1] Bioconductor
#>   scales               1.1.1    2020-05-11 [1] CRAN (R 4.2.0)
#>   scater               1.23.6   2022-03-15 [1] Bioconductor
#>   scatterplot3d        0.3-41   2018-03-14 [1] CRAN (R 4.2.0)
#>   scuttle              1.5.1    2022-03-15 [1] Bioconductor
#>   sessioninfo        * 1.2.2    2021-12-06 [1] CRAN (R 4.2.0)
#>   shiny                1.7.1    2021-10-02 [1] CRAN (R 4.2.0)
#>   shinyWidgets         0.6.4    2022-02-06 [1] CRAN (R 4.2.0)
#>   SingleCellExperiment * 1.17.2 2021-11-18 [1] Bioconductor
#>   spam                 2.8-0    2022-01-06 [1] CRAN (R 4.2.0)
#>   sparseMatrixStats    1.7.0    2021-10-26 [1] Bioconductor
#>   SpatialExperiment  * 1.5.4    2022-03-11 [1] Bioconductor
#>   spatialLIBD        * 1.7.14   2022-03-31 [1] Github (LieberInstitute/spatialLIBD@154a594)
#>   stringi              1.7.6    2021-11-29 [1] CRAN (R 4.2.0)
#>   stringr              1.4.0    2019-02-10 [1] CRAN (R 4.2.0)
#>   SummarizedExperiment * 1.25.3 2021-12-08 [1] Bioconductor
#>   testthat             3.1.3    2022-03-29 [1] CRAN (R 4.2.0)
#>   tibble               3.1.6    2021-11-07 [1] CRAN (R 4.2.0)
#>   tidyr                1.2.0    2022-02-01 [1] CRAN (R 4.2.0)
#>   tidyselect           1.1.2    2022-02-21 [1] CRAN (R 4.2.0)
#>   usethis              2.1.5    2021-12-09 [1] CRAN (R 4.2.0)
#>   utf8                 1.2.2    2021-07-24 [1] CRAN (R 4.2.0)
#>   vctrs                0.4.0    2022-03-30 [1] CRAN (R 4.2.0)
#>   vipor                0.4.5    2017-03-22 [1] CRAN (R 4.2.0)
#>   viridis              0.6.2    2021-10-13 [1] CRAN (R 4.2.0)
#>   viridisLite          0.4.0    2021-04-13 [1] CRAN (R 4.2.0)
#>   withr                2.5.0    2022-03-03 [1] CRAN (R 4.2.0)
#>   xfun                 0.30     2022-03-02 [1] CRAN (R 4.2.0)
#>   XML                  3.99-0.9 2022-02-24 [1] CRAN (R 4.2.0)
#>   xml2                 1.3.3    2021-11-30 [1] CRAN (R 4.2.0)
#>   xtable               1.8-4    2019-04-21 [1] CRAN (R 4.2.0)
#>   XVector              0.35.0   2021-10-26 [1] Bioconductor
#>   yaml                 2.3.5    2022-02-21 [1] CRAN (R 4.2.0)
```

```
#>  zlibbioc              1.41.0   2021-10-26 [1] Bioconductor
#>
#>  [1] /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/library
#>
#>  -------------------------------------------------------------------------------
```

9   Bibliography

This vignette was generated using *BiocStyle* (Oleś, 2021) with *knitr* (Xie, 2022) and *rmarkdown* (Allaire, Xie, McPherson, et al., 2022) running behind the scenes.

Citations made with *RefManageR* (McLean, 2017).

[1] J. Allaire, Y. Xie, J. McPherson, et al. rmarkdown: Dynamic Documents for R. R package version 2.13. 2022. URL: https://github.com/rstudio/rmarkdown.

[2] M. Lawrence, R. Gentleman, and V. Carey. "rtracklayer: an R package for interfacing with genome browsers". In: Bioinformatics 25 (2009), pp. 1841-1842. DOI: 10.1093/bioinformatics/btp328. URL: http://bioinformatics.oxfordjournals.org/content/25/14/1841.abstract.

[3] K. R. Maynard, L. Collado-Torres, L. M. Weber, et al. "Transcriptome-scale spatial gene expression in the human dorsolateral prefrontal cortex". In: Nature Neuroscience (2021). DOI: 10.1038/s41593-020-00787-0. URL: https://www.nature.com/articles/s41593-020-00787-0.

[4] M. W. McLean. "RefManageR: Import and Manage BibTeX and BibLaTeX References in R". In: The Journal of Open Source Software (2017). DOI: 10.21105/joss.00338.

[5] A. Oleś. BiocStyle: Standard styles for vignettes and other Bioconductor documents. R package version 2.23.1. 2021. URL: https://github.com/Bioconductor/BiocStyle.

[6] B. Pardo, A. Spangler, L. M. Weber, et al. "spatialLIBD: an R/Bioconductor package to visualize spatially-resolved transcriptomics data". In: bioRxiv (2021). DOI: 10.1101/2021.04.29.440149. URL: https://www.biorxiv.org/content/10.1101/2021.04.29.440149v1.

[7] R Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing. Vienna, Austria, 2022. URL: https://www.R-project.org/.

[8] D. Righelli, D. Risso, H. Crowell, et al. SpatialExperiment: S4 Class for Spatially Resolved Transcriptomics Data. R package version 1.5.4. 2022. URL: https://github.com/drighelli/SpatialExperiment.

[9] L. Shepherd and M. Morgan. BiocFileCache: Manage Files Across Sessions. R package version 2.3.4. 2022.

[10] H. Wickham. lobstr: Visualize R Data Structures with Trees. R package version 1.1.1. 2019. URL: https://CRAN.R-project.org/package=lobstr.

[11] H. Wickham. "testthat: Get Started with Testing". In: The R Journal 3 (2011), pp. 5–10. URL: https://journal.r-project.org/archive/2011-1/RJournal_2011-1_Wickham.pdf.

[12] H. Wickham, W. Chang, R. Flight, et al. sessioninfo: R Session Information. R package version 1.2.2. 2021. URL: https://CRAN.R-project.org/package=sessioninfo.

[13] Y. Xie. knitr: A General-Purpose Package for Dynamic Report Generation in R. R package version 1.38. 2022. URL: https://yihui.org/knitr/.