Spatial transcriptomics data analysis: theory and practice

Eleftherios Zormpas, Dr Simon J. Cockell

2023-07-20

Contents

Welcome

This book will guide you through the practical steps of the in-person tutorial IP2 for the ISMB/ECCB 2023 conference in Lyon named: *"Spatial transcriptomics data analysis: theory and practice"*.

Abstract

Recent technological advances have led to the application of RNA Sequencing *in situ*. This allows for whole-transcriptome characterisation, at approaching single-cell resolution, while retaining the spatial information inherent in the intact tissue. Since tissues are congregations of intercommunicating cells, identifying local and global patterns of spatial association is imperative to elucidate the processes which underlie tissue function. Performing spatial data analysis requires particular considerations of the distinct properties of data with a spatial dimension, which gives rise to an association with a different set of *statistical* and *inferential* considerations.

In this comprehensive tutorial, we will introduce users to spatial transcriptomics (STx) technologies and current pipelines of STx data analysis inside the **Bioconductor** framework. Furthermore, we will introduce attendees to the underlying features of spatial data analysis and how they can effectively utilise space to extract in-depth information from STx datasets.

Learning objectives

Participants in this tutorial will gain understanding of the core technologies for undertaking a spatial transcriptomics experiment, and the common tools used for the analysis of this data. In particular, participants will appreciate the strengths of geospatial data analysis methods in relation to this type of data. Specific learning objectives will include:

1. Describe and discuss core technologies for spatial transcriptomics
2. Make use of key computational technologies to process and analyse STx data

3. Apply an analysis strategy to obtain derived results and data visualisations
4. Appreciate the principles underlying spatial data analysis
5. Understand some of the methods available for spatial data analysis
6. Apply said methods to an example STx data set

Chapter 1

Practical session 1

In this practical session you will familiarise yourself with some example spatial transcriptomics (STx) data and the common features of such data.

1.1 About this documentation

This handbook is designed to walk you through the practical elements of today's tutorial. All of the code you need to accomplish the basic tasks thoughout the day is presented in full, there are some 'stretch goals' in some of the tutorials where only a template is provided. This is not a typing tutorial, so feel free to copy and paste where necessary. The tutorials are written in `rmarkdown` and presented on bookdown.org.

1.2 Posit Cloud

You should have received an invite to the Posit Cloud Space for today's tutorial. Accepting this invite will give you access to the 4 RStudio projects for the 4 sessions we will run today. Each project has the required packages pre-installed, and the data files uploaded. These projects are set up as "Assignments" so that you get your own copy of the workspace.

1.3 Import 10X Visium data

In this tutorial we will be using data from the STexampleData package that contains a small collection of STx datasets from different technologies, including SlideSeq V2, seqFISH and 10x Genomics Visium. These datasets are provided in the `SpatialExperiment` format - described below.

The specific dataset used for this tutorial is a single sample from the dorsolateral prefrontal cortex (DLPFC) - a 10x Genomics Visium dataset that was published by Maynard et al. [2021].

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

```
library(SpatialExperiment)
library(STexampleData)
library(ggplot2)
library(ggspavis)

# Load the object
spe <- Visium_humanDLPFC()
```

1.4 Explore data types

There is a long history of encapsulating expression data in S3 and S4 objects in R, going back to the `ExpressionSet` class in Biobase which was designed to store a matrix of microarray data alongside associated experimental metadata. This concept of storing all the relevant data and metadata in a single object has persisted through the development of RNA-Seq analysis (e.g. `SummarizedExperiment`) and into the age of single-cell transcriptomics (e.g. `SingleCellExperiment` - see below).

1.4.1 SpatialExperiment class

For the first part of this tutorial (practical sessions 1 and 2), we will be using the SpatialExperiment S4 class from Bioconductor as the main data structure for storing and manipulating datasets.

`SpatialExperiment` is a specialized object class that supports the storing of spatially-resolved transcriptomics datasets within the Bioconductor framework. It builds on the SingleCellExperiment class [Amezquita et al., 2020] for single-cell RNA sequencing data, which itself extends the `RangedSummarizedExperiment` class. More specifically, `SpatialExperiment` has extra customisations to store spatial information (i.e., spatial coordinates and images).

An overview of the `SpatialExperiment` object structure is is presented in 1.1. In brief, the `SpatialExperiment` object consists of the below five parts:

1. `assays`: gene expression counts

2. `rowData`: information about features, usually genes

3. `colData`: information on spots (non-spatial and spatial metadata)

4. `spatialCoords`: spatial coordinates

5. `imgData`: image data

NOTE: For spot-based STx data (i.e., 10x Genomics Visium), a single `assay` named `counts` is used.

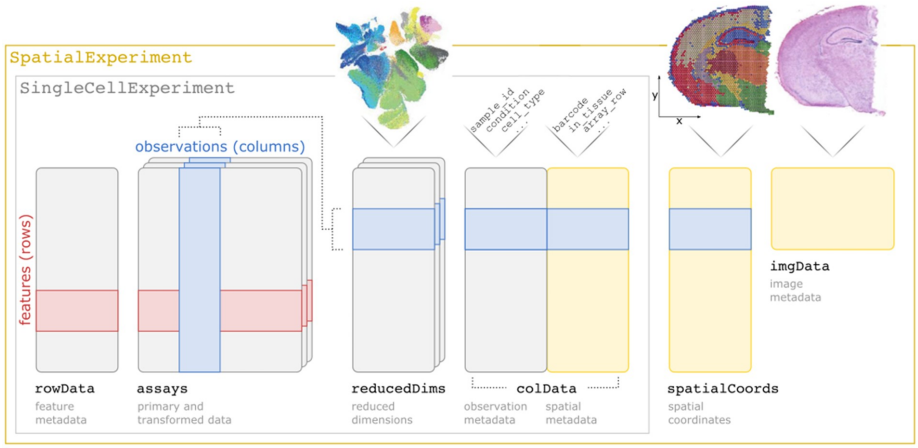


Figure 1.1: Overview of the 'SpatialExperiment' object class structure.

For more details, see the related publication from Righelli et al., 2021 describing the `SpatialExperiment` [Righelli et al., 2022].

1.4.2 Inspect the object

```
## Check the object's structure
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
```

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

```
## [1] 33538  4992
```

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

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

1.4.3 Counts table and gene metadata

```
## Have a look at the counts table
assay(spe)[1:6,1:4]
```

```
## 6 x 4 sparse Matrix of class "dgTMatrix"
##                 AAACAACGAATAGTTC-1 AAACAAGTATCTCCCA-1 AAACAATCTACTAGCA-1
## ENSG00000243485                  .                  .                  .
## ENSG00000237613                  .                  .                  .
## ENSG00000186092                  .                  .                  .
## ENSG00000238009                  .                  .                  .
## ENSG00000239945                  .                  .                  .
## ENSG00000239906                  .                  .                  .
##                 AAACACCAATAACTGC-1
## ENSG00000243485                  .
## ENSG00000237613                  .
## ENSG00000186092                  .
## ENSG00000238009                  .
## ENSG00000239945                  .
## ENSG00000239906                  .
```

As we can see here the counts table is an object of class `dgTMatrix` which is a sparse matrix. This is because much like scRNA-seq data, STx data include many zeros. As a result, to make the counts table as light as possible we resort to using sparse matrices. This next code chunk examines a part of the matrix that includes genes with some level of expression:

```
assay(spe)[20:40, 2000:2010]
```

```
## 21 x 11 sparse Matrix of class "dgTMatrix"
##
## ENSG00000223764 . . . . . . . . . . .
## ENSG00000187634 . . . . . . . . . . .
## ENSG00000188976 . . 2 . . . . . 1 1
## ENSG00000187961 . . . . . . . . . . .
## ENSG00000187583 . . . . . . . . . . .
```

```
## ENSG00000187642 . . . . . .
## ENSG00000272512 . . . . . .
## ENSG00000188290 1 . . . . . 2 .
## ENSG00000187608 . 1 . . . . 2 . . 1 .
## ENSG00000224969 . . . . . .
## ENSG00000188157 . 1 . . 2 . . . . 1 .
## ENSG00000273443 . . . . . .
## ENSG00000237330 . . . . . .
## ENSG00000131591 . . . . . 1 .
## ENSG00000223823 . . . . . .
## ENSG00000272141 . . . . . .
## ENSG00000205231 . . . . . .
## ENSG00000162571 . . . . . .
## ENSG00000186891 . . . 1 . . . . . .
## ENSG00000186827 . . . . . .
## ENSG00000078808 . 1 2 . 1 . . . . 1 .
```

```
assay(spe)[33488:33508, 2000:2010]
```

```
## 21 x 11 sparse Matrix of class "dgTMatrix"
##
## ENSG00000160294  .  .   .  .   .   .  .  .  .   .  .
## ENSG00000228137  .  .   .  .   .   .  .  .  .   .  .
## ENSG00000239415  .  .   .  .   .   .  .  .  .   .  .
## ENSG00000182362  .  .   .  .   .   .  .  .  1   .  .
## ENSG00000160298  .  .   .  .   .   .  .  .  .   .  .
## ENSG00000160299  .  .   1  .   1   .  .  .  .   .  .
## ENSG00000160305  .  .   .  .   .   2  .  .  .   .  .
## ENSG00000160307  1  3   1  1   4   5  1  1  .   2  1
## ENSG00000160310  .  .   .  .   1   .  .  .  .   2  .
## ENSG00000198888 17 44  71 16 154  97 12 14 32 167  6
## ENSG00000198763 16 59  64 11 116  63 11 12 18 123  6
## ENSG00000198804 37 85 155 25 252 176 24 27 38 335 12
## ENSG00000198712 23 79 120 23 214 170 22 25 48 242 10
## ENSG00000228253  2  .   3  .   1   .  .  1  1   6  .
## ENSG00000198899 20 39  93  9 136 108 20 18 25 165  7
## ENSG00000198938 27 59 133 20 216 120 22 26 43 232  9
## ENSG00000198840  5 27  33  5  71  39  8 11 12  78  .
## ENSG00000212907  2  .   4  2   7   5  .  1  1   9  .
## ENSG00000198886 15 65  95  9 183  98 18 19 33 178  7
## ENSG00000198786  2 10  10  3  20  14  1  2  2  25  4
## ENSG00000198695  1  1   3  .   2   2  .  .  .   1  .
```

The levels of expression of different genes in the same spots differ significantly with many low values being present. We have to remember here that this data is not as yet normalized, and is therefore affected by systematic factors such as library size. Nonetheless, what is demonstrated here is typical for STx data (as

it is for scRNA-seq data) - many genes will show low expression in individual spots.

To continue our exploration of the information stored in the `SpatialExperiment` object:

```
## Have a look at the genes 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
```

1.4.4 Coordinates table and spot metadata

The data that distinguished a `SpatialExperiment` object is the coordinate data which describes the spatial location of each spot.

```
## Check the 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
```

```
## 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
```

1.4.5 Image metadata

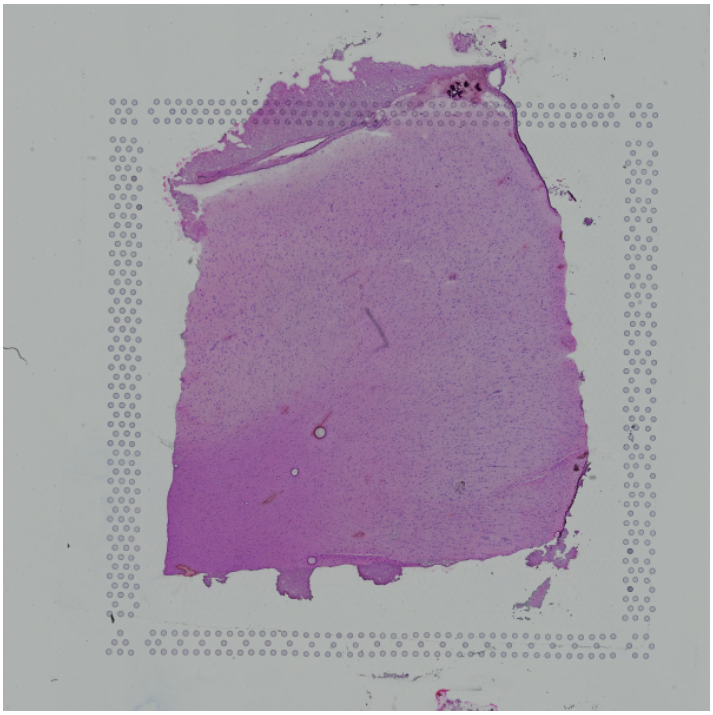
Finally, the `SpatialExperiment` object also contains the image data from the STx experiment, giving the coordinates we looked at in the previous section some context in terms of the tissue of origin.

```
## Have a look at the 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
```

As well as this (fairly basic) metadata, the `spe` object also contains the image itself, which the `SpatialExperiment` class allows us to access, like so:

```
## retrieve the image
spi <- getImg(spe)
## "plot" the image
plot(imgRaster(spi))
```

