Best Practices for Spatial Transcriptomics Analysis with Bioconductor

2023-09-13

Table of contents

Welcome

This is the website for the online book **'Best Practices for Spatial Transcriptomics Analysis with Bioconductor'**.

This book provides several examples of computational analysis workflows for spatially resolved-transcriptomics (SRT) data, using the Bioconductor framework within the R programming language. The chapters contain details on individual analysis steps as well as complete workflows, with example datasets and R code that can be run on your own laptop.

The book is organized into several parts, including background, individual analysis steps, and example workflows.

Additional details on single-cell analysis workflows as well as introductory material on R and Bioconductor can be found in the related book Orchestrating Single-Cell Analysis with Bioconductor (OSCA).

Part I

Introduction

This book provides interactive examples of computational analysis workflows for spatially-resolved transcriptomics data using the Bioconductor framework within the R programming language.

Contents

Chapters are organized into several parts:

- **Introduction:** introduction, background, and R/Bioconductor data classes
- **Analysis steps:** chapters on individual analysis steps
- **Workflows:** complete workflows for several example datasets
- **Appendix:** related resources, contributors and acknowledgments, and references

Who this book is for

Overall, the aim of this book is to interactively demonstrate analysis workflows at a sufficient level of detail that allows readers to get started with analyzing spatially-resolved transcriptomics data and to adapt or extend these workflows to their own datasets. While we will showcase a number of methods available through Bioconductor or CRAN for individual analysis steps, we do not intend to provide a comprehensive review of all available methods for each step.

This book is intended for readers who have some experience with R, but does not necessarily assume familiarity with Bioconductor. The examples and workflows include R code required to download data, set up data objects, perform analyses, and visualize results.

For readers who are new to R and Bioconductor, additional useful resources include:

- The Orchestrating Single-Cell Analysis with Bioconductor (OSCA) book (Amezquita et al. 2019), which contains comprehensive resources on analysis workflows for single-cell data, as well as additional introductory material on R and Bioconductor.
- The R for Data Science online book provides an excellent introduction to R.
- Data Carpentry and Software Carpentry provide online lesson materials on R programming, the Unix shell, and version control.
- The R/Bioconductor Data Science Team at LIBD has a detailed guide of free resources and videos to learn more about R and Bioconductor, as well as YouTube videos and LIBD rstats club sessions, including some on the basics of Bioconductor and infrastructure for storing gene expression data.

Additional details on data preprocessing procedures for spatially-resolved transcriptomics data from the 10x Genomics Visium platform are provided in the following online book (using tools outside R and Bioconductor):

- Visium Data Preprocessing

Bioconductor

Bioconductor is an *open source* and *open development* project, providing a cohesive and flexible framework for analyzing high-throughput genomic data in R (Huber et al. 2015). The Bioconductor project consists of more than 2,000 contributed R packages, as well as core infrastructure maintained by the Bioconductor Core Team, providing a rich analysis environment for users.

One of the main advantages of Bioconductor is the modularity and open development philosophy. R packages are contributed by numerous research groups, with the Bioconductor Core Team coordinating the overall project and maintaining core infrastructure, build testing, and development guidelines. A key feature is that contributed packages use consistent data structures, enabling users to integrate packages into analysis workflows. Bioconductor packages also include comprehensive documentation, including long-form tutorials or vignettes.

This modular and open development approach allows data analysts to build analysis workflows that integrate the latest state-of-the-art methods developed by research groups around the world. Any research group can contribute new packages to Bioconductor by following the contribution guidelines provided on the Bioconductor website.

Contributions

We welcome suggestions for updates to the analysis and workflow chapters. Suggestions may be provided as GitHub issues.

All methods included in the code examples must be available as R packages from either Bioconductor or CRAN. This ensures compatibility with the existing workflows and provides users with guarantees regarding ease of installation, long-term availability, stability, and maintenance through the Bioconductor or CRAN systems. By following this strategy, we aim to showcase key methods for individual analysis steps and demonstrate flexible analysis workflows at a sufficient level of detail to enable readers to adapt or extend these workflows to their own datasets.

Additional methods (e.g. available from GitHub) may also be described within the analysis chapters, but these methods will not be included in the code examples. Methods available as

`pip` installable Python packages will also be integrated into the examples in future updates using reticulate.

1 Spatially-resolved transcriptomics

Spatially-resolved transcriptomics (SRT) refers to high-throughput genomic technologies that enable the measurement of large sets of genes at a large number of spatial locations (e.g. up to thousands of genes at thousands of measurement locations), usually on two-dimensional tissue sections. A number of technological platforms have been developed, each with their unique advantages. SRT was named the Method of the Year 2020 by Nature Methods, and has found widespread application in numerous biological systems.

For recent reviews of available platforms, computational analysis methods, and outstanding challenges, see Bressan et al. 2023 or Moses et al. 2022.

1.1 Overview

In this book, we concentrate on commercially available platforms, since these are most widely used and available. Here, we provide a short overview and introductory details on the platforms used to generate the datasets discussed in this book.

1.2 Sequencing-based platforms

Sequencing-based platforms capture RNAs at a set of spatial measurement locations, tag RNAs with spatial barcodes, and generate a readout by next-generation sequencing.

These platforms can provide up to full-transcriptome feature resolution due to the use of next-generation sequencing. The spatial resolution depends on the size of the capture locations and tissue cell density, and may include multiple cells per measurement location.

1.2.1 10x Genomics Visium

The 10x Genomics Visium platform measures transcriptome-wide gene expression at a grid of spatial locations (referred to as spots) on a tissue capture area. Either fresh-frozen or formalin-fixed paraffin-embedded (FFPE) tissue may be used. Each spot contains millions of spatially-barcoded capture oligonucleotides, which bind to mRNAs from the tissue. A cDNA

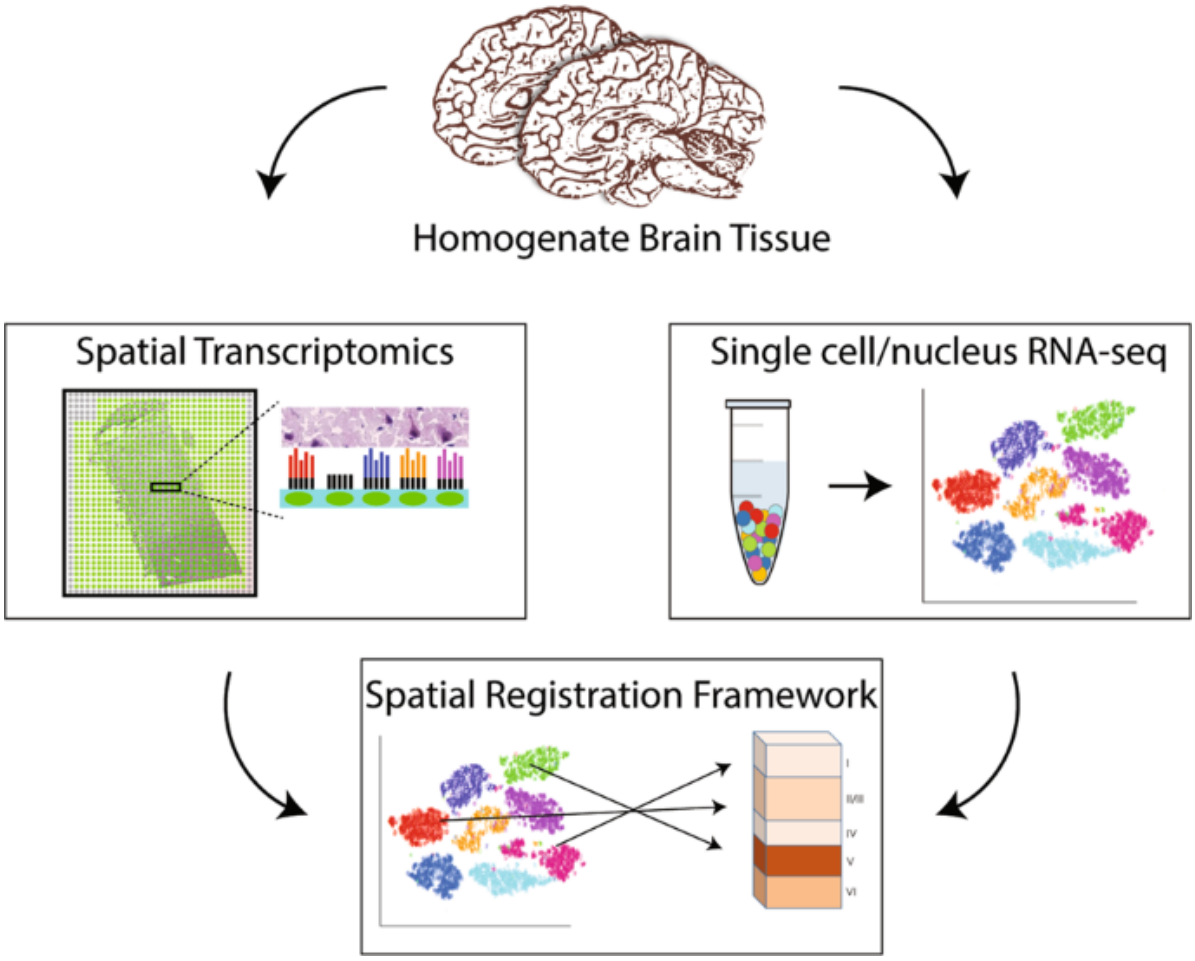


Figure 1.1: Illustration of potential uses of spatially-resolved transcriptomics data and how it can be combined with single cell/nucleus RNA-seq to study tissues of interest such as the human brain. Source: Maynard et al, *Neuropsychopharmacol.*, 2020.

library is then generated for sequencing, which includes the spatial barcodes, allowing reads to be mapped back to their spatial locations.

The array dimensions are 6.5mm x 6.5mm, with around 5000 barcoded spots. Spots are 55µm in diameter and spaced 100µm center-to-center in a hexagonal grid arrangement. The number of cells per spot depends on the tissue cell density, e.g. around 0-10 for human brain or ~50 for mouse brain. Each Visium slide contains four tissue capture areas. The following figure provides an illustration.

Histology images generated from hematoxylin and eosin (H&E) staining can be used to identify anatomical and cell morphological features such as the number of cells per spot.

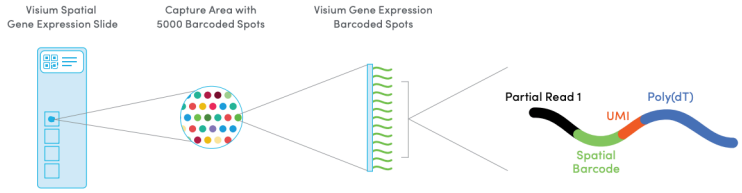


Figure 1.2: Schematic of 10x Genomics Visium platform. Source: 10x Genomics Visium website

1.3 Molecule-based platforms

Molecule-based platforms identify individual RNAs by in situ hybridization (ISH) or in situ sequencing (ISS) at up to sub-cellular spatial resolution. Counts may be aggregated to cellular (single-cell) spatial resolution by applying manual or data-driven cell segmentation algorithms.

Feature resolution is generally on the order of 100s to 1000s of RNAs, which may be selected as targeted panels of RNAs of interest.

1.3.1 10x Genomics Xenium

Details available on the 10x Genomics website.

1.3.2 Vizgen MERSCOPE

Details available on the Vizgen website.

1.3.3 NanoString CosMx

Details available on the NanoString website.

2 Bioconductor data classes

Bioconductor provides several data classes for storing and manipulating SRT datasets. By relying on these standardized data classes, we can build analysis workflows that can easily connect methods and packages developed by different groups.

Below, we describe the Bioconductor data classes used in this book.

2.1 SpatialExperiment class

The SpatialExperiment class is the core data class that we use in this book. This class allows us to store datasets at the spot or cell level, e.g. 10x Genomics Visium data (spot level) or data from molecule-based platforms that has been aggregated to the cell level.

SpatialExperiment builds on the more general SingleCellExperiment class (Amezquita et al. 2019) for single-cell RNA sequencing data, with additional customizations to store spatial information, such as spatial coordinates and image files.

A summary of the `SpatialExperiment` object structure is shown in the following schematic. Briefly, a `SpatialExperiment` object consists of (i) `assays` containing expression counts, (ii) `rowData` containing information on features, i.e. genes, (iii) `colData` containing information on spots or cells, including nonspatial and spatial metadata, (iv) `spatialCoords` containing spatial coordinates, and (v) `imgData` containing image data. For spot-based SRT data (e.g. 10x Genomics Visium), a single `assay` named `counts` is used.

For more details, see the Bioconductor vignette for the SpatialExperiment class or the associated paper (Righelli, Weber, and Crowell et al. 2022).

2.2 Molecule-based data

Molecule-based SRT datasets include additional information such as the locations of individual mRNA molecules and segmentation boundaries for cells.

The following classes provide additional functionality to store and manipulate this information.

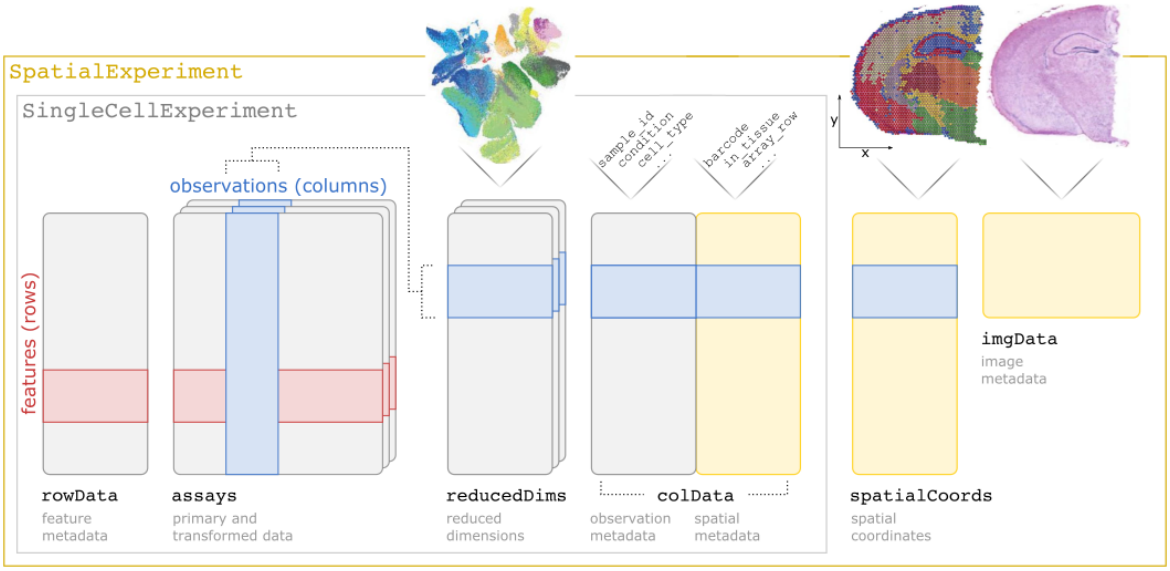


Figure 2.1: Overview of the `SpatialExperiment` object class for storing spatially-resolved transcriptomics datasets in the Bioconductor framework.

Note that these classes extend `SpatialExperiment` and are compatible with `SpatialExperiment` for aggregated cell-level analyses.

2.2.1 MoleculeExperiment

`MoleculeExperiment` is available as a Bioconductor package and described in detail in a paper by Couto et al. (2023).

2.2.2 SpatialFeatureExperiment

`SpatialFeatureExperiment` is available as a Bioconductor package and described in detail in a paper by Moses et al. (2023).

Part II

Analysis steps

This part consists of several chapters for steps in a computational analysis pipeline for spatially-resolved transcriptomics (SRT) data. This includes quality control (QC), normalization, feature selection, dimensionality reduction, clustering, identifying marker genes, and spot-level deconvolution.

These steps require that the raw data has been loaded into R. In the previous part, we provide instructions and examples showing how to do this for the 10x Genomics Visium platform.

Throughout these chapters, we will rely on Bioconductor data classes, especially `SpatialExperiment`, to connect the steps in the pipeline. Following the Bioconductor principle of modularity, you can substitute alternative methods for individual steps if you prefer, as long as these can interface with the `SpatialExperiment` structure.

Load data

In the following analysis chapters, we use a pre-prepared dataset where we have previously applied data preprocessing procedures (using tools outside R and Bioconductor) and saved the object in the `SpatialExperiment` format. This is available from the STexampleData package.

The dataset consists of a single sample of human brain from the dorsolateral prefrontal cortex (DLPFC) region, measured using the 10x Genomics Visium platform, sourced from Maynard et al. (2021). The dataset is also described in more detail in [Visium human DLPFC workflow].

Here, we show how to load the data from the `STexampleData` package.

```
library(SpatialExperiment)
library(STexampleData)

# load object
spe <- Visium_humanDLPFC()
```

SpatialExperiment object

Next, we inspect the `SpatialExperiment` object. For more details, see Bioconductor data classes.

```
# check object
spe
```

```
class: SpatialExperiment
dim: 33538 4992
metadata(0):
assays(1): counts
rownames(33538): ENSG00000243485 ENSG00000237613 ... ENSG00000277475
  ENSG00000268674
rowData names(3): gene_id gene_name feature_type
colnames(4992): AAACAACGAATAGTTC-1 AAACAAGTATCTCCCA-1 ...
  TTGTTTGTATTACACG-1 TTGTTTGTGTAAATTC-1
colData names(7): barcode_id sample_id ... ground_truth cell_count
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
```

```
# number of features (rows) and spots (columns)
dim(spe)
```

```
[1] 33538  4992
```

```
# names of 'assay' tables
assayNames(spe)
```

```
[1] "counts"
```

```
# features metadata
head(rowData(spe))
```

```
DataFrame with 6 rows and 3 columns
                       gene_id   gene_name    feature_type
                   <character> <character>     <character>
ENSG00000243485 ENSG00000243485 MIR1302-2HG Gene Expression
ENSG00000237613 ENSG00000237613     FAM138A Gene Expression
ENSG00000186092 ENSG00000186092       OR4F5 Gene Expression
ENSG00000238009 ENSG00000238009  AL627309.1 Gene Expression
ENSG00000239945 ENSG00000239945  AL627309.3 Gene Expression
ENSG00000239906 ENSG00000239906  AL627309.2 Gene Expression
```

```
# spot-level metadata
head(colData(spe))
```

```
DataFrame with 6 rows and 7 columns
                         barcode_id     sample_id in_tissue array_row
                        <character>   <character> <integer> <integer>
AAACAACGAATAGTTC-1 AAACAACGAATAGTTC-1 sample_151673         0         0
AAACAAGTATCTCCCA-1 AAACAAGTATCTCCCA-1 sample_151673         1        50
AAACAATCTACTAGCA-1 AAACAATCTACTAGCA-1 sample_151673         1         3
AAACACCAATAACTGC-1 AAACACCAATAACTGC-1 sample_151673         1        59
AAACAGAGCGACTCCT-1 AAACAGAGCGACTCCT-1 sample_151673         1        14
AAACAGCTTTCAGAAG-1 AAACAGCTTTCAGAAG-1 sample_151673         1        43
                   array_col ground_truth cell_count
                   <integer>  <character>  <integer>
AAACAACGAATAGTTC-1        16           NA         NA
AAACAAGTATCTCCCA-1       102       Layer3          6
AAACAATCTACTAGCA-1        43       Layer1         16
AAACACCAATAACTGC-1        19           WM          5
AAACAGAGCGACTCCT-1        94       Layer3          2
AAACAGCTTTCAGAAG-1         9       Layer5          4
```

```
# spatial coordinates
head(spatialCoords(spe))
```

```
                   pxl_col_in_fullres pxl_row_in_fullres
AAACAACGAATAGTTC-1               3913               2435
AAACAAGTATCTCCCA-1               9791               8468
AAACAATCTACTAGCA-1               5769               2807
AAACACCAATAACTGC-1               4068               9505
AAACAGAGCGACTCCT-1               9271               4151
AAACAGCTTTCAGAAG-1               3393               7583
```

```
# image metadata
imgData(spe)
```

```
DataFrame with 2 rows and 4 columns
      sample_id    image_id   data scaleFactor
    <character> <character> <list>   <numeric>
```

```
1 sample_151673      lowres   ####   0.0450045
2 sample_151673       hires   ####   0.1500150
```

3 Quality control

3.1 Background

Quality control (QC) procedures at the spot level aim to remove low-quality spots before further analysis. Low-quality spots can occur due to problems during library preparation or other experimental procedures. Examples include large proportions of dead cells due to cell damage during library preparation, and low mRNA capture efficiency due to inefficient reverse transcription or PCR amplification.

These spots are usually removed prior to further analysis, since otherwise they tend to create problems during downstream analyses such as clustering. For example, problematic spots that are not removed could show up as separate clusters, which may be misidentified as distinct cell types.

Low-quality spots can be identified according to several characteristics, including:

- library size (i.e. total UMI counts per spot)
- number of expressed features (i.e. number of genes with non-zero UMI counts per spot)
- proportion of reads mapping to mitochondrial genes (a high proportion indicates cell damage)
- number of cells per spot (unusually high values can indicate problems)

Low library size or low number of expressed features can indicate poor mRNA capture rates, e.g. due to cell damage and missing mRNAs, or low reaction efficiency. A high proportion of mitochondrial reads indicates cell damage, e.g. partial cell lysis leading to leakage and missing cytoplasmic mRNAs, with the resulting reads therefore concentrated on the remaining mitochondrial mRNAs that are relatively protected inside the mitochondrial membrane. Unusually high numbers of cells per spot can indicate problems during cell segmentation.

The first three characteristics listed above are also used for QC in scRNA-seq data. However, the expected distributions for high-quality spots are different (compared to high-quality cells in scRNA-seq), since spots may contain zero, one, or multiple cells.

3.2 Load data

```
library(SpatialExperiment)
library(STexampleData)

# load object
spe <- Visium_humanDLPFC()
```

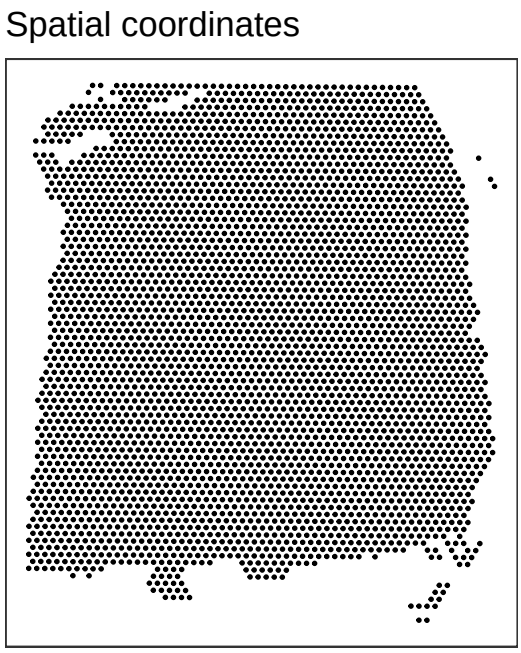
3.3 Plot data

As an initial check, plot the spatial coordinates (spots) in x-y dimensions on the tissue slide, to check that the object has loaded correctly and that the orientation is as expected.

We use visualization functions from the ggspavis package to generate plots.

```
library(ggspavis)
```

```
# plot spatial coordinates (spots)
plotSpots(spe)
```



3.4 Calculate QC metrics

We calculate the QC metrics described above with a combination of methods from the `scater` (McCarthy et al. 2017) package (for metrics that are also used for scRNA-seq data, where we treat spots as equivalent to cells) and our own functions.

The QC metrics from `scater` can be calculated and added to the `SpatialExperiment` object as follows. Here, we also identify mitochondrial reads using their gene names, and pass these as an argument to `scater`.

First, we subset the object to keep only spots over tissue. The remaining spots are background spots, which we are not interested in.

```
library(scater)
```

```
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
dim(spe)
```

```
[1] 33538  3639
```

```
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
table(is_mito)
```

```
is_mito
FALSE  TRUE
33525    13
```

```
rowData(spe)$gene_name[is_mito]
```

```
 [1] "MT-ND1"  "MT-ND2"  "MT-CO1"  "MT-CO2"  "MT-ATP8" "MT-ATP6" "MT-CO3"
 [8] "MT-ND3"  "MT-ND4L" "MT-ND4"  "MT-ND5"  "MT-ND6"  "MT-CYB"
```

```
# calculate per-spot QC metrics and store in colData
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
head(colData(spe))
```

```
DataFrame with 6 rows and 13 columns
                         barcode_id     sample_id in_tissue array_row
                        <character>   <character> <integer> <integer>
AAACAAGTATCTCCCA-1 AAACAAGTATCTCCCA-1 sample_151673         1        50
AAACAATCTACTAGCA-1 AAACAATCTACTAGCA-1 sample_151673         1         3
AAACACCAATAACTGC-1 AAACACCAATAACTGC-1 sample_151673         1        59
AAACAGAGCGACTCCT-1 AAACAGAGCGACTCCT-1 sample_151673         1        14
AAACAGCTTTCAGAAG-1 AAACAGCTTTCAGAAG-1 sample_151673         1        43
AAACAGGGTCTATATT-1 AAACAGGGTCTATATT-1 sample_151673         1        47
                   array_col ground_truth cell_count       sum  detected
                   <integer>  <character>  <integer> <numeric> <numeric>
AAACAAGTATCTCCCA-1       102       Layer3          6      8458      3586
AAACAATCTACTAGCA-1        43       Layer1         16      1667      1150
AAACACCAATAACTGC-1        19           WM          5      3769      1960
AAACAGAGCGACTCCT-1        94       Layer3          2      5433      2424
AAACAGCTTTCAGAAG-1         9       Layer5          4      4278      2264
AAACAGGGTCTATATT-1        13       Layer6          6      4004      2178
                   subsets_mito_sum subsets_mito_detected subsets_mito_percent
                          <numeric>             <numeric>            <numeric>
AAACAAGTATCTCCCA-1             1407                    13              16.6351
AAACAATCTACTAGCA-1              204                    11              12.2376
AAACACCAATAACTGC-1              430                    13              11.4089
AAACAGAGCGACTCCT-1             1316                    13              24.2223
AAACAGCTTTCAGAAG-1              651                    12              15.2174
AAACAGGGTCTATATT-1              621                    13              15.5095
                       total
                   <numeric>
AAACAAGTATCTCCCA-1      8458
AAACAATCTACTAGCA-1      1667
AAACACCAATAACTGC-1      3769
AAACAGAGCGACTCCT-1      5433
AAACAGCTTTCAGAAG-1      4278
AAACAGGGTCTATATT-1      4004
```

3.5 Selecting thresholds

The simplest option to apply the QC metrics is to select thresholds for each metric, and remove any spots that do not meet the thresholds for one or more metrics. Exploratory visualizations can be used to help select appropriate thresholds, which may differ depending on the dataset.

Here, we use visualizations to select thresholds for several QC metrics in our human DLPFC dataset: (i) library size, (ii) number of expressed genes, (iii) proportion of mitochondrial reads,

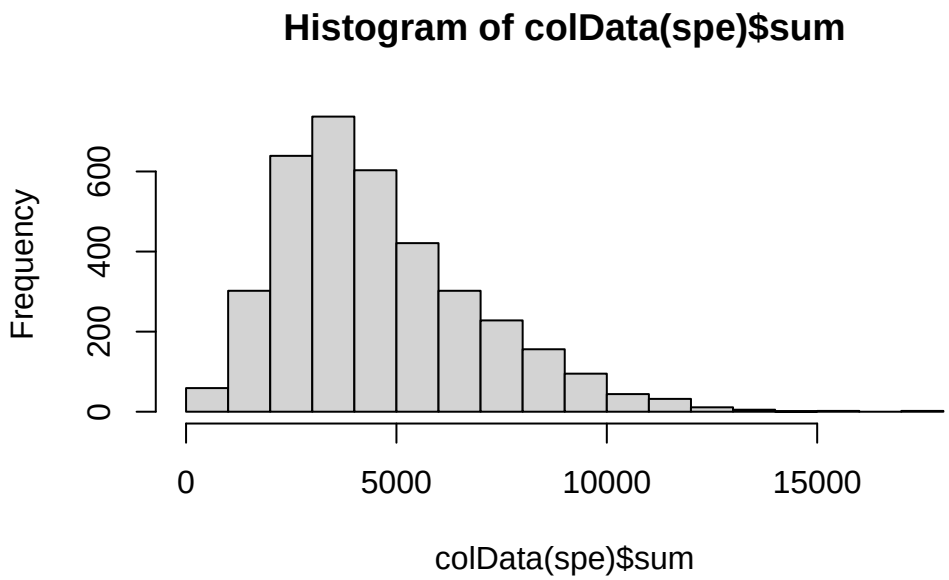
and (iv) number of cells per spot.

3.5.1 Library size

Library size represents the total sum of UMI counts per spot. This is included in the column labeled `sum` in the `scater` output.

Plot a histogram of the library sizes across spots.

```
# histogram of library sizes
hist(colData(spe)$sum, breaks = 20)
```

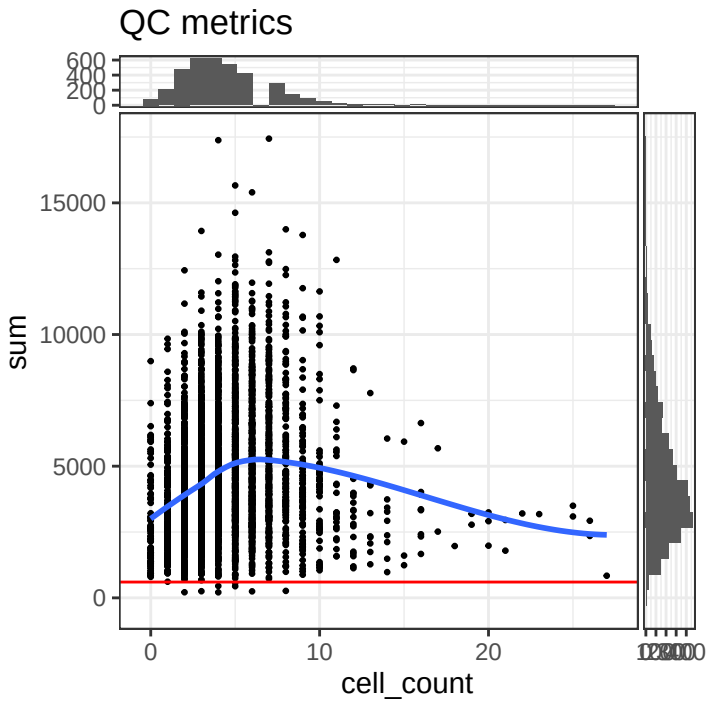


The distribution is relatively smooth, and there are no obvious issue such as a spike at very low library sizes.

We also plot the library sizes against the number of cells per spot (which is available for this dataset). This is to check that we are not inadvertently removing a biologically meaningful group of spots. The horizontal line (argument `threshold`) shows our first guess at a possible filtering threshold for library size based on the histogram.

```
# plot library size vs. number of cells per spot
plotQC(spe, type = "scatter",
       metric_x = "cell_count", metric_y = "sum",
       threshold_y = 600)
```

```
`geom_smooth()` using formula = 'y ~ x'
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



The plot shows that setting a filtering threshold for library size (e.g. at the value shown) does not appear to select for any obvious biologically consistent group of spots.

We set a relatively arbitrary threshold of 600 UMI counts per spot, and then check the number of spots below this threshold.

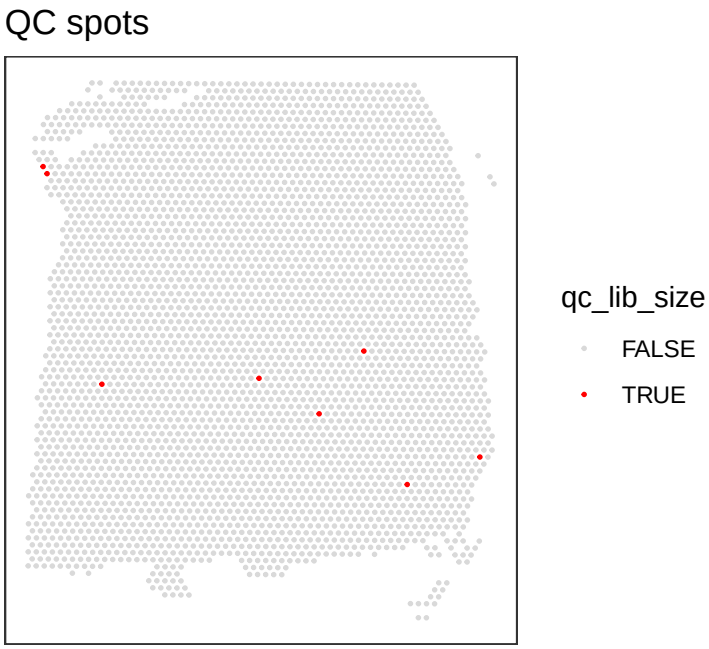
```
# select QC threshold for library size
qc_lib_size <- colData(spe)$sum < 600
table(qc_lib_size)
```

```
qc_lib_size
FALSE  TRUE
 3631     8
```

```
colData(spe)$qc_lib_size <- qc_lib_size
```

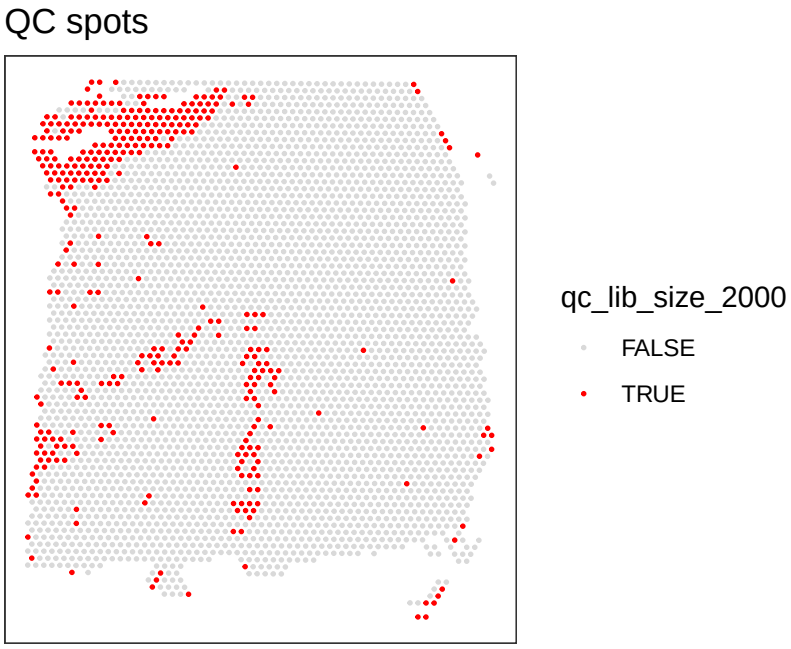
Finally, we also check that the discarded spots do not have any obvious spatial pattern that correlates with known biological features. Otherwise, removing these spots could indicate that we have set the threshold too high, and are removing biologically informative spots.

```
# check spatial pattern of discarded spots
plotQC(spe, type = "spots",
       discard = "qc_lib_size")
```



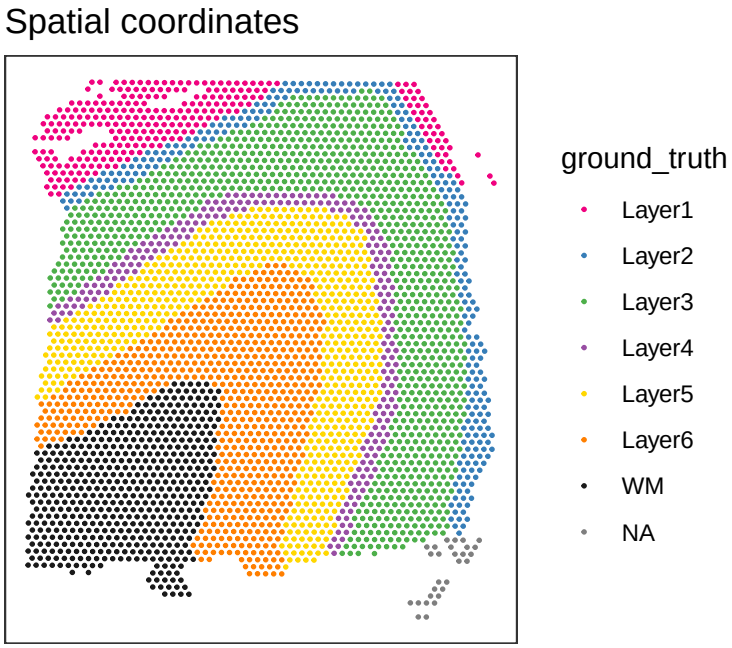
As an aside, here we can also illustrate what happens if we set the threshold too high. For example, if we set the threshold to 2000 UMI counts per spot – which may also seem like a reasonable value based on the histogram and scatterplot – then we see a possible spatial pattern in the discarded spots, matching the cortical layers. This illustrates the importance of interactively checking exploratory visualizations when choosing these thresholds.

```
# check spatial pattern of discarded spots if threshold is too high
qc_lib_size_2000 <- colData(spe)$sum < 2000
colData(spe)$qc_lib_size_2000 <- qc_lib_size_2000
plotQC(spe, type = "spots",
       discard = "qc_lib_size_2000")
```



For reference, here are the ground truth (manually annotated) cortical layers in this dataset.

```
# plot ground truth (manually annotated) layers
plotSpots(spe, annotate = "ground_truth",
          palette = "libd_layer_colors")
```

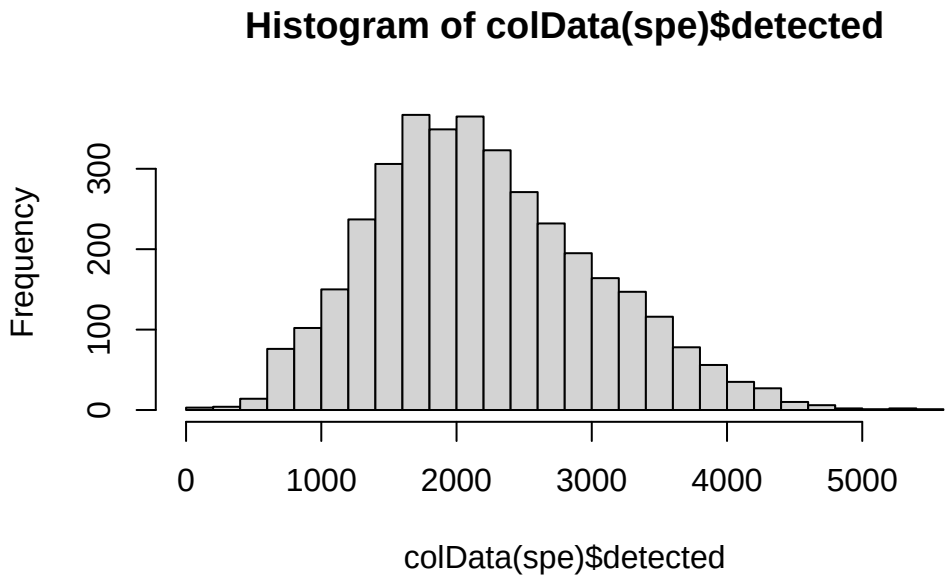


3.5.2 Number of expressed features

The number of expressed features refers to the number of genes with non-zero UMI counts per spot. This is stored in the column `detected` in the `scater` output.

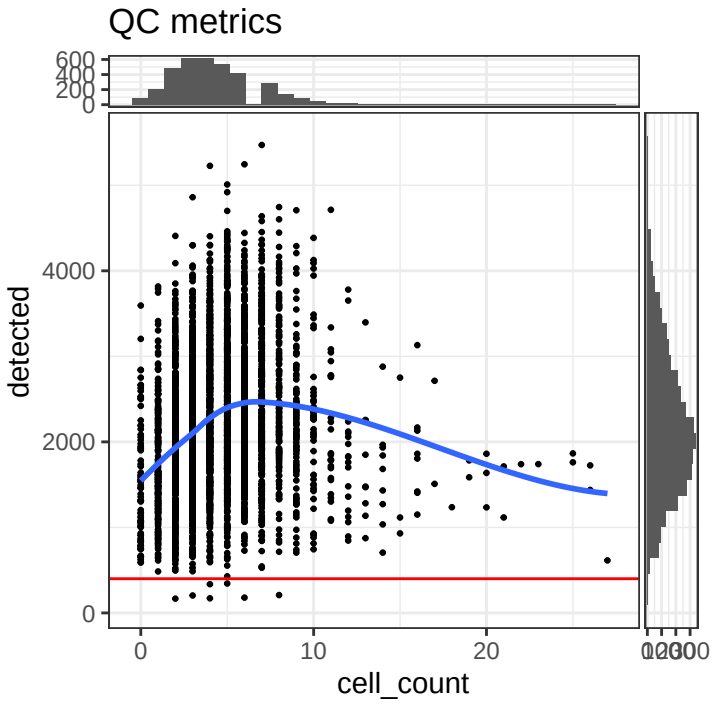
We use a similar sequence of visualizations to choose a threshold for this QC metric.

```
# histogram of numbers of expressed genes
hist(colData(spe)$detected, breaks = 20)
```




```
# plot number of expressed genes vs. number of cells per spot
plotQC(spe, type = "scatter",
       metric_x = "cell_count", metric_y = "detected",
       threshold_y = 400)
```

```
`geom_smooth()` using formula = 'y ~ x'
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



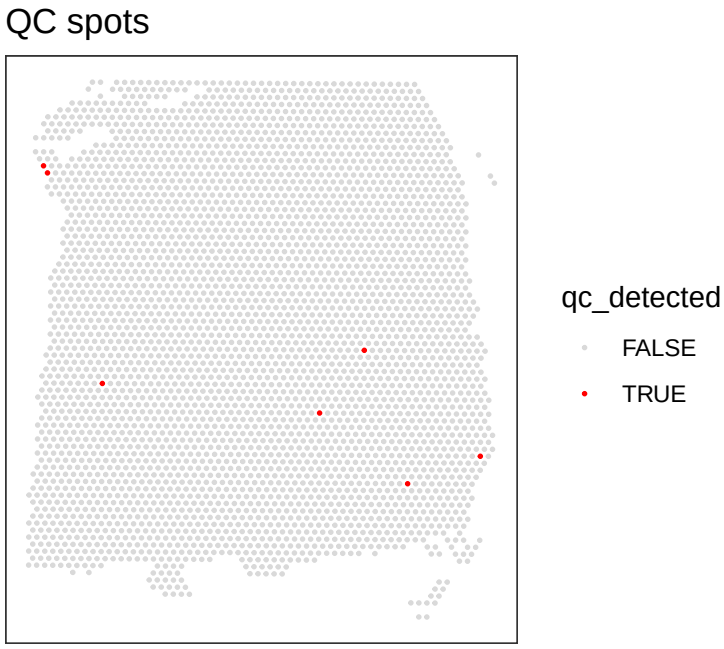
Based on the plots, we select a threshold of 400 expressed genes per spot.

```
# select QC threshold for number of expressed genes
qc_detected <- colData(spe)$detected < 400
table(qc_detected)
```

```
qc_detected
FALSE  TRUE
 3632     7
```

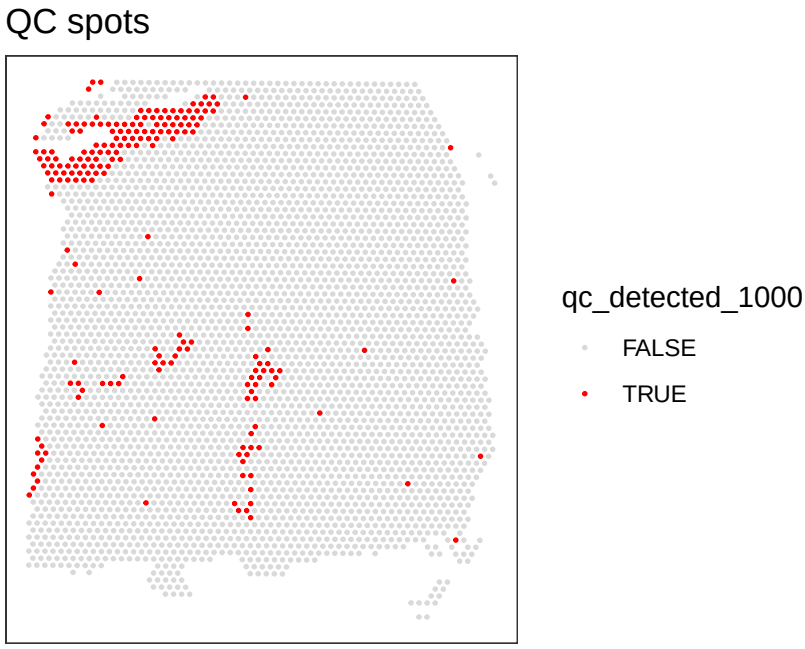
```
colData(spe)$qc_detected <- qc_detected
```

```
# check spatial pattern of discarded spots
plotQC(spe, type = "spots",
       discard = "qc_detected")
```



Again, we also check what happens when we set the threshold too high.

```
# check spatial pattern of discarded spots if threshold is too high
qc_detected_1000 <- colData(spe)$detected < 1000
colData(spe)$qc_detected_1000 <- qc_detected_1000
plotQC(spe, type = "spots",
       discard = "qc_detected_1000")
```

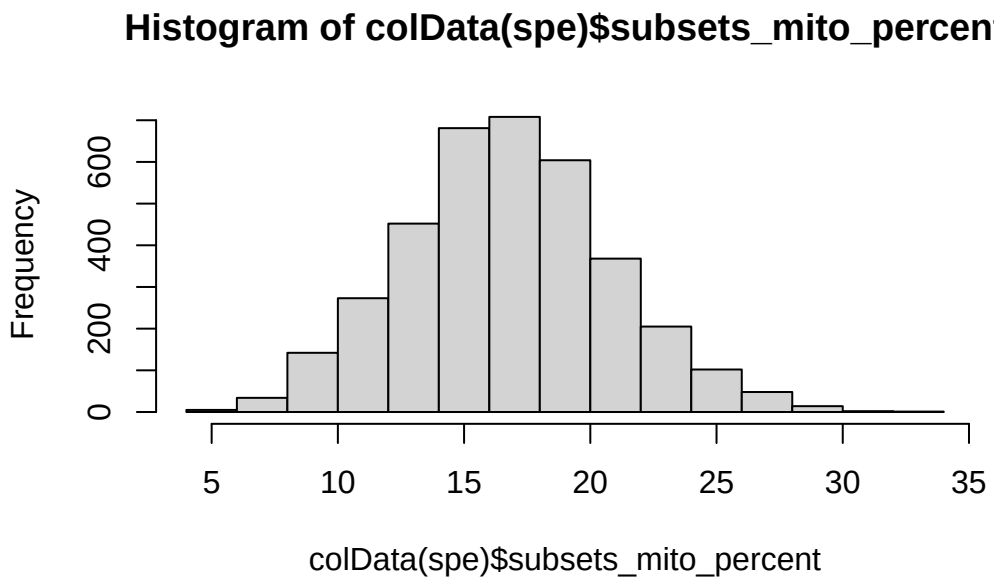


3.5.3 Proportion of mitochondrial reads

A high proportion of mitochondrial reads indicates cell damage.

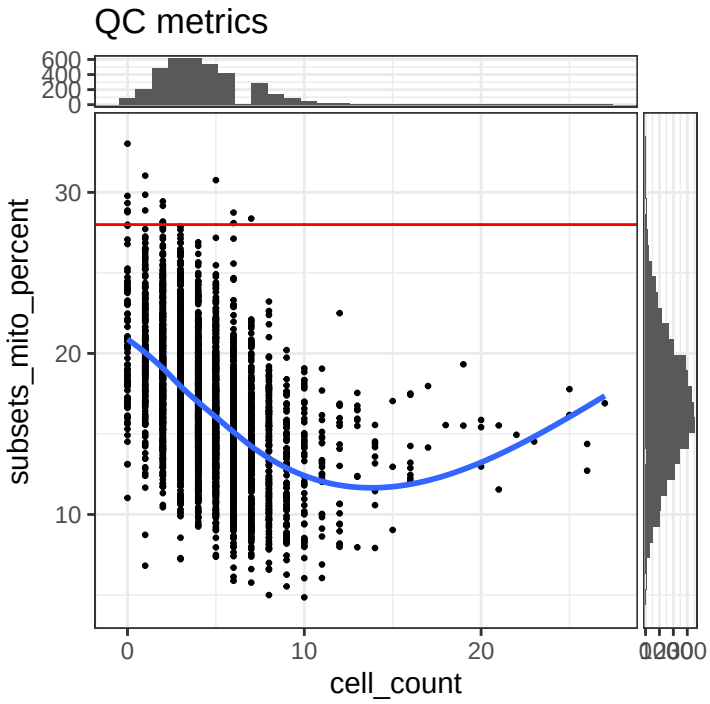
We investigate the proportions of mitochondrial reads across spots, and select an appropriate threshold. The proportions of mitochondrial reads per spot are stored in the column `subsets_mito_percent` in the `scater` output.

```
# histogram of mitochondrial read proportions
hist(colData(spe)$subsets_mito_percent, breaks = 20)
```




```
# plot mitochondrial read proportion vs. number of cells per spot
plotQC(spe, type = "scatter",
       metric_x = "cell_count", metric_y = "subsets_mito_percent",
       threshold_y = 28)
```

```
`geom_smooth()` using formula = 'y ~ x'
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



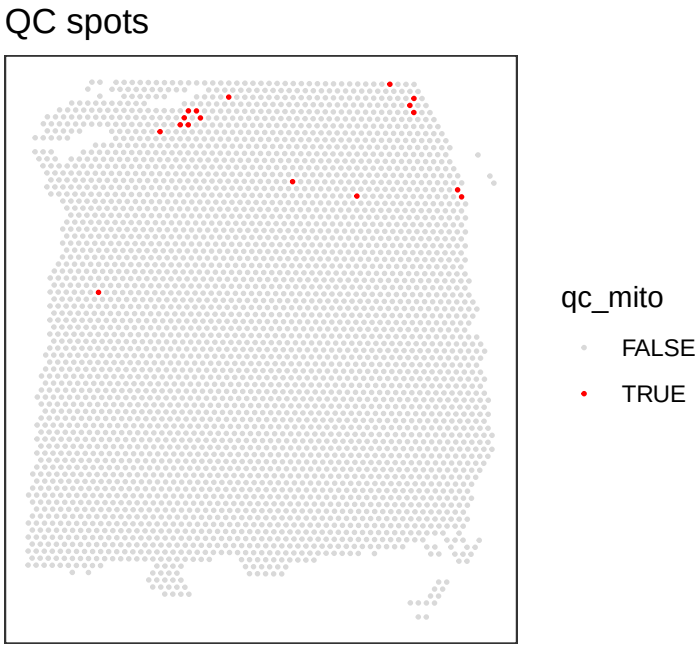
We select a threshold of 28% for the mitochondrial read proportion.

```
# select QC threshold for mitochondrial read proportion
qc_mito <- colData(spe)$subsets_mito_percent > 28
table(qc_mito)
```

```
qc_mito
FALSE  TRUE
 3622    17
```

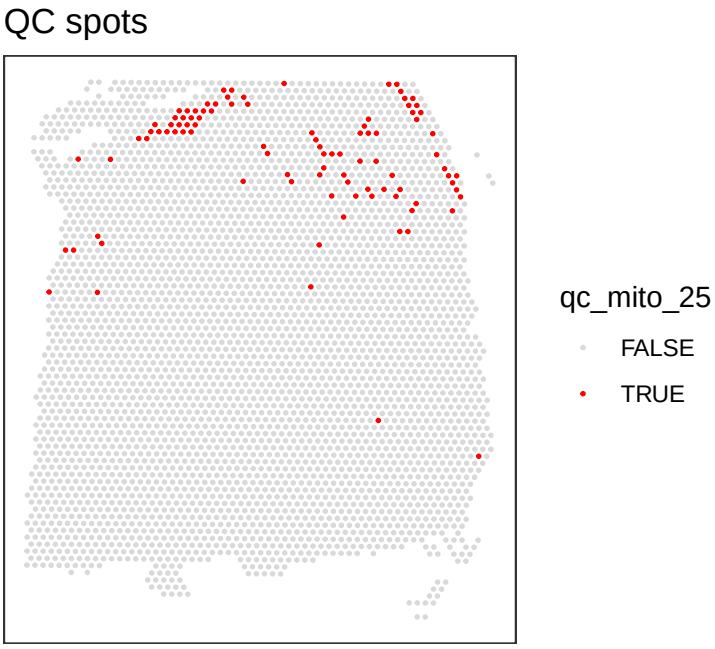
```
colData(spe)$qc_mito <- qc_mito
```

```
# check spatial pattern of discarded spots
plotQC(spe, type = "spots",
       discard = "qc_mito")
```



We also check what happens when we set the threshold too low.

```
# check spatial pattern of discarded spots if threshold is too high
qc_mito_25 <- colData(spe)$subsets_mito_percent > 25
colData(spe)$qc_mito_25 <- qc_mito_25
plotQC(spe, type = "spots",
       discard = "qc_mito_25")
```

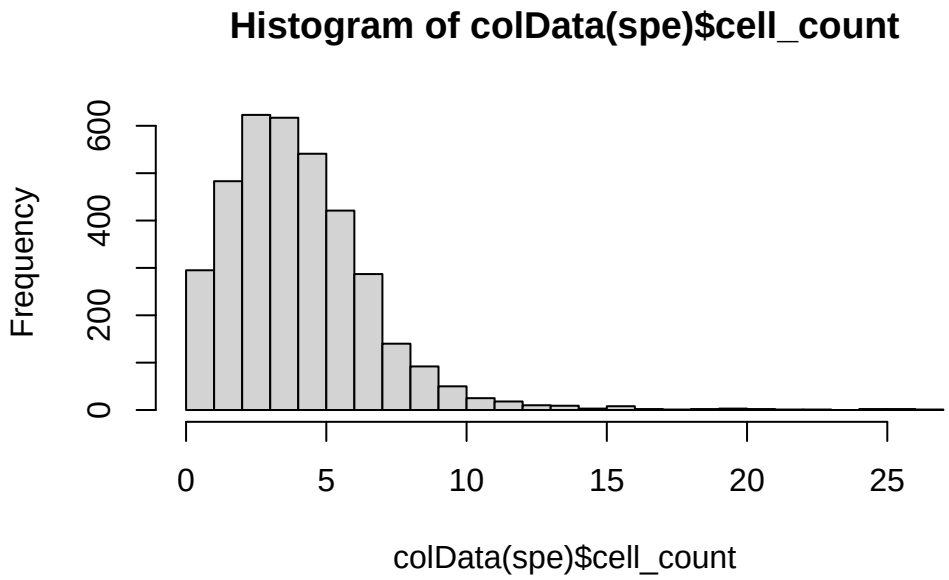


3.5.4 Number of cells per spot

The number of cells per spot depends on the tissue type and organism.

Here, we check for any outlier values that could indicate problems during cell segmentation.

```
# histogram of cell counts
hist(colData(spe)$cell_count, breaks = 20)
```

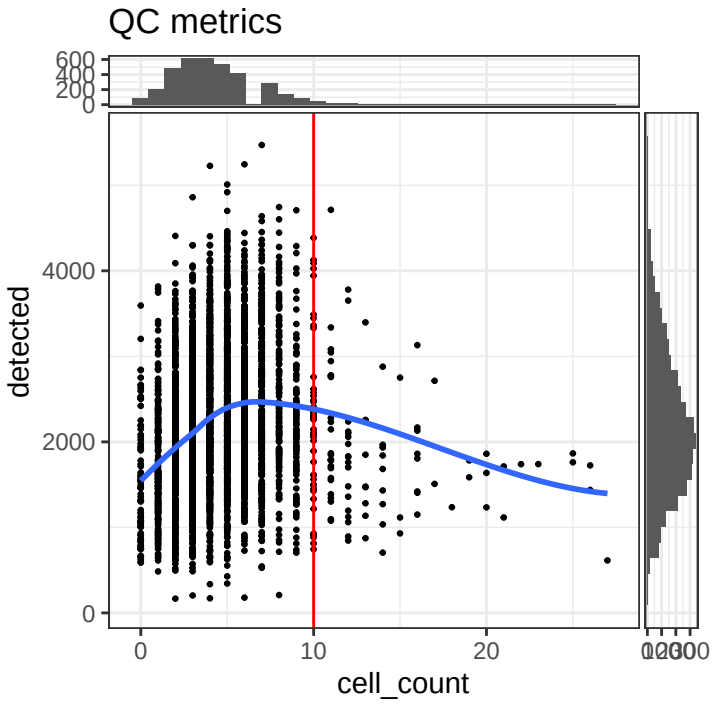



```
# distribution of cells per spot
tbl_cells_per_spot <- table(colData(spe)$cell_count)
```

We see a tail of very high values, which could indicate problems for these spots. These values are also visible on the scatterplots. Here, we again plot the number of expressed genes vs. cell count, with an added trend.

```
# plot number of expressed genes vs. number of cells per spot
plotQC(spe, type = "scatter",
       metric_x = "cell_count", metric_y = "detected",
       threshold_x = 10)
```

```
`geom_smooth()` using formula = 'y ~ x'
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



In particular, we see that the spots with very high cell counts also have low numbers of expressed genes. This indicates that the experiments have failed for these spots, and they should be removed.

We select a threshold of 10 cells per spot. The number of spots above this threshold is relatively small, and there is a clear downward trend in the number of expressed genes above this threshold.

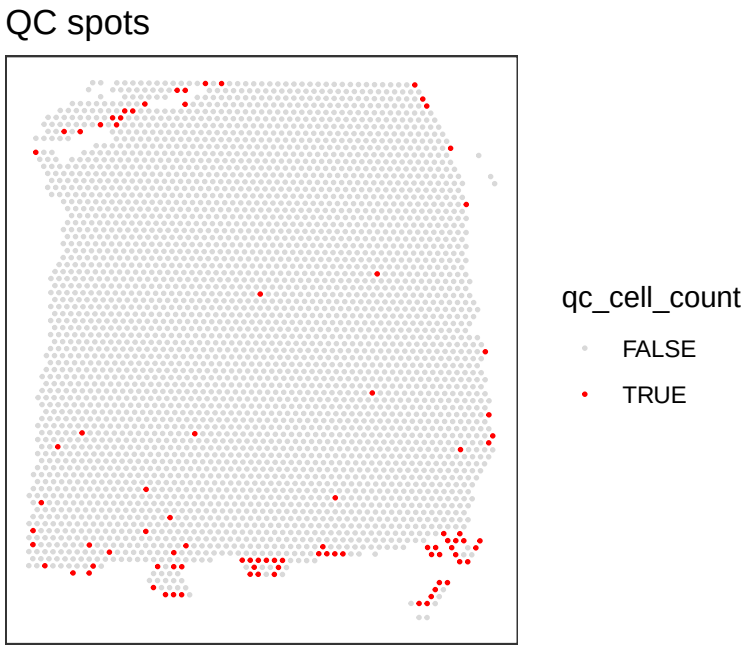
```
# select QC threshold for number of cells per spot
qc_cell_count <- colData(spe)$cell_count > 10
table(qc_cell_count)
```

```
qc_cell_count
FALSE  TRUE
 3549    90
```

```
colData(spe)$qc_cell_count <- qc_cell_count
```

```
# check spatial pattern of discarded spots
plotQC(spe, type = "spots",
```

```
        discard = "qc_cell_count")
```



While there is a spatial pattern to the discarded spots, it does not appear to be correlated with the known biological features (cortical layers). The discarded spots are all on the edges of the tissue. It seems plausible that something has gone wrong with the cell segmentation on the edges of the images, so it makes sense to remove these spots.

3.5.5 Remove low-quality spots

Now that we have calculated several QC metrics and selected thresholds for each one, we can combine the sets of low-quality spots, and remove them from our object.

We also check again that the combined set of discarded spots does not correspond to any obvious biologically relevant group of spots.

```
# number of discarded spots for each metric
apply(cbind(qc_lib_size, qc_detected, qc_mito, qc_cell_count), 2, sum)
```

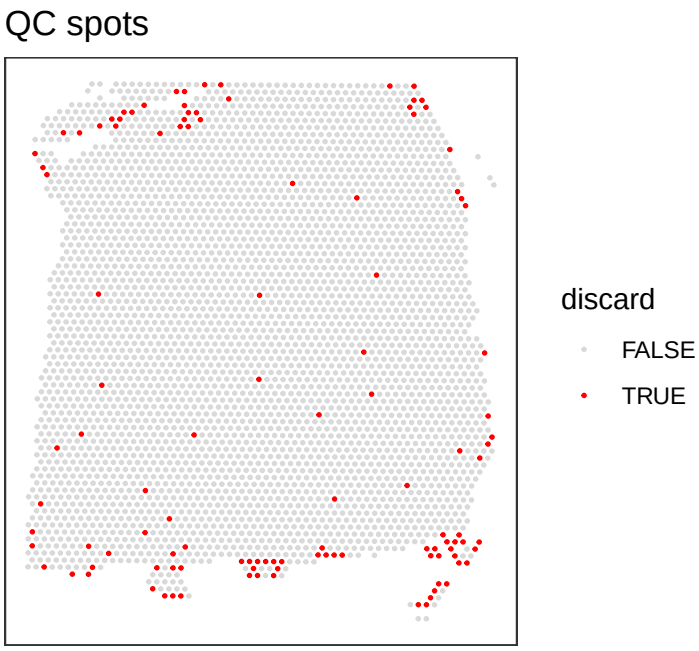
```
qc_lib_size   qc_detected       qc_mito qc_cell_count
          8             7            17            90
```

```
# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
table(discard)
```

```
discard
FALSE  TRUE
 3524   115
```

```
# store in object
colData(spe)$discard <- discard
```

```
# check spatial pattern of combined set of discarded spots
plotQC(spe, type = "spots",
       discard = "discard")
```




```
# remove combined set of low-quality spots
spe <- spe[, !colData(spe)$discard]
dim(spe)
```

```
[1] 33538  3524
```

3.6 Zero-cell and single-cell spots

A particular characteristic of Visium data is that spots can contain zero, one, or multiple cells.

We could also imagine other filtering procedures such as (i) removing spots with zero cells, or (ii) restricting the analysis to spots containing a single cell (which would make the data more similar to scRNA-seq).

However, this would discard a large amount of information. Below, we show the distribution of cells per spot again (up to the filtering threshold of 10 cells per spot from above).

```
# distribution of cells per spot
tbl_cells_per_spot[1:13]
```

```
 0   1   2   3   4   5   6   7   8   9  10  11  12
84 211 483 623 617 541 421 287 140  92  50  25  18
```

```
# as proportions
prop_cells_per_spot <- round(tbl_cells_per_spot / sum(tbl_cells_per_spot), 2)
prop_cells_per_spot[1:13]
```

```
   0    1    2    3    4    5    6    7    8    9   10   11   12
0.02 0.06 0.13 0.17 0.17 0.15 0.12 0.08 0.04 0.03 0.01 0.01 0.00
```

Only 6% of spots contain a single cell. If we restricted the analysis to these spots only, we would be discarding most of the data.