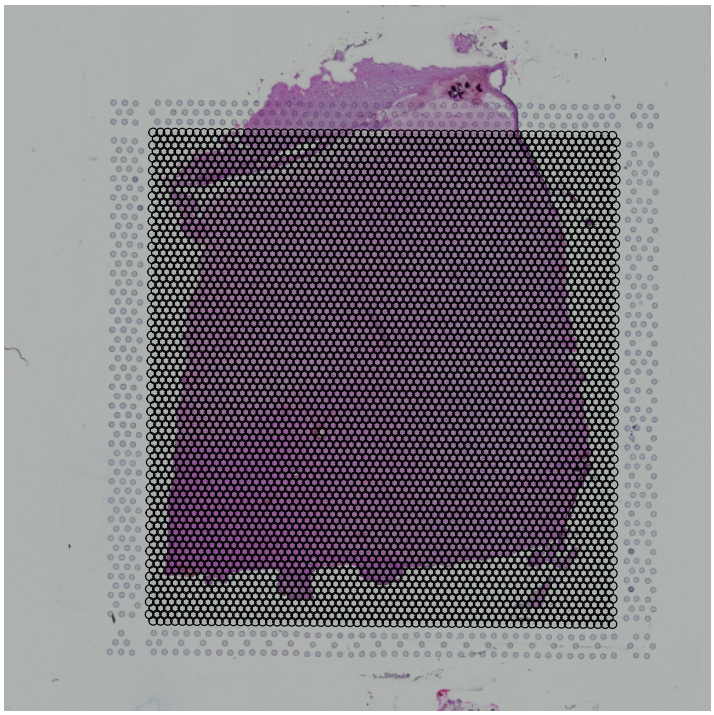


We can also use the scaling factors in the `imgData` to plot the locations of the Visium spots over the image. The position of a point in an image does not map directly to the spot location in cartesian coordinates, as it is the top-left of an image that is (0,0), not the bottom-left. In order to manage this, we need to transform the y-axis coordinates.

```
## "Plot" the image
plot(imgRaster(spi))
## Extract the spot locations
spot_coords <- spatialCoords(spe) %>% as.data.frame
## Scale by low-res factor
lowres_scale <- imgData(spe)[imgData(spe)$image_id == 'lowres', 'scaleFactor']
spot_coords$x_axis <- spot_coords$pxl_col_in_fullres * lowres_scale
spot_coords$y_axis <- spot_coords$pxl_row_in_fullres * lowres_scale
## lowres image is 600x600 pixels
dim(imgRaster(spi))
```

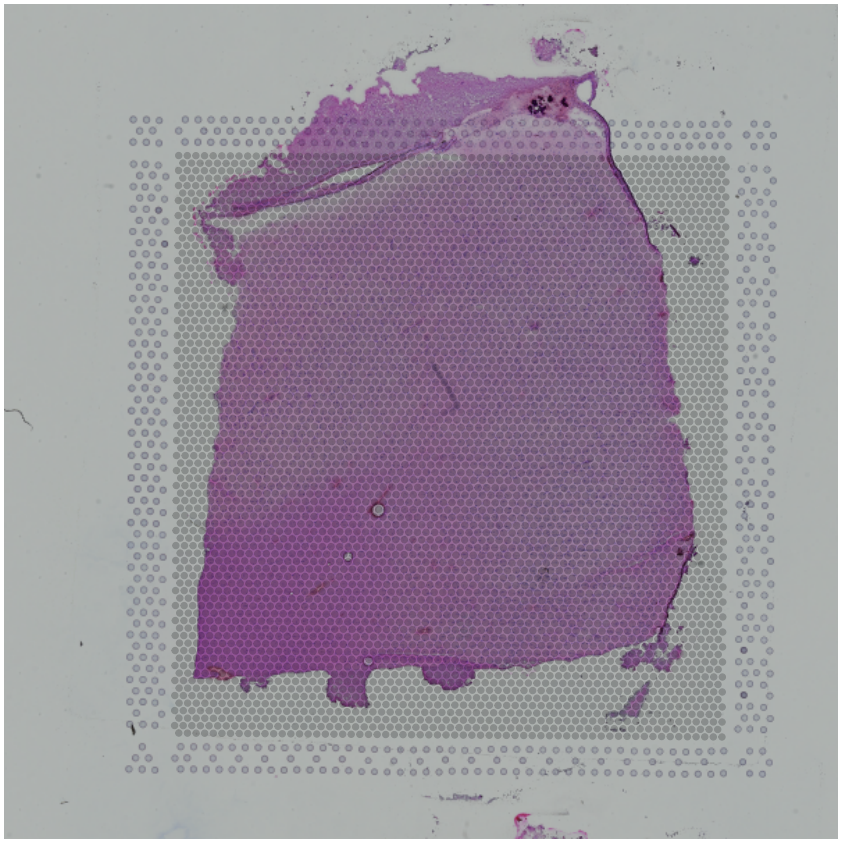
```
## [1] 600 600
```

```
## flip the Y axis
spot_coords$y_axis <- abs(spot_coords$y_axis - (ncol(imgRaster(spi)) + 1))
points(x=spot_coords$x_axis, y=spot_coords$y_axis)
```



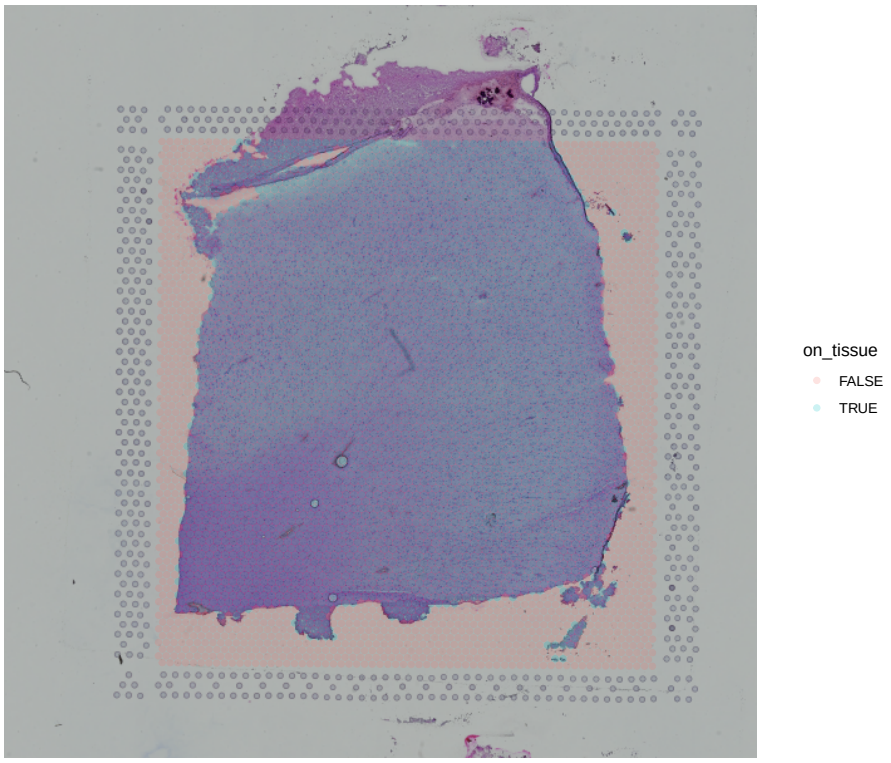
An equivalent plot, using ggplot2 as the plotting library:

```
ggplot(mapping = aes(1:600, 1:600)) +
  annotation_raster(imgRaster(spi), xmin = 1, xmax = 600, ymin = 1, ymax = 600) +
  geom_point(data=spot_coords, aes(x=x_axis, y=y_axis), alpha=0.2) + xlim(1, 600) + ylim(1, 600)
  coord_fixed() +
  theme_void()
```



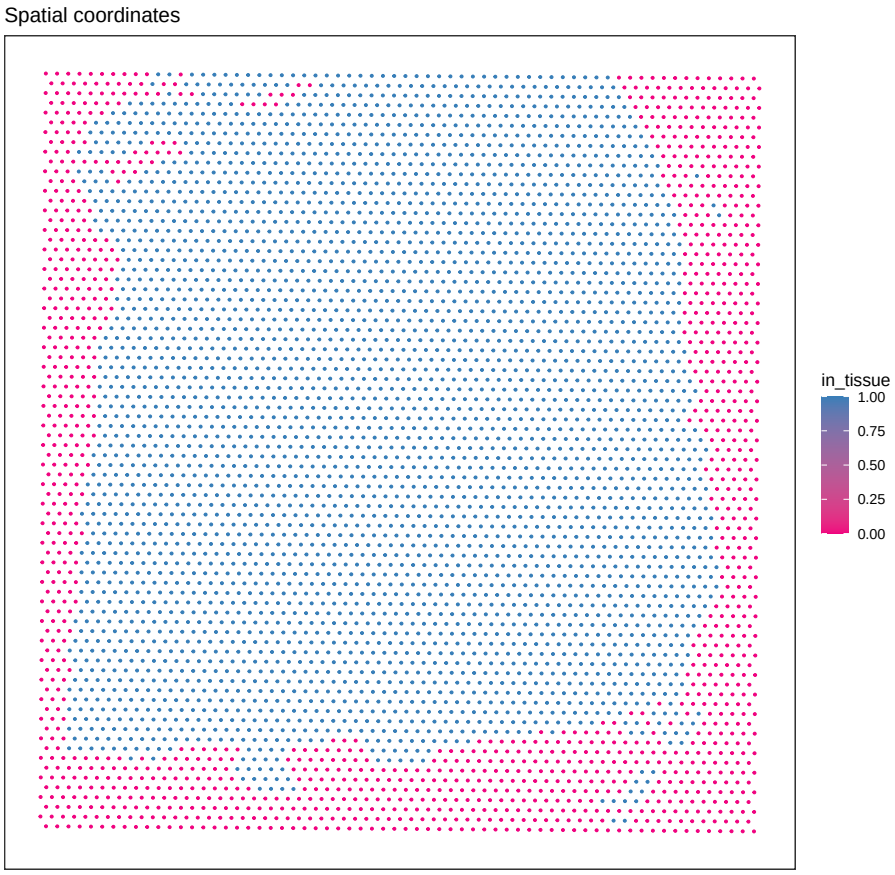
We can also extract additional metadata to make these plots more informative - for instance, the annotation from `colData` that flags whether a spot is "on tissue" or not can be used to colour the spots like so:

```
## Add the annotation to the coordinate data frame
spot_coords$on_tissue <- as.logical(colData(spe)$in_tissue)

ggplot(mapping = aes(1:600, 1:600)) +
  annotation_raster(imgRaster(spi), xmin = 1, xmax = 600, ymin = 1, ymax = 600) +
  geom_point(data=spot_coords, aes(x=x_axis, y=y_axis, colour=on_tissue), alpha=0.2) +
  coord_fixed() +
  theme_void()
```



Having to manually extract the relevant information from the `SpatialExperiment` object to generate plots like this does not generally make sense, and defies the point of using a data class that can encapsulate this information. We can instead use a package like `ggspavis`, which is explicitly built for generating visualisations of STx data directly from the `SpatialExperiment` object. We will make extensive use of this package during the next tutorial as we work through quality control processes for STx data. The pre-built nature of these plots is convenient, though it prevents users from achieving tasks like adding the tissue image to the plot. For many users the convenience will outweigh any issues this presents, though it is worth being aware of how to build visualisations from the ground up.

```
plotSpots(spe, in_tissue = NULL, annotate='in_tissue', size=0.5)
```



1.5 Conclusion

This first practical session has been a pretty straight-forward examination of an example Visium dataset. We've demonstrated where in this object the data and metadata are stored, how to extract it and make simple use of it.

Chapter 2

Practical session 2

Having previously introduced some of the Bioconductor ecosystem for storing and manipulating STx data, in this second session we will focus on some of the most common STx analysis tasks - particularly quality control assessment and associated spot- and gene- level filtering. We will also consider some global methods of STx analysis, including dimensionality reduction and clustering. All of the methods demonstrated here continue to focus on interoperable packages available via Bioconductor.

```
## Load packages {-}
library(SpatialExperiment)
library(STexampleData)
library(ggspavis)
library(ggplot2)
library(scater)
library(scran)
library(igraph)
library(pheatmap)
library(ggExtra)
```

- `ggspavis` is a Bioconductor package that includes visualization functions for spatially resolved transcriptomics datasets stored in `SpatialExperiment` format from spot-based (e.g., 10x Genomics Visium) platforms (Weber and Crowell [2022]).
- `scater` is also a Bioconductor package that is a selection of tools for doing various analyses of scRNA-seq gene expression data, with a focus on quality control and visualization which has extended applications to STx data too. It is based on the `SingleCellExperiment` and `SpatialExperiment` classes and thus is interoperable with many other Bioconductor packages such as `scran`, `scuttle` and `iSEE`.

```
## Reload the example dataset
spe <- Visium_humanDLPFC()
```

```
## see ?STexampleData and browseVignettes('STexampleData') for documentation
```

```
## loading from cache
```

2.1 Spot-level Quality Control

Considered quality control (QC) procedures are essential for analysing any high-throughput data in molecular biology. The removal of noise and low quality data from complex datasets can improve the reliability of downsrtream analyses. STx is no different in this regard, and QC can be undertaken in 2 main places - spot-level and gene-level. Here, we focus on spot-level QC.

Spot-level quality control (sQC) procedures are employed to eliminate low-quality spots before conducting further analyses. Low-quality spots may result from issues during library preparation or other experimental procedures, such as a high percentage of dead cells due to cell damage during library preparation, or low mRNA capture efficiency caused by ineffective reverse transcription or PCR amplification. Keeping these spots usually leads to creating problems during downstream analyses.