Removing the spots containing zero cells (2% of spots) would also be problematic, since these spots can also contain biologically meaningful information. For example, in this brain dataset, the regions between cell bodies consists of neuropil (dense networks of axons and dendrites). In our paper (Maynard et al. 2021), we explore the information contained in these neuropil spots.

3.7 Quality control at gene level

The sections above consider quality control at the spot level. In some datasets, it may also be appropriate to apply quality control procedures or filtering at the gene level. For example, certain genes may be biologically irrelevant for downstream analyses.

However, here we make a distinction between quality control and feature selection. Removing biologically uninteresting genes (such as mitochondrial genes) may also be considered as part of feature selection, since there is no underlying experimental procedure that has failed. Therefore, we will discuss gene-level filtering in the Feature selection chapter.

4 Normalization

4.1 Background

Here we apply normalization methods developed for scRNA-seq data, treating each spot as equivalent to one cell.

4.2 Previous steps

Code to run steps from the previous chapters to generate the ***SpatialExperiment*** *object required for this chapter.*

```
# LOAD DATA

library(SpatialExperiment)
library(STexampleData)
spe <- Visium_humanDLPFC()

# QUALITY CONTROL (QC)

library(scater)
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
# calculate per-spot QC metrics
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
# select QC thresholds
qc_lib_size <- colData(spe)$sum < 600
qc_detected <- colData(spe)$detected < 400
qc_mito <- colData(spe)$subsets_mito_percent > 28
qc_cell_count <- colData(spe)$cell_count > 10
# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
```

```
colData(spe)$discard <- discard
# filter low-quality spots
spe <- spe[, !colData(spe)$discard]
```

4.3 Logcounts

Calculate log-transformed normalized counts (abbreviated as "logcounts") using library size factors.

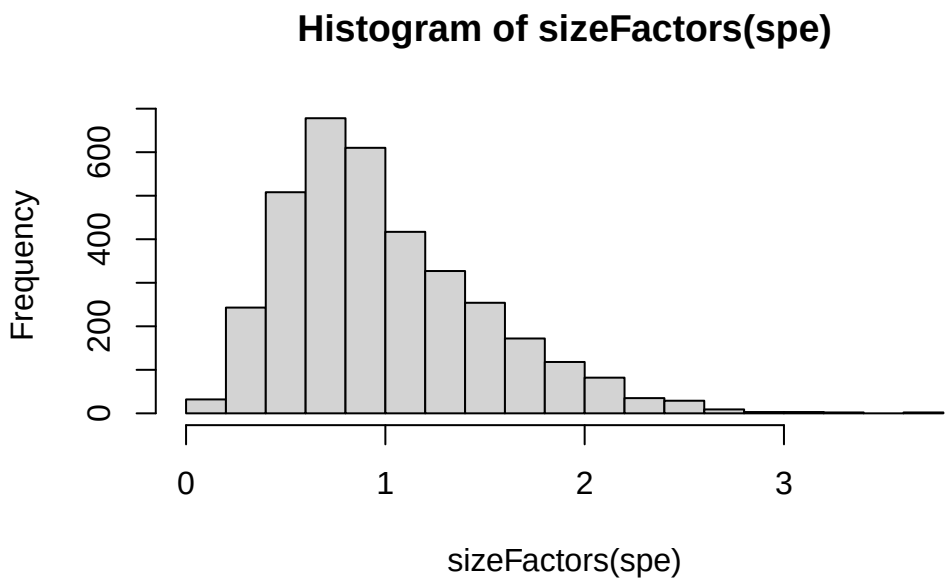
We apply the methods implemented in the `scater` (McCarthy et al. 2017) and `scran` (Lun, McCarthy, and Marioni 2016) packages, which were originally developed for scRNA-seq data, making the assumption here that these methods can be applied to SRT data by treating spots as equivalent to cells.

We use the library size factors methodology since this is the simplest approach, and can easily be applied to SRT data. Alternative approaches that are populare for scRNA-seq data, including normalization by deconvolution, are more difficulty to justify in the context of spot-based SRT data since (i) spots may contain multiple cells from more than one cell type, and (ii) datasets can contain multiple samples (e.g. multiple Visium slides, resulting in sample-specific clustering).

```
library(scran)

# calculate library size factors
spe <- computeLibraryFactors(spe)

summary(sizeFactors(spe))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1321  0.6312  0.9000  1.0000  1.2849  3.7582
```

```
hist(sizeFactors(spe), breaks = 20)
```




```
# calculate logcounts and store in object
spe <- logNormCounts(spe)

# check
assayNames(spe)
```

```
[1] "counts"    "logcounts"
```

```
dim(counts(spe))
```

```
[1] 33538  3524
```

```
dim(logcounts(spe))
```

```
[1] 33538  3524
```

5 Feature selection

5.1 Background

Here we apply feature selection methods to identify highly variable genes (HVGs) or spatially variable genes (SVGs), which can then be investigated individually or used as the input for further downstream analyses.

5.2 Previous steps

Code to run steps from the previous chapters to generate the ***SpatialExperiment*** *object required for this chapter.*

```
# LOAD DATA

library(SpatialExperiment)
library(STexampleData)
spe <- Visium_humanDLPFC()

# QUALITY CONTROL (QC)

library(scater)
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
# calculate per-spot QC metrics
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
# select QC thresholds
qc_lib_size <- colData(spe)$sum < 600
qc_detected <- colData(spe)$detected < 400
qc_mito <- colData(spe)$subsets_mito_percent > 28
qc_cell_count <- colData(spe)$cell_count > 10
# combined set of discarded spots
```

```
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
colData(spe)$discard <- discard
# filter low-quality spots
spe <- spe[, !colData(spe)$discard]

# NORMALIZATION

library(scran)
# calculate logcounts using library size factors
spe <- logNormCounts(spe)
```

5.3 Highly variable genes (HVGs)

We use methods from `scran` (Lun et al. 2016) to identify a set of top highly variable genes (HVGs), which can be used to define major cell types. These methods were originally developed for single-cell RNA sequencing (scRNA-seq) data, so here we are making the assumption that spots can be treated as equivalent to cells.

Note that HVGs are defined based only on molecular features (i.e. gene expression), and do not take any spatial information into account. If the biologically meaningful spatial information in this dataset mainly reflects spatial distributions of major cell types, then relying on HVGs for downstream analyses may be sufficient. But if there are additional important spatial features in the dataset, then it may be more meaningful to investigate spatially variable genes (SVGs).

To identify HVGs, we first remove mitochondrial genes, since these are very highly expressed in this dataset and are not of main biological interest.

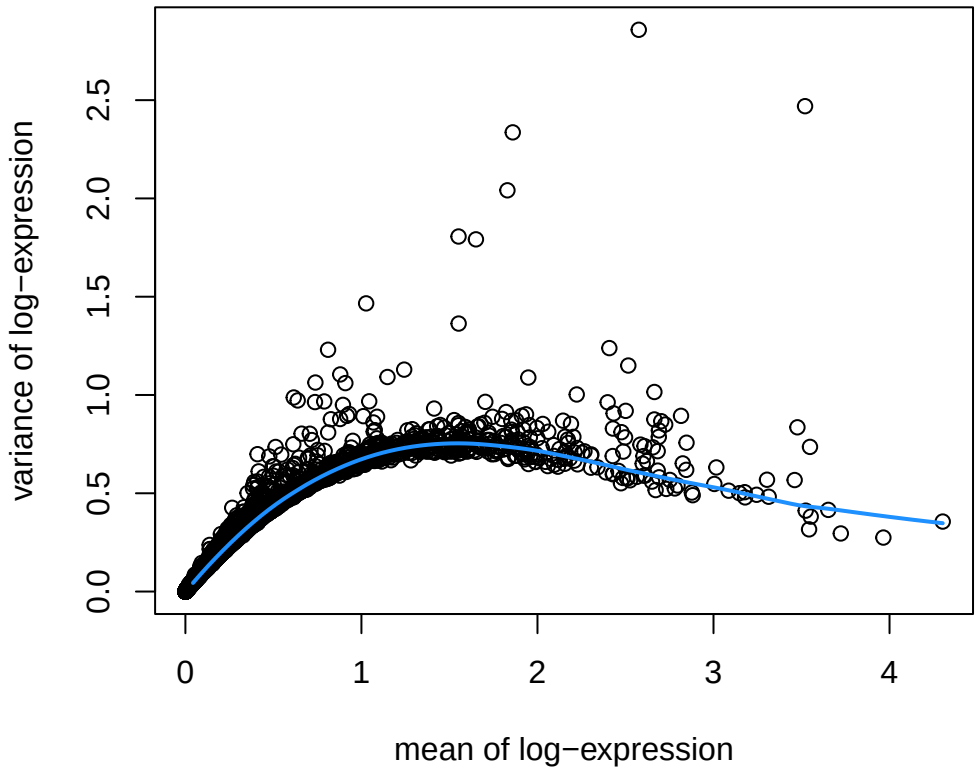
```
# remove mitochondrial genes
spe <- spe[!is_mito, ]
dim(spe)
```

```
[1] 33525  3524
```

Then, we apply methods from `scran`. This gives us a list of HVGs, which can be used for further downstream analyses. The parameter `prop` defines how many HVGs we want. For example `prop = 0.1` returns the top 10% of genes.

```
library(scran)

# fit mean-variance relationship
```

```
dec <- modelGeneVar(spe)

# visualize mean-variance relationship
fit <- metadata(dec)
plot(fit$mean, fit$var,
     xlab = "mean of log-expression", ylab = "variance of log-expression")
curve(fit$trend(x), col = "dodgerblue", add = TRUE, lwd = 2)
```




```
# select top HVGs
top_hvgs <- getTopHVGs(dec, prop = 0.1)
length(top_hvgs)
```

```
[1] 1438
```

5.4 Spatially variable genes (SVGs)

Alternatively, we can apply methods to identify spatially variable genes (SVGs) instead of HVGs. SVGs are defined as genes with a spatially correlated patterns of expression.

Simple approaches to identify SVGs include standard statistical measures such as Moran's I statistic or Geary's C statistic, which can be used to rank genes by the observed spatial autocorrelation.

Sophisticated statistical methods to identify SVGs adapted to the properties of SRT data have also been developed, including the following:

- **nnSVG**: available as an R package from Bioconductor and described by Weber et al. (2023)
- **SPARK-X**: available as an R package from GitHub and described by Zhou et al. (2021)
- **SPARK**: available as an R package from GitHub and described by Sun et al. (2020)
- **SpatialDE**: available as a Python package from GitHub and described by Svensson et al. (2018)

6 Dimensionality reduction

6.1 Background

In this chapter, we apply dimensionality reduction methods to visualize the data and to generate inputs for further downstream analyses.

6.2 Previous steps

Code to run steps from the previous chapters to generate the ***SpatialExperiment*** *object required for this chapter.*

```
# LOAD DATA

library(SpatialExperiment)
library(STexampleData)
spe <- Visium_humanDLPFC()

# QUALITY CONTROL (QC)

library(scater)
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
# calculate per-spot QC metrics
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
# select QC thresholds
qc_lib_size <- colData(spe)$sum < 600
qc_detected <- colData(spe)$detected < 400
qc_mito <- colData(spe)$subsets_mito_percent > 28
qc_cell_count <- colData(spe)$cell_count > 10
# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
```

```
colData(spe)$discard <- discard
# filter low-quality spots
spe <- spe[, !colData(spe)$discard]

# NORMALIZATION

library(scran)
# calculate logcounts using library size factors
spe <- logNormCounts(spe)

# FEATURE SELECTION

# remove mitochondrial genes
spe <- spe[!is_mito, ]
# fit mean-variance relationship
dec <- modelGeneVar(spe)
# select top HVGs
top_hvgs <- getTopHVGs(dec, prop = 0.1)
```

6.3 Principal component analysis (PCA)

Apply principal component analysis (PCA) to the set of top highly variable genes (HVGs) to reduce the dimensionality of the dataset, and retain the top 50 principal components (PCs) for further downstream analyses.

This is done for two reasons: (i) to reduce noise due to random variation in expression of biologically uninteresting genes, which are assumed to have expression patterns that are independent of each other, and (ii) to improve computational efficiency during downstream analyses.

We use the computationally efficient implementation of PCA provided in the `scater` package (McCarthy et al. 2017). This implementation uses randomization, and therefore requires setting a random seed for reproducibility.

```
# compute PCA
set.seed(123)
spe <- runPCA(spe, subset_row = top_hvgs)

reducedDimNames(spe)
```

```
[1] "PCA"
```

```
dim(reducedDim(spe, "PCA"))
```

```
[1] 3524   50
```

6.4 Uniform Manifold Approximation and Projection (UMAP)

We also run UMAP (McInnes, Healy, and Melville 2018) on the set of top 50 PCs and retain the top 2 UMAP components, which will be used for visualization purposes.

```
# compute UMAP on top 50 PCs
set.seed(123)
spe <- runUMAP(spe, dimred = "PCA")

reducedDimNames(spe)
```

```
[1] "PCA"  "UMAP"
```

```
dim(reducedDim(spe, "UMAP"))
```

```
[1] 3524    2
```

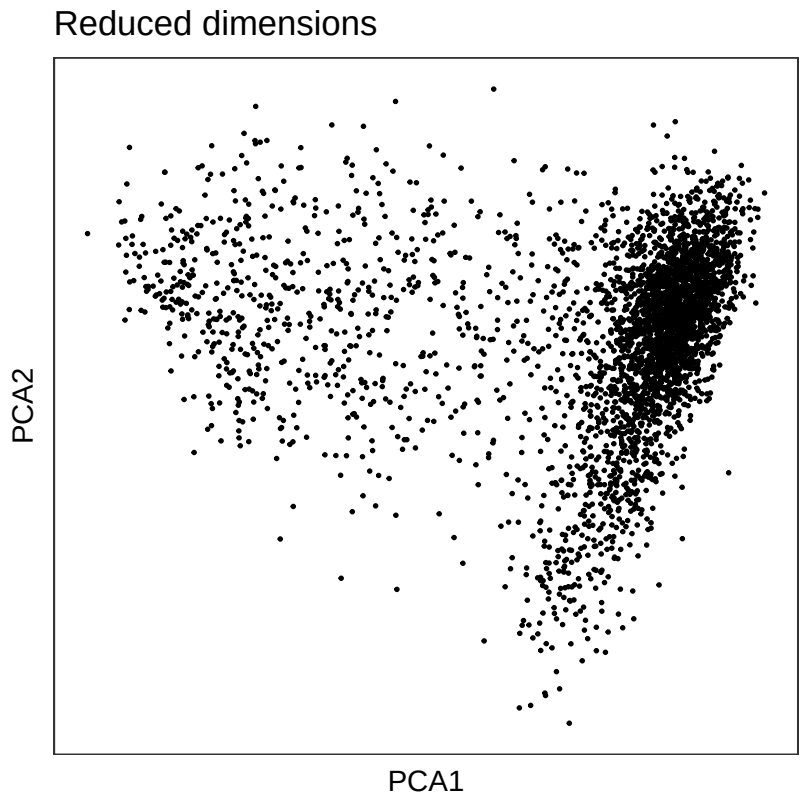
```
# update column names for easier plotting
colnames(reducedDim(spe, "UMAP")) <- paste0("UMAP", 1:2)
```

6.5 Visualizations

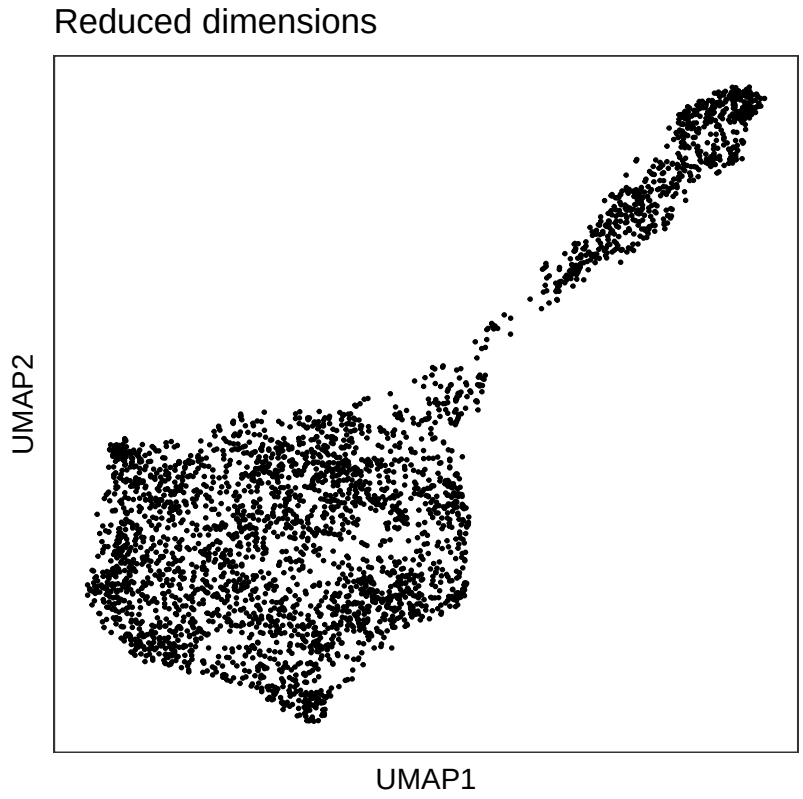
Generate plots using plotting functions from the ggspavis package. In the next chapter on clustering, we will add cluster labels to these reduced dimension plots.

```
library(ggspavis)
```

```
# plot top 2 PCA dimensions
plotDimRed(spe, type = "PCA")
```




```
# plot top 2 UMAP dimensions
plotDimRed(spe, type = "UMAP")
```


Reduced dimensions
UMAP2
UMAP1

7 Clustering

7.1 Overview

Clustering is used in single-cell and spatial analysis to identify cell types.

Cell types and subtypes can be defined at various resolutions, depending on biological context. For the purposes of clustering, this means that the desired number of clusters also depends on biological context.

In the spatial context, we may be interested in e.g. (i) identifying cell types or subtypes that occur in biologically interesting spatial patterns, or (ii) identifying major cell types and performing subsequent analyses within these cell types.

7.2 Previous steps

Code to run steps from the previous chapters to generate the ***SpatialExperiment*** *object required for this chapter.*

```
# LOAD DATA

library(SpatialExperiment)
library(STexampleData)
spe <- Visium_humanDLPFC()

# QUALITY CONTROL (QC)

library(scater)
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
# calculate per-spot QC metrics
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
# select QC thresholds
```

```r
qc_lib_size <- colData(spe)$sum < 600
qc_detected <- colData(spe)$detected < 400
qc_mito <- colData(spe)$subsets_mito_percent > 28
qc_cell_count <- colData(spe)$cell_count > 10
# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
colData(spe)$discard <- discard
# filter low-quality spots
spe <- spe[, !colData(spe)$discard]

# NORMALIZATION

library(scran)
# calculate logcounts using library size factors
spe <- logNormCounts(spe)

# FEATURE SELECTION

# remove mitochondrial genes
spe <- spe[!is_mito, ]
# fit mean-variance relationship
dec <- modelGeneVar(spe)
# select top HVGs
top_hvgs <- getTopHVGs(dec, prop = 0.1)

# DIMENSIONALITY REDUCTION

# compute PCA
set.seed(123)
spe <- runPCA(spe, subset_row = top_hvgs)
# compute UMAP on top 50 PCs
set.seed(123)
spe <- runUMAP(spe, dimred = "PCA")
# update column names
colnames(reducedDim(spe, "UMAP")) <- paste0("UMAP", 1:2)
```

7.3 Non-spatial clustering on HVGs

We can perform clustering by applying standard clustering methods developed for single-cell RNA sequencing data, using molecular features (gene expression). Here, we apply graph-based

clustering using the Walktrap method implemented in `scran` (Lun et al. 2016), applied to the top 50 PCs calculated on the set of top HVGs.

In the context of spatial data, this means we assume that biologically informative spatial distribution patterns of cell types can be detected from the molecular features (gene expression).

```
# graph-based clustering
set.seed(123)
k <- 10
g <- buildSNNGraph(spe, k = k, use.dimred = "PCA")
g_walk <- igraph::cluster_walktrap(g)
clus <- g_walk$membership
table(clus)
```

```
clus
   1    2    3    4    5    6    7
 338  312 1146  978  274  116  360
```

```
# store cluster labels in column 'label' in colData
colLabels(spe) <- factor(clus)
```

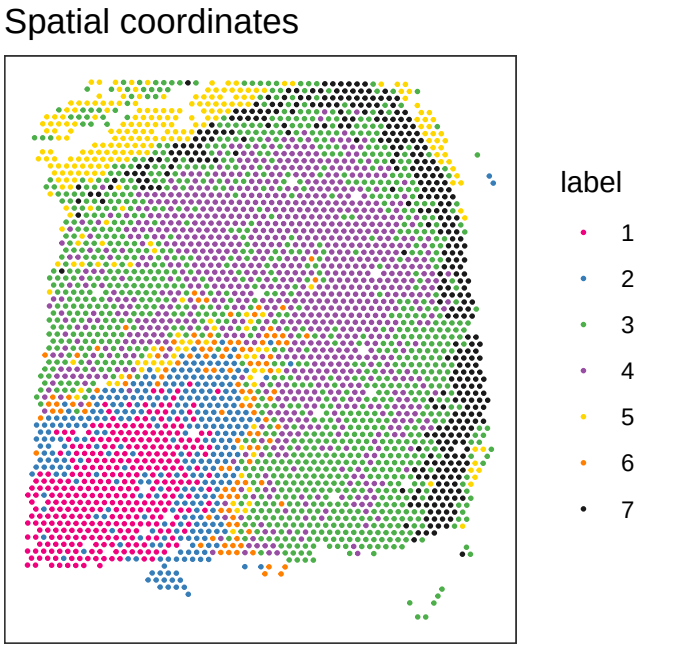
Visualize the clusters by plotting in (i) spatial (x-y) coordinates on the tissue slide, and (ii) reduced dimension space (PCA or UMAP). We use plotting functions from the ggspavis package.

For reference, we also display the ground truth (manually annotated) labels available for this dataset (in spatial coordinates).

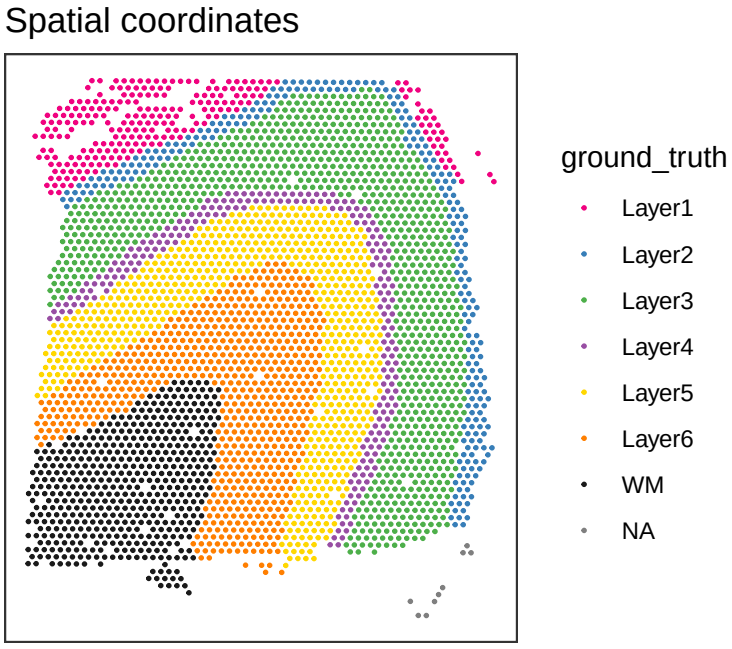
From the visualizations, we can see that the clustering reproduces the known biological structure (cortical layers), although not perfectly. The clusters are also separated in UMAP space, but again not perfectly.

```
library(ggspavis)
```

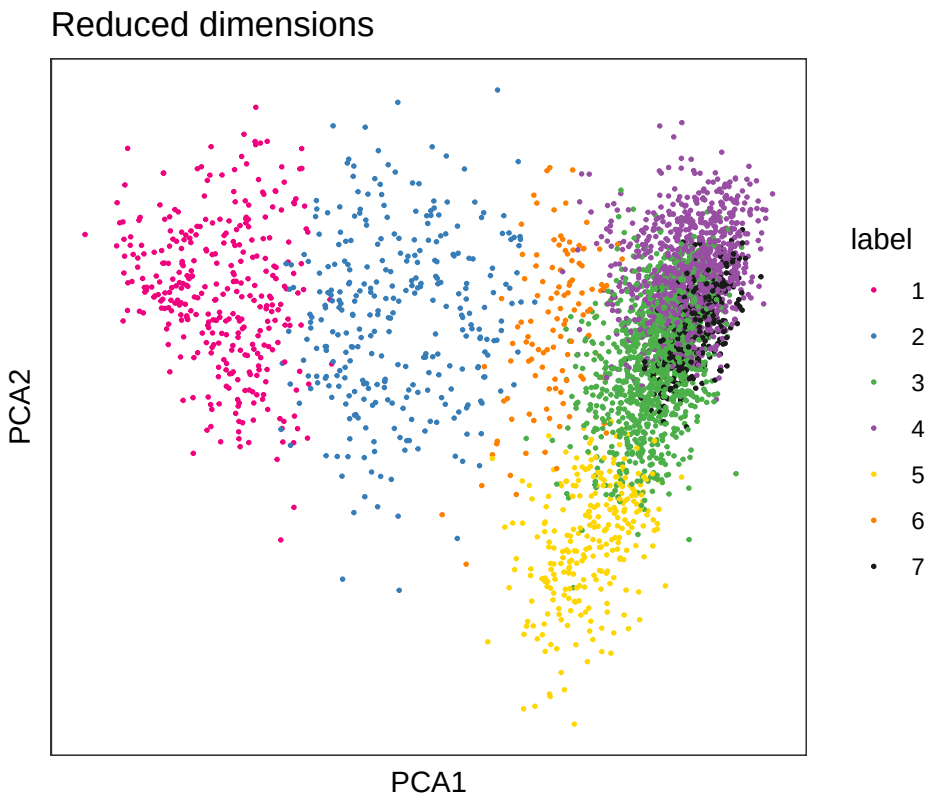
```
# plot clusters in spatial x-y coordinates
plotSpots(spe, annotate = "label",
          palette = "libd_layer_colors")
```



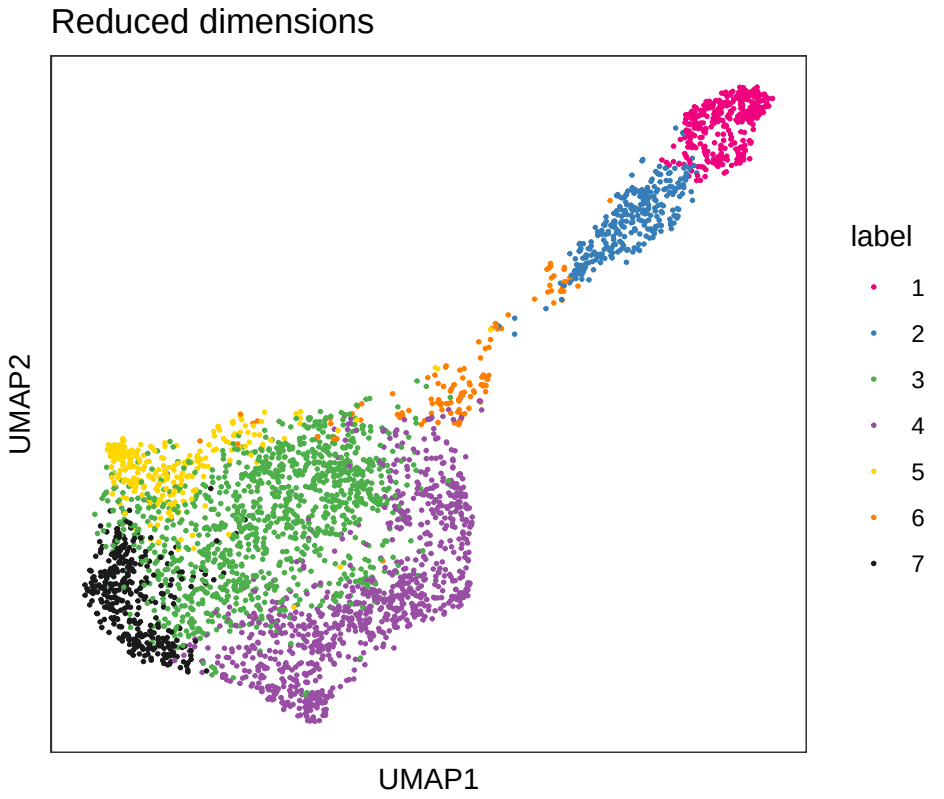

```
# plot ground truth labels in spatial coordinates
plotSpots(spe, annotate = "ground_truth",
          palette = "libd_layer_colors")
```



```
# plot clusters in PCA reduced dimensions
plotDimRed(spe, type = "PCA",
           annotate = "label", palette = "libd_layer_colors")
```




```
# plot clusters in UMAP reduced dimensions
plotDimRed(spe, type = "UMAP",
           annotate = "label", palette = "libd_layer_colors")
```



7.4 Spatially-aware clustering

In SRT data, we can also perform clustering that takes spatial information into account, for example to identify spatially compact or spatially connected clusters.

A simple strategy is to perform graph-based clustering on a set of features (columns) that includes both molecular features (gene expression) and spatial features (x-y coordinates). In this case, a crucial tuning parameter is the relative amount of scaling between the two data modalities – if the scaling is chosen poorly, either the molecular or spatial features will dominate the clustering. Depending on data availability, further modalities could also be included. In this section, we will include some examples on this clustering approach.

More sophisticated approaches for spatially-aware clustering include the following:

- **BayesSpace**: available as an R package from Bioconductor and described by Zhao et al. (2021)
- **SpaGCN**: available as a Python package from GitHub and described by Hu et al. (2021)
- **PRECAST**: available as an R package from CRAN and described by Liu and Liao et al. (2023)

8 Marker genes

8.1 Background

In this chapter, we identify marker genes for each cluster by testing for differential expression between clusters.

8.2 Previous steps

Code to run steps from the previous chapters to generate the `SpatialExperiment` object required for this chapter.

```
# LOAD DATA

library(SpatialExperiment)
library(STexampleData)
spe <- Visium_humanDLPFC()

# QUALITY CONTROL (QC)

library(scater)
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
# calculate per-spot QC metrics
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
# select QC thresholds
qc_lib_size <- colData(spe)$sum < 600
qc_detected <- colData(spe)$detected < 400
qc_mito <- colData(spe)$subsets_mito_percent > 28
qc_cell_count <- colData(spe)$cell_count > 10
# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
```

```r
colData(spe)$discard <- discard
# filter low-quality spots
spe <- spe[, !colData(spe)$discard]

# NORMALIZATION

library(scran)
# calculate logcounts using library size factors
spe <- logNormCounts(spe)

# FEATURE SELECTION

# remove mitochondrial genes
spe <- spe[!is_mito, ]
# fit mean-variance relationship
dec <- modelGeneVar(spe)
# select top HVGs
top_hvgs <- getTopHVGs(dec, prop = 0.1)

# DIMENSIONALITY REDUCTION

# compute PCA
set.seed(123)
spe <- runPCA(spe, subset_row = top_hvgs)
# compute UMAP on top 50 PCs
set.seed(123)
spe <- runUMAP(spe, dimred = "PCA")
# update column names
colnames(reducedDim(spe, "UMAP")) <- paste0("UMAP", 1:2)

# CLUSTERING

# graph-based clustering
set.seed(123)
k <- 10
g <- buildSNNGraph(spe, k = k, use.dimred = "PCA")
g_walk <- igraph::cluster_walktrap(g)
clus <- g_walk$membership
colLabels(spe) <- factor(clus)
```

8.3 Marker genes

Identify marker genes by testing for differential gene expression between clusters.

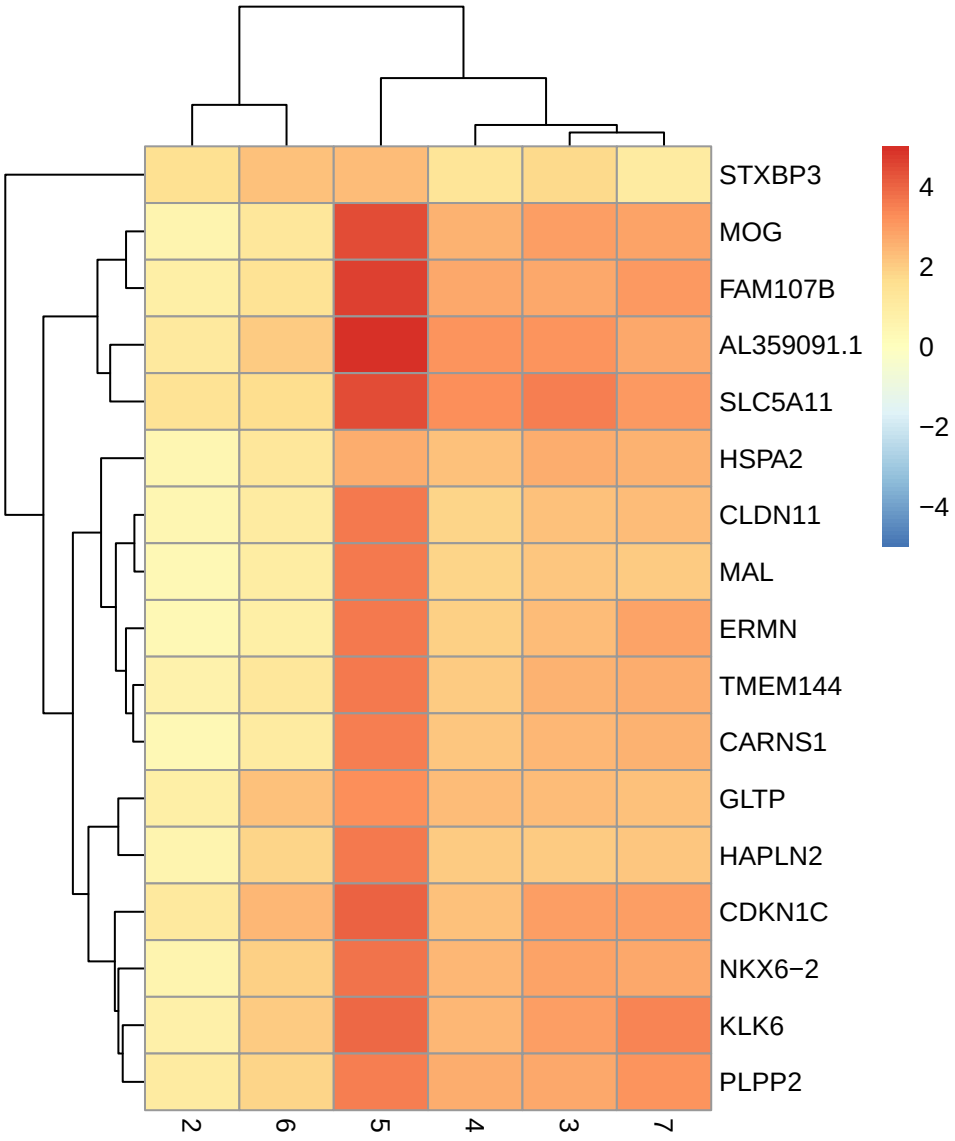
We use the `findMarkers` implementation in `scran` (Lun, McCarthy, and Marioni 2016), using a binomial test, which tests for genes that differ in the proportion expressed vs. not expressed between clusters. This is a more stringent test than the default t-tests, and tends to select genes that are easier to interpret and validate experimentally.

```
library(scran)
library(scater)
library(pheatmap)
```

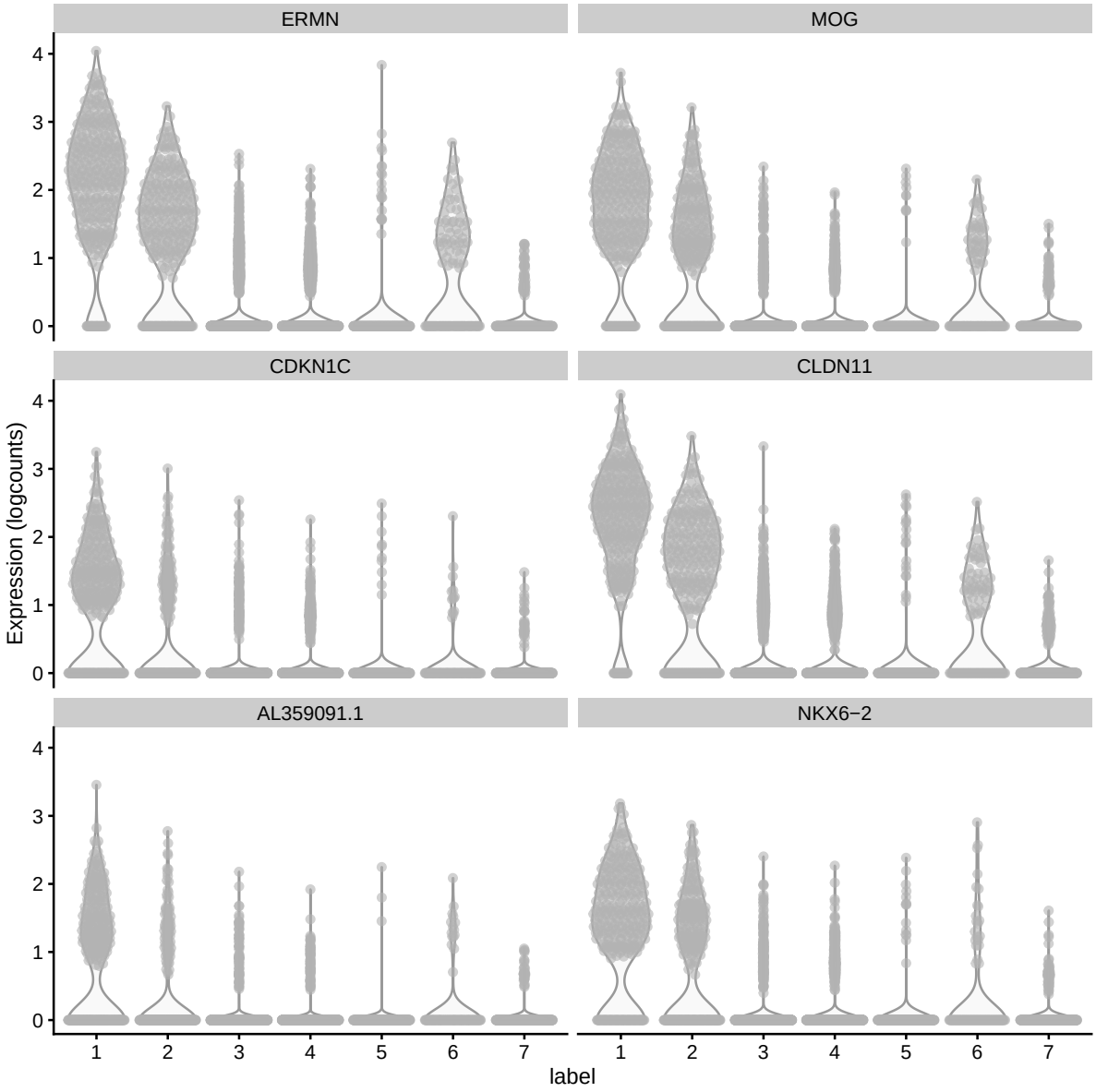
```
# set gene names as row names for easier plotting
rownames(spe) <- rowData(spe)$gene_name

# test for marker genes
markers <- findMarkers(spe, test = "binom", direction = "up")

# returns a list with one DataFrame per cluster
markers
```

```
List of length 7
names(7): 1 2 3 4 5 6 7
```

```
# plot log-fold changes for one cluster over all other clusters
# selecting cluster 1
interesting <- markers[[1]]
best_set <- interesting[interesting$Top <= 5, ]
logFCs <- getMarkerEffects(best_set)

pheatmap(logFCs, breaks = seq(-5, 5, length.out = 101))
```




```
# plot log-transformed normalized expression of top genes for one cluster
top_genes <- head(rownames(interesting))

plotExpression(spe, x = "label", features = top_genes)
```


ERMN
MOG
CDKN1C
CLDN11
AL359091.1
NKX6−2
Expression (logcounts)
label
0
1
2
3
4
5
6
7

9 Spot-level deconvolution

9.1 Background

Spot-level SRT data can contain zero, one, or multiple cells per spot, depending on the tissue density and technological platform used. This characteristic of the data influences several steps in analysis pipelines, especially Quality control and Clustering.

This is also a unique characteristic of SRT data, which is distinct from single-cell data, so here we cannot easily apply existing methods from single-cell pipelines.

In this section, we will demonstrate methods to deconvolve cell types per spot. Several methods exist from the bulk RNA-seq literature, as well as new methods designed for SRT data.

9.2 Previous steps

Code to run steps from the previous chapters to generate the `SpatialExperiment` *object required for this chapter.*

```
# LOAD DATA

library(SpatialExperiment)
library(STexampleData)
spe <- Visium_humanDLPFC()
```

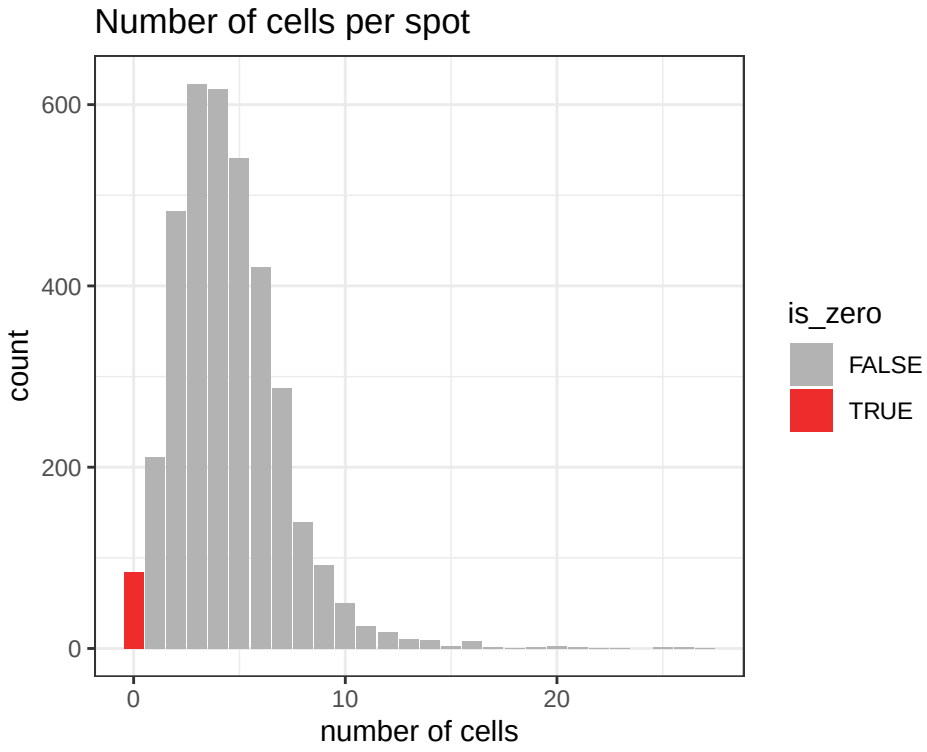
9.3 Number of cells per spot

The following figure provides an overview of the number of cells per spot in this dataset, which is known in this dataset and stored in a column in `colData` in the `SpatialExperiment` object. We use a visualization function from ggspavis to generate the plot.

We see that spots in this dataset contain around 0-10 cells, with a mode of 3. Therefore, it is plausible that some spots contain multiple cell types, and spot-level deconvolution could improve downstream analyses by deconvolving these cell types.

```r
library(ggspavis)

# plot number of cells per spot
plotQC(spe, type = "bar", metric_x = "cell_count") +
  xlab("number of cells") +
  ggtitle("Number of cells per spot")
```

```
Warning: Removed 1353 rows containing non-finite values (`stat_count()`).
Removed 1353 rows containing non-finite values (`stat_count()`).
```



Part III

Workflows

This part contains several longer chapters describing complete workflows or comparisons of methods for individual analysis steps.

The workflow chapters each describe a complete analysis workflow for a single dataset, illustrating how the individual analysis steps fit together. For example, workflows may demonstrate an analysis for a dataset from a specific technological platform or tissue type. The workflows reuse code from the individual analysis chapters in a condensed format for consistency. For more details on individual analysis steps, see the chapters in the previous part.

The comparison chapters compare alternative methods for a specific analysis step, such as spatially-aware clustering. Methods are compared in terms of performance using a given dataset and evaluation metrics. This provides a way to showcase new methods as these are developed and implemented in the Bioconductor framework, and provides readers with information on which methods may be appropriate for their analyses.

10 Human DLPFC workflow

This workflow analyzes one sample of human brain from the dorsolateral prefrontal cortex (DLPFC) region, measured using the 10x Genomics Visium platform. This is a condensed version of the analyses shown in the individual analysis chapters in the previous part. For more details on the individual steps, see the previous chapters.

10.1 Description of dataset

This is a 10x Genomics Visium dataset generated from healthy human brain samples from the dorsolateral prefrontal cortex (DLPFC) region.

In the full dataset, there are 12 samples in total, from 3 individuals, with 2 pairs of spatially adjacent replicates (serial sections) per individual (4 samples per individual). The individuals and spatially adjacent replicates can be used as blocking factors. Each sample spans the six layers of the cortex plus white matter in a perpendicular tissue section.

For the examples in this workflow and the analysis chapters, we use a single sample from this dataset (sample 151673), to keep the computational requirements to compile the book manageable.

For more details on the dataset, see Maynard et al. (2021). The full dataset is publicly available through the spatialLIBD Bioconductor package. The dataset can also be explored interactively through the spatialLIBD Shiny web app.

10.2 Load data

Here, we load a single sample from this dataset (sample 151673), which is available as a `SpatialExperiment` object from the STexampleData package.

```
library(SpatialExperiment)
library(STexampleData)

# load object
spe <- Visium_humanDLPFC()
```

```
spe
```

```
class: SpatialExperiment
dim: 33538 4992
metadata(0):
assays(1): counts
rownames(33538): ENSG00000243485 ENSG00000237613 ... ENSG00000277475
  ENSG00000268674
rowData names(3): gene_id gene_name feature_type
colnames(4992): AAACAACGAATAGTTC-1 AAACAAGTATCTCCCA-1 ...
  TTGTTTGTATTACACG-1 TTGTTTGTGTAAATTC-1
colData names(7): barcode_id sample_id ... ground_truth cell_count
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
```

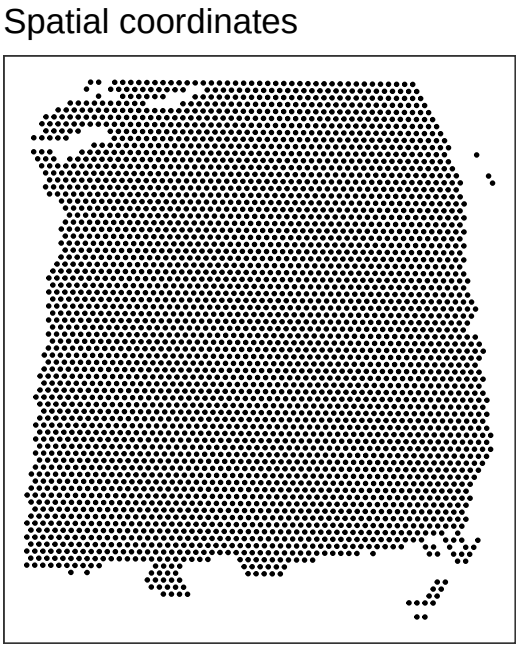
10.3 Plot data

As an initial check, plot the spatial coordinates (spots) in x-y dimensions on the tissue slide, to check that the object has loaded correctly and that the orientation is as expected.

We use visualization functions from the ggspavis package to generate plots.

```
library(ggspavis)
```

```
# plot spatial coordinates (spots)
plotSpots(spe)
```



10.4 Quality control (QC)

First, we subset the object to keep only spots over tissue. The remaining spots are background spots, which we exclude.

```
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
dim(spe)
```

```
[1] 33538  3639
```

Next, calculate spot-level QC metrics using the `scater` package (McCarthy et al. 2017), and store the QC metrics in `colData`. See Quality control for more details, including explanations of the QC metrics.

```
library(scater)
```

```
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
table(is_mito)
```

```
is_mito
FALSE  TRUE
33525    13
```

```
rowData(spe)$gene_name[is_mito]
```

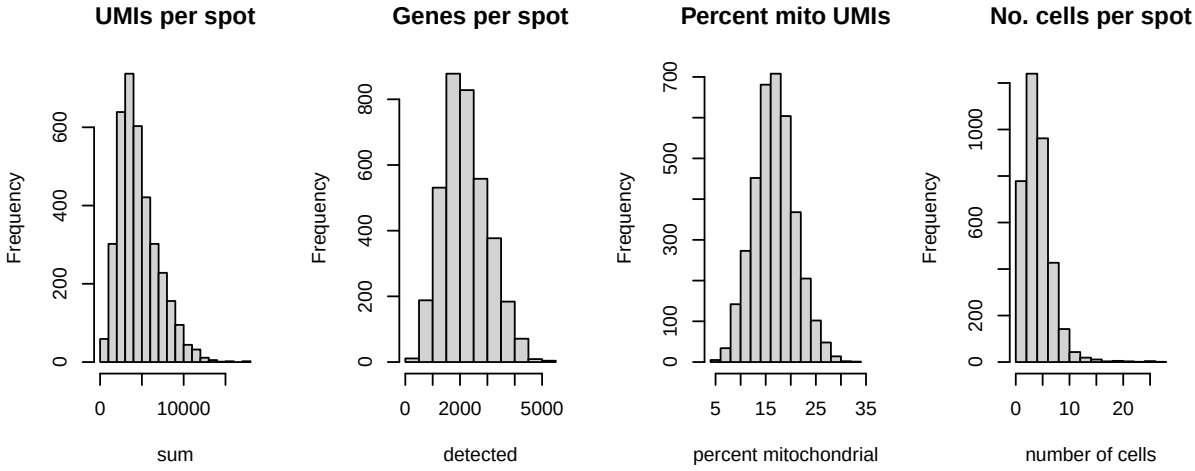
```
 [1] "MT-ND1"  "MT-ND2"  "MT-CO1"  "MT-CO2"  "MT-ATP8" "MT-ATP6" "MT-CO3"
 [8] "MT-ND3"  "MT-ND4L" "MT-ND4"  "MT-ND5"  "MT-ND6"  "MT-CYB"
```

```
# calculate per-spot QC metrics and store in colData
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
head(colData(spe), 3)
```

```
DataFrame with 3 rows and 13 columns
                         barcode_id     sample_id in_tissue array_row
                        <character>   <character> <integer> <integer>
AAACAAGTATCTCCCA-1 AAACAAGTATCTCCCA-1 sample_151673         1        50
AAACAATCTACTAGCA-1 AAACAATCTACTAGCA-1 sample_151673         1         3
AAACACCAATAACTGC-1 AAACACCAATAACTGC-1 sample_151673         1        59
                   array_col ground_truth cell_count       sum  detected
                   <integer>  <character>  <integer> <numeric> <numeric>
AAACAAGTATCTCCCA-1       102       Layer3          6      8458      3586
AAACAATCTACTAGCA-1        43       Layer1         16      1667      1150
AAACACCAATAACTGC-1        19           WM          5      3769      1960
                   subsets_mito_sum subsets_mito_detected subsets_mito_percent
                          <numeric>             <numeric>            <numeric>
AAACAAGTATCTCCCA-1             1407                    13              16.6351
AAACAATCTACTAGCA-1              204                    11              12.2376
AAACACCAATAACTGC-1              430                    13              11.4089
                       total
                   <numeric>
AAACAAGTATCTCCCA-1      8458
AAACAATCTACTAGCA-1      1667
AAACACCAATAACTGC-1      3769
```

Select filtering thresholds for the QC metrics by examining distributions using histograms. For additional details, including further exploratory visualizations to select the thresholds, see Quality control. Here, we use relatively relaxed thresholds, since the additional exploratory visualizations showed that more stringent thresholds tended to remove groups of spots corresponding to biologically meaningful regions.

```
# histograms of QC metrics
par(mfrow = c(1, 4))
hist(colData(spe)$sum, xlab = "sum", main = "UMIs per spot")
hist(colData(spe)$detected, xlab = "detected", main = "Genes per spot")
hist(colData(spe)$subsets_mito_percent, xlab = "percent mitochondrial", main = "Percent mi
hist(colData(spe)$cell_count, xlab = "number of cells", main = "No. cells per spot")
```




```
par(mfrow = c(1, 1))

# select QC thresholds
qc_lib_size <- colData(spe)$sum < 600
qc_detected <- colData(spe)$detected < 400
qc_mito <- colData(spe)$subsets_mito_percent > 28
qc_cell_count <- colData(spe)$cell_count > 10

# number of discarded spots for each metric
apply(cbind(qc_lib_size, qc_detected, qc_mito, qc_cell_count), 2, sum)
```

```
qc_lib_size   qc_detected       qc_mito qc_cell_count
          8             7            17            90
```

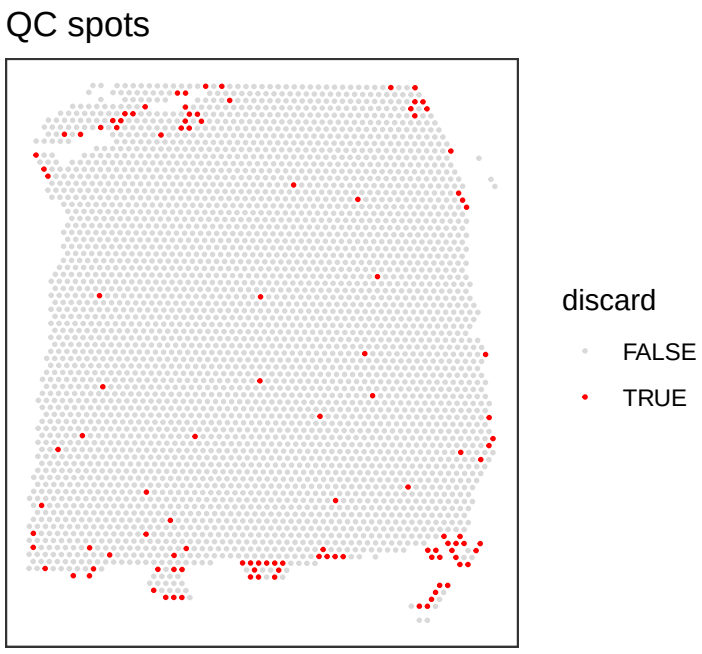
```
# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
table(discard)
```

```
discard
FALSE  TRUE
 3524   115
```

```
# store in object
colData(spe)$discard <- discard
```

Plot the set of discarded spots in the spatial x-y coordinates, to confirm that the spatial distribution of the discarded spots does not correspond to any biologically meaningful regions, which would indicate that we are removing biologically informative spots.

```
# check spatial pattern of discarded spots
plotQC(spe, type = "spots", discard = "discard")
```



There is some concentration of discarded spots at the edge of the tissue region, which may be due to tissue damage. Importantly, the discarded spots do not correspond to any of the cortical layers of interest.

We filter out the low-quality spots from the object.

```
# filter low-quality spots
spe <- spe[, !colData(spe)$discard]
dim(spe)
```

```
[1] 33538  3524
```

10.5 Normalization

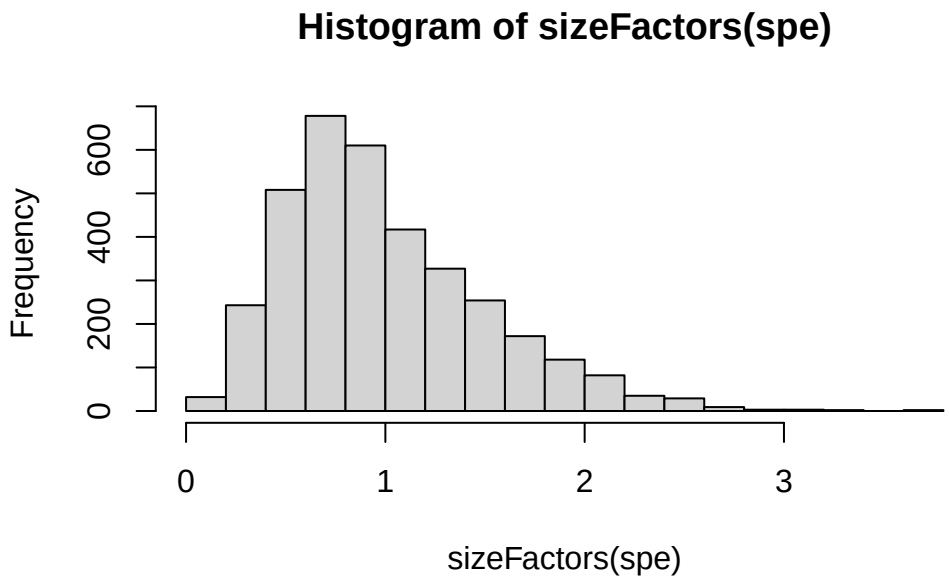
Calculate log-transformed normalized counts (logcounts) with the library size factors methodology, using methods from `scater` (McCarthy et al. 2017) and `scran` (Lun, McCarthy, and Marioni 2016), making the assumption that spots can be treated as equivalent to cells. For more details, see Normalization.

```
library(scran)
```

```
# calculate library size factors
spe <- computeLibraryFactors(spe)

summary(sizeFactors(spe))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1321  0.6312  0.9000  1.0000  1.2849  3.7582
```

```
hist(sizeFactors(spe), breaks = 20)
```



```
# calculate logcounts and store in object
spe <- logNormCounts(spe)

assayNames(spe)
```

```
[1] "counts"    "logcounts"
```

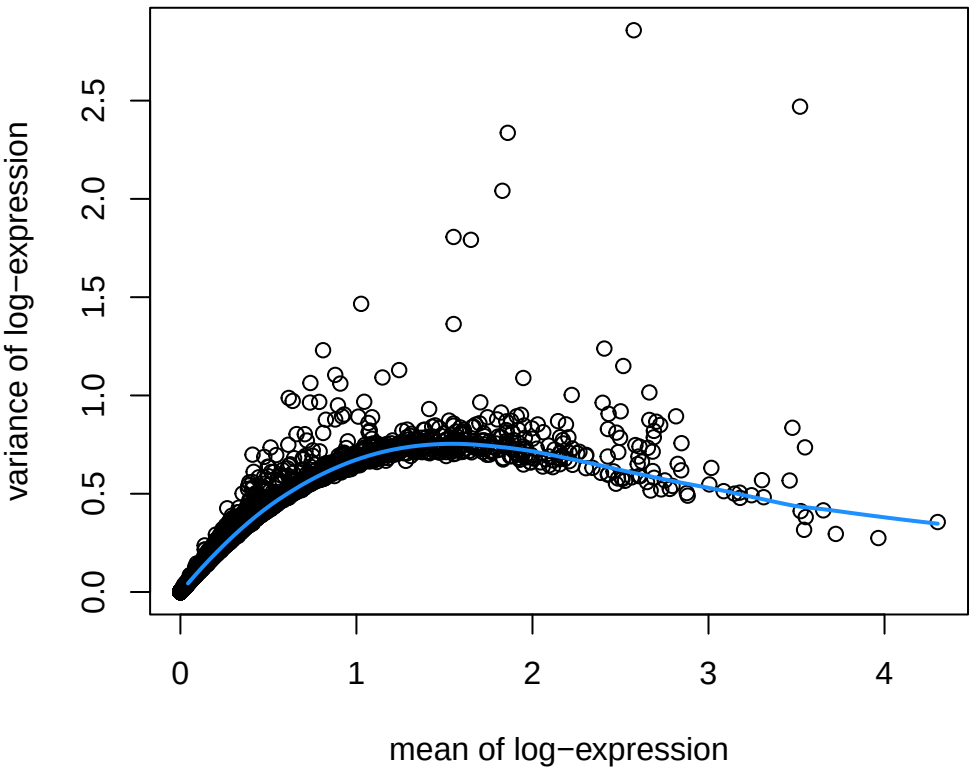
10.6 Feature selection

Identify a set of top highly variable genes (HVGs), which will be used to define cell types. We use methods from `scran` (Lun, McCarthy, and Marioni 2016), treating spots as equivalent to single cells, and considering only molecular features (gene expression) as described in Feature selection. We also first filter out mitochondrial genes, since these are very highly expressed and not of main biological interest here.

```
# remove mitochondrial genes
spe <- spe[!is_mito, ]
dim(spe)
```

```
[1] 33525  3524
```

```
# fit mean-variance relationship
dec <- modelGeneVar(spe)

# visualize mean-variance relationship
fit <- metadata(dec)
plot(fit$mean, fit$var,
     xlab = "mean of log-expression", ylab = "variance of log-expression")
curve(fit$trend(x), col = "dodgerblue", add = TRUE, lwd = 2)
```




```
# select top HVGs
top_hvgs <- getTopHVGs(dec, prop = 0.1)
length(top_hvgs)
```

```
[1] 1438
```

10.7 Spatially-aware feature selection

Alternatively, run nnSVG to identify a set of top spatially variable genes (SVGs) instead of HVGs.