We can identify low-quality spots using several characteristics that are also used in cell-level QC for scRNA-sq data, including:

1. **library size** (total of UMI counts per spot will vary due to sequencing *-like different samples in a bulk RNA-seq-*, or due to number of cells in the spot)
2. **number of expressed genes** (i.e. number of genes with non-zero UMI counts per spot)
3. **proportion of reads mapping to mitochondrial genes** (a high proportion indicates putative cell damage)

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.

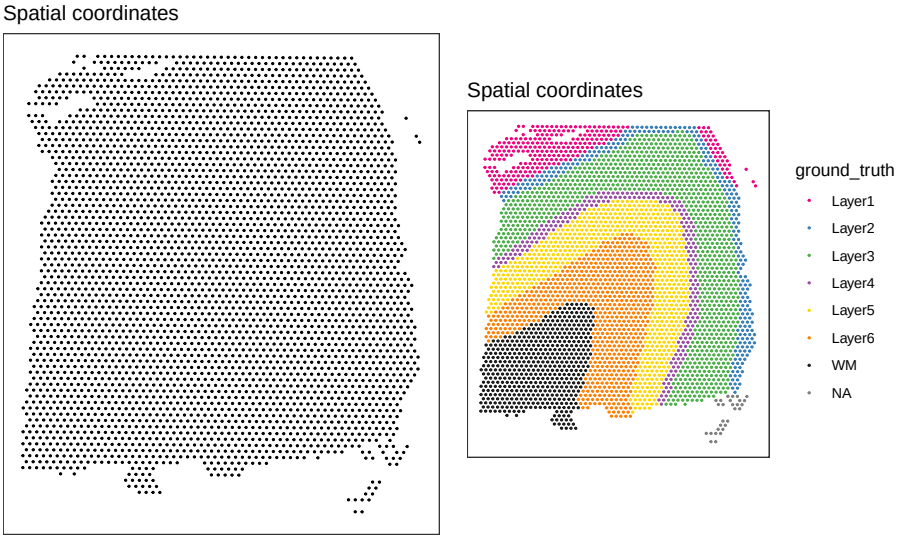
The idea of using scRNA-seq QC metrics in STx data comes from the fact that if we remove space and effectively treat each spot as a single cell, the two datasets share common features. We need to bear in mind, however, that the expected distributions for high-quality *spots* are different (compared to high-quality *cells* in scRNA-seq), since spots may contain zero, one, or multiple cells.

A few publications for further reading that can help you understand the quality controls: McCarthy et al. [2017] and Amezquita et al. [2020].

2.1.1 Plot tissue map

The dorso-lateral prefrontal cortex (DLPFC) is a functional brain region in primates involved in executive function. It consists of six layers of neurons that differ in their cell types, density and connections. The DLPFC dataset we looked at in session one, and will be here using comes with manual annotation of these layers (and the adjacent white matter - WM) by the authors Maynard et al. [2021]. We can plot the tissue map with and without the annotations to get a complete view.

```
## Plot spatial coordinates without annotations
plotSpots(spe)

## Plot spatial coordinates with annotations
plotSpots(spe,
          annotate = "ground_truth")
```



2.1.2 Calculating QC metrics

We will calculate the three main QC metrics described above using methods from the `scater` [McCarthy et al., 2017] package, and investigate their influence on the DLPFC dataset with some plots from `ggspavis`, along with some additional plots of our own.

At present, the dataset contains both on- and off-tissue spots - we plotted these

in the previous practical. For any future analysis though we are only interested in the on-tissue spots. Therefore, before we run any calculations we want to remove the off-tissue spots.

NOTE: the on- or off-tissue information for each spot can be found in the `colData` of the `spe` object and in the `in_tissue` column where *0 = off-tissue* and *1 = on-tissue*.

```
## Dataset dimensions before the filtering
dim(spe)
```

```
## [1] 33538  4992
```

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

```
## [1] 33538  3639
```

The next thing we need to do before we make decisions on how to quality *"trim"* the dataset is to calculate the percentage per spot of mitochodrial gene expression and store this information inside the `colData`. First of all, find the mitochrondrial genes - their gene names start with "MT-" or "mt-".

```
## Classify genes as "mitochondrial" (is_mito == TRUE)
## or not (is_mito == FALSE)
is_mito <- grepl("(^MT-)|(^mt-)", rowData(spe)$gene_name)
rowData(spe)$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"
```

Then find what proportion of reads in a spot's library are attributable to the expression of these genes. This uses a function, `addPerCellQC()` from `scater` (which in this instance is actually a wrapper around `scuttle`).

```
## 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
```

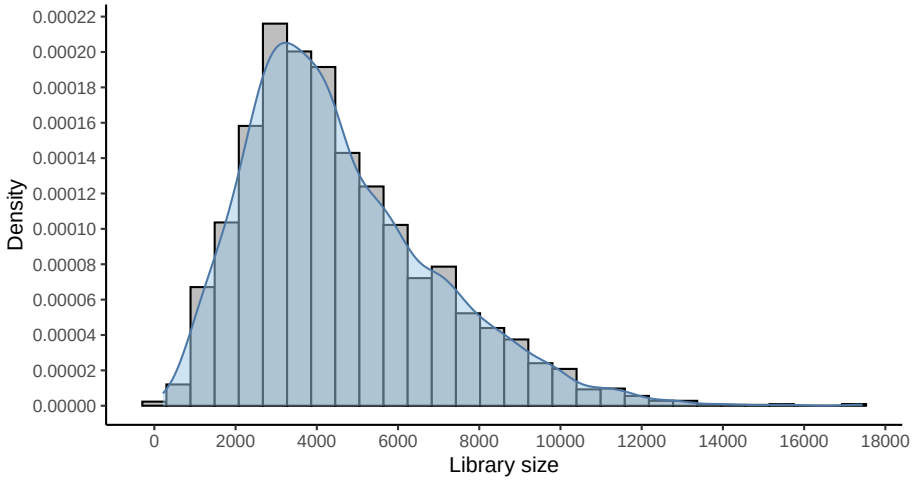
After calculating a required metric, we need to apply a cut-off threshold for the metric to decide whether or not to keep each spot. It is important to consider an individual dataset on its own merits, as it might need slightly different cut-off values to be applied. As a result we cannot rely on identifying a single value to use every time and we need to rely on plotting these metrics and making a decision on a dataset-by-dataset basis.

2.1.3 Library size threshold plot

We can plot a histogram of the library sizes across spots. The library size is the number of UMI counts in each spot. We can find this information in the `sum` column in the `colData`.

```
## Density and histogram of library sizes
ggplot(data = as.data.frame(colData(spe)),
       aes(x = sum)) +
  geom_histogram(aes(y = after_stat(density)),
                 colour = "black",
                 fill = "grey") +
  geom_density(alpha = 0.5,
               adjust = 1.0,
               fill = "#A0CBE8",
               colour = "#4E79A7") +
```

```
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  xlab("Library size") +
  ylab("Density") +
  theme_classic()
```

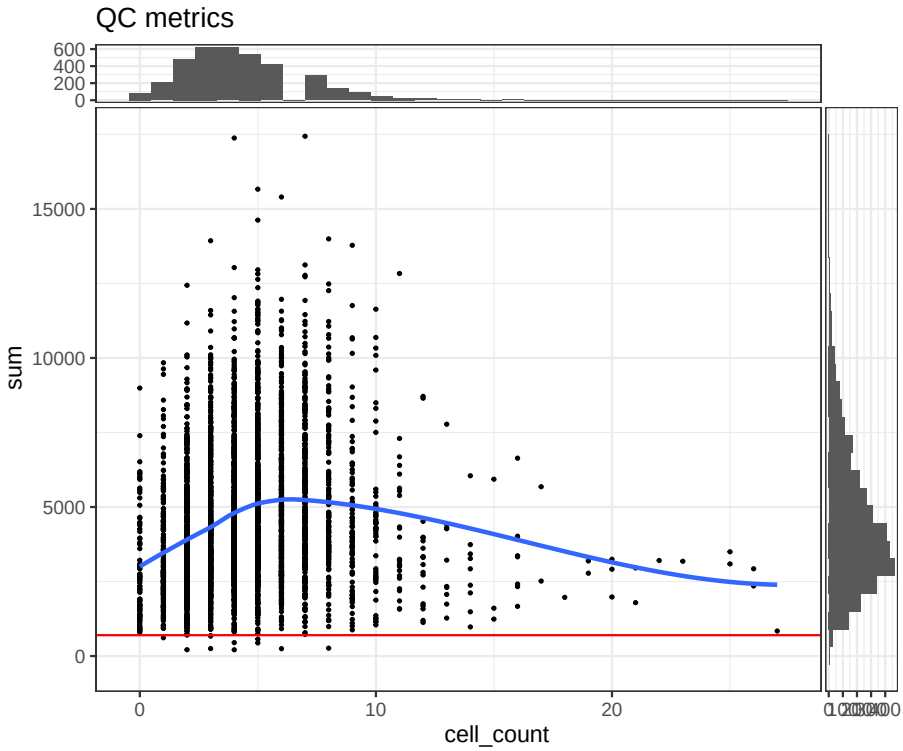


As we can see there are no obvious issues with the library sizes. An example of an issue could be a high frequency of small libraries which would indicate poor experimental output. Generally we do not want to keep spots with too small libraries.

If the dataset we are analysing contains the number of cells that are present in each spot (this one does), then it makes sense to also plot the library sizes against the number of cells per spot. In that way we are making sure that we don't remove any spots that may have biological meaning. In many cases though the datasets do not have such information unless we can generate it using a nuclei segmentation tool to extract this information from the H&E images.

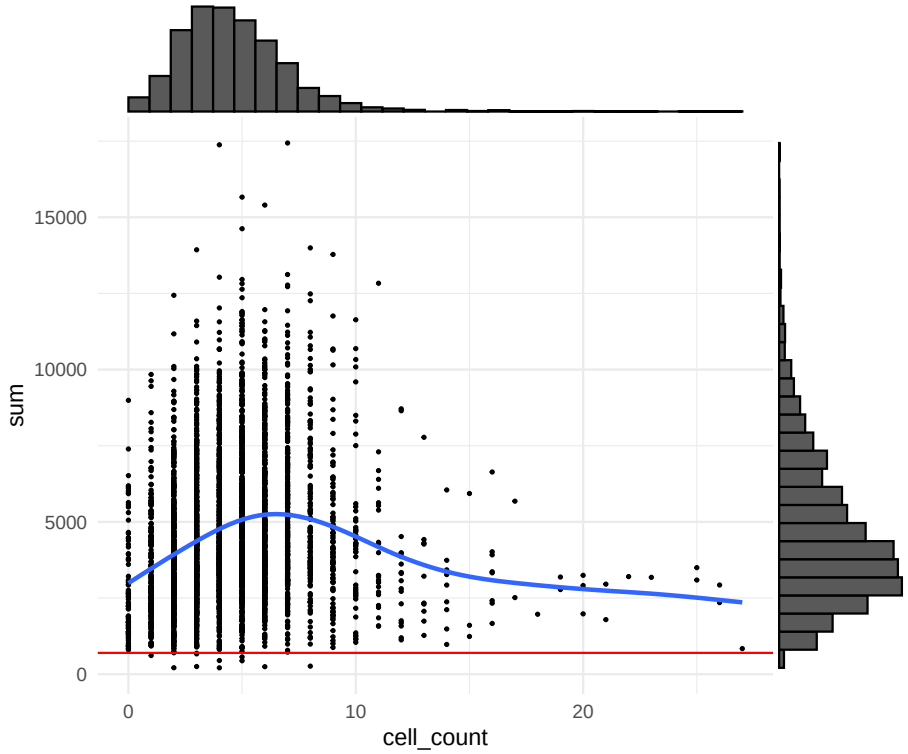
The horizontal red line (argument `threshold` in the `plotQC` function) shows a first guess at a possible filtering threshold for library size based on the above histogram.

```
## Scatter plot, library size against number of cells per spot
plotQC(spe, type = "scatter",
       metric_x = "cell_count", metric_y = "sum",
       threshold_y = 700)
```



NOTE: The `ggspavis` plots for QC are convenient, but not very configurable. As can be seen from the "missing" bin in the top histogram here, the default configuration provided is not always the best. A ggplot2 alternative (using `ggExtra` to provide the marginal histograms) is also provided here.

```
p = ggplot(as.data.frame(colData(spe)), aes(x=cell_count, y=sum)) +
  geom_point(size=0.5) +
  geom_smooth(se=FALSE) +
  geom_hline(yintercept = 700, colour='red') +
  theme_minimal()
ggMarginal(p, type='histogram', margins = 'both')
```



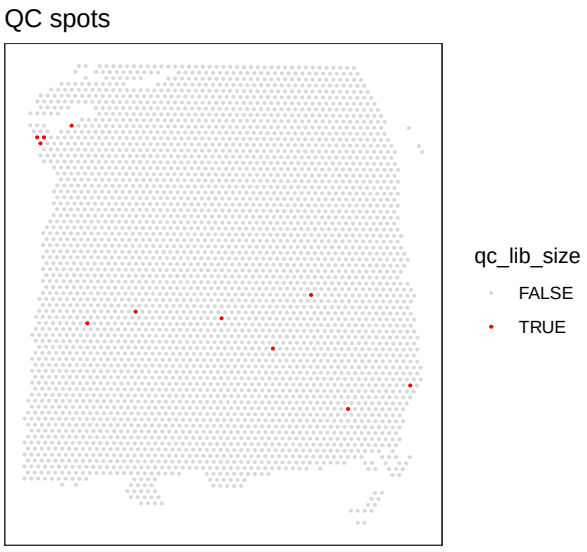
We need to keep in mind here that the threshold is, to an extent, arbitrary. It is therefore important to look at the number of spots that are left out of the dataset by this choice of cut-off value, and also have a look at their putative spatial patterns. If we filtered out spots with biological relevance, then we should observe some patterns on the tissue map that correlate with some of the known biological structures of the tissue. If we do observe such a phenomenon, we have probably set our threshold too high (i.e. not permissive enough).

```
## Select library size threshold
qc_lib_size <- colData(spe)$sum < 700
## Check how many spots are filtered out
table(qc_lib_size)
```

```
## qc_lib_size
## FALSE  TRUE
##  3628    11
```

```
## Add threshold in colData
colData(spe)$qc_lib_size <- qc_lib_size

## Check putative spatial patterns of removed spots
plotQC(spe, type = "spots",
```

```
        discard = "qc_lib_size")
```



As an optional exercise, try to illustrate what happens if we set the threshold too high (i.e., 2000 UMI counts).