Here, we run nnSVG using a small subset of the dataset for faster runtime. We select a subset of the data by subsampling on the set of spots and including stringent filtering for low-expressed genes. For a full analysis, we recommend running `nnSVG` on all spots and using default filtering parameters (for Visium data from human brain tissue), which takes around 45 minutes for one Visium slide on a standard laptop using multiple cores.

```
library(nnSVG)
```

```
# subsample spots
n <- 100
set.seed(123)
ix <- sample(seq_len(n), n)

spe_nnSVG <- spe[, ix]

# filter low-expressed and mitochondrial genes
# using very stringent filtering parameters for faster runtime in this example
# note: for a full analysis, use alternative filtering parameters (e.g. defaults)
spe_nnSVG <- filter_genes(
  spe_nnSVG, filter_genes_ncounts = 10, filter_genes_pcspots = 3
)
```

```
Gene filtering: removing mitochondrial genes
```

```
removed 0 mitochondrial genes
```

```
Gene filtering: retaining genes with at least 10 counts in at least 3% (n = 3) of spatial lo
```

```
removed 33353 out of 33525 genes due to low expression
```

```
# re-calculate logcounts after filtering
# using library size factors
spe_nnSVG <- logNormCounts(spe_nnSVG)

# run nnSVG
# using a single core for compatibility on build system
# note: for a full analysis, use multiple cores
set.seed(123)
spe_nnSVG <- nnSVG(spe_nnSVG, n_threads = 1)

# investigate results

# show results
head(rowData(spe_nnSVG), 3)
```

```
DataFrame with 3 rows and 17 columns
                     gene_id   gene_name    feature_type   sigma.sq
                 <character> <character>     <character>  <numeric>
ENSG00000074800 ENSG00000074800        ENO1 Gene Expression 0.00840777
ENSG00000171603 ENSG00000171603      CLSTN1 Gene Expression 0.43102027
ENSG00000162545 ENSG00000162545     CAMK2N1 Gene Expression 0.15373440
                   tau.sq       phi    loglik   runtime      mean       var
                <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
ENSG00000074800  0.518011  15.82454  -109.808     0.019   1.75820  0.530929
ENSG00000171603  0.298698  18.12310  -123.212     0.022   2.07433  0.741077
ENSG00000162545  0.547736   2.38046  -119.576     0.016   2.63576  0.702846
                   spcov   prop_sv loglik_lm   LR_stat      rank      pval
                <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
ENSG00000074800 0.0521523 0.0159716  -109.735 -0.145454       167 1.0000000
ENSG00000171603 0.3164987 0.5906665  -126.409  6.394009        86 0.0408845
ENSG00000162545 0.1487576 0.2191603  -123.760  8.368951        70 0.0152302
                    padj
                <numeric>
ENSG00000074800 1.0000000
ENSG00000171603 0.0817690
ENSG00000162545 0.0374227
```

```
# number of significant SVGs
table(rowData(spe_nnSVG)$padj <= 0.05)
```

```
FALSE  TRUE
   96    76
```

```
# show results for top n SVGs
rowData(spe_nnSVG)[order(rowData(spe_nnSVG)$rank)[1:6], ]
```

```
DataFrame with 6 rows and 17 columns
                     gene_id   gene_name    feature_type  sigma.sq    tau.sq
                 <character> <character>     <character> <numeric> <numeric>
ENSG00000197971 ENSG00000197971         MBP Gene Expression   3.92430  0.175336
ENSG00000123560 ENSG00000123560        PLP1 Gene Expression   3.23544  0.461590
ENSG00000109846 ENSG00000109846       CRYAB Gene Expression   1.89968  0.281795
ENSG00000173786 ENSG00000173786         CNP Gene Expression   2.28793  0.402524
```

```
ENSG00000131095 ENSG00000131095        GFAP Gene Expression   2.23709  0.461297
ENSG00000160307 ENSG00000160307       S100B Gene Expression   1.24179  0.155737
                      phi    loglik   runtime      mean       var     spcov
                <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
ENSG00000197971   0.93014  -118.372     0.022   3.78073   2.76535  0.523969
ENSG00000123560   1.03983  -140.269     0.018   2.86143   3.02555  0.628613
ENSG00000109846   1.88555  -126.692     0.016   1.86058   1.88123  0.740785
ENSG00000173786   1.02675  -129.206     0.017   1.79558   1.97274  0.842396
ENSG00000131095   2.49083  -149.004     0.019   1.94543   2.71281  0.768823
ENSG00000160307   5.76279  -129.722     0.015   1.82695   1.46946  0.609953
                  prop_sv loglik_lm   LR_stat      rank        pval        padj
                <numeric> <numeric> <numeric> <numeric>   <numeric>   <numeric>
ENSG00000197971  0.957231  -192.250  147.7556         1 0.00000e+00 0.00000e+00
ENSG00000123560  0.875146  -196.746  112.9549         2 0.00000e+00 0.00000e+00
ENSG00000109846  0.870824  -172.988   92.5906         3 0.00000e+00 0.00000e+00
ENSG00000173786  0.850388  -175.363   92.3137         4 0.00000e+00 0.00000e+00
ENSG00000131095  0.829047  -191.291   84.5739         5 0.00000e+00 0.00000e+00
ENSG00000160307  0.888562  -160.636   61.8281         6 3.75255e-14 1.07573e-12
```

```
# identify top-ranked SVG
rowData(spe_nnSVG)$gene_name[which(rowData(spe_nnSVG)$rank == 1)]
```

```
[1] "MBP"
```

10.8 Dimensionality reduction

Run principal component analysis (PCA) on the set of top HVGs, and retain the top 50 principal components (PCs) for further downstream analyses. This is done both to reduce noise and to improve computational efficiency. We also run UMAP on the set of top 50 PCs and retain the top 2 UMAP components for visualization purposes.

We use the computationally efficient implementation of PCA available in `scater` (McCarthy et al. 2017), which uses randomization, and therefore requires setting a random seed for reproducibility.

```
# compute PCA
set.seed(123)
spe <- runPCA(spe, subset_row = top_hvgs)

reducedDimNames(spe)
```

```
[1] "PCA"
```

```
dim(reducedDim(spe, "PCA"))
```

```
[1] 3524   50
```

```
# compute UMAP on top 50 PCs
set.seed(123)
spe <- runUMAP(spe, dimred = "PCA")

reducedDimNames(spe)
```

```
[1] "PCA"  "UMAP"
```

```
dim(reducedDim(spe, "UMAP"))
```

```
[1] 3524    2
```

```
# update column names for easier plotting
colnames(reducedDim(spe, "UMAP")) <- paste0("UMAP", 1:2)
```

10.9 Clustering

Next, we perform clustering to define cell types. Here, we use molecular features (gene expression) only, as described in Clustering. We apply graph-based clustering using the Walktrap method implemented in **scran** (Lun, McCarthy, and Marioni 2016), applied to the top 50 PCs calculated on the set of top HVGs.

```
# graph-based clustering
set.seed(123)
k <- 10
g <- buildSNNGraph(spe, k = k, use.dimred = "PCA")
g_walk <- igraph::cluster_walktrap(g)
clus <- g_walk$membership
table(clus)
```

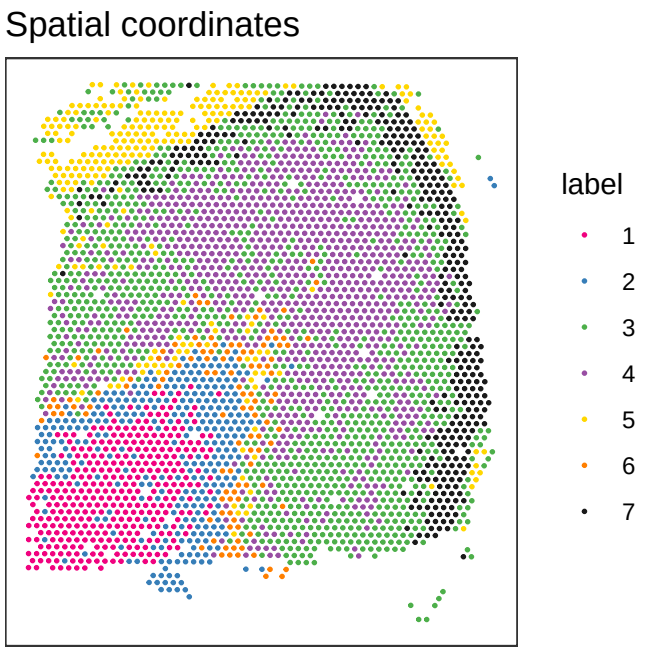
```
clus
   1    2    3    4    5    6    7
 338  312 1146  978  274  116  360
```

```
# store cluster labels in column 'label' in colData
colLabels(spe) <- factor(clus)
```

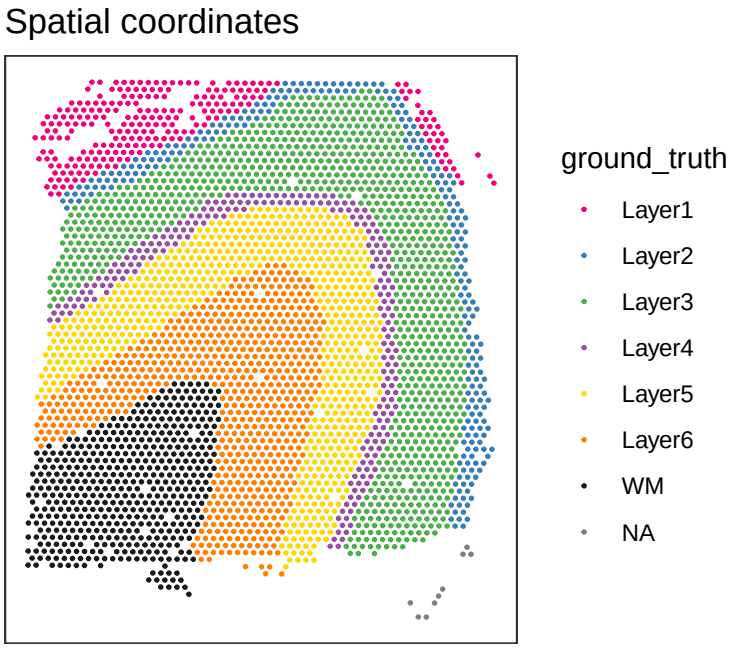
Visualize the clusters by plotting in spatial (x-y) coordinates on the tissue slide, and in UMAP dimensions.

From the visualizations, we can see that the clustering reproduces the known biological structure (cortical layers), although not perfectly. The clusters are also separated in UMAP space, but again not perfectly.

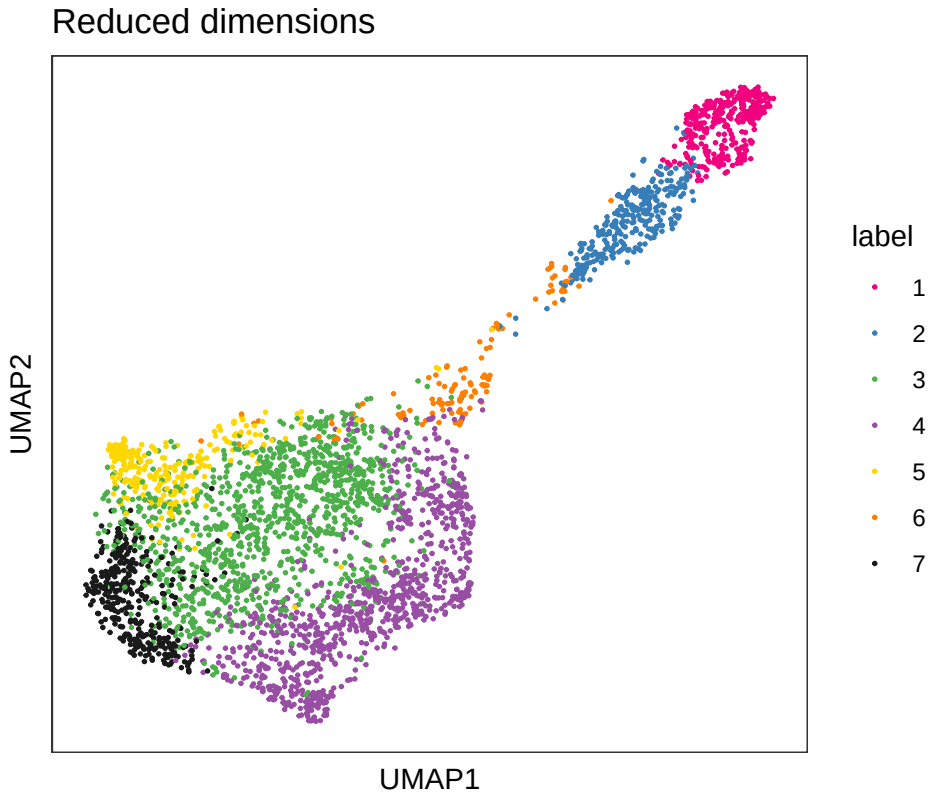
```
# plot clusters in spatial x-y coordinates
plotSpots(spe, annotate = "label",
          palette = "libd_layer_colors")
```




```
# plot ground truth labels in spatial coordinates
plotSpots(spe, annotate = "ground_truth",
          palette = "libd_layer_colors")
```




```
# plot clusters in UMAP reduced dimensions
plotDimRed(spe, type = "UMAP",
           annotate = "label", palette = "libd_layer_colors")
```



10.10 Marker genes

Identify marker genes by testing for differential gene expression between clusters. We use the `findMarkers` implementation in `scran` (Lun, McCarthy, and Marioni 2016), using a binomial test, which tests for genes that differ in the proportion expressed vs. not expressed between clusters. This is a more stringent test than the default t-tests, and tends to select genes that are easier to interpret and validate experimentally.

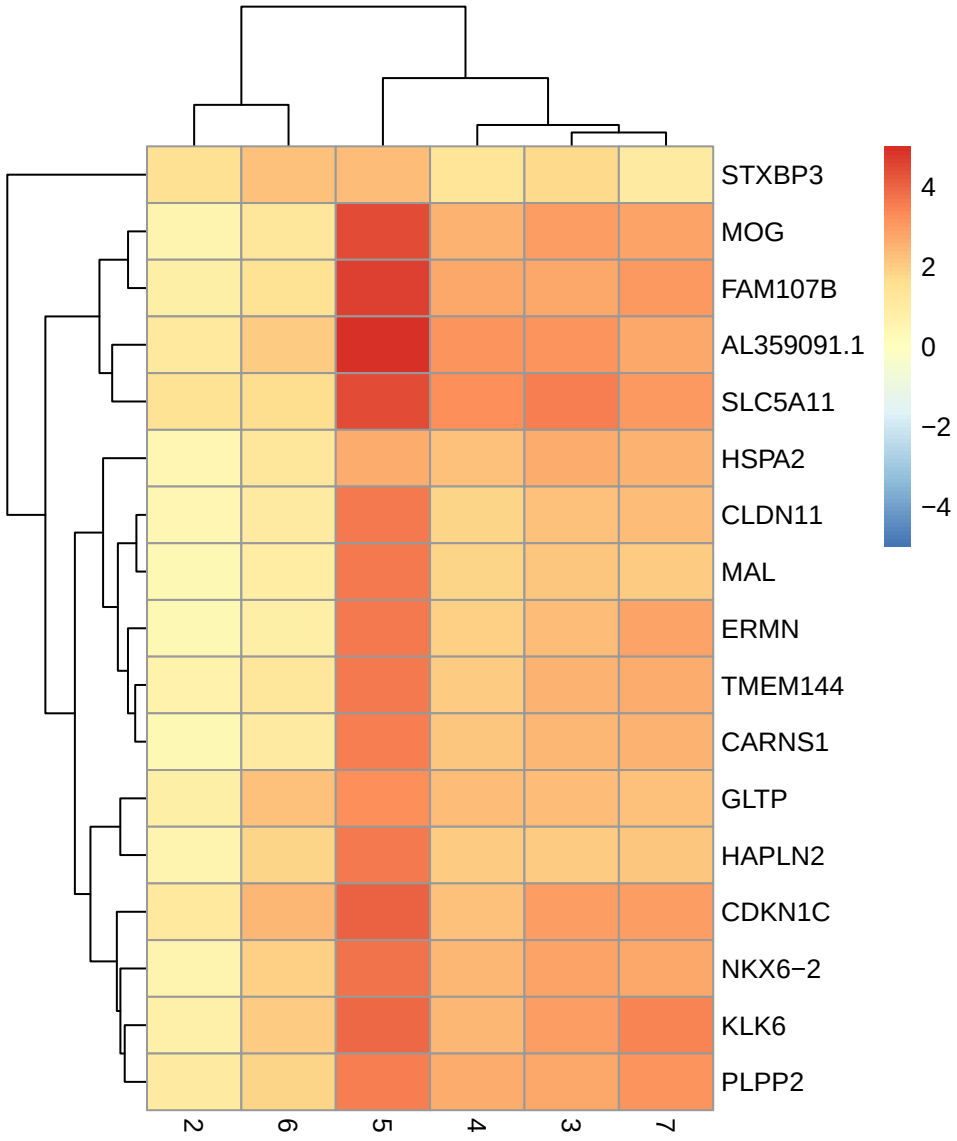
```
# set gene names as row names for easier plotting
rownames(spe) <- rowData(spe)$gene_name

# test for marker genes
markers <- findMarkers(spe, test = "binom", direction = "up")

# returns a list with one DataFrame per cluster
markers
```

```
List of length 7
```

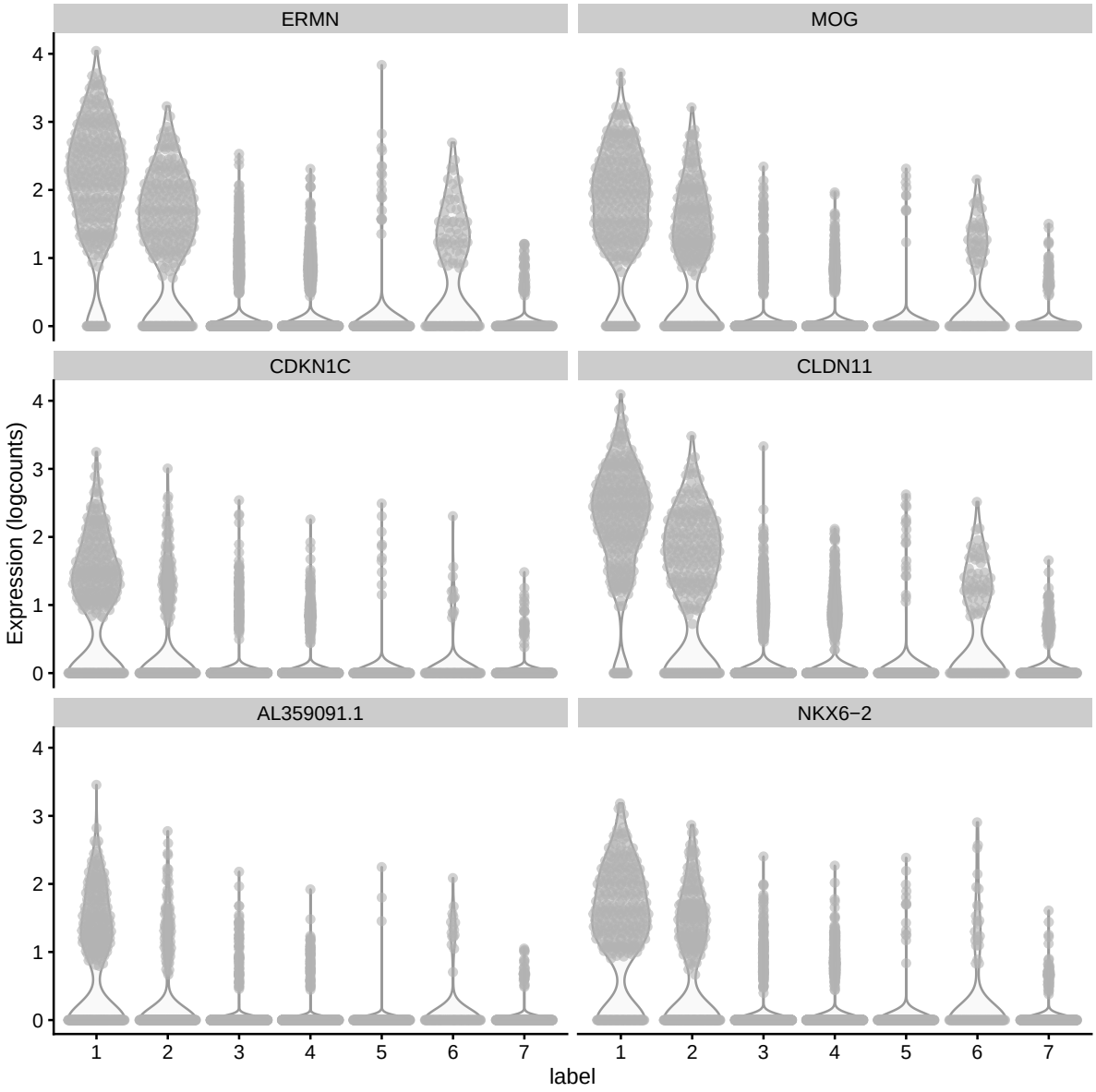
```
names(7): 1 2 3 4 5 6 7
```

```
library(pheatmap)
```

```
# plot log-fold changes for one cluster over all other clusters
# selecting cluster 1
interesting <- markers[[1]]
best_set <- interesting[interesting$Top <= 5, ]
logFCs <- getMarkerEffects(best_set)

pheatmap(logFCs, breaks = seq(-5, 5, length.out = 101))
```




```
# plot log-transformed normalized expression of top genes for one cluster
top_genes <- head(rownames(interesting))

plotExpression(spe, x = "label", features = top_genes)
```


ERMN
MOG
CDKN1C
CLDN11
AL359091.1
NKX6−2
Expression (logcounts)
label
0
1
2
3
4
5
6
7

11 Mouse coronal workflow

This workflow analyzes a mouse coronal brain section dataset from the 10x Genomics Visium platform. This dataset was generated by 10x Genomics, and the raw data files are publicly available from the 10x Genomics website.

11.1 Description of dataset

This dataset measures transcriptome-wide gene expression on a Visium slide spanning one hemisphere of a mouse coronal brain section. For experimental details, see the 10x Genomics website.

Due to the small size of the mouse brain and the dimensions of the Visium slide (6.5mm x 6.5mm), the measurements span an entire brain hemisphere. Therefore, we can use this dataset to compare gene expression profiles between major anatomical regions of the mouse brain. Due to the small size of cells in the mouse brain, each spot can contain up to 50 cells. In this dataset, we do not know the exact number of cells per spot.

11.2 Load data

The dataset is available in `SpatialExperiment` format from the STexampleData package.

```
library(SpatialExperiment)
library(STexampleData)

# load object
spe <- Visium_mouseCoronal()
spe
```

```
class: SpatialExperiment
dim: 32285 4992
metadata(0):
assays(1): counts
```

```
rownames(32285): ENSMUSG00000051951 ENSMUSG00000089699 ...
  ENSMUSG00000095019 ENSMUSG00000095041
rowData names(3): gene_id gene_name feature_type
colnames(4992): AAACAACGAATAGTTC-1 AAACAAGTATCTCCCA-1 ...
  TTGTTTGTATTACACG-1 TTGTTTGTGTAAATTC-1
colData names(5): barcode_id sample_id in_tissue array_row array_col
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
```

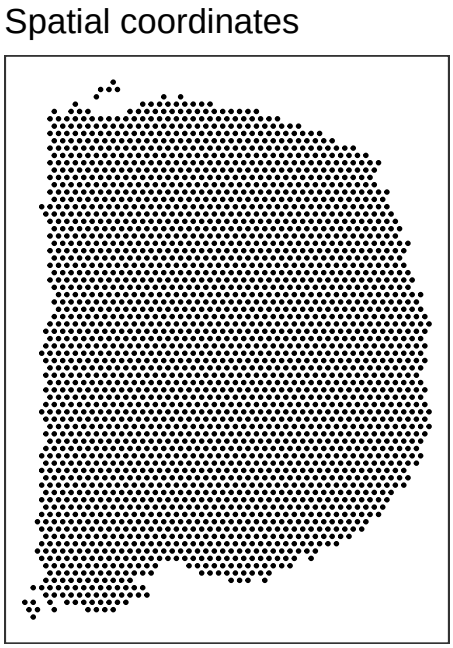
11.3 Plot data

As an initial check, plot the spatial coordinates (spots) in x-y dimensions on the tissue slide. This confirms that the object has loaded correctly, and the orientation matches the 10x Genomics website.

We use visualization functions from the ggspavis package to generate plots.

```
library(ggspavis)
```

```
# plot spatial coordinates (spots)
plotSpots(spe)
```



11.4 Quality control (QC)

Subset object to keep only spots over tissue.

```
# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]
dim(spe)
```

```
[1] 32285  2702
```

Calculate spot-level QC metrics using the `scater` package (McCarthy et al. 2017), and store the QC metrics in `colData`.

```
library(scater)
```

```
# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
table(is_mito)
```

```
is_mito
FALSE  TRUE
32272    13
```

```
rowData(spe)$gene_name[is_mito]
```

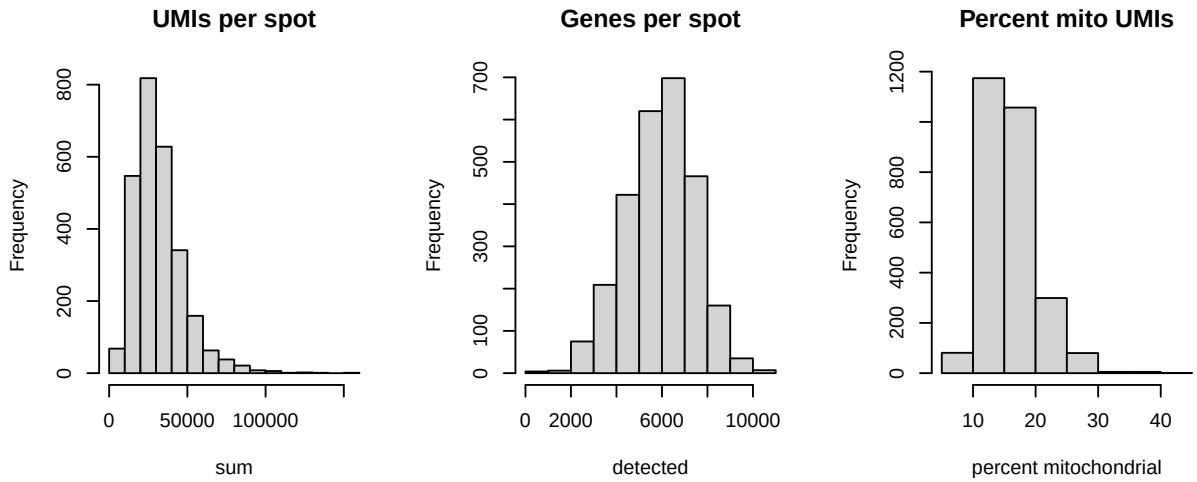
```
 [1] "mt-Nd1"  "mt-Nd2"  "mt-Co1"  "mt-Co2"  "mt-Atp8" "mt-Atp6" "mt-Co3"
 [8] "mt-Nd3"  "mt-Nd4l" "mt-Nd4"  "mt-Nd5"  "mt-Nd6"  "mt-Cytb"
```

```
# calculate per-spot QC metrics and store in colData
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))
head(colData(spe), 3)
```

```
DataFrame with 3 rows and 11 columns
                         barcode_id   sample_id in_tissue array_row array_col
                        <character> <character> <integer> <integer> <integer>
AAACAAGTATCTCCCA-1 AAACAAGTATCTCCCA-1    sample01         1        50       102
AAACAATCTACTAGCA-1 AAACAATCTACTAGCA-1    sample01         1         3        43
AAACACCAATAACTGC-1 AAACACCAATAACTGC-1    sample01         1        59        19
                         sum  detected subsets_mito_sum subsets_mito_detected
                   <numeric> <numeric>        <numeric>             <numeric>
AAACAAGTATCTCCCA-1     20935      5230             4036                    13
AAACAATCTACTAGCA-1     14789      3646             3419                    13
AAACACCAATAACTGC-1     34646      6272             5068                    13
                   subsets_mito_percent     total
                              <numeric> <numeric>
AAACAAGTATCTCCCA-1              19.2787     20935
AAACAATCTACTAGCA-1              23.1185     14789
AAACACCAATAACTGC-1              14.6280     34646
```

Select filtering thresholds for the QC metrics by examining distributions using histograms.

```
# histograms of QC metrics
par(mfrow = c(1, 3))
hist(colData(spe)$sum, xlab = "sum", main = "UMIs per spot")
hist(colData(spe)$detected, xlab = "detected", main = "Genes per spot")
hist(colData(spe)$subsets_mito_percent, xlab = "percent mitochondrial", main = "Percent mi
```




```
par(mfrow = c(1, 1))

# select QC thresholds
qc_lib_size <- colData(spe)$sum < 5000
qc_detected <- colData(spe)$detected < 1000
qc_mito <- colData(spe)$subsets_mito_percent > 30

# number of discarded spots for each QC metric
apply(cbind(qc_lib_size, qc_detected, qc_mito), 2, sum)
```

```
qc_lib_size qc_detected     qc_mito
          9           4          11
```

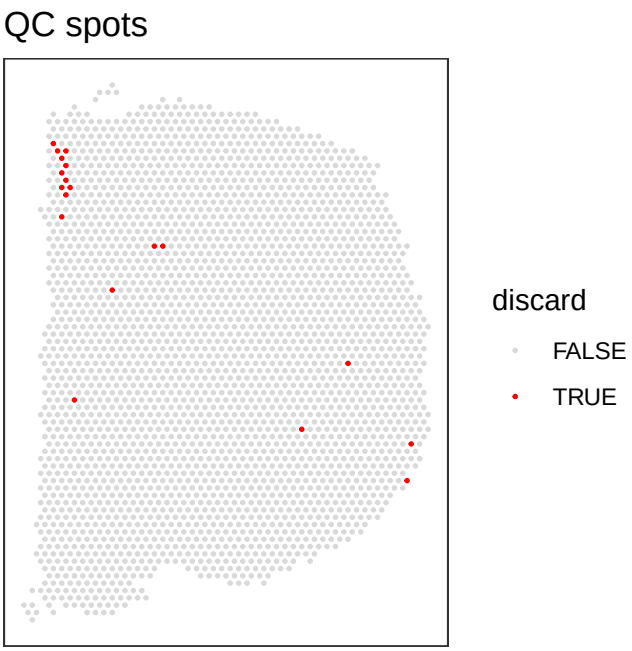
```
# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito
table(discard)
```

```
discard
FALSE  TRUE
 2683    19
```

```
# store in object
colData(spe)$discard <- discard
```

Plot discarded spots in x-y coordinates on the tissue slide to check if there is any biologically meaningful spatial pattern. This would be problematic, since it would mean we are removing biologically informative spots.

```
# check spatial pattern of discarded spots
plotQC(spe, type = "spots", discard = "discard")
```



There is one small region with some concentrated discarded spots at the top-left. However, this does not appear to correspond to any specific known anatomical region of interest. We assume that these are low-quality spots, and filtering them out will not cause problems in the biological interpretation.

We filter out the low-quality spots from the object.

```
# filter low-quality spots
spe <- spe[, !colData(spe)$discard]
dim(spe)
```

```
[1] 32285  2683
```

11.5 Normalization

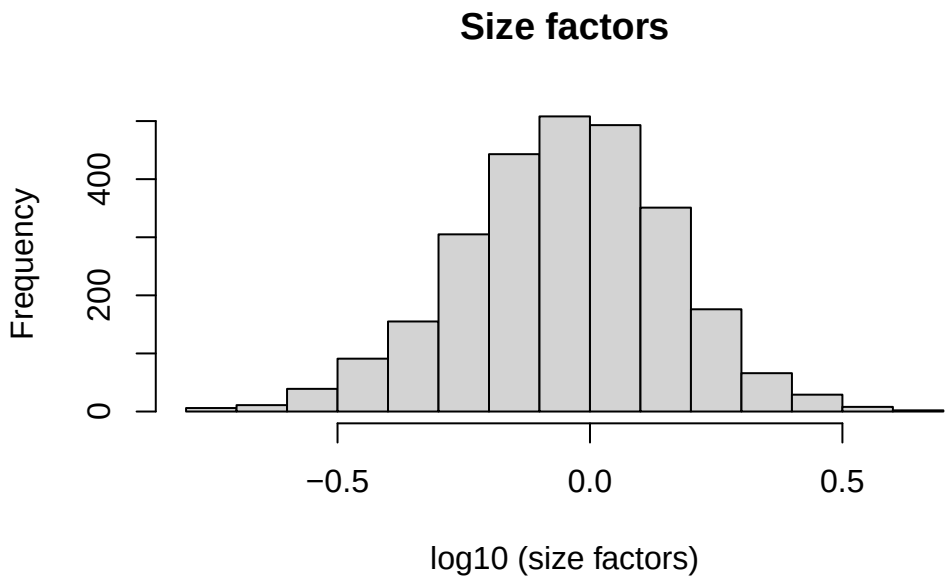
Next, we calculate log-transformed normalized counts (logcounts) with the library size factors methodology, using methods from `scater` (McCarthy et al. 2017) and `scran` (Lun, McCarthy, and Marioni 2016).

```
library(scran)
```

```
# calculate library size factors
spe <- computeLibraryFactors(spe)

summary(sizeFactors(spe))
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
 0.1615  0.6597  0.9083  1.0000  1.2342  4.8973
```

```
hist(log10(sizeFactors(spe)), xlab = "log10 (size factors)", main = "Size factors")
```




```
# calculate logcounts and store in object
spe <- logNormCounts(spe)

assayNames(spe)
```

```
[1] "counts"    "logcounts"
```

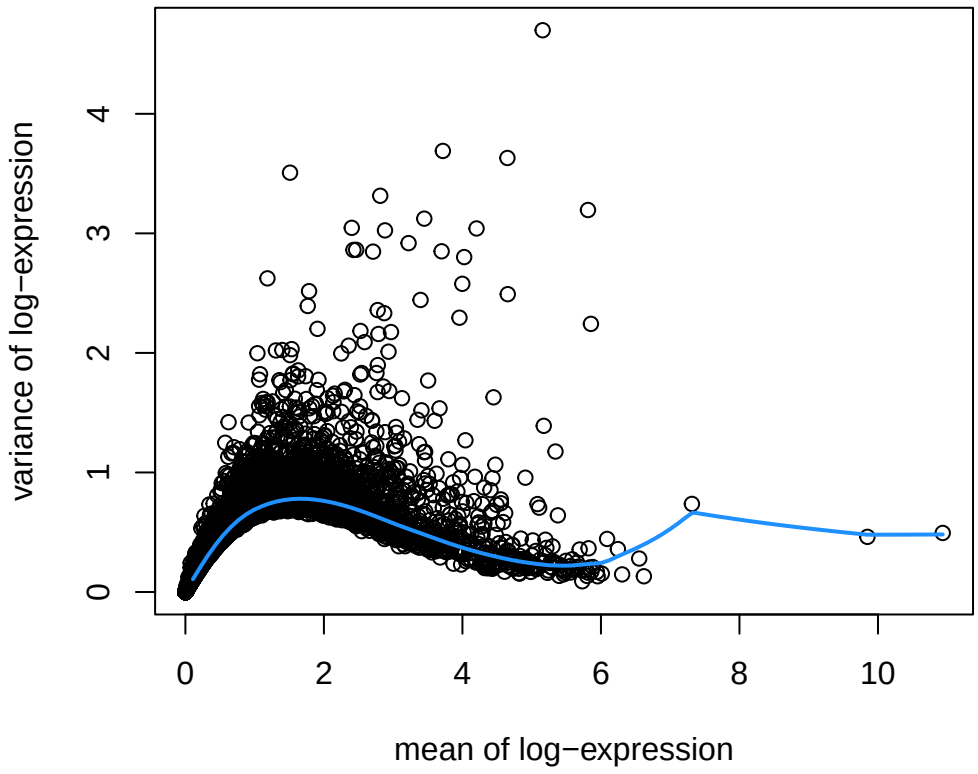
11.6 Feature selection

Identify a set of top highly variable genes (HVGs), which will be used to define cell types. We use methods from `scran` (Lun, McCarthy, and Marioni 2016), and first filter out mitochondrial genes (since these are very highly expressed and not of biological interest here).

```
# remove mitochondrial genes
spe <- spe[!is_mito, ]
dim(spe)
```

```
[1] 32272  2683
```

```
# fit mean-variance relationship
dec <- modelGeneVar(spe)

# visualize mean-variance relationship
fit <- metadata(dec)
plot(fit$mean, fit$var,
     xlab = "mean of log-expression", ylab = "variance of log-expression")
curve(fit$trend(x), col = "dodgerblue", add = TRUE, lwd = 2)
```




```
# select top HVGs
top_hvgs <- getTopHVGs(dec, prop = 0.1)
length(top_hvgs)
```

```
[1] 1274
```

Note there are a few extremely highly expressed genes, which influence the fitted mean-variance relationship. We check the names of these genes to decide whether they should be removed as outliers.

```
# identify outlier genes
rev(sort(fit$mean))[1:3]
```

```
ENSMUSG00000115783 ENSMUSG00000098178 ENSMUSG00000024661
         10.934946           9.847532           7.313407
```

```
outlier_ids <- names(rev(sort(fit$mean))[1:3])

rowData(spe)[outlier_ids, ]
```

```
DataFrame with 3 rows and 3 columns
                            gene_id   gene_name    feature_type
                        <character> <character>     <character>
ENSMUSG00000115783 ENSMUSG00000115783         Bc1 Gene Expression
ENSMUSG00000098178 ENSMUSG00000098178     Gm42418 Gene Expression
ENSMUSG00000024661 ENSMUSG00000024661        Fth1 Gene Expression
```

These appear to be biologically meaningful genes, so we leave them in.

11.7 Spatially-aware feature selection

Alternatively, run nnSVG to identify a set of top spatially variable genes (SVGs) instead of HVGs.

Here, we run nnSVG using a small subset of the dataset for faster runtime. We select a subset of the data by subsampling on the set of spots and including stringent filtering for low-expressed genes. For a full analysis, we recommend running `nnSVG` on all spots and using alternative filtering parameters (for Visium data from mouse brain tissue), which takes around 45 minutes for one Visium slide on a standard laptop using multiple cores.

```
library(nnSVG)
```

```
# subsample spots
n <- 100
set.seed(123)
ix <- sample(seq_len(n), n)

spe_nnSVG <- spe[, ix]

# filter low-expressed and mitochondrial genes
# using very stringent filtering parameters for faster runtime in this example
# note: for a full analysis, use alternative filtering parameters (e.g. defaults)
spe_nnSVG <- filter_genes(
  spe_nnSVG, filter_genes_ncounts = 50, filter_genes_pcspots = 5
)
```

```
Gene filtering: removing mitochondrial genes
```

```
removed 0 mitochondrial genes
```

```
Gene filtering: retaining genes with at least 50 counts in at least 5% (n = 5) of spatial lo
```

```
removed 32074 out of 32272 genes due to low expression
```

```
# re-calculate logcounts after filtering
# using library size factors
spe_nnSVG <- logNormCounts(spe_nnSVG)

# run nnSVG
# using a single core for compatibility on build system
# note: for a full analysis, use multiple cores
set.seed(123)
spe_nnSVG <- nnSVG(spe_nnSVG, n_threads = 1)

# investigate results

# show results
head(rowData(spe_nnSVG), 3)
```

```
DataFrame with 3 rows and 17 columns
                            gene_id   gene_name    feature_type  sigma.sq
                        <character> <character>     <character> <numeric>
ENSMUSG00000061518 ENSMUSG00000061518       Cox5b Gene Expression 0.0210665
ENSMUSG00000073702 ENSMUSG00000073702       Rpl31 Gene Expression 0.0688871
ENSMUSG00000046330 ENSMUSG00000046330      Rpl37a Gene Expression 0.0549986
                      tau.sq       phi    loglik   runtime      mean       var
                   <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
ENSMUSG00000061518 0.1202984  13.53194  -43.6472     0.023   4.98425  0.142977
ENSMUSG00000073702 0.0860842   5.84110  -40.3003     0.013   4.83432  0.143616
ENSMUSG00000046330 0.1133114   6.47579  -48.2811     0.018   5.35620  0.166734
                       spcov   prop_sv loglik_lm   LR_stat      rank      pval
                   <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
ENSMUSG00000061518 0.0291204  0.149022  -44.1377  0.981135       184 0.6122789
ENSMUSG00000073702 0.0542917  0.444515  -44.3607  8.120691       147 0.0172431
ENSMUSG00000046330 0.0437844  0.326770  -51.8235  7.084636       151 0.0289462
                        padj
```

```
                   <numeric>
ENSMUSG00000061518 0.6588653
ENSMUSG00000073702 0.0232254
ENSMUSG00000046330 0.0379559
```

```
# number of significant SVGs
table(rowData(spe_nnSVG)$padj <= 0.05)
```

```
FALSE  TRUE
   46   152
```

```
# show results for top n SVGs
rowData(spe_nnSVG)[order(rowData(spe_nnSVG)$rank)[1:6], ]
```

```
DataFrame with 6 rows and 17 columns
                            gene_id   gene_name    feature_type  sigma.sq
                        <character> <character>     <character> <numeric>
ENSMUSG00000046447 ENSMUSG00000046447     Camk2n1 Gene Expression   1.88342
ENSMUSG00000053310 ENSMUSG00000053310        Nrgn Gene Expression   3.17506
ENSMUSG00000018593 ENSMUSG00000018593       Sparc Gene Expression   2.04163
ENSMUSG00000032532 ENSMUSG00000032532         Cck Gene Expression   2.98145
ENSMUSG00000070570 ENSMUSG00000070570     Slc17a7 Gene Expression   2.64081
ENSMUSG00000024617 ENSMUSG00000024617      Camk2a Gene Expression   1.41249
                       tau.sq       phi    loglik   runtime      mean
                    <numeric> <numeric> <numeric> <numeric> <numeric>
ENSMUSG00000046447 6.28372e-02   2.05206   -99.021     0.020   5.35338
ENSMUSG00000053310 2.47772e-01   3.26420  -151.225     0.022   4.85871
ENSMUSG00000018593 7.63186e-02   3.68722  -124.793     0.020   3.61331
ENSMUSG00000032532 2.98145e-08   5.07869  -147.464     0.017   4.19282
ENSMUSG00000070570 3.20107e-01   4.15601  -155.620     0.016   3.62933
ENSMUSG00000024617 9.79965e-02   3.74573  -113.269     0.024   4.11652
                         var     spcov   prop_sv loglik_lm   LR_stat      rank
                   <numeric> <numeric> <numeric> <numeric> <numeric> <numeric>
ENSMUSG00000046447   1.50820  0.256357  0.967714  -161.937  125.8321         1
ENSMUSG00000053310   3.71912  0.366737  0.927612  -207.066  111.6804         2
ENSMUSG00000018593   1.87664  0.395442  0.963966  -172.866   96.1453         3
ENSMUSG00000032532   2.72400  0.411820  1.000000  -191.496   88.0644         4
ENSMUSG00000070570   3.00604  0.447757  0.891889  -196.423   81.6050         5
```

```
ENSMUSG00000024617   1.17870  0.288711  0.935123  -149.612   72.6864         6
                         pval        padj
                    <numeric>   <numeric>
ENSMUSG00000046447 0.00000e+00 0.00000e+00
ENSMUSG00000053310 0.00000e+00 0.00000e+00
ENSMUSG00000018593 0.00000e+00 0.00000e+00
ENSMUSG00000032532 0.00000e+00 0.00000e+00
ENSMUSG00000070570 0.00000e+00 0.00000e+00
ENSMUSG00000024617 1.11022e-16 3.66374e-15
```

```
# identify top-ranked SVG
rowData(spe_nnSVG)$gene_name[which(rowData(spe_nnSVG)$rank == 1)]
```

```
[1] "Camk2n1"
```

11.8 Dimensionality reduction

Run principal component analysis (PCA) on the set of top HVGs using `scater` (McCarthy et al. 2017), and retain the top 50 principal components (PCs) for downstream analyses. Also run UMAP on the top 50 PCs, and retain the top 2 UMAP components for visualization purposes.

```
# compute PCA
set.seed(123)
spe <- runPCA(spe, subset_row = top_hvgs)

reducedDimNames(spe)
```

```
[1] "PCA"
```

```
dim(reducedDim(spe, "PCA"))
```

```
[1] 2683   50
```

```
# compute UMAP on top 50 PCs
set.seed(123)
```

```
spe <- runUMAP(spe, dimred = "PCA")

reducedDimNames(spe)
```

```
[1] "PCA"  "UMAP"
```

```
dim(reducedDim(spe, "UMAP"))
```

```
[1] 2683    2
```

```
# update column names for easier plotting
colnames(reducedDim(spe, "UMAP")) <- paste0("UMAP", 1:2)
```

11.9 Clustering

Perform clustering to define cell types. We apply graph-based clustering using the Walktrap method implemented in `scran` (Lun, McCarthy, and Marioni 2016), applied to the top 50 PCs calculated on the set of top HVGs.

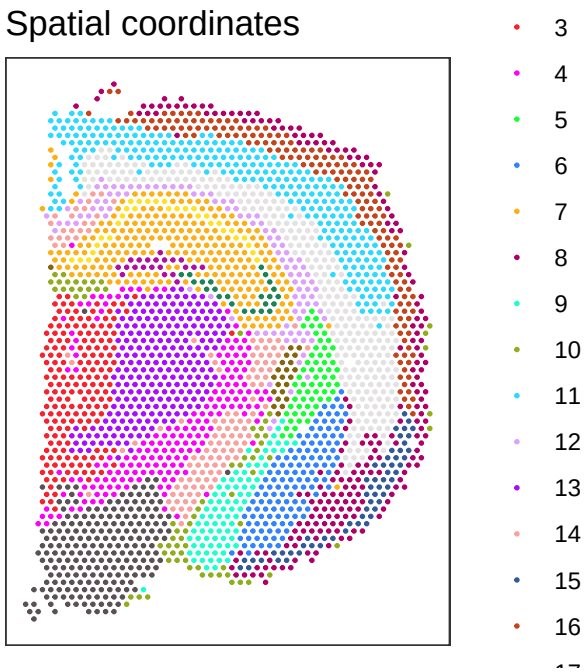
```
# graph-based clustering
set.seed(123)
k <- 10
g <- buildSNNGraph(spe, k = k, use.dimred = "PCA")
g_walk <- igraph::cluster_walktrap(g)
clus <- g_walk$membership
table(clus)
```

```
clus
  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
218 255 176 179  60 127 240 217  83  77 271 100 241 125  57 160  23  22  21  31
```

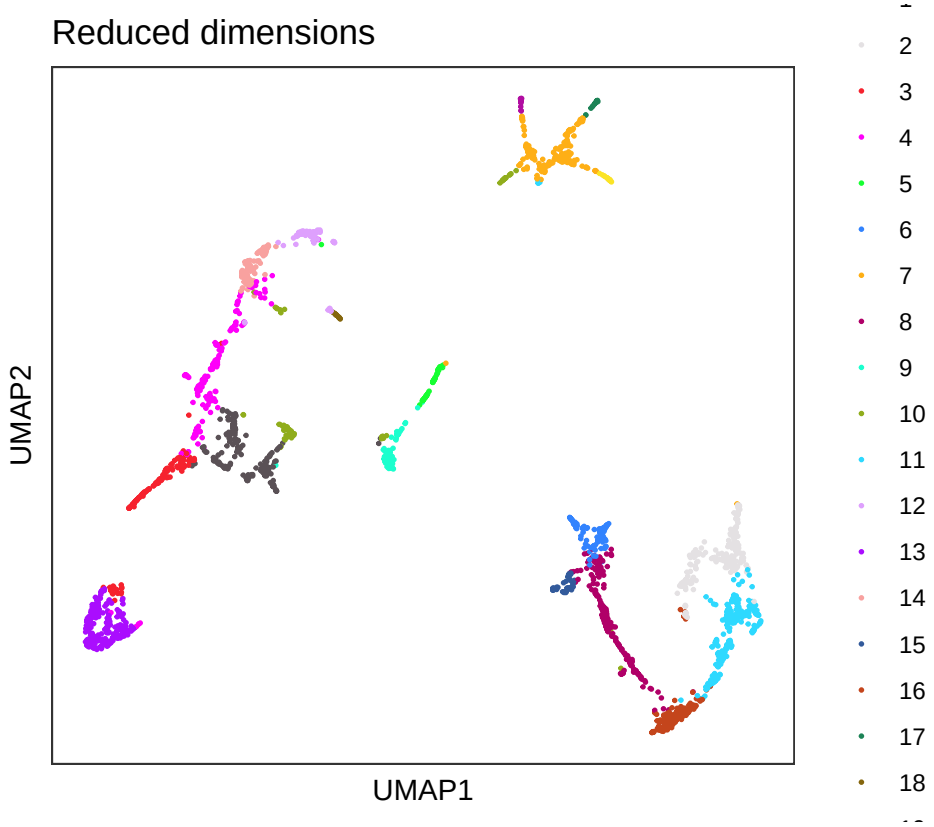
```
# store cluster labels in column 'label' in colData
colLabels(spe) <- factor(clus)
```

Visualize the clusters by plotting in (i) spatial (x-y) coordinates on the tissue slide, and (ii) UMAP dimensions.

```r
# define custom color palette
colors <- unname(palette.colors(palette = "Polychrome 36"))

# plot clusters in spatial x-y coordinates
plotSpots(spe, annotate = "label",
          palette = colors)
```




```r
# plot clusters in UMAP dimensions
plotDimRed(spe, type = "UMAP",
           annotate = "label", palette = colors)
```



11.10 Marker genes

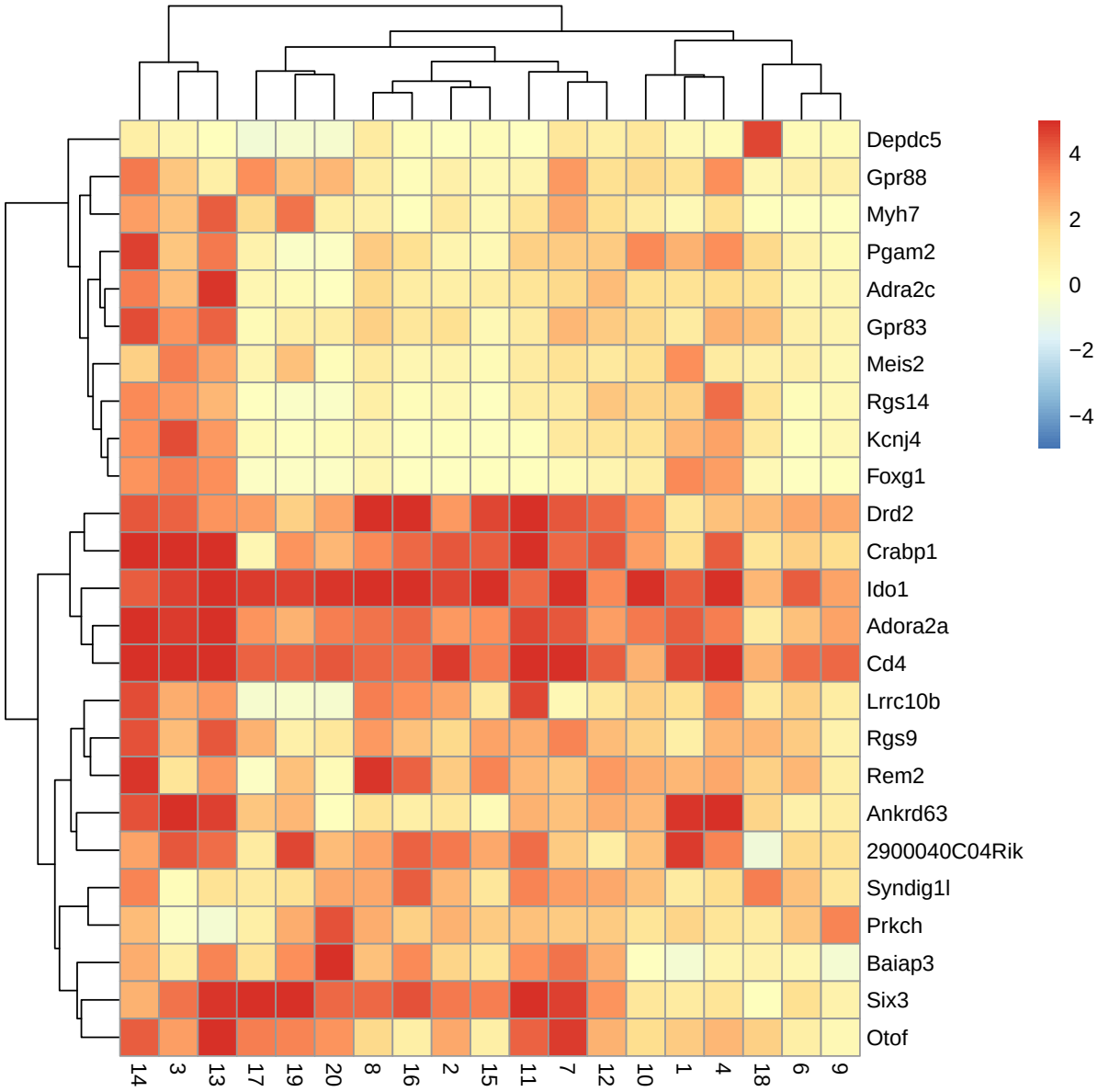
Identify marker genes by testing for differential gene expression between clusters, using the binomial test implemented in `findMarkers` in `scran` (Lun, McCarthy, and Marioni 2016).

```
# set gene names as row names for easier plotting
rownames(spe) <- rowData(spe)$gene_name

# test for marker genes
markers <- findMarkers(spe, test = "binom", direction = "up")

# returns a list with one DataFrame per cluster
markers
```

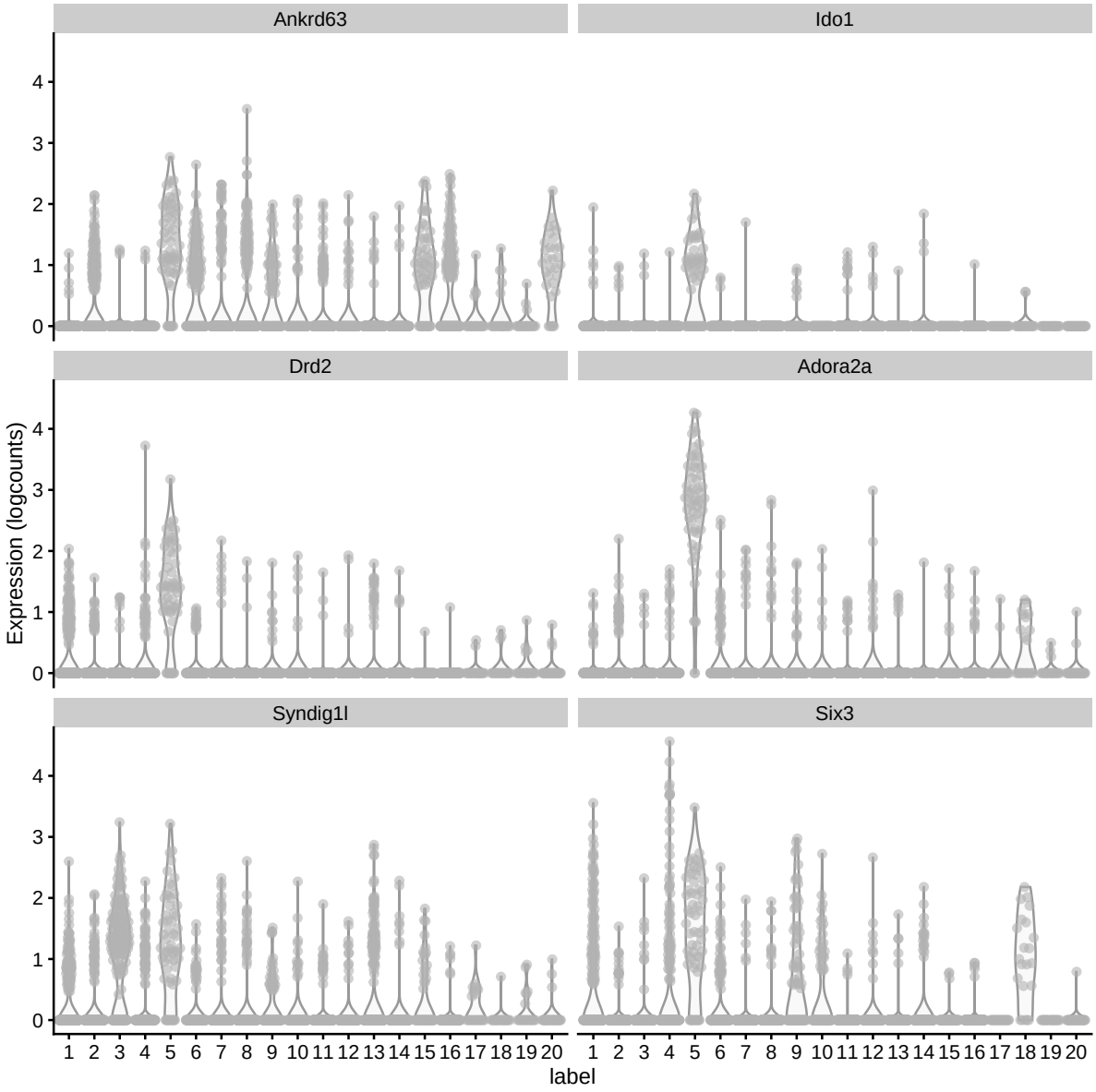
```
List of length 20
names(20): 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

```
library(pheatmap)
```

```
# plot log-fold changes for one cluster over all other clusters
# selecting cluster 5
interesting <- markers[[5]]
best_set <- interesting[interesting$Top <= 5, ]
logFCs <- getMarkerEffects(best_set)

pheatmap(logFCs, breaks = seq(-5, 5, length.out = 101))
```




```
# plot log-transformed normalized expression of top genes for one cluster
top_genes <- head(rownames(interesting))

plotExpression(spe, x = "label", features = top_genes)
```


Ankrd63
Ido1
Drd2
Adora2a
Syndig1l
Six3
Expression (logcounts)
label
0
1
2
3
4
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

12 spatialLIBD workflow

12.1 Overview

In the previous workflow, Human DLPFC workflow, you practiced some of the basics with a portion of the postmortem human brain dataset Maynard et al. (2021). The goal of this workflow is to learn what steps you need to carry out in order to create an interactive website to visualize this type of data. For this, we'll use the `spatialLIBD` Bioconductor package Pardo et al. (2021).

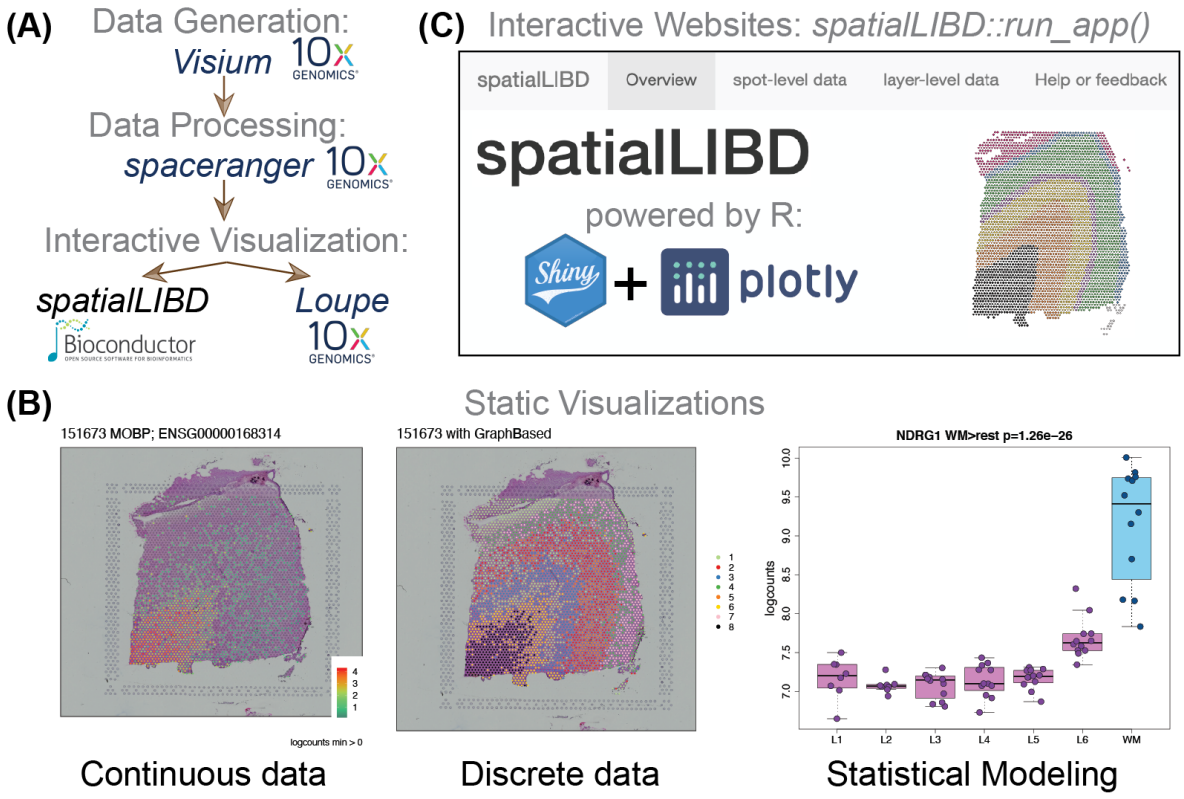


Figure 12.1: `spatialLIBD` overview. Source: Pardo et al, *bioRxiv*, 2021.

12.2 spatialLIBD

12.2.1 Why use spatialLIBD?

Before we dive into the R code, let's first revisit why you might want to use `spatialLIBD`. This package has a function, `spatialLIBD::run_app(spe)`, which will create an interactive website using a `SpatialExperiment` object (`spe`). The interactive website it creates has several features that were initially designed for a specific dataset Maynard et al. (2021) and later made flexible for any dataset Pardo et al. (2021). These features include panels to visualize spots from the Visium platform by 10x Genomics:

- for one tissue section at a time, either with interactive or static versions
- multiple tissue sections at a time, either interactively or statically

Both options work with continuous and discrete variables such as the gene expression and clusters, respectively. The interactive version for discrete variables such as clusters is useful if you want to manually annotate Visium spots, as it was done in the initial project Maynard et al. (2021). `spatialLIBD` allows users to download the annotated spots and resume your spot annotation work later.

Visualizing genes or clusters across multiple tissue sections can be quite useful. For example, here we show the expression levels of *PCP4* across two sets of spatially adjacent replicates. *PCP4* is a marker gene for layer 5 in the grey matter of the dorsolateral prefrontal cortex (DLPFC) in the human brain. Spatially adjacent replicates are about 10 microns apart from each other and visualizations like the one below help assess the technical variability in the Visium technology.

You can try out a `spatialLIBD`-powered website yourself by opening it on your browser [1].

12.2.2 Want to learn more about spatialLIBD?

If you are interested in learning more about `spatialLIBD`, please check the `spatialLIBD` Bioconductor landing page or the `pkgdown` documentation website. In particular, we have two vignettes documents:

- Introduction to spatialLIBD
- Using spatialLIBD with 10x Genomics public datasets

You can also read more about `spatialLIBD` in the associated publication.

[1] Check https://github.com/LieberInstitute/spatialLIBD#shiny-website-mirrors in case you need to use a mirror. `shiny`-powered websites work best on browsers such as Google Chrome and Mozilla Firefox, among others.

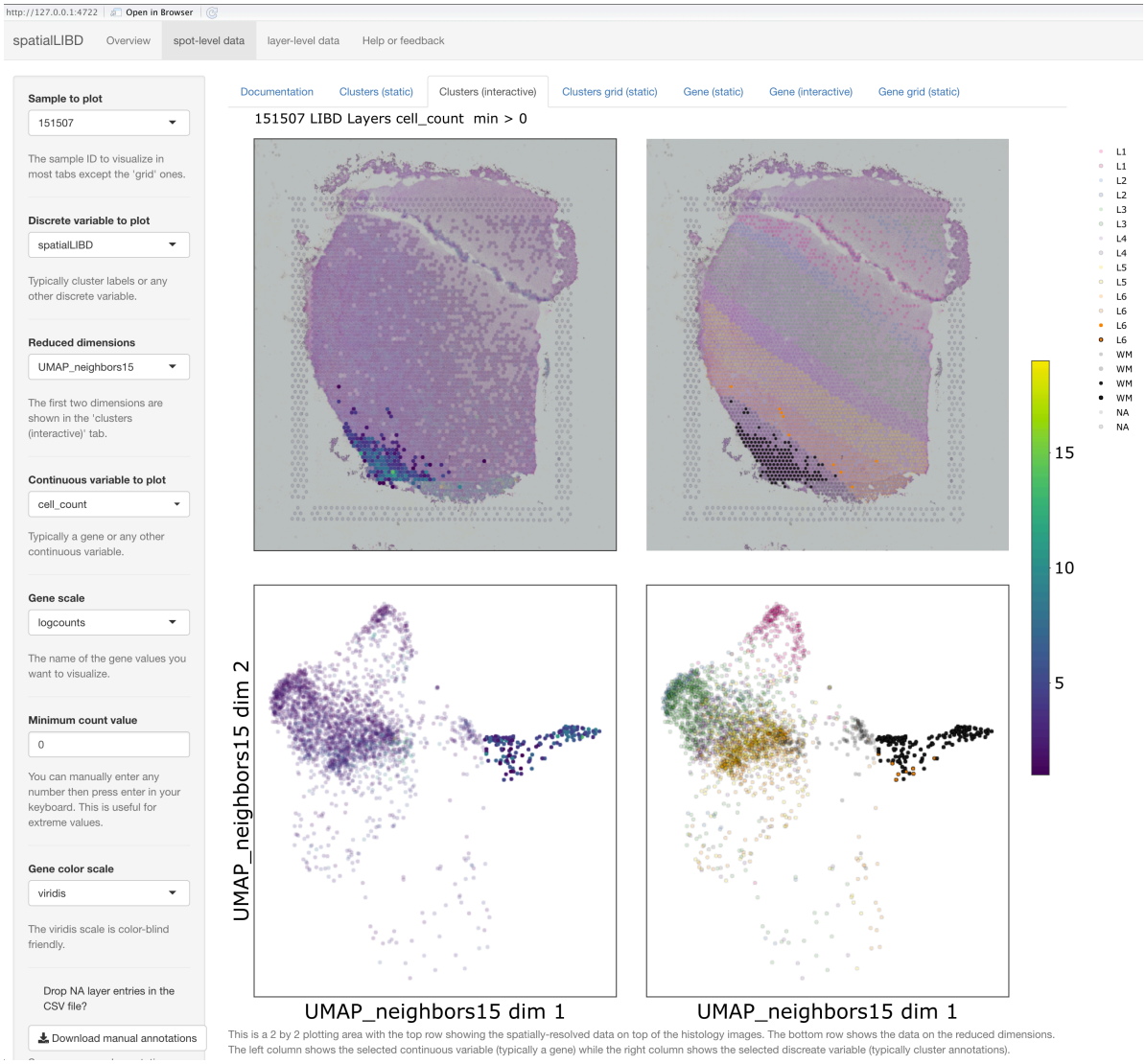


Figure 12.2: Screenshot of the 'clusters (interactive)' section of the 'spot-level data' panel created with the full spatialLIBD dataset. The website was created with `spatialLIBD::run_app(spatialLIBD::fetch_data('spe'))` version 1.4.0 and then using the lasso selection, we selected a set of spots in the UMAP interactive plot colored by the estimated number of cells per spot (`cell_count`) on the bottom left, which automatically updated the other three plots.

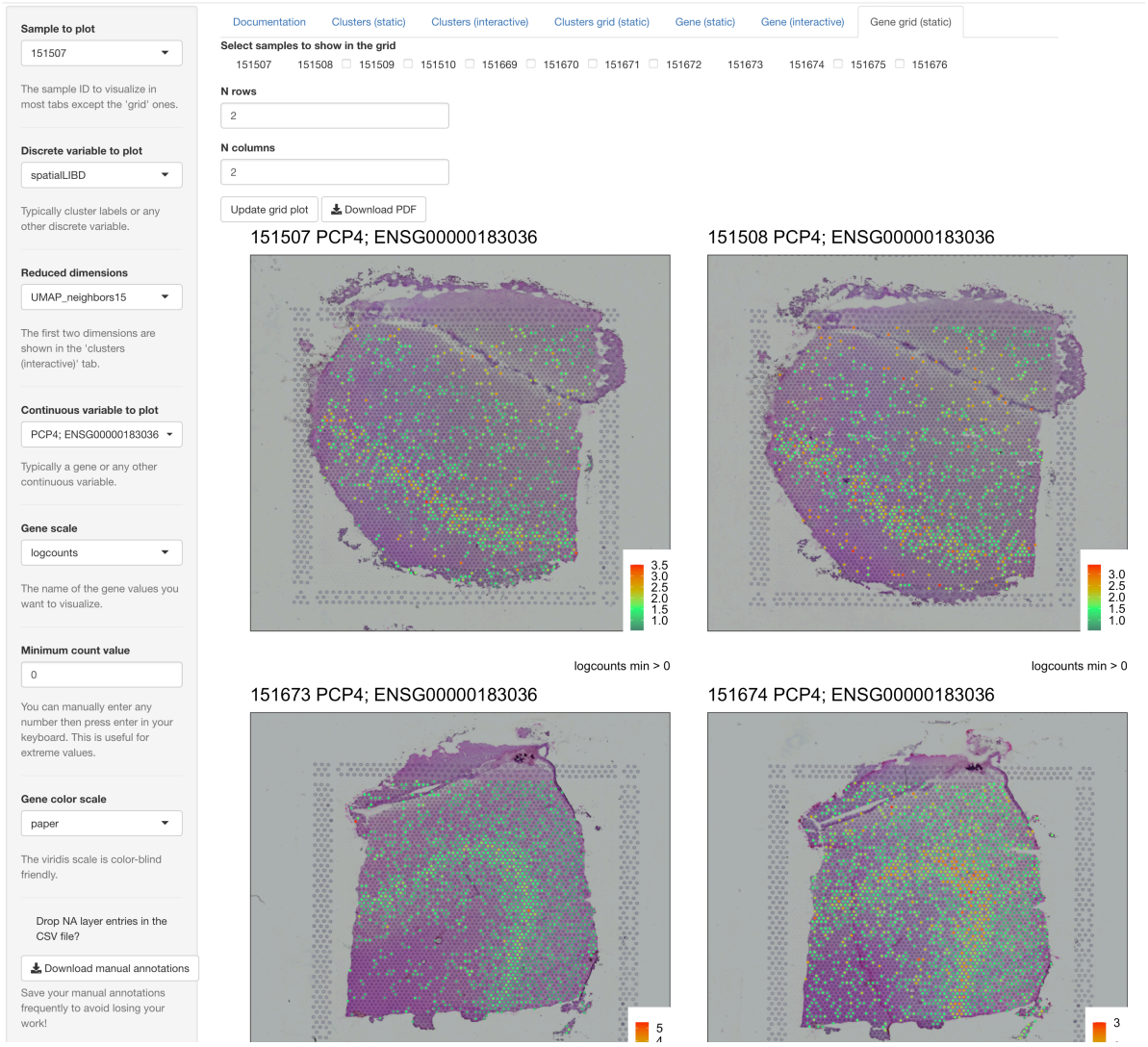


Figure 12.3: Screenshot of the 'gene grid (static)' section of the 'spot-level data' panel created with the full spatialLIBD dataset. The website was created with `spatialLIBD::run_app(spatialLIBD::fetch_data('spe'))` version 1.4.0, selecting the *PCP4* gene, selecting the *paper* gene color scale, changing the number of rows and columns in the grid 2, selecting two pairs of spatially adjacent replicate samples (151507, 151508, 151673, and 151674), and clicking on the *upgrade grid plot* button. Note that the default *viridis* gene color scale is color-blind friendly.