NOTE: For reference, remember the ground truth layers in this dataset that we plotted at the beginning of this session.

```
## Select library size threshold
code...
## Check how many spots are filtered out
code...
## Add threshold in colData
code...

## Check putative spatial patterns of removed spots
plotQC(...)
```

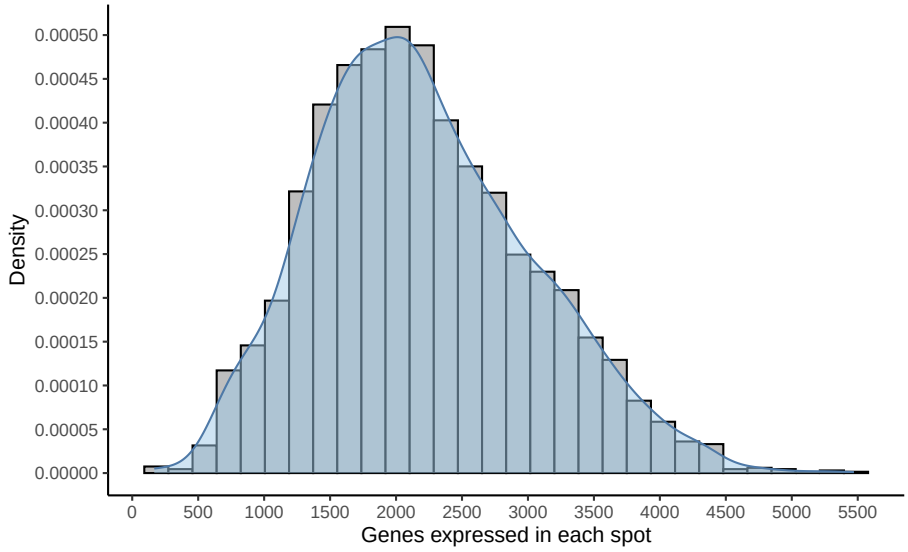
2.1.4 Number of expressed genes

As we did with the library sizes, we can plot a histogram of the number of expressed genes across spots. A gene is "expressed" in a spot if it has at least one count in it. We can find this information in the `detected` column in the `colData`.

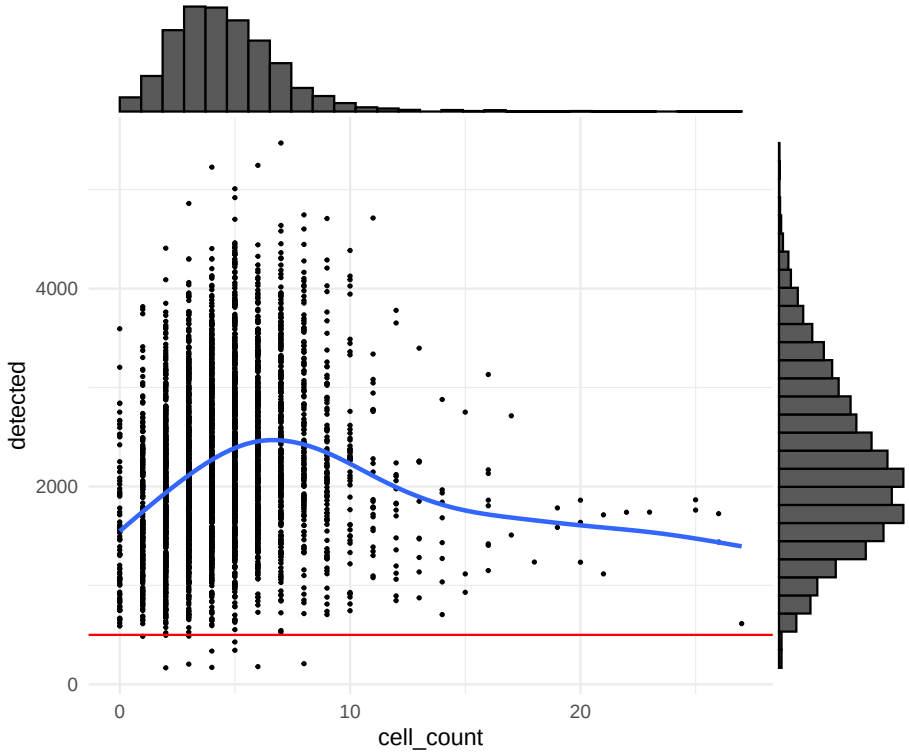
We will follow the same logic for the plots as we did for the library size earlier.

```
## Density and histogram of expressed genes
ggplot(data = as.data.frame(colData(spe)),
```

```
      aes(x = detected)) +
  geom_histogram(aes(y = after_stat(density)),
                 colour = "black",
                 fill = "grey") +
  geom_density(alpha = 0.5,
               adjust = 1.0,
               fill = "#A0CBE8",
               colour = "#4E79A7") +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  xlab("Genes expressed in each spot") +
  ylab("Density") +
  theme_classic()
```




```
# plot number of expressed genes vs. number of cells per spot
p = ggplot(as.data.frame(colData(spe)), aes(x=cell_count, y=detected)) +
  geom_point(size=0.5) +
  geom_smooth(se=FALSE) +
  geom_hline(yintercept = 500, colour='red') +
  theme_minimal()
ggMarginal(p, type='histogram', margins = 'both')
```

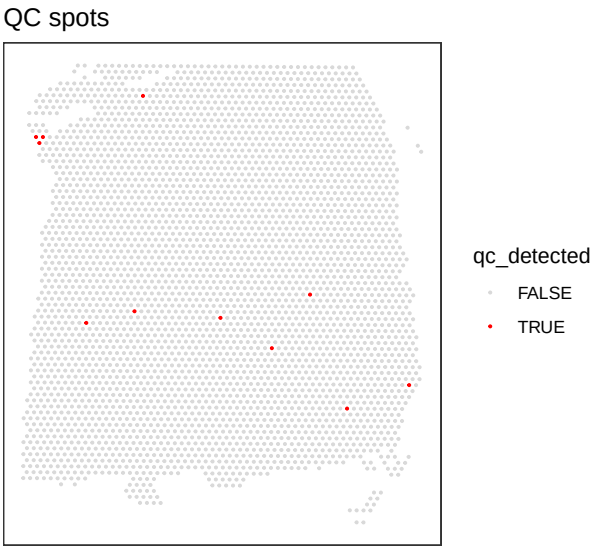


Finally, again as before, we apply the chosen threshold to flag spots with (in this case) fewer than 500 expressed genes.

```
## Select expressed genes threshold
qc_detected <- colData(spe)$detected < 500
## Check how many spots are filtered out
table(qc_detected)
```

```
## qc_detected
## FALSE  TRUE
##  3628    11
```

```
## Add threshold in colData
colData(spe)$qc_detected <- qc_detected

## Check for putative spatial pattern of removed spots
plotQC(spe, type = "spots",
       discard = "qc_detected")
```



Again, an optional exercise is provided to see the effects of an over-enthusiastic filter - to illustrate what happens if we set the threshold too high (i.e., 1000 expressed genes).

NOTE: For reference, remember the ground truth layers in this dataset that we plotted at the beginning of this session.

```
## Select library size threshold
code...
## Check how many spots are filtered out
code...
## Add threshold in colData
code...

## Check putative spatial patterns of removed spots
plotQC(...)
```

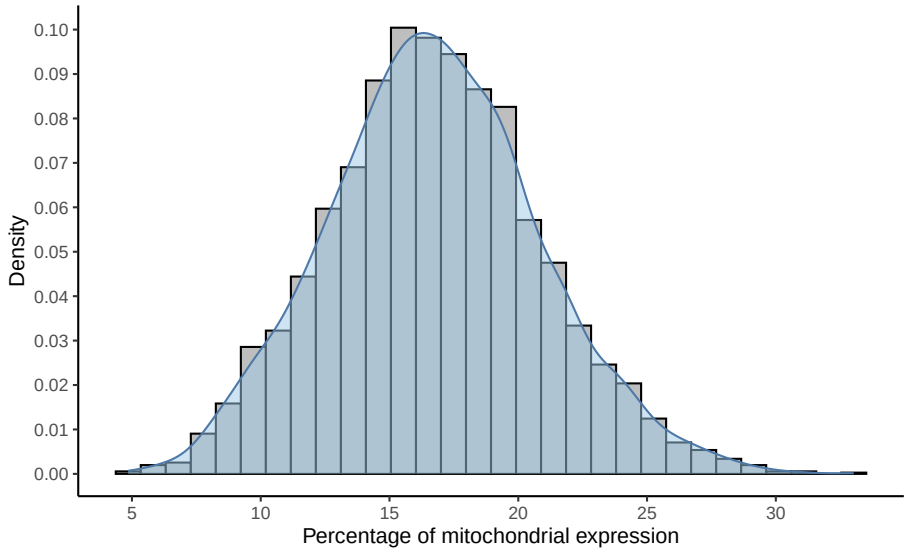
2.1.5 Percentage of mitochondrial expression

As we briefly touched on at the beginning, a high proportion of mitochondrial reads indicates low cell quality, probably due to cell damage.

We calculated this data earlier on in this session, and can now investigate the percentage of mitochondrial expression across spots by looking at the column `subsets_mito_percent` in the `colData`.

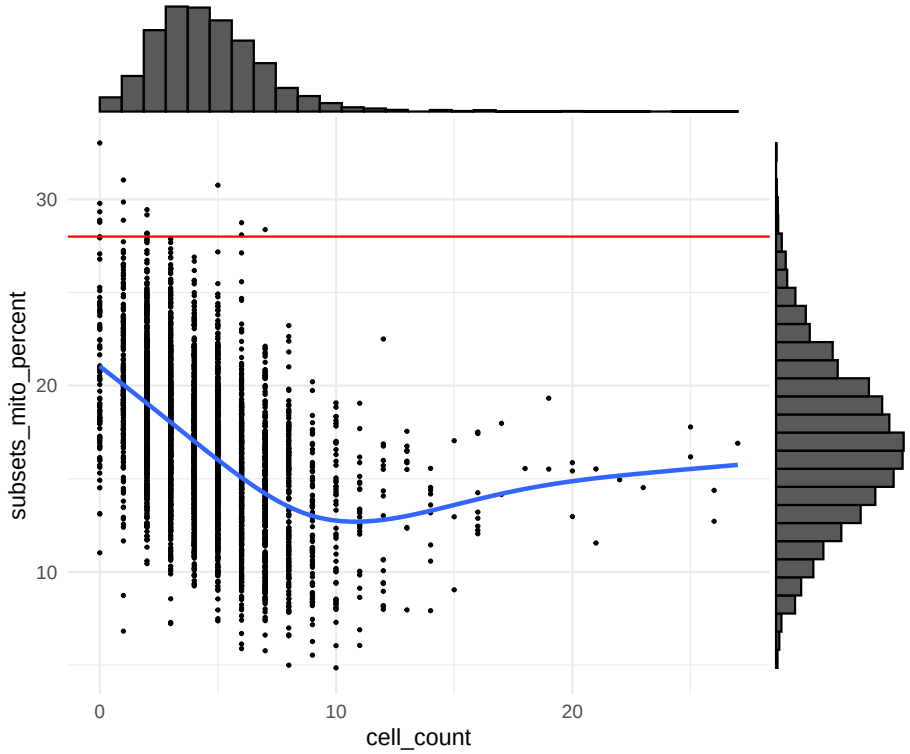
```
## Density and histogram of percentage of mitochondrial expression
ggplot(data = as.data.frame(colData(spe)),
       aes(x = subsets_mito_percent)) +
```

```
  geom_histogram(aes(y = after_stat(density)),
                 colour = "black",
                 fill = "grey") +
  geom_density(alpha = 0.5,
               adjust = 1.0,
               fill = "#A0CBE8",
               colour = "#4E79A7") +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  xlab("Percentage of mitochondrial expression") +
  ylab("Density") +
  theme_classic()
```



In this instance, a higher percentage of mitochondrial expression is the thing to avoid, so the threshold is an upper bound, rather than the lower bounds we have observed so far. Our suggestion this time is to cut-off at 28%.

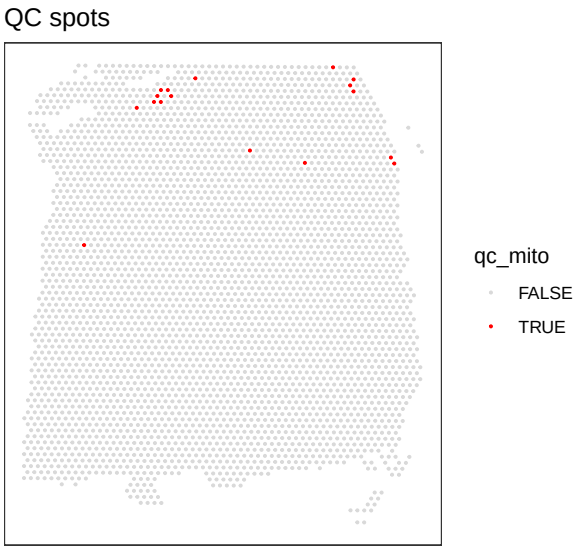
```
# plot mitochondrial read proportion vs. number of cells per spot
p = ggplot(as.data.frame(colData(spe)), aes(x=cell_count, y=subsets_mito_percent)) +
  geom_point(size=0.5) +
  geom_smooth(se=FALSE) +
  geom_hline(yintercept = 28, colour='red') +
  theme_minimal()
ggMarginal(p, type='histogram')
```




```
## Select expressed genes threshold
qc_mito <- colData(spe)$subsets_mito_percent > 28
## Check how many spots are filtered out
table(qc_mito)
```

```
## qc_mito
## FALSE  TRUE
##  3622    17
```

```
## Add threshold in colData
colData(spe)$qc_mito <- qc_mito

## Check for putative spatial pattern of removed spots
plotQC(spe, type = "spots",
       discard = "qc_mito")
```



Again, try to illustrate what happens if we set the threshold too low (i.e., 20 0r 25%).

NOTE: For reference, remember the ground truth layers in this dataset that we plotted at the beginning of this session.

```
## Select library size threshold
code...
## Check how many spots are filtered out
code...
## Add threshold in colData
code...

## Check putative spatial patterns of removed spots
plotQC(...)
```

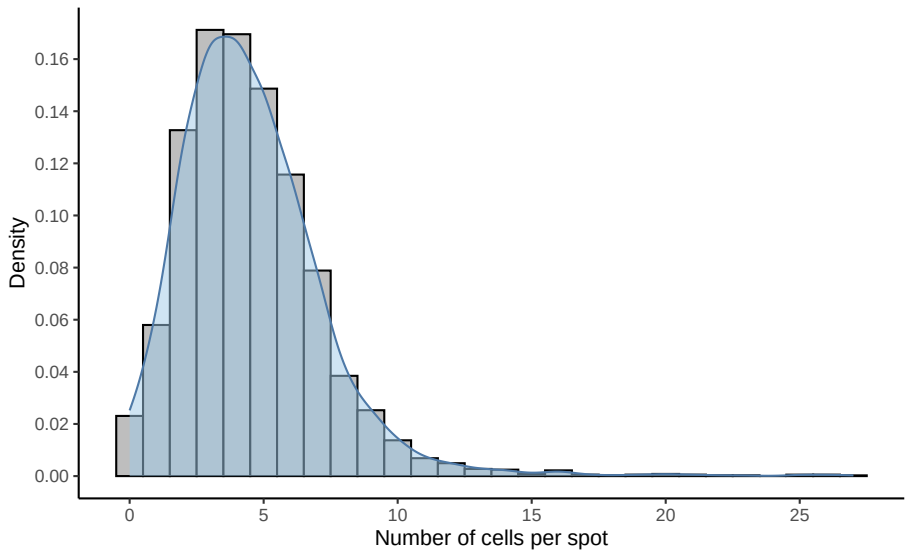
2.1.6 Number of cells per spot

As previously mentioned, number of cells per spot is an attribute that not all datasets include. Nonetheless, it can be useful to further control the quality of the dataset prior to any downstream analysis. Of course, the number of cells per spot depends on the tissue type and organism and according to 10X Genomics, each spot typically contains between 0 and 10 cells.

The DPFLC dataset does contain information on the number of cells per spot (acquired by processing and cell segmentation of high-resolution histology images obtained prior on-slide cDNA synthesis, see Maynard et al. [2021] for details). To investigate the number of cells in each spot looking for any outlier values that could indicate problems we need to take a look in the column

`cell_count` in `colData`.

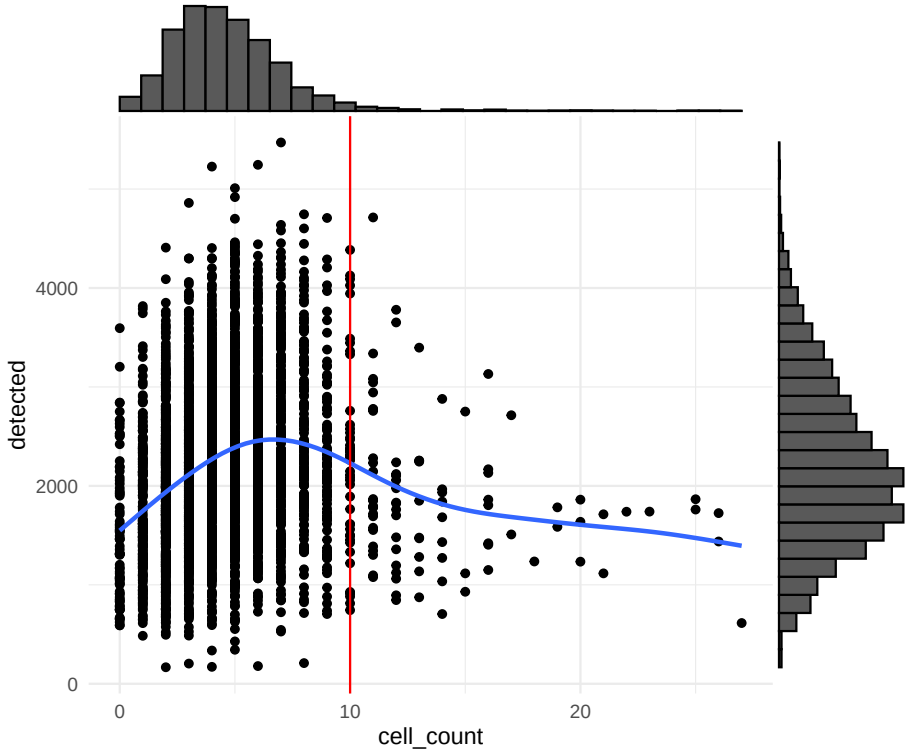
```
## Density and histogram of the number of cells in each spot
ggplot(data = as.data.frame(colData(spe)),
       aes(x = cell_count)) +
  geom_histogram(aes(y = after_stat(density)),
                 binwidth = 1,
                 colour = "black",
                 fill = "grey") +
  geom_density(alpha = 0.5,
               adjust = 1.5,
               fill = "#A0CBE8",
               colour = "#4E79A7") +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  xlab("Number of cells per spot") +
  ylab("Density") +
  theme_classic()
```




```
## Have a look at the values
table(colData(spe)$cell_count)
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19
##  84 211 483 623 617 541 421 287 140  92  50  25  18  10   9   3   8   2   1   2
##  20  21  22  23  25  26  27
##   3   2   1   1   2   2   1
```

```
# plot number of expressed genes vs. number of cells per spot
p = ggplot(as.data.frame(colData(spe)), aes(x=cell_count, y=detected)) +
  geom_point() +
  geom_smooth(se=FALSE) +
  geom_vline(xintercept = 10, colour='red') +
  theme_minimal()
ggMarginal(p, type='histogram')
```

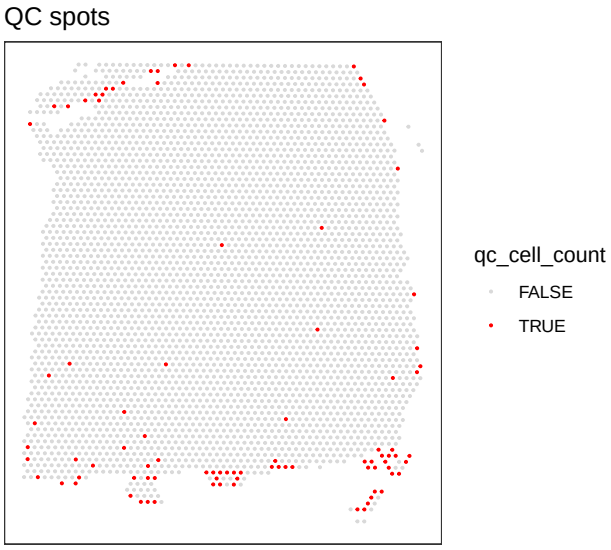


As we can see from both the histogram and the scatter plot there is a tail of very high values, which could indicate problems for these spots. More specifically, we can see from the scatter plot that most of the spots with very high cell counts also tend to have lower numbers of expressed genes. This indicates problems with the experiment on these spots, and they should be removed.

```
## Select expressed genes threshold
qc_cell_count <- colData(spe)$cell_count > 10
## Check how many spots are filtered out
table(qc_cell_count)
```

```
## qc_cell_count
## FALSE  TRUE
##  3549    90
```

```
## Add threshold in colData
colData(spe)$qc_cell_count <- qc_cell_count

## Check for putative spatial pattern of removed 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 typically at 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.

2.1.7 Remove low-quality spots

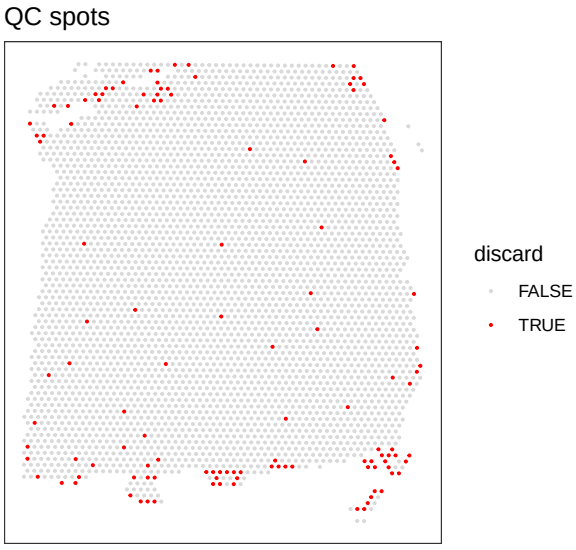
All the steps so far have flagged spots with potential issues - before proceeding with analysis, we want to remove these spots from our SpatialExperiment object. Since we have calculated different spot-level QC metrics and selected thresholds for each one, we can combine them to identify a set of low-quality spots, and remove them from our `spe` object in a single step.

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

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

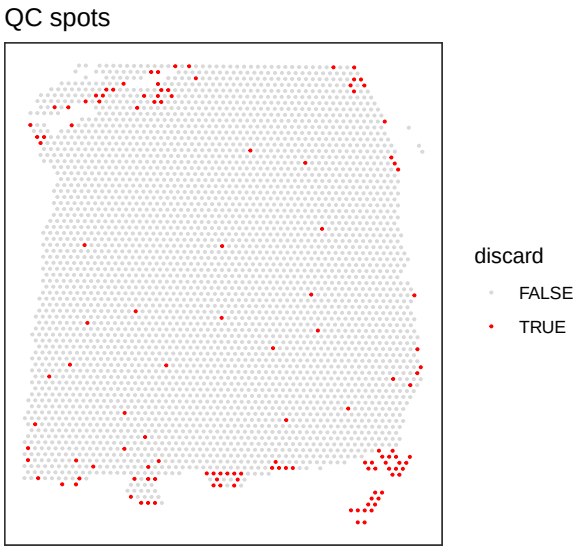
```
##   qc_lib_size   qc_detected       qc_mito qc_cell_count
##            11            11            17            90
```

```
## Combine together the set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count
## Store the set in the object
colData(spe)$discard <- discard

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



Since this dataset has also manual annotation (remember)) we see that there are locations that are not annotated (marked with `NA`). We could further remove those locations to reduce potential noise and further increase the quality of the dataset.

```
## Select locations without annotation
qc_NA_spots <- is.na(colData(spe)$ground_truth)
## Combine together the set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito | qc_cell_count | qc_NA_spots
## Store the set in the object
colData(spe)$discard <- discard

## Check the 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]
```

2.2 Normalisation of counts

2.2.1 Background

Normalisation is applied in STx data for the same reason as any other RNA-Seq technique - the differences observed in the count data can arise from a range of systematic factors, not just a physiologically-relevant change in expression. The primary systematic effect is that of library size (or in the case of STx, counts/UMIs per spot). `scater` corrects for library size by scaling the sizes across all spots such that the mean library size is 1. Normalized counts are then calculated as a ratio of observed count to library size factor.

Secondly, a log-transformation is applied to the scaled counts - this transformation is commonly applied as it stabilises the variance across the range of transcriptomics data (otherwise the variance is dominated by highly expressed genes) and it facilitates comparisons of expression by rendering positive and negative changes symmetrical and found by subtraction rather than division. Since $log2(0)$ is undefined, a *pseudocount* is added to each observed count to avoid this error - a pseudocount of 1 is typically applied, as $log2(0+1) = 0$.