```
citation("spatialLIBD")[1]
```

```
Pardo B, Spangler A, Weber LM, Hicks SC, Jaffe AE, Martinowich K,
Maynard KR, Collado-Torres L (2022). "spatialLIBD: an R/Bioconductor
package to visualize spatially-resolved transcriptomics data." _BMC
Genomics_. doi:10.1186/s12864-022-08601-w
<https://doi.org/10.1186/s12864-022-08601-w>,
<https://doi.org/10.1186/s12864-022-08601-w>.

A BibTeX entry for LaTeX users is

  @Article{,
    title = {spatialLIBD: an R/Bioconductor package to visualize spatially-resolved transcri
    author = {Brenda Pardo and Abby Spangler and Lukas M. Weber and Stephanie C. Hicks and A
    year = {2022},
    journal = {BMC Genomics},
    doi = {10.1186/s12864-022-08601-w},
    url = {https://doi.org/10.1186/s12864-022-08601-w},
  }
```

12.2.2.1 Recordings

If you prefer to watch recordings of presentations related to the dataset Maynard et al. (2021) or `spatialLIBD` Pardo et al. (2021), check the following [2]:

These slides were part of our 2021-04-27 webinar for BioTuring that you can watch on YouTube:

A recording of an earlier version of this talk is also available on YouTube.

You might also be interested in this video demonstration of `spatialLIBD` for the LIBD rstats club.

12.3 Code prerequisites

Ok, let's get started! First we need to re-create the `spe` object from the Human DLPFC workflow in the previous chapter. That chapter included code for visualizing results along the way, which we'll skip here. Thus, we'll use the following packages in addition to `spatialLIBD`:

[2] Originally available at https://github.com/LieberInstitute/spatialLIBD/blob/master/inst/app/www/documentation_spe.md#sli and-videos.

- SpatialExperiment: for storing our data in a common object
- STexampleData: for accessing the example data
- scater: for quality control checks
- scran: for normalization, dimension reduction, and clustering
- igraph: for clustering algorithms
- BiocFileCache: for downloading and storing data
- rtracklayer: for importing gene annotation files
- lobstr: for checking object memory usage

You'll need to have the R version compatible with `bioc-release` [3] installed in your computer, as documented by Bioconductor. Alternatively, you can use the Bioconductor docker images. Next, if you haven't installed these packages, please do so with the following R code.

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

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

## Install the required R packages for this workflow
BiocManager::install(c(
    "SpatialExperiment",
    "STexampleData",
    "scater",
    "scran",
    "igraph",
    "BiocFileCache",
    "rtracklayer",
    "lobstr",
    "spatialLIBD"
))
```

We can now run the following R code to re-make the `spe` object from the Human DLPFC workflow. This will take a bit of time.

```
## Load packages required for the
## "Visium human DLPFC workflow"
library("SpatialExperiment")
```

[3] This book is under development, so right now you actually need to use `bioc-devel`. Once this book is submitted to Bioconductor, then it'll work with `bioc-release`. TODO: remove this comment when the book is available on `bioc-release`.

```
library("STexampleData")
library("scater")
library("scran")
library("igraph")
library("BiocFileCache")
library("rtracklayer")
library("lobstr")
library("spatialLIBD")

## Start tracking time
time_start <- Sys.time()

# load object
spe <- Visium_humanDLPFC()

# subset to keep only spots over tissue
spe <- spe[, colData(spe)$in_tissue == 1]

# identify mitochondrial genes
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)

# calculate per-spot QC metrics and store in colData
spe <- addPerCellQC(spe, subsets = list(mito = is_mito))

# select QC thresholds
qc_lib_size <- colData(spe)$sum < 600
qc_detected <- colData(spe)$detected < 400
qc_mito <- colData(spe)$subsets_mito_percent > 28
qc_cell_count <- colData(spe)$cell_count > 10

# combined set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count

# store in object
colData(spe)$discard <- discard

# filter low-quality spots
spe <- spe[, !colData(spe)$discard]

# calculate logcounts using library size factors
spe <- logNormCounts(spe)
```

```r
# remove mitochondrial genes
spe <- spe[!is_mito, ]

# fit mean-variance relationship
dec <- modelGeneVar(spe)

# select top HVGs
top_hvgs <- getTopHVGs(dec, prop = 0.1)

# compute PCA
set.seed(123)
spe <- runPCA(spe, subset_row = top_hvgs)

# compute UMAP on top 50 PCs
set.seed(123)
spe <- runUMAP(spe, dimred = "PCA")

# update column names for easier plotting
colnames(reducedDim(spe, "UMAP")) <- paste0("UMAP", 1:2)

# graph-based clustering
set.seed(123)
k <- 10
g <- buildSNNGraph(spe, k = k, use.dimred = "PCA")
g_walk <- igraph::cluster_walktrap(g)
clus <- g_walk$membership

# store cluster labels in column 'label' in colData
colLabels(spe) <- factor(clus)

# set gene names as row names for easier plotting
rownames(spe) <- rowData(spe)$gene_name

# test for marker genes
markers <- findMarkers(spe, test = "binom", direction = "up")

## Find the interesting markers for each cluster
interesting <- sapply(markers, function(x) x$Top <= 5)
colnames(interesting) <- paste0("gene_interest_", seq_len(length(markers)))
rowData(spe) <- cbind(rowData(spe), interesting)
```

```
## How long this code took to run
time_prereqs <- Sys.time()
time_prereqs - time_start
```

```
Time difference of 1.3031 mins
```

12.4 Prepare for spatialLIBD

Now that we have a `spe` object with quality control information, dimension reduction results, clustering data, among other things, we can proceed to visualize the object using `spatialLIBD`. Well, almost. First we need to modify the `spe` object, similar to steps we need to carry out when using spatialLIBD with 10x Genomics public datasets

12.4.1 Basic information

```
## Add some information used by spatialLIBD
spe$key <- paste0(spe$sample_id, "_", colnames(spe))
spe$sum_umi <- colSums(counts(spe))
spe$sum_gene <- colSums(counts(spe) > 0)
```

12.4.2 Gene annotation

Since the gene information is missing, we'll add the gene annotation data from Gencode although you would ideally add this information from the same gene annotation you used for running `spaceranger`.

```
## Download the Gencode v32 GTF file and cache it
bfc <- BiocFileCache::BiocFileCache()
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
```

```
"/Users/lukas/Library/Caches/org.R-project.R/R/BiocFileCache/693465337611_gencode.v32.annotat
```

```
## 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(rowData(spe)$gene_id, gtf$gene_id)
table(is.na(match_genes))
```

```
FALSE  TRUE
33267   258
```

```
## 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")]

## Save the "interest"ing columns from our original spe object
interesting <- rowData(spe)[, grepl("interest", colnames(rowData(spe)))]

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

## Add back the "interest" coolumns
```

```
rowData(spe) <- cbind(rowData(spe), interesting)

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

```
GRanges object with 33267 ranges and 12 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
              ...      ...               ...    ... .      ...      ...
  ENSG00000160298    chr21 46300181-46323875      - |   HAVANA     gene
  ENSG00000160299    chr21 46324141-46445769      + |   HAVANA     gene
  ENSG00000160305    chr21 46458891-46570015      + |   HAVANA     gene
  ENSG00000160307    chr21 46598604-46605208      - |   HAVANA     gene
  ENSG00000160310    chr21 46635595-46665124      + |   HAVANA     gene
                        gene_id   gene_name      gene_type gene_interest_1
                     <character> <character>    <character>       <logical>
  ENSG00000243485 ENSG00000243485 MIR1302-2HG         lncRNA            TRUE
  ENSG00000237613 ENSG00000237613     FAM138A         lncRNA            TRUE
  ENSG00000186092 ENSG00000186092       OR4F5 protein_coding            TRUE
  ENSG00000238009 ENSG00000238009  AL627309.1         lncRNA            TRUE
  ENSG00000239945 ENSG00000239945  AL627309.3         lncRNA            TRUE
              ...             ...         ...            ...             ...
  ENSG00000160298 ENSG00000160298    C21orf58 protein_coding           FALSE
  ENSG00000160299 ENSG00000160299        PCNT protein_coding           FALSE
  ENSG00000160305 ENSG00000160305       DIP2A protein_coding           FALSE
  ENSG00000160307 ENSG00000160307       S100B protein_coding           FALSE
  ENSG00000160310 ENSG00000160310       PRMT2 protein_coding           FALSE
                  gene_interest_2 gene_interest_3 gene_interest_4
                        <logical>       <logical>       <logical>
  ENSG00000243485            TRUE            TRUE            TRUE
  ENSG00000237613            TRUE            TRUE            TRUE
  ENSG00000186092            TRUE            TRUE            TRUE
  ENSG00000238009            TRUE            TRUE            TRUE
  ENSG00000239945            TRUE            TRUE            TRUE
              ...             ...             ...             ...
  ENSG00000160298           FALSE           FALSE           FALSE
```

```
ENSG00000160299           FALSE           FALSE           FALSE
ENSG00000160305           FALSE           FALSE           FALSE
ENSG00000160307           FALSE           FALSE           FALSE
ENSG00000160310           FALSE           FALSE           FALSE
                gene_interest_5 gene_interest_6 gene_interest_7
                      <logical>       <logical>       <logical>
ENSG00000243485            TRUE            TRUE            TRUE
ENSG00000237613            TRUE            TRUE            TRUE
ENSG00000186092            TRUE            TRUE            TRUE
ENSG00000238009            TRUE            TRUE            TRUE
ENSG00000239945            TRUE            TRUE            TRUE
            ...             ...             ...             ...
ENSG00000160298           FALSE           FALSE           FALSE
ENSG00000160299           FALSE           FALSE           FALSE
ENSG00000160305           FALSE           FALSE           FALSE
ENSG00000160307           FALSE           FALSE           FALSE
ENSG00000160310           FALSE           FALSE           FALSE
-------
seqinfo: 25 sequences from an unspecified genome; no seqlengths
```

Now that we have the gene annotation information, we can use it to add a few more pieces to our `spe` object that `spatialLIBD` will use. For example, we want to enable users to search genes by either their gene symbol or their Ensembl ID. We also like to visualize the amount and percent of the mitochondrial gene expression.

```
## 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
```

12.4.3 Extra information and filtering

Now that we have the full gene annotation information we need, we can proceed to add some last touches as well as filter the object to reduce the memory required for visualizing the data.

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

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

## Number of genes with no counts
length(no_expr)
```

```
[1] 11596
```

```
## Compute the percent of genes with no counts
length(no_expr) / nrow(spe) * 100
```

```
[1] 34.85737
```

```
spe <- spe[-no_expr, , drop = FALSE]

## Remove spots without counts
summary(spe$sum_umi)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    537    2414    3466    3870    4938   15862
```

```
## 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]
}
```

We *think* that we are ready to proceed to making our interactive website. Let's use the `spatialLIBD::check_spe()` function, just to verify that we are right. If we aren't, then it'll try to tell us what we missed.

```
## Run check_spe() function
spatialLIBD::check_spe(spe)
```

```
class: SpatialExperiment
dim: 21671 3524
metadata(0):
assays(2): counts logcounts
rownames(21671): ENSG00000243485 ENSG00000238009 ... ENSG00000160307
  ENSG00000160310
rowData names(13): source type ... gene_interest_7 gene_search
colnames(3524): AAACAAGTATCTCCCA-1 AAACACCAATAACTGC-1 ...
  TTGTTTCCATACAACT-1 TTGTTTGTGTAAATTC-1
colData names(22): barcode_id sample_id ... expr_chrM_ratio
  ManualAnnotation
reducedDimNames(2): PCA UMAP
mainExpName: NULL
altExpNames(0):
spatialCoords names(2) : pxl_col_in_fullres pxl_row_in_fullres
imgData names(4): sample_id image_id data scaleFactor
```

```
## End tracking time
time_end <- Sys.time()

## How long this code took to run
time_end - time_prereqs
```

```
Time difference of 38.73031 secs
```

Creating our final `spe` object took 1.94860501686732 to run. So you might want to save this object for later use.

```
saveRDS(spe, file = "spe_workflow_Visium_spatialLIBD.rds")
```

You can then re-load it with the following code on a later session.

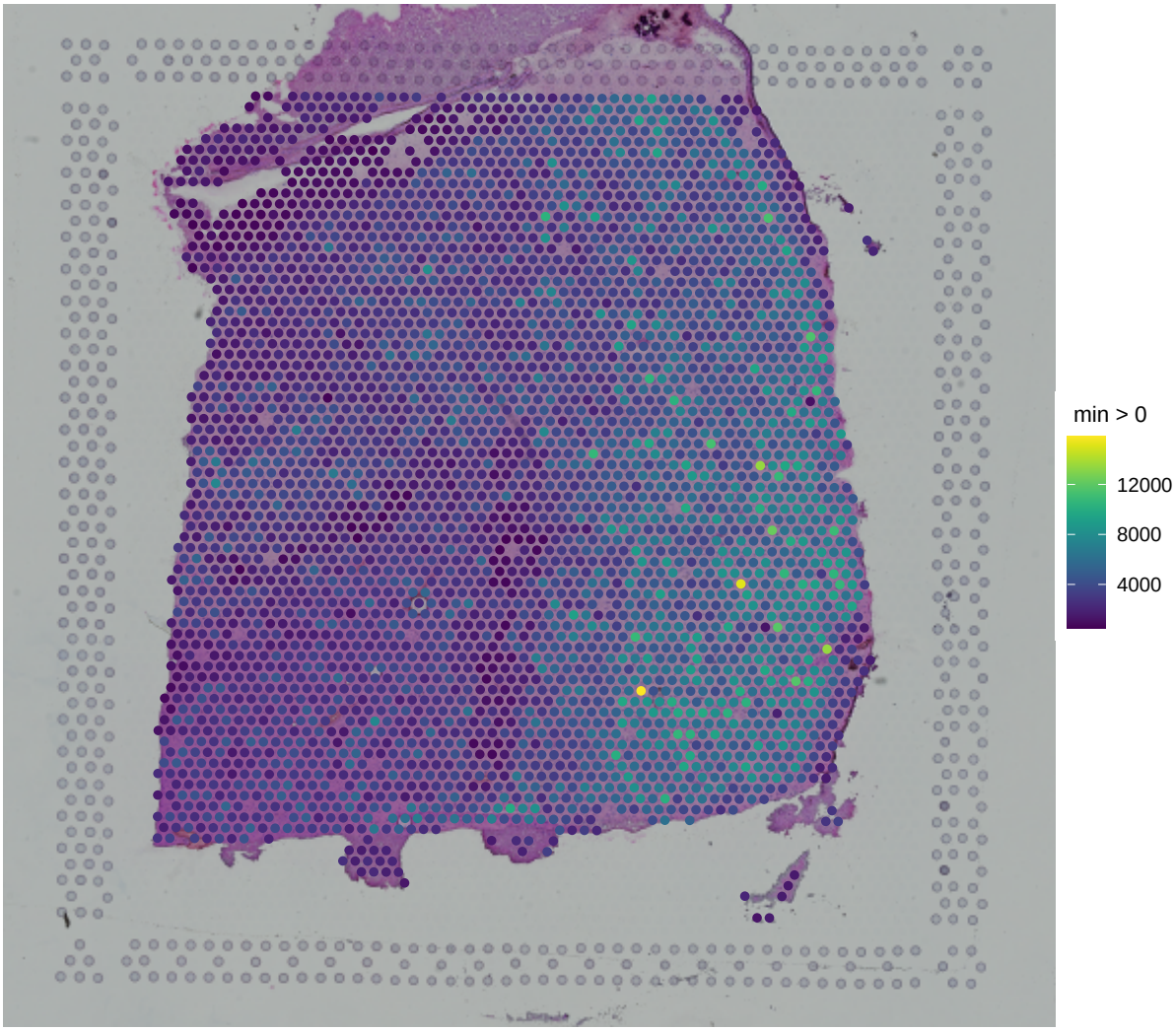
```
spe <- readRDS("spe_workflow_Visium_spatialLIBD.rds")
```

12.5 Explore the data

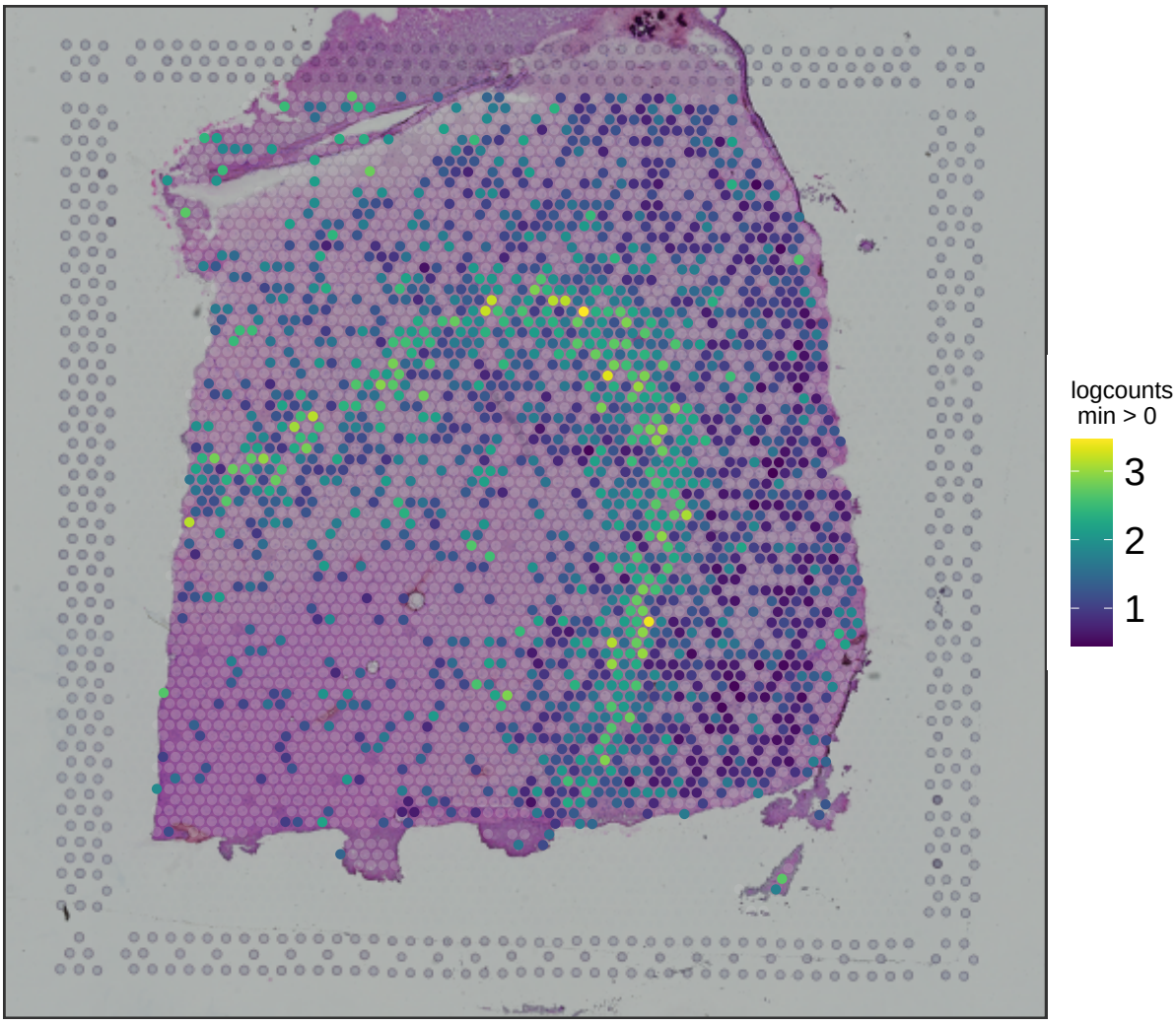
In order to visualize the data, we can then use `spatialLIBD::vis_gene()`. Note that we didn't need to do all that hard work just for that. But well, this is a nice quick check before we try launching our interactive website.

```
## Sum of UMI
spatialLIBD::vis_gene(
    spe = spe,
    sampleid = "sample_151673",
    geneid = "sum_umi"
)
```

sample_151673 sum_umi




```
## PCP4, a layer 5 marker gene
spatialLIBD::vis_gene(
    spe = spe,
    sampleid = "sample_151673",
    geneid = rowData(spe)$gene_search[which(rowData(spe)$gene_name == "PCP4")]
)
```



As we wanted let's proceed to visualize the data interactively with a `spatialLIBD`-powered website. We have lots of variables to choose from. We'll specify which are our continuous and discrete variables in our `spatialLIBD::run_app()` call.

```
## Explore all the variables we can use
colData(spe)
```

```
DataFrame with 3524 rows and 22 columns
                           barcode_id     sample_id in_tissue array_row
```

```
                       <character>   <character> <integer> <integer>
AAACAAGTATCTCCCA-1 AAACAAGTATCTCCCA-1 sample_151673         1        50
AAACACCAATAACTGC-1 AAACACCAATAACTGC-1 sample_151673         1        59
AAACAGAGCGACTCCT-1 AAACAGAGCGACTCCT-1 sample_151673         1        14
AAACAGCTTTCAGAAG-1 AAACAGCTTTCAGAAG-1 sample_151673         1        43
AAACAGGGTCTATATT-1 AAACAGGGTCTATATT-1 sample_151673         1        47
...                             ...           ...       ...       ...
TTGTTGTGTGTCAAGA-1 TTGTTGTGTGTCAAGA-1 sample_151673         1        31
TTGTTTCACATCCAGG-1 TTGTTTCACATCCAGG-1 sample_151673         1        58
TTGTTTCATTAGTCTA-1 TTGTTTCATTAGTCTA-1 sample_151673         1        60
TTGTTTCCATACAACT-1 TTGTTTCCATACAACT-1 sample_151673         1        45
TTGTTTGTGTAAATTC-1 TTGTTTGTGTAAATTC-1 sample_151673         1         7
                   array_col ground_truth cell_count       sum  detected
                   <integer>  <character>  <integer> <numeric> <numeric>
AAACAAGTATCTCCCA-1       102       Layer3          6      8458      3586
AAACACCAATAACTGC-1        19           WM          5      3769      1960
AAACAGAGCGACTCCT-1        94       Layer3          2      5433      2424
AAACAGCTTTCAGAAG-1         9       Layer5          4      4278      2264
AAACAGGGTCTATATT-1        13       Layer6          6      4004      2178
...                      ...          ...        ...       ...       ...
TTGTTGTGTGTCAAGA-1        77       Layer5          3      3966      1982
TTGTTTCACATCCAGG-1        42           WM          3      4324      2170
TTGTTTCATTAGTCTA-1        30           WM          4      2761      1560
TTGTTTCCATACAACT-1        27       Layer6          3      2322      1343
TTGTTTGTGTAAATTC-1        51       Layer2          5      6281      2927
                   subsets_mito_sum subsets_mito_detected subsets_mito_percent
                          <numeric>             <numeric>            <numeric>
AAACAAGTATCTCCCA-1             1407                    13              16.6351
AAACACCAATAACTGC-1              430                    13              11.4089
AAACAGAGCGACTCCT-1             1316                    13              24.2223
AAACAGCTTTCAGAAG-1              651                    12              15.2174
AAACAGGGTCTATATT-1              621                    13              15.5095
...                             ...                   ...                  ...
TTGTTGTGTGTCAAGA-1              789                    13             19.89410
TTGTTTCACATCCAGG-1              370                    12              8.55689
TTGTTTCATTAGTCTA-1              314                    12             11.37269
TTGTTTCCATACAACT-1              476                    13             20.49957
TTGTTTGTGTAAATTC-1              991                    13             15.77774
                       total   discard sizeFactor    label
                   <numeric> <logical>  <numeric> <factor>
AAACAAGTATCTCCCA-1      8458     FALSE   1.822839        3
AAACACCAATAACTGC-1      3769     FALSE   0.812282        1
AAACAGAGCGACTCCT-1      5433     FALSE   1.170902        3
```

```
AAACAGCTTTCAGAAG-1      4278     FALSE   0.921980        3
AAACAGGGTCTATATT-1      4004     FALSE   0.862928        6
...                      ...       ...        ...      ...
TTGTTGTGTGTCAAGA-1      3966     FALSE   0.854739        4
TTGTTTCACATCCAGG-1      4324     FALSE   0.931894        1
TTGTTTCATTAGTCTA-1      2761     FALSE   0.595041        1
TTGTTTCCATACAACT-1      2322     FALSE   0.500429        2
TTGTTTGTGTAAATTC-1      6281     FALSE   1.353660        7
                                    key   sum_umi  sum_gene expr_chrM
                              <character> <numeric> <integer> <numeric>
AAACAAGTATCTCCCA-1 sample_151673_AAACAA..      7051      3573         0
AAACACCAATAACTGC-1 sample_151673_AAACAC..      3339      1947         0
AAACAGAGCGACTCCT-1 sample_151673_AAACAG..      4117      2411         0
AAACAGCTTTCAGAAG-1 sample_151673_AAACAG..      3627      2252         0
AAACAGGGTCTATATT-1 sample_151673_AAACAG..      3383      2165         0
...                                   ...       ...       ...       ...
TTGTTGTGTGTCAAGA-1 sample_151673_TTGTTG..      3177      1969         0
TTGTTTCACATCCAGG-1 sample_151673_TTGTTT..      3954      2158         0
TTGTTTCATTAGTCTA-1 sample_151673_TTGTTT..      2447      1548         0
TTGTTTCCATACAACT-1 sample_151673_TTGTTT..      1846      1330         0
TTGTTTGTGTAAATTC-1 sample_151673_TTGTTT..      5290      2914         0
                   expr_chrM_ratio ManualAnnotation
                         <numeric>      <character>
AAACAAGTATCTCCCA-1               0               NA
AAACACCAATAACTGC-1               0               NA
AAACAGAGCGACTCCT-1               0               NA
AAACAGCTTTCAGAAG-1               0               NA
AAACAGGGTCTATATT-1               0               NA
...                            ...              ...
TTGTTGTGTGTCAAGA-1               0               NA
TTGTTTCACATCCAGG-1               0               NA
TTGTTTCATTAGTCTA-1               0               NA
TTGTTTCCATACAACT-1               0               NA
TTGTTTGTGTAAATTC-1               0               NA
```

```
## Run our shiny app
if (interactive()) {
    spatialLIBD::run_app(
        spe,
        sce_layer = NULL,
        modeling_results = NULL,
        sig_genes = NULL,
```

```
        title = "OSTA spatialLIBD workflow example",
        spe_discrete_vars = c("ground_truth", "label", "ManualAnnotation"),
        spe_continuous_vars = c(
            "cell_count",
            "sum_umi",
            "sum_gene",
            "expr_chrM",
            "expr_chrM_ratio",
            "sum",
            "detected",
            "subsets_mito_sum",
            "subsets_mito_detected",
            "subsets_mito_percent",
            "total",
            "sizeFactor"
        ),
        default_cluster = "label"
    )
}
```