Here we will be using methods from the `scater` [McCarthy et al., 2017] and `scran` [Lun et al., 2016] packages that calculate logcounts using library size factors. The library size factors approach is arguably the simplest approach for STx data. Other approaches used in scRNA-seq are more difficult to justify their use in STx because of two main reasons:

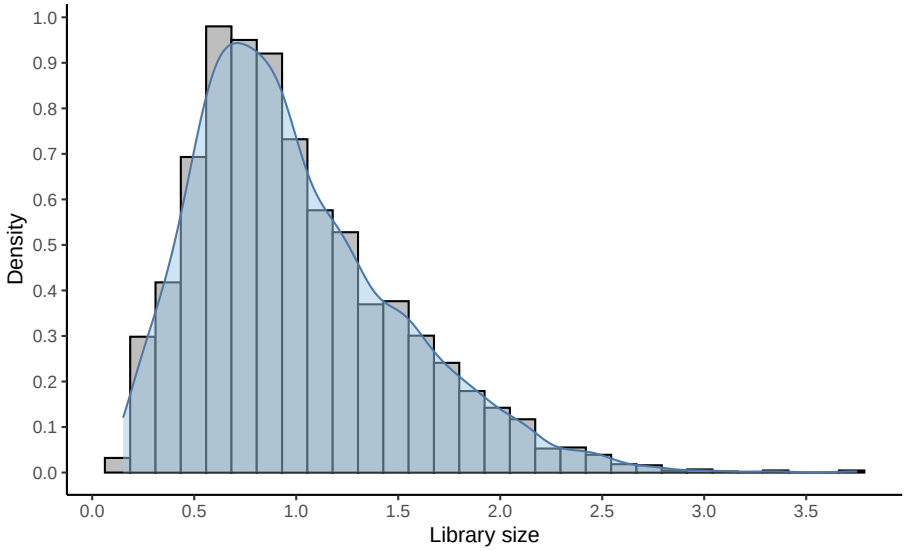
1. Spots can contain multiple cells of different cell-types.
2. Datasets can include multiple tissue samples which will lead to different clusterings.

```
## Calculate library size factors
spe <- computeLibraryFactors(spe)
## Have a look at the size factors
summary(sizeFactors(spe))
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.1514  0.6326  0.9011  1.0000  1.2849  3.7500
```

As described above, the mean size factor is 1.0.

```
## Density and histogram of library sizes
ggplot(data = data.frame(sFact = sizeFactors(spe)),
       aes(x = sFact)) +
  geom_histogram(aes(y = after_stat(density)),
                 colour = "black",
                 fill = "grey") +
  geom_density(alpha = 0.5,
               adjust = 1.0,
               fill = "#A0CBE8",
               colour = "#4E79A7") +
  scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
  scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
  xlab("Library size") +
  ylab("Density") +
  theme_classic()
```



The log-transformation that takes place is a log2-transformation and in order to avoid - *Infinity* values we add a pseudo value of 1. Both the log2- transformation and the pseudocount of 1 are defaults in this method.

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

## Check that a new assay has been added
assayNames(spe)
```

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

2.3 Selecting genes

2.3.1 Background

Gene selection - or alternatively "feature selection" - is applied to identify genes that are likely to be informative for downstream analyses. The most common feature selection method is the definition of highly variable genes (HVGs). The assumption is that since we quality-controlled and normalised our dataset, the genes with high variability are the ones that contain high levels of biological variability too. Since here we have a spatial dataset we can also try to identify spatially variable genes too (SVGs).

It is important to note that HVGs are identified solely from the gene expression data. Spatial information does not play a role in finding HVGs. STx data pose a dilemma; does the meaningful spatial information reflect only spatial distribution of major cell types or does it reflect additional important spatial

features? If we believe the former, relying on HVGs can be enough. If the second also holds true though, it is important to identify SVGs as well.

2.3.2 Highly Variable Genes (HVGs)

Here we will be using methods from the `scran` package [Lun et al., 2016] to identify a set of HVGs. Again, here we need to remember that `scran` methods were developed for scRNA-seq and we are performing the analysis under the assumption that the spots of an STx experiment can be treated as single cells.

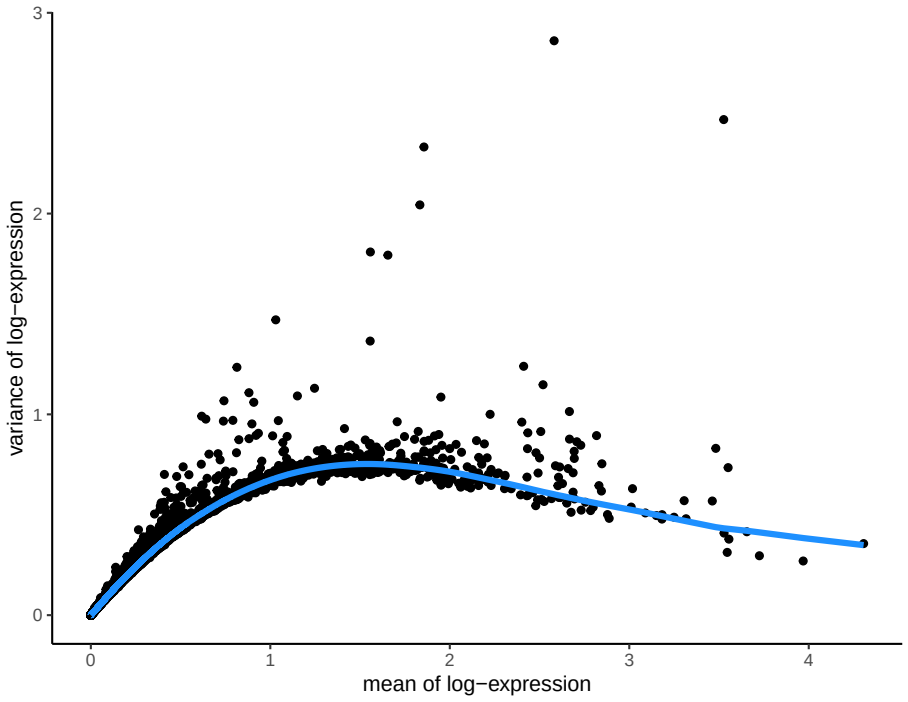
In this dataset, the mitochondrial genes are too highly expressed and are not of major biological interest. As a result, if we are to identify true HVGs, we first need to remove the mitochondrial genes.

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

Then, we apply methods from `scran` that give a list of HVGs, which can be used for further downstream analyses.

First we model the variance of the log-expression profiles for each gene, decomposing it into technical and biological components based on a fitted mean-variance trend.

```
## Fit mean-variance relationship
dec <- modelGeneVar(spe)
## Visualize mean-variance relationship
fit <- metadata(dec)
fit_df <- data.frame(mean = fit$mean,
                     var = fit$var,
                     trend = fit$trend(fit$mean))

ggplot(data = fit_df,
       aes(x = mean, y = var)) +
  geom_point() +
  geom_line(aes(y = trend), colour = "dodgerblue", linewidth = 1.5) +
  labs(x = "mean of log-expression",
       y = "variance of log-expression") +
  theme_classic()
```



The `trend` function that we used above is returned from the `modelGeneVar` function and returns the fitted value of the trend at any value of the mean. The "biological" variance of a gene is what remains when the fitted variance for a gene of that expression value is subtracted from the total variance (so genes above the blue trend line have a positive biological variance).

We select the top 10% of genes based on their biological variability The parameter `prop` defines how many HVGs we want. For example `prop = 0.1` returns the top 10% of genes. `prop = 1.0` would return all genes with a positive biological variability.

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

## How many HVGs?
length(top_hvgs)
```

```
## [1] 1429
```

NOTE - we will return to feature selection in the next practical, as it is a complicated process with significant impacts on the chosen downstream analysis.

2.3.3 Spatially variable genes (SVGs)

SVGs are genes with a highly spatially correlated pattern of expression, which varies along with the spatial distribution of a tissue structure of interest. This phenomenon is also called *spatial autocorrelation* and underlies all types of spatial data, as we will discuss later.

The field of geography has developed some statistical measures to calculate spatial autocorrelation. Examples of these are Moran's I [Mor, 1950] and Geary's C [Gea, 1954] that can be used to rank genes by the observed spatial autocorrelation to identify SVGs.

Several sophisticated new statistical methods to identify SVGs in STx data have also recently been developed. These include SpatialDE [Svensson et al., 2018], SPARK [Sun et al., 2020], and SPARK-X [Zhu et al., 2021].

2.3.4 Integration of HVGs and SVGs

A recent benchmark paper [Li et al., 2022] showed that integrating HVGs and SVGs to generate a combined set of features can improve downstream clustering performance in STx data. This confirms that SVGs contain additional biologically relevant information that is not captured by HVGs in these datasets. For example, a simple way to combine these features is to concatenate columns of principal components (PCs) calculated on the set of HVGs and the set of SVGs (excluding overlapping HVGs), and then using the combined set of features for further downstream analyses [Li et al., 2022].

2.4 Dimensionality reduction

2.4.1 Background

STx data, just like bulk and single-cell transcriptomics, is captured in high-dimensional space. The reduction of this complexity can be helpful for a number of applications. Principal Components Analysis (PCA) assumes linearity in the data and has historically been used for dimensionality reduction. More modern techniques, such as Uniform Manifold Approximation and Projection (UMAP) [McInnes et al., 2018] and t-Stochastic Neighbor Embedding (tSNE) [van der Maaten and Hinton, 2008] do not assume linearity and provide some performance advantages.

The main practical difference between the output of these techniques is that the distances between the data points in PCA space are interpretable and can be used for clustering, while the distances in a UMAP/tSNE embedding are not interpretable in this way. As a result, we will be using PCA to reduce the dimensions of our dataset to assist clustering and UMAP to further reduce the principal components (PCs) in a two-dimensional space and produce better visualisations for the PCA.

Dimensionality reduction prior to clustering has two main advantages, firstly it reduces dataset noise from random variation. Secondly it improves the computational efficiency of downstream analyses such as clustering. In an STx experiment, like the one we are analysing here, we have more than 3,000 spots and almost 1,500 HVGs. As as result, each spot has 1,500 attributes (dimensions) as a basis for subsequent clustering. This large number of variables that differentiate or cluster together spots gives rise to the *curse of dimensionality* [Keogh and Mueen, 2017]. This principle states that data points (spots) with a large number of features appear equidistant in attribute space resulting in poor clustering output.

2.4.2 PCA: Principal component analysis

Here we will use an efficient implementation of PCA provided in the `scater` package [McCarthy et al., 2017] and retain the top 50 PCs for further downstream analyses. The random seed is required for reproducibility reasons because this implementation uses randomisation.

```
## Set seed
set.seed(987)
## Compute PCA
spe <- runPCA(spe, subset_row = top_hvgs)
## Check correctness - names
reducedDimNames(spe)
```

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

```
## Check correctness - dimensions
dim(reducedDim(spe, "PCA"))
```

```
## [1] 3511   50
```

2.4.3 UMAP: Uniform Manifold Approximation and Projection

Here we will also run UMAP - using `scater`'s implementation - on the 50 PCs generated above and retain the top 2 UMAP components to visualise results.

```
## Set seed
set.seed(987)
## Compute UMAP on top 50 PCs
spe <- runUMAP(spe, dimred = "PCA")
## Check correctness - names
reducedDimNames(spe)
```

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

```
## Check correctness - dimensions
dim(reducedDim(spe, "UMAP"))
```

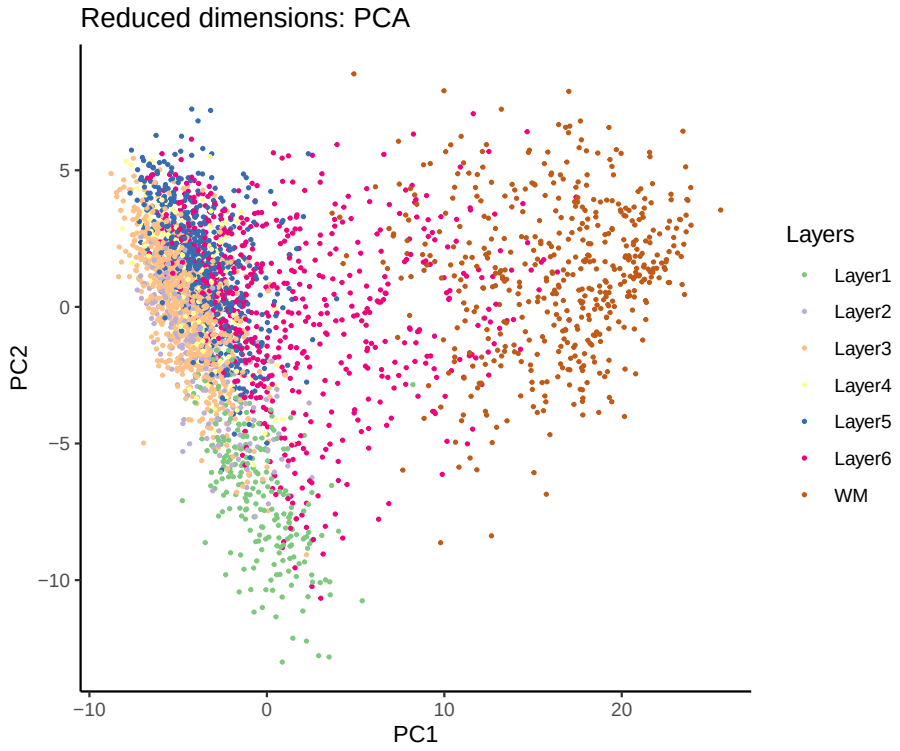
```
## [1] 3511    2
```

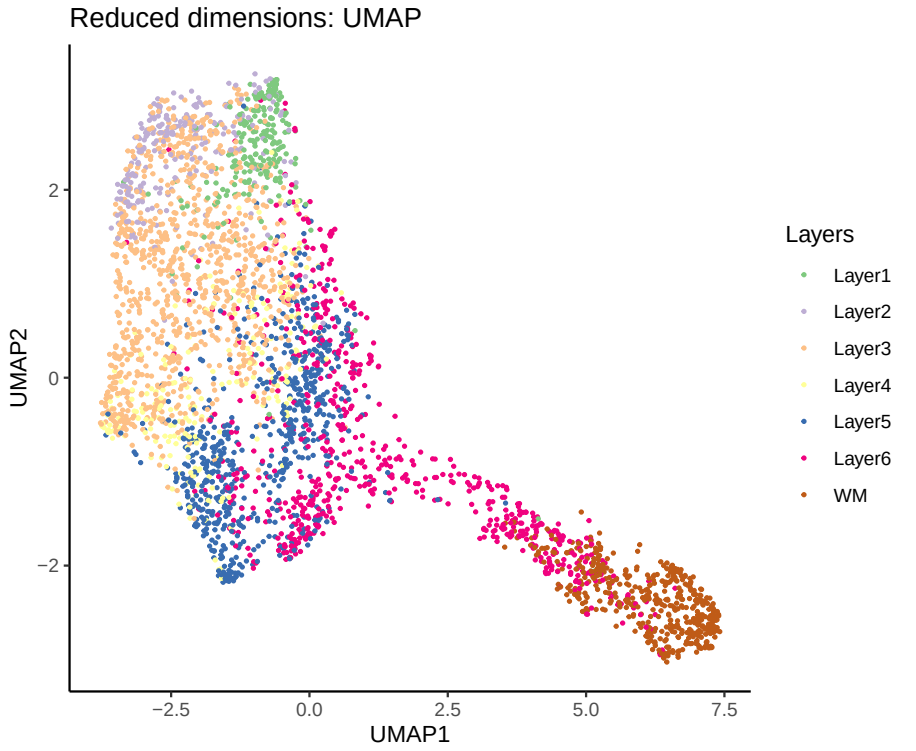
```
## Update column names for easier plotting
colnames(reducedDim(spe, "UMAP")) <- paste0("UMAP", 1:2)
```

2.4.4 UMAP visualisations

We can generate plots either using plotting functions from the ggspavis package or `ggplot2` package. When clustering later, we will add cluster labels to these reduced dimension plots for an off-tissue visualisation.

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

ggplot(data = as.data.frame(spe@int_colData@listData$reducedDims$PCA),
       aes(x = PC1, y = PC2, colour = spe@colData$ground_truth)) +
  geom_point(size = 0.5) +
  scale_colour_brewer(type = "qual") +
  labs(title = "Reduced dimensions: PCA",
       x = "PC1",
       y = "PC2",
       colour = "Layers") +
  theme_classic()

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

ggplot(data = as.data.frame(spe@int_colData@listData$reducedDims$UMAP),
       aes(x = UMAP1, y = UMAP2, colour = spe@colData$ground_truth)) +
  geom_point(size = 0.5) +
  scale_colour_brewer(type = "qual") +
  labs(title = "Reduced dimensions: UMAP",
       x = "UMAP1",
       y = "UMAP2",
       colour = "Layers") +
  theme_classic()
```


Reduced dimensions: PCA
PC2
PC1
5
0
−5
−10
−10
0
10
20
Layers
Layer1
Layer2
Layer3
Layer4
Layer5
Layer6
WM



2.5 Clustering

2.5.1 Background

The clustering of observations into statistically similar groups is a well-established application in both bulk and single-cell RNA-Seq analysis. Clustering is a helpful tool because it structures and orders the data, allowing useful insights to be gained from complex, multivariate datasets and use those insights to classify the observed data or to generate hypotheses.

Common clustering methods are applied to ST data based on correlation or statistical distance of gene expression measurements. As we briefly mentioned above, the dimensionality of STx data means that sample distances in gene expression space tend to be small and not reliable for identifying clusters, so feature selection (gene selection) or dimensionality reduction approaches (i.e., PCA, UMAP) tend to be taken before clustering.

Common approaches to clustering gene expression data include k-means, hierarchical and Louvain algorithms, and all have been applied to the clustering of ST data. Some of these methods are implemented in some of the most popular single-cell analysis packages, such as `Seurat` [Hao et al., 2021] and `scran` [Lun

et al., 2016] and have been used for clustering in a number of ST studies.

2.5.2 Clustering on HVGs

Here, we apply graph-based clustering to the top 50 PCs calculated on the set of selected HVGs, using the Walktrap method implemented in `scran` [Lun et al., 2016]. To do so, we assume that (i) each spot is equal to a cell and (ii) we can detect from the gene expression the biologically informative spatial distribution patterns of cell types.

```
## Set seed
set.seed(987)
## Set number of Nearest-Neighbours (NNs)
k <- 10
## Build the k-NN graph
g <- buildSNNGraph(spe, k = k, use.dimred = "PCA")
## Run walktrap clustering
g_walk <- igraph::cluster_walktrap(g)
## Get the cluster labels
clus <- g_walk$membership
## Check how many
table(clus)
```

```
## clus
##   1   2   3   4   5   6
## 350 354 661 895 366 885
```

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

2.5.3 HVGs clustering visualisations

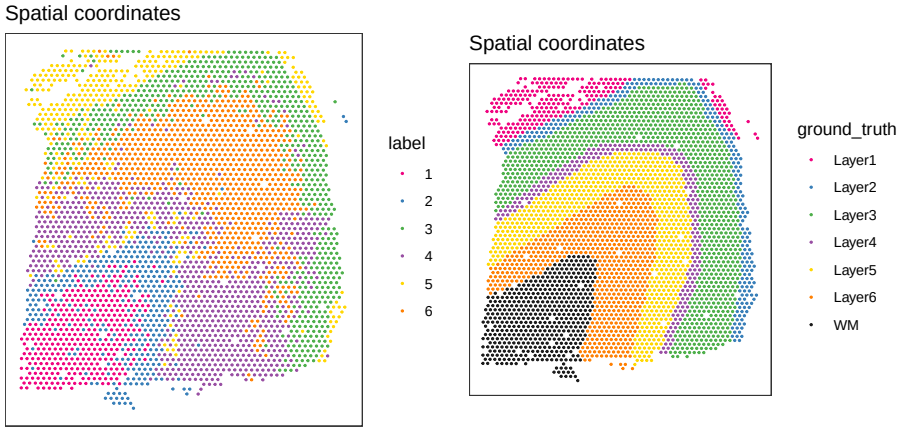
We can visualise the clusters in two ways:

1. plotting in spatial coordinates on the tissue map
2. plotting in the UMAP/PCA embeddings.

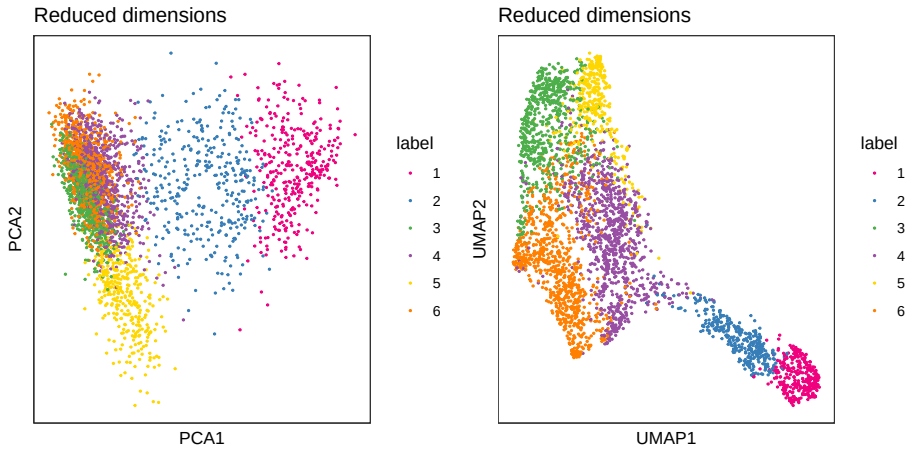
We can use plotting functions either from the ggspavis package.

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

```
## Plot in tissue map
plotSpots(spe, annotate = "label",
          palette = "libd_layer_colors")

## Plot ground truth in tissue map
plotSpots(spe, annotate = "ground_truth",
          palette = "libd_layer_colors")
```




```
## Plot clusters in PCA space
plotDimRed(spe, type = "PCA",
           annotate = "label", palette = "libd_layer_colors")

## Plot clusters in UMAP space
plotDimRed(spe, type = "UMAP",
           annotate = "label", palette = "libd_layer_colors")
```



From the visualizations, we can see that the clustering reproduces, to an extent, the known biological structure of the tissue, but not perfectly. One reason for this could be the fact that each spot may be comprised of a number different cells whose gene expression profiles are diluted in the overall profile of the spot,

thus leading to low-quality clustering.

2.5.4 Spatially-aware clustering

In STx 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.

2.6 Inter-cluster differentially expressed genes (DGEs)

2.6.1 Background

Here, we will identify differentially expressed genes between clusters.

We will use the `findMarkers` implementation from the `scran` [Lun et al., 2016]. This implementation uses 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.

2.6.2 DGEs identification

```
## Set gene names as row names ease of plotting
rownames(spe) <- rowData(spe)$gene_name
## Test for DGEs
markers <- findMarkers(spe, test = "binom", direction = "up")
## Check output
markers
```

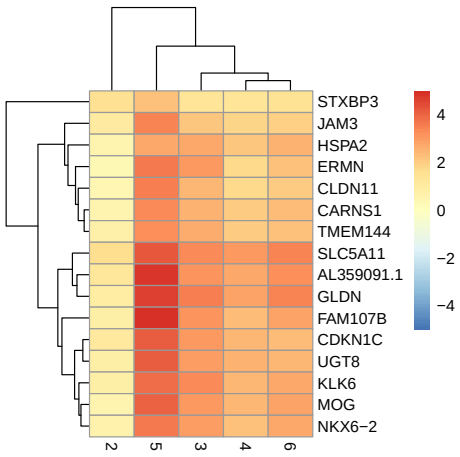
```
## List of length 6
## names(6): 1 2 3 4 5 6
```

The output from the `findMarkers` implementation is a list of length equal to the number of clusters. Each element of the list contains the Log-Fold-Change (LogFC) of each gene between one cluster and all others.

2.6.3 DGEs visualisation

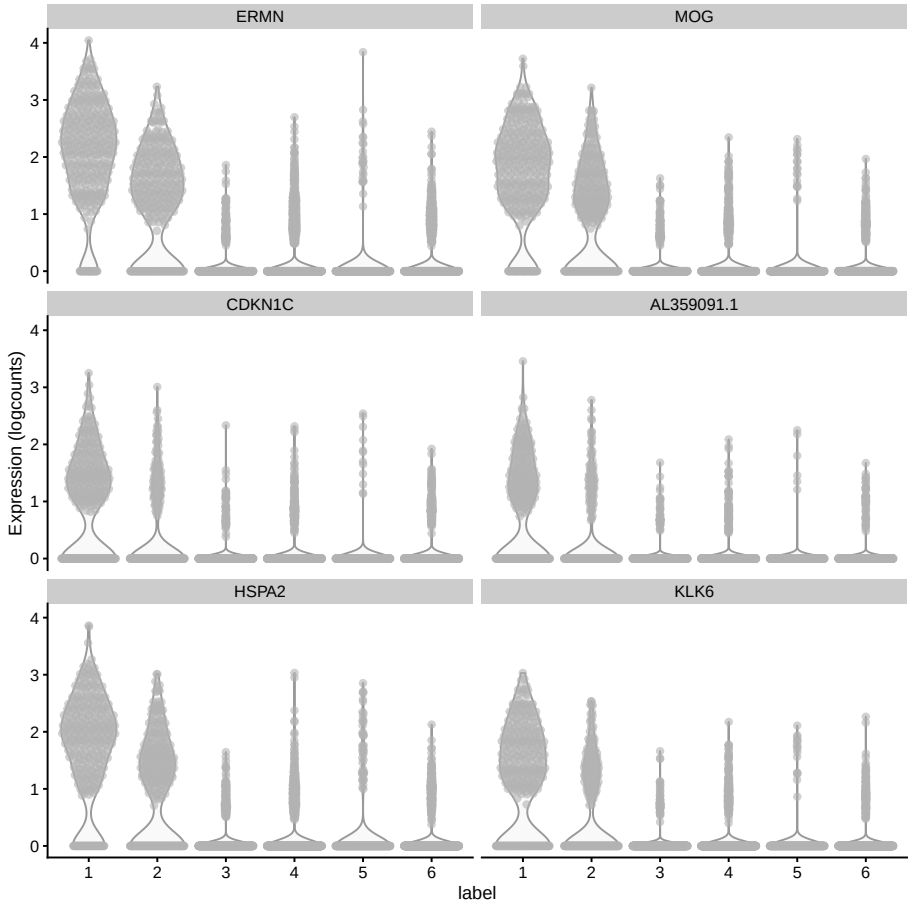
Here we will plot LogFCs for cluster 1 against all other clusters

```
## Select cluster 1 genes
interesting <- markers[[1]]
## Get the top genes
best_set <- interesting[interesting$Top <= 5, ]
## Calculate the effect
logFCs <- getMarkerEffects(best_set)
## Plot a heat map
pheatmap(logFCs, breaks = seq(-5, 5, length.out = 101))
```



Below we will plot the log-transformed normalised expression of the top genes for one cluster alongside their expression in the other clusters.