12.6 Sharing your website

Now that you have created a `spatialLIBD`-powered website, you might be interested in sharing it. To do so, it'll be useful to have saved a small `spe` object using `saveRDS()` like we did earlier. The smaller the object, the better in terms of performance. For example, you might want to remove the lowly expressed genes to save space. One function you can use to measure how big your object is is `lobstr::obj_size()` as shown below.

```
## Object size
lobstr::obj_size(spe) / 1024^2 ## Convert to MB
```

```
293.12 B
```

If your data is small enough, you might want to share your website by hosting on shinyapps.io by RStudio, which you can try for free. Once you have created your account, you'll want to create an `app.R` file like the one we have on the `spatialLIBD_demo` directory.

```
library("spatialLIBD")
 library("markdown") # for shinyapps.io
```

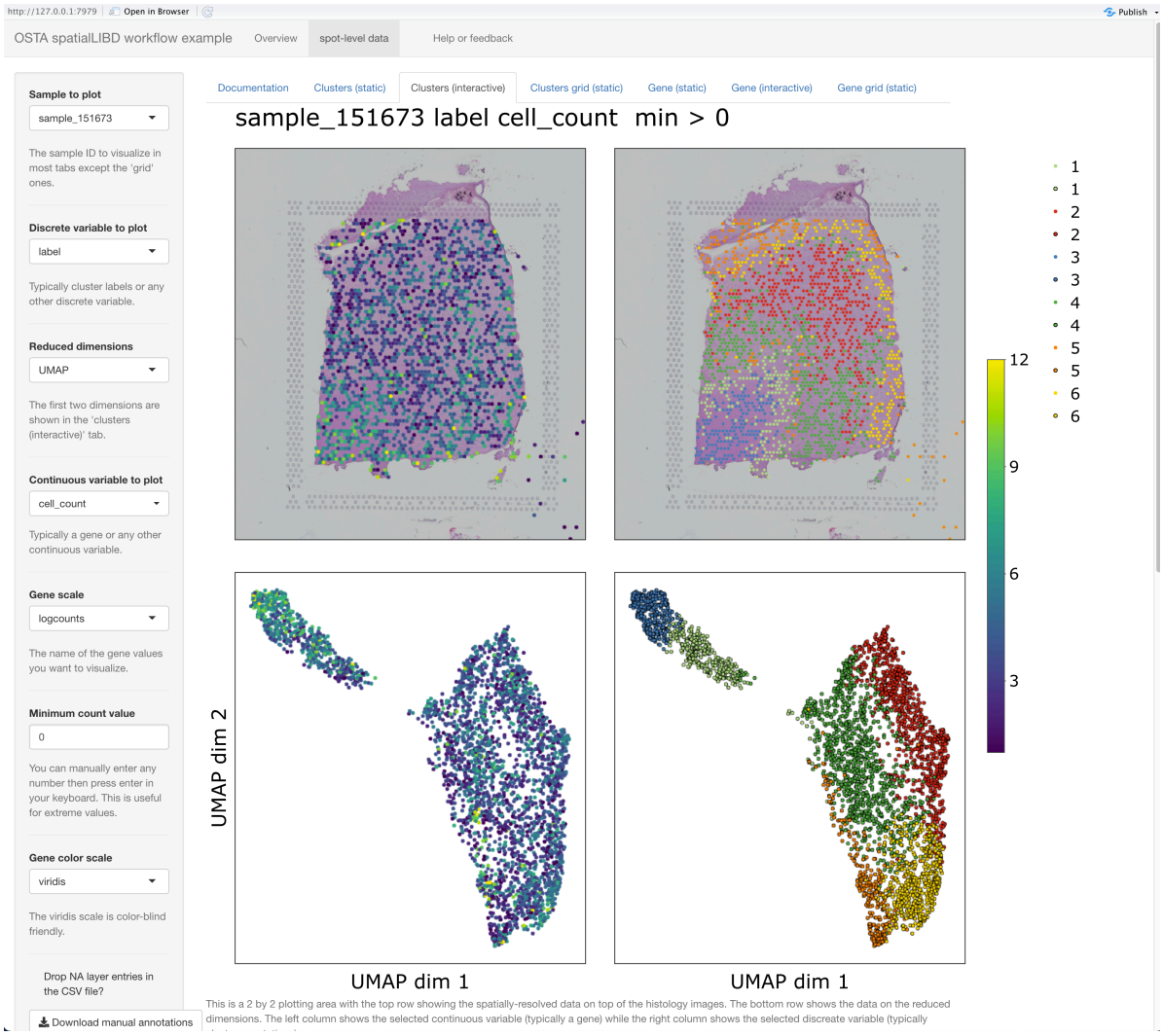


Figure 12.4: Screenshot of the 'clusters (interactive)' section of the 'spot-level data' panel created with with the data from this workflow.

```
## 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
spe <- readRDS("spe_workflow_Visium_spatialLIBD.rds")

## Deploy the website
spatialLIBD::run_app(
    spe,
    sce_layer = NULL,
    modeling_results = NULL,
    sig_genes = NULL,
    title = "OSTA spatialLIBD workflow example",
    spe_discrete_vars = c("ground_truth", "label", "ManualAnnotation"),
    spe_continuous_vars = c(
        "cell_count",
        "sum_umi",
        "sum_gene",
        "expr_chrM",
        "expr_chrM_ratio",
        "sum",
        "detected",
        "subsets_mito_sum",
        "subsets_mito_detected",
        "subsets_mito_percent",
        "total",
        "sizeFactor"
    ),
    default_cluster = "label",
    docs_path = "www"
)
```

You can then open R in a new session in the same directory where you saved the `app.R` file, run the code and click on the "publish" blue button at the top right of your RStudio window. You'll then need to upload the `app.R` file, your `spe_workflow_Visium_spatialLIBD.rds` file and the files under the `www` directory which enable you to customize your `spatialLIDB` website.

The RStudio prompts will guide you along the process for authenticating to your shinyapps.io account, which will involve copy pasting some code that starts with

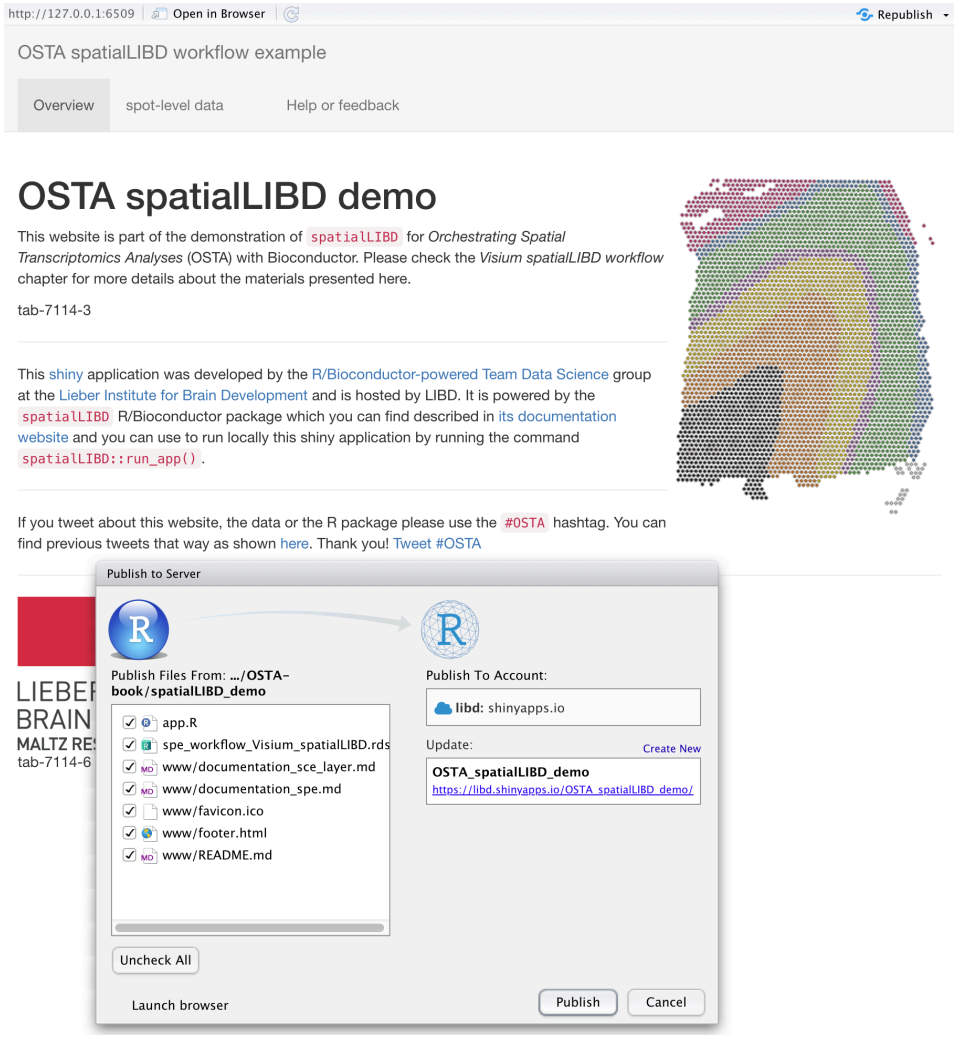


Figure 12.5: Screenshot of the RStudio window for publishing your spatialLIBD-powered website to shinyapps.io

`rsconnect::setAccountInfo()`. Alternatively, you can create a `deploy.R` script and write the code for uploading your files to shinyapps.io as shown below.

```
library('rsconnect')

 ## Or you can go to your shinyapps.io account and copy this
 ## Here we do this to keep our information hidden.
 load(".deploy_info.Rdata")
 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
 rsconnect::deployApp(
     appFiles = c(
         "app.R",
         "spe_workflow_Visium_spatialLIBD.rds",
         dir("www", full.names = TRUE)
     ),
     appName = 'OSTA_spatialLIBD_demo',
     account = 'libd',
     server = 'shinyapps.io'
 )
```

Note that we have copied the default `www` directory files from the `spatialLIBD` repository and adapted them to our liking. We then use these files with `spatialLIBD::run_app(docs_path)` in our `app.R` script. These files help us control portions of our `spatialLIBD`-powered website and customize it to our liking.

If you follow this workflow, 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
- *Advanced* `Max Connections`: 15

The `Max Worker Processes` determines how many R sessions are open per instance. Then `Max Connections` specifies the number of connections to each R session. The `Instance Size` determines the memory available. In this case, 8000 / 300 is approximately 27, but we decided to be conservative and set the total number of users per instance to be 15. This is why it can be important to reduce the size of your `spe` object before sharing the website. Alternatively, you can rent an AWS Instance and deploy your app there, which is how http://spatial.libd.org/spatialLIBD is hosted along with these error configuration files.

12.7 Wrapping up

Thank you for reading this far! In this workflow we showed you:

- why you might be interested in using `spatialLIBD`,
- we re-used the `spe` object from the Human DLPFC workflow chapter,
- we adapted the `spe` object to make it compatible with `spatialLIBD`,
- we created an interactive website in our laptops,
- we shared the website with others using shinyapps.io.

Overall, we hope that you found this information useful and we wish you the best luck with exploring and annotating your own Visium data!

R session information

Here's the R session information for this workflow.

```
options(width = 120)
sessioninfo::session_info()
```

```
- Session info -------------------------------------------------------------------------------
 setting  value
 version  R version 4.3.1 (2023-06-16)
 os       macOS Monterey 12.6.7
 system   x86_64, darwin20
 ui       X11
 language (EN)
 collate  en_US.UTF-8
 ctype    en_US.UTF-8
 tz       Australia/Perth
 date     2023-09-13
 pandoc   3.1.1 @ /Applications/quarto/bin/tools/ (via rmarkdown)
```

```
- Packages ---------------------------------------------------------------------
 package              * version   date (UTC) lib source
 abind                  1.4-5     2016-07-21 [1] CRAN (R 4.3.0)
 AnnotationDbi          1.62.2    2023-07-02 [1] Bioconductor
 AnnotationHub        * 3.8.0     2023-04-25 [1] Bioconductor
 attempt                0.3.1     2020-05-03 [1] CRAN (R 4.3.0)
 beachmat               2.16.0    2023-04-25 [1] Bioconductor
 beeswarm               0.4.0     2021-06-01 [1] CRAN (R 4.3.0)
 benchmarkme            1.0.8     2022-06-12 [1] CRAN (R 4.3.0)
 benchmarkmeData        1.0.4     2020-04-23 [1] CRAN (R 4.3.0)
 Biobase              * 2.60.0    2023-04-25 [1] Bioconductor
 BiocFileCache        * 2.8.0     2023-04-26 [1] Bioconductor
 BiocGenerics         * 0.46.0    2023-04-25 [1] Bioconductor
 BiocIO                 1.10.0    2023-05-11 [1] Bioconductor
 BiocManager            1.30.22   2023-08-08 [1] CRAN (R 4.3.0)
 BiocNeighbors          1.18.0    2023-04-25 [1] Bioconductor
 BiocParallel           1.34.2    2023-05-22 [1] Bioconductor
 BiocSingular           1.16.0    2023-04-25 [1] Bioconductor
 BiocVersion            3.17.1    2022-11-15 [1] Bioconductor
 Biostrings             2.68.1    2023-05-16 [1] Bioconductor
 bit                    4.0.5     2022-11-15 [1] CRAN (R 4.3.0)
 bit64                  4.0.5     2020-08-30 [1] CRAN (R 4.3.0)
 bitops                 1.0-7     2021-04-24 [1] CRAN (R 4.3.0)
 blob                   1.2.4     2023-03-17 [1] CRAN (R 4.3.0)
 bluster                1.10.0    2023-04-25 [1] Bioconductor
 bslib                  0.5.1     2023-08-11 [1] CRAN (R 4.3.0)
 cachem                 1.0.8     2023-05-01 [1] CRAN (R 4.3.0)
 cli                    3.6.1     2023-03-23 [1] CRAN (R 4.3.0)
 cluster                2.1.4     2022-08-22 [1] CRAN (R 4.3.1)
 codetools              0.2-19    2023-02-01 [1] CRAN (R 4.3.1)
 colorspace             2.1-0     2023-01-23 [1] CRAN (R 4.3.0)
 config                 0.3.2     2023-08-30 [1] CRAN (R 4.3.0)
 cowplot                1.1.1     2020-12-30 [1] CRAN (R 4.3.0)
 crayon                 1.5.2     2022-09-29 [1] CRAN (R 4.3.0)
 curl                   5.0.2     2023-08-14 [1] CRAN (R 4.3.0)
 data.table             1.14.8    2023-02-17 [1] CRAN (R 4.3.0)
 DBI                    1.1.3     2022-06-18 [1] CRAN (R 4.3.0)
 dbplyr               * 2.3.3     2023-07-07 [1] CRAN (R 4.3.0)
 DelayedArray           0.26.7    2023-07-28 [1] Bioconductor
 DelayedMatrixStats     1.22.6    2023-08-28 [1] Bioconductor
 digest                 0.6.33    2023-07-07 [1] CRAN (R 4.3.0)
 doParallel             1.0.17    2022-02-07 [1] CRAN (R 4.3.0)
```

```
dotCall64                1.0-2     2022-10-03 [1] CRAN (R 4.3.0)
dplyr                    1.1.3     2023-09-03 [1] CRAN (R 4.3.0)
dqrng                    0.3.1     2023-08-30 [1] CRAN (R 4.3.0)
DT                       0.28      2023-05-18 [1] CRAN (R 4.3.0)
edgeR                    3.42.4    2023-06-01 [1] Bioconductor
ellipsis                 0.3.2     2021-04-29 [1] CRAN (R 4.3.0)
evaluate                 0.21      2023-05-05 [1] CRAN (R 4.3.0)
ExperimentHub          * 2.8.1     2023-07-12 [1] Bioconductor
fansi                    1.0.4     2023-01-22 [1] CRAN (R 4.3.0)
farver                   2.1.1     2022-07-06 [1] CRAN (R 4.3.0)
fastmap                  1.1.1     2023-02-24 [1] CRAN (R 4.3.0)
fields                   15.2      2023-08-17 [1] CRAN (R 4.3.0)
filelock                 1.0.2     2018-10-05 [1] CRAN (R 4.3.0)
FNN                      1.1.3.2   2023-03-20 [1] CRAN (R 4.3.0)
foreach                  1.5.2     2022-02-02 [1] CRAN (R 4.3.0)
generics                 0.1.3     2022-07-05 [1] CRAN (R 4.3.0)
GenomeInfoDb           * 1.36.2    2023-08-25 [1] Bioconductor
GenomeInfoDbData         1.2.10    2023-05-10 [1] Bioconductor
GenomicAlignments        1.36.0    2023-05-11 [1] Bioconductor
GenomicRanges          * 1.52.0    2023-04-25 [1] Bioconductor
ggbeeswarm               0.7.2     2023-04-29 [1] CRAN (R 4.3.0)
ggplot2                * 3.4.3     2023-08-14 [1] CRAN (R 4.3.0)
ggrepel                  0.9.3     2023-02-03 [1] CRAN (R 4.3.0)
glue                     1.6.2     2022-02-24 [1] CRAN (R 4.3.0)
golem                    0.4.1     2023-06-05 [1] CRAN (R 4.3.0)
gridExtra                2.3       2017-09-09 [1] CRAN (R 4.3.0)
gtable                   0.3.4     2023-08-21 [1] CRAN (R 4.3.0)
htmltools                0.5.6     2023-08-10 [1] CRAN (R 4.3.0)
htmlwidgets              1.6.2     2023-03-17 [1] CRAN (R 4.3.0)
httpuv                   1.6.11    2023-05-11 [1] CRAN (R 4.3.0)
httr                     1.4.7     2023-08-15 [1] CRAN (R 4.3.0)
igraph                 * 1.5.1     2023-08-10 [1] CRAN (R 4.3.0)
interactiveDisplayBase   1.38.0    2023-04-25 [1] Bioconductor
IRanges                * 2.34.1    2023-06-22 [1] Bioconductor
irlba                    2.3.5.1   2022-10-03 [1] CRAN (R 4.3.0)
iterators                1.0.14    2022-02-05 [1] CRAN (R 4.3.0)
jquerylib                0.1.4     2021-04-26 [1] CRAN (R 4.3.0)
jsonlite                 1.8.7     2023-06-29 [1] CRAN (R 4.3.0)
KEGGREST                 1.40.0    2023-04-25 [1] Bioconductor
knitr                    1.43      2023-05-25 [1] CRAN (R 4.3.0)
labeling                 0.4.3     2023-08-29 [1] CRAN (R 4.3.0)
later                    1.3.1     2023-05-02 [1] CRAN (R 4.3.0)
lattice                  0.21-8    2023-04-05 [1] CRAN (R 4.3.1)
```

```
lazyeval                0.2.2     2019-03-15 [1] CRAN (R 4.3.0)
lifecycle               1.0.3     2022-10-07 [1] CRAN (R 4.3.0)
limma                   3.56.2    2023-06-06 [1] Bioconductor
lobstr                * 1.1.2     2022-06-22 [1] CRAN (R 4.3.0)
locfit                  1.5-9.8   2023-06-11 [1] CRAN (R 4.3.0)
magick                  2.7.5     2023-08-07 [1] CRAN (R 4.3.0)
magrittr                2.0.3     2022-03-30 [1] CRAN (R 4.3.0)
maps                    3.4.1     2022-10-30 [1] CRAN (R 4.3.0)
Matrix                  1.6-1     2023-08-14 [1] CRAN (R 4.3.0)
MatrixGenerics        * 1.12.3    2023-07-31 [1] Bioconductor
matrixStats           * 1.0.0     2023-06-02 [1] CRAN (R 4.3.0)
memoise                 2.0.1     2021-11-26 [1] CRAN (R 4.3.0)
metapod                 1.8.0     2023-04-25 [1] Bioconductor
mime                    0.12      2021-09-28 [1] CRAN (R 4.3.0)
munsell                 0.5.0     2018-06-12 [1] CRAN (R 4.3.0)
paletteer               1.5.0     2022-10-19 [1] CRAN (R 4.3.0)
pillar                  1.9.0     2023-03-22 [1] CRAN (R 4.3.0)
pkgconfig               2.0.3     2019-09-22 [1] CRAN (R 4.3.0)
plotly                  4.10.2    2023-06-03 [1] CRAN (R 4.3.0)
png                     0.1-8     2022-11-29 [1] CRAN (R 4.3.0)
prettyunits             1.1.1     2020-01-24 [1] CRAN (R 4.3.0)
promises                1.2.1     2023-08-10 [1] CRAN (R 4.3.0)
purrr                   1.0.2     2023-08-10 [1] CRAN (R 4.3.0)
R6                      2.5.1     2021-08-19 [1] CRAN (R 4.3.0)
rappdirs                0.3.3     2021-01-31 [1] CRAN (R 4.3.0)
RColorBrewer            1.1-3     2022-04-03 [1] CRAN (R 4.3.0)
Rcpp                    1.0.11    2023-07-06 [1] CRAN (R 4.3.0)
RCurl                   1.98-1.12 2023-03-27 [1] CRAN (R 4.3.0)
rematch2                2.1.2     2020-05-01 [1] CRAN (R 4.3.0)
restfulr                0.0.15    2022-06-16 [1] CRAN (R 4.3.0)
rjson                   0.2.21    2022-01-09 [1] CRAN (R 4.3.0)
rlang                   1.1.1     2023-04-28 [1] CRAN (R 4.3.0)
rmarkdown               2.24      2023-08-14 [1] CRAN (R 4.3.0)
Rsamtools               2.16.0    2023-05-11 [1] Bioconductor
RSQLite                 2.3.1     2023-04-03 [1] CRAN (R 4.3.0)
rstudioapi              0.15.0    2023-07-07 [1] CRAN (R 4.3.0)
rsvd                    1.0.5     2021-04-16 [1] CRAN (R 4.3.0)
rtracklayer           * 1.60.1    2023-08-15 [1] Bioconductor
S4Arrays                1.0.6     2023-08-30 [1] Bioconductor
S4Vectors             * 0.38.1    2023-05-02 [1] Bioconductor
sass                    0.4.7     2023-07-15 [1] CRAN (R 4.3.0)
ScaledMatrix            1.8.1     2023-05-03 [1] Bioconductor
scales                  1.2.1     2022-08-20 [1] CRAN (R 4.3.0)
```

```
scater                 * 1.28.0    2023-04-25 [1] Bioconductor
scran                  * 1.28.2    2023-07-23 [1] Bioconductor
scuttle                * 1.10.2    2023-08-04 [1] Bioconductor
sessioninfo              1.2.2     2021-12-06 [1] CRAN (R 4.3.0)
shiny                    1.7.5     2023-08-12 [1] CRAN (R 4.3.0)
shinyWidgets             0.7.6     2023-01-08 [1] CRAN (R 4.3.0)
SingleCellExperiment   * 1.22.0    2023-04-25 [1] Bioconductor
spam                     2.9-1     2022-08-07 [1] CRAN (R 4.3.0)
sparseMatrixStats        1.12.2    2023-07-02 [1] Bioconductor
SpatialExperiment      * 1.11.2    2023-09-02 [1] Bioconductor
spatialLIBD            * 1.12.0    2023-04-27 [1] Bioconductor
statmod                  1.5.0     2023-01-06 [1] CRAN (R 4.3.0)
STexampleData          * 1.8.0     2023-04-27 [1] Bioconductor
SummarizedExperiment   * 1.30.2    2023-06-06 [1] Bioconductor
tibble                   3.2.1     2023-03-20 [1] CRAN (R 4.3.0)
tidyr                    1.3.0     2023-01-24 [1] CRAN (R 4.3.0)
tidyselect               1.2.0     2022-10-10 [1] CRAN (R 4.3.0)
utf8                     1.2.3     2023-01-31 [1] CRAN (R 4.3.0)
uwot                     0.1.16    2023-06-29 [1] CRAN (R 4.3.0)
vctrs                    0.6.3     2023-06-14 [1] CRAN (R 4.3.0)
vipor                    0.4.5     2017-03-22 [1] CRAN (R 4.3.0)
viridis                  0.6.4     2023-07-22 [1] CRAN (R 4.3.0)
viridisLite              0.4.2     2023-05-02 [1] CRAN (R 4.3.0)
withr                    2.5.0     2022-03-03 [1] CRAN (R 4.3.0)
xfun                     0.40      2023-08-09 [1] CRAN (R 4.3.0)
XML                      3.99-0.14 2023-03-19 [1] CRAN (R 4.3.0)
xtable                   1.8-4     2019-04-21 [1] CRAN (R 4.3.0)
XVector                  0.40.0    2023-04-25 [1] Bioconductor
yaml                     2.3.7     2023-01-23 [1] CRAN (R 4.3.0)
zlibbioc                 1.46.0    2023-04-25 [1] Bioconductor

[1] /Library/Frameworks/R.framework/Versions/4.3-x86_64/Resources/library
```

A Related resources

In this chapter, we highlight several related resources from the Bioconductor and other communities.

A.1 Data preprocessing procedures

Details on data preprocessing procedures for spatially-resolved transcriptomics data from the 10x Genomics Visium platform are provided in the following online book (using tools outside R and Bioconductor):

- Visium Data Preprocessing

A.2 Resources for other spatially-resolved platforms

Workflows and other resources for other spatially-resolved platforms:

Analysis workflow for IMC data

- Online book providing a workflow highlighting the use of common R/Bioconductor packages to analyze single-cell data obtained from segmented imaging mass cytometry (IMC) images. Examples focus on IMC data and can also be applied to images obtained by other highly-multiplexed imaging technologies, e.g. CODEX, MIBI, and mIF.
- Authors: Nils Eling, Vito Zanotelli, Michelle Daniel, Daniel Schulz, Jonas Windhager

VectraPolarisData

- Bioconductor data package providing two multiplex single-cell imaging datasets collected on Vectra Polaris and Vectra 3 instruments.
- Authors: Julia Wrobel, Tusharkanti Ghosh

A.3 Data structures

SpatialExperiment

- S4 class for storing spatially-resolved transcriptomics (SRT) data, which is used as the basis for the examples in this book. See Chapter Bioconductor data classes.
- Authors: Dario Righelli, Lukas M. Weber, Helena L. Crowell, Brenda Pardo, Leonardo Collado-Torres, Shila Ghazanfar, Aaron T. L. Lun, Stephanie C. Hicks, Davide Risso

SpatialFeatureExperiment

- S4 class extending `SpatialExperiment` to incorporate geometries and geometry operations with the `sf` R package.
- Authors: Lambda Moses, Lior Pachter

A.4 Statistical concepts

Modern Statistics for Modern Biology

- Online textbook on concepts in modern statistics for high-throughput and high-dimensional biology, including chapter on image data and spatial statistics.
- Authors: Susan Holmes, Wolfgang Huber

B Contributors and acknowledgments

B.1 Best Practices for Spatial Transcriptomics Analysis with Bioconductor

- Lukas M. Weber, *Johns Hopkins Bloomberg School of Public Health, Baltimore, MD, USA*
- Abby Spangler, *Lieber Institute for Brain Development, Baltimore, MD, USA*
- Madhavi Tippani, *Lieber Institute for Brain Development, Baltimore, MD, USA*
- Leonardo Collado-Torres, *Lieber Institute for Brain Development, Baltimore, MD, USA*
- Stephanie C. Hicks, *Johns Hopkins Bloomberg School of Public Health, Baltimore, MD, USA*

As well as GitHub contributors who have helped us improve this book over time. Thank you for your contributions!

B.2 SpatialExperiment

- Dario Righelli, *Department of Statistical Sciences, University of Padova, Padova, Italy*
- Helena L. Crowell, *Department of Molecular Life Sciences, University of Zurich, Zurich, Switzerland*
- Aaron Lun, *Genentech, South San Francisco, CA, USA*
- Stephanie C. Hicks, *Johns Hopkins Bloomberg School of Public Health, Baltimore, MD, USA*
- Davide Risso, *Department of Statistical Sciences, University of Padova, Padova, Italy*

B.3 GitHub Actions workflow

The GitHub Actions workflow used to build the online version of the book is based on `biocthis` and uses the Bioconductor Docker images.

References

Lun, Aaron T L, Davis J McCarthy, and John C Marioni. 2016. "A Step-by-Step Workflow for Low-Level Analysis of Single-Cell RNA-seq Data with Bioconductor." *F1000Research* 5: 2122. https://doi.org/10.12688/f1000research.9501.2.

Maynard, Kristen R., Leonardo Collado-Torres, Lukas M. Weber, Cedric Uytingco, Brianna K. Barry, Stephen R. Williams, Joseph L. Catallini II, et al. 2021. "Transcriptome-Scale Spatial Gene Expression in the Human Dorsolateral Prefrontal Cortex." *Nature Neuroscience*. https://doi.org/10.1038/s41593-020-00787-0.

McCarthy, Davis J, Kieran R Campbell, Aaron T L Lun, and Quin F Wills. 2017. "Scater: Pre-Processing, Quality Control, Normalization and Visualization of Single-Cell RNA-seq Data in R." *Bioinformatics* 33 (8): 1179–86. https://doi.org/10.1093/bioinformatics/btw777.

McInnes, Leland, John Healy, and James Melville. 2018. "UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction." *arXiv* 1802.03426 (v2). https://arxiv.org/abs/1802.03426.

Pardo, Brenda, Abby Spangler, Lukas M. Weber, Stephanie C. Hicks, Andrew E. Jaffe, Keri Martinowich, Kristen R. Maynard, and Leonardo Collado-Torres. 2021. "spatialLIBD: An r/Bioconductor Package to Visualize Spatially-Resolved Transcriptomics Data." *bioRxiv*. https://doi.org/10.1101/2021.04.29.440149.