```
## Select genes
top_genes <- head(rownames(interesting))
## Plot expression
plotExpression(spe, x = "label", features = top_genes)
```



2.7 Putting it all together

```
# clear workspace from previous chapters
rm(list = ls(all = TRUE))

# 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)

# 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)

# MARKER GENES
# test for marker genes
rownames(spe) <- rowData(spe)$gene_name
markers <- findMarkers(spe, test = "binom", direction = "up")
```

Chapter 3

Practical session 3

This practical session will demonstrate the application of the most commonly used spatial analysis tools to STx data, and how we work with coordinate data alongside expression data.

3.1 Load packages

- `spdep` is a collection of functions to create spatial weights matrix objects from polygon *contiguities*, from point patterns by distance and tessellations. It is used for summarizing these objects, and for permitting their use in spatial data analysis like regional aggregation and tests for spatial *autocorrelation*.
- `sf` (*Simple Features for R*) is a package that offers support for simple features, a standardized way to encode spatial vector data.
- `GWmodel` is a suite of models that fit situations when data are not described well by some global model, but where there are spatial regions where a suitably localised calibration provides a better description.

3.2 Background

3.2.1 Main geocomputational data structures

There are three main data structures that we need to have ready before we undertake a geocomputational approach to STx data analysis. Namely these are; (1) geometries (point and polygon), (2) neighbours lists and (3) distance matrices.

1. Spatial geometries can be points, lines, polygons and pixels. Polygons consist of a multitude of points connected by lines and can have many

forms like circle, hexagon, non-canonical polygon etc.

2. Neighbour lists are special types of lists that contain information about the neighbours of each polygon. The neighbours can be defined either by adjacency or by distance.

3. Distance matrices contain the distances between different points and can be either weighted or un-weighted. The weighted distances are usually objective to each point and its neighbours. Meaning that the closer or farther a neighbour is from the point of focus, the weight of their distance changes according to an applied kernel. Usually in the case of STx data, like the ones generated by the 10X Visium platform, the un-weighted distance between two points is expressed in pixels and we acquire it from the `spaceranger` output.

3.2.2 The `sf` objects

Package `sf` represents simple features as native R objects. All functions and methods in `sf` that operate on spatial data are prefixed by *st_*, which refers to *spatial type*. Simple features are implemented as R native data, using simple data structures (S3 classes, lists, matrix, vector). The typical use of `sf` involves reading, manipulating and writing of sets of features, with attributes and geometries.

As attributes are typically stored in `data.frame` objects (or the very similar `tbl_df`), we will also store feature geometries in a `data.frame` column. Since geometries are not single-valued, they are put in a list-column, a list of length equal to the number of records in the `data.frame`, with each list element holding the simple feature geometry of that feature. The three classes used to represent simple features are:

- `sf`, the table (`data.frame`) with feature attributes and feature geometries, which contains
- `sfc`, the list-column with the geometries for each feature (record), which is composed of
- `sfg`, the feature geometry of an individual simple feature.

3.2.2.1 Simple feature geometry types

The following seven simple feature types are the most common:

type	description
`POINT`	zero-dimensional geometry containing a single point
`LINESTRING`	sequence of points connected by straight, non-self intersecting line pieces; one-dimensional geometry

type	description
POLYGON	geometry with a positive area (two-dimensional); sequence of points form a closed, non-self intersecting ring; the first ring denotes the exterior ring, zero or more subsequent rings denote holes in this exterior ring
MULTIPOINT	set of points; a MULTIPOINT is simple if no two Points in the MULTIPOINT are equal
MULTILINESTRING	set of linestrings
MULTIPOLYGON	set of polygons
GEOMETRYCOLLECTION	set of geometries of any type except GEOMETRYCOLLECTION

Each of the geometry types can also be a (typed) empty set, containing zero coordinates (for POINT the standard is not clear how to represent the empty geometry). Empty geometries can be thought of as being the analogue to missing (NA) attributes, NULL values or empty lists.

3.2.2.2 sf: objects with simple features

As we usually do not work with geometries of single simple features, but with datasets consisting of sets of features with attributes, the two are put together in sf (simple feature) objects. The following command reads a test dataset called nc from a file that is contained in the sf package:

```
nc <- st_read(system.file("shape/nc.shp", package = "sf"))
```

```
## Reading layer `nc' from data source
##   `/home/sjcockell/R/x86_64-pc-linux-gnu-library/4.3/sf/shape/nc.shp'
##   using driver `ESRI Shapefile'
## Simple feature collection with 100 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## Geodetic CRS:  NAD27
```

The short report printed gives the file name, the driver (ESRI Shapefile), mentions that there are 100 features (records, represented as rows) and 14 fields (attributes, represented as columns).

This object is of class:

```
class(nc)
```

```
## [1] "sf"         "data.frame"
```

meaning it extends (and "is" a) data.frame, but with a single list-column with geometries, which is held in the column with name:

```
attr(nc, "sf_column")
```

```
## [1] "geometry"
```

If we print the first three features, we see their attribute values and an abridged version of the geometry

```
print(nc[9:15], n = 3)
```

which would give the following output:

```
## Simple feature collection with 100 features and 6 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID):    4267
## proj4string:    +proj=longlat +datum=NAD27 +no_defs
## precision:      double (default; no precision model)
## First 3 features:
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79                           geom
## 1  1091     1       10  1364     0      19 MULTIPOLYGON(((-81.47275543...
## 2   487     0       10   542     3      12 MULTIPOLYGON(((-81.23989105...
## 3  3188     5      208  3616     6     260 MULTIPOLYGON(((-80.45634460...
```

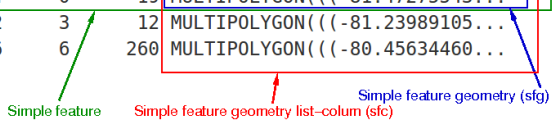


Figure 3.1: Overview of the 'sf' object.

In the output we see:

- in green a simple feature: a single record, or `data.frame` row, consisting of attributes and geometry
- in blue a single simple feature geometry (an object of class `sfg`)
- in red a simple feature list-column (an object of class `sfc`, which is a column in the `data.frame`)
- that although geometries are native R objects, they are printed as well-known text

It is also possible to create `data.frame` objects with geometry list-columns that are not of class `sf`, e.g. by:

```
nc.no_sf <- as.data.frame(nc)
class(nc.no_sf)
```

```
## [1] "data.frame"
```

However, such objects:

- no longer register which column is the geometry list-column
- no longer have a plot method, and
- lack all of the other dedicated methods for class `sf`

3.2.2.3 sfc: simple feature geometry list-column

The column in the `sf` data.frame that contains the geometries is a list, of class `sfc`. We can retrieve the geometry list-column in this case by using standard `data.frame` notation like `nc$geom` or `nc[[15]]`, but the more general way uses `st_geometry`:

```
(nc_geom <- st_geometry(nc))
```

```
## Geometry set for 100 features
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## Geodetic CRS:  NAD27
## First 5 geometries:

## MULTIPOLYGON (((-81.47276 36.23436, -81.54084 3...

## MULTIPOLYGON (((-81.23989 36.36536, -81.24069 3...

## MULTIPOLYGON (((-80.45634 36.24256, -80.47639 3...

## MULTIPOLYGON (((-76.00897 36.3196, -76.01735 36...

## MULTIPOLYGON (((-77.21767 36.24098, -77.23461 3...
```

Geometries are printed in abbreviated form, but we can view a complete geometry by selecting it, e.g. the first one by:

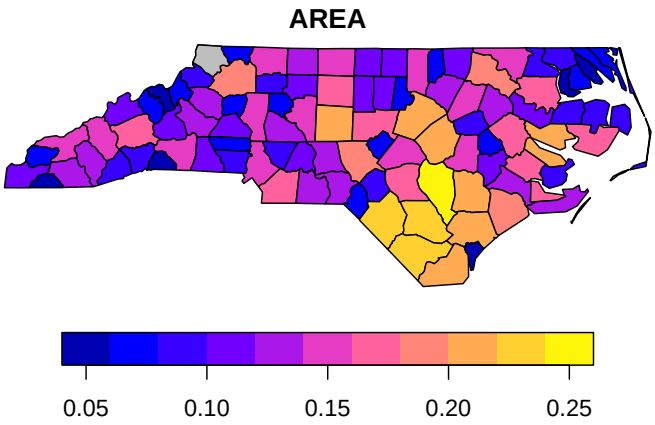
```
nc_geom[[1]]
```

```
## MULTIPOLYGON (((-81.47276 36.23436, -81.54084 36.27251, -81.56198 36.27359, -81.63306 36.34069
```

The way this is printed is called *well-known text*, and is part of the standards. The word `MULTIPOLYGON` is followed by three parentheses, because it can consist of multiple polygons, in the form of `MULTIPOLYGON(POL1,POL2)`, where `POL1` might consist of an exterior ring and zero or more interior rings, as of `(EXT1,HOLE1,HOLE2)`. Sets of coordinates are held together with parentheses, so we get `((crds_ext)(crds_hole1)(crds_hole2))` where `crds_` is a comma-separated set of coordinates of a ring. This leads to the case above, where `MULTIPOLYGON(((crds_ext)))` refers to the exterior ring (1), without holes (2), of the first polygon (3) - hence three parentheses.

We can see there is a single polygon with no rings:

```
par(mar = c(0,0,1,0))
plot(nc[1], reset = FALSE) # reset = FALSE: we want to add to a plot with a legend
plot(nc[1,1], col = 'grey', add = TRUE)
```



Following the `MULTIPOLYGON` data structure, in R we have a list of lists of lists of matrices. For instance, we get the first 3 coordinate pairs of the second exterior ring (first ring is always exterior) for the geometry of feature 4 by:

```
nc_geom[[4]][[2]][[1]][1:3,]
```

```
##           [,1]     [,2]
## [1,] -76.02717 36.55672
## [2,] -75.99866 36.55665
## [3,] -75.91192 36.54253
```

Geometry columns have their own class,

```
class(nc_geom)
```

```
## [1] "sfc_MULTIPOLYGON" "sfc"
```

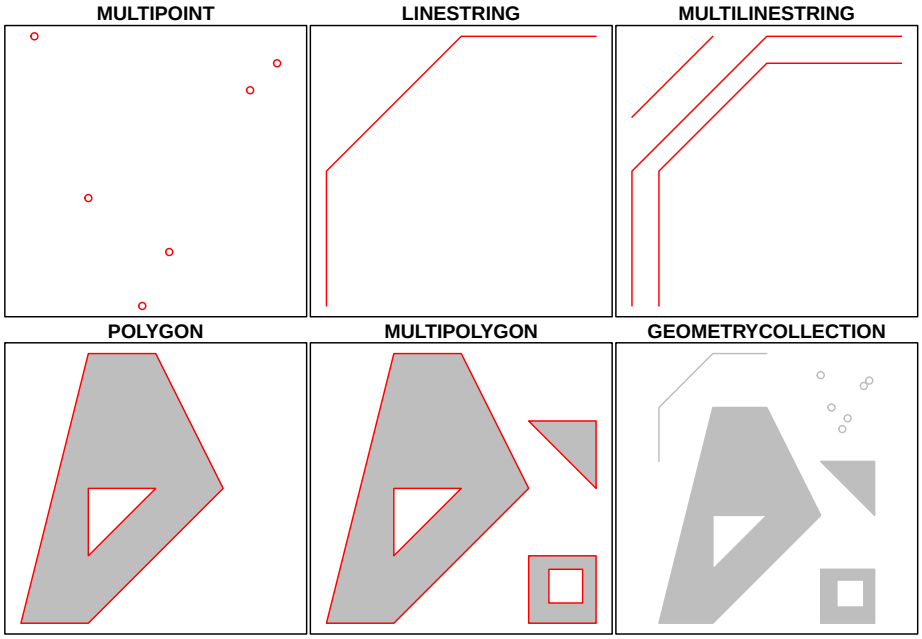
3.2.2.4 sfg: simple feature geometry

Simple feature geometry (`sfg`) objects carry the geometry for a single feature, e.g. a point, linestring or polygon.

Simple feature geometries are implemented as R native data, using the following rules

1. a single POINT is a numeric vector
2. a set of points, e.g. in a LINESTRING or ring of a POLYGON is a `matrix`, each row containing a point
3. any other set is a `list`

The below figure illustrates the different types of geometries:



Geometries can also be empty, as in

```
(x <- st_geometrycollection())
## GEOMETRYCOLLECTION EMPTY
length(x)
## [1] 0
```

The above are taken from the very well written, well-descriptive and thorough `sf` *package vignette.*

3.3 Data structures preparation

For this practical we will be using a human steatotic kidney dataset from the Liver Atlas [Guilliams et al., 2022]. Specifically we will use the JBO019 sample.

3.3.1 Load new dataset

Note - Between now and section 3.8 nothing new is introduced (this is a repetition of the QC carried out in practical 2, but with this new liver dataset). Feel free to skip over this section and use the code block just before section 3.8 to run this QC quickly for this data.

First we generate the `SpatialFeaturesExperiment` object which is an extension of the `SpatialExperiment` (SPE) object that we used in the 2nd practical session. The difference is that the SFE object has incorporated the `sf` object structure and thus can accommodate the use of geocomputational tools.

```
sampleDir <- "./data/spaceranger_outs/Human_Liver_Steatotic/JBO019_Results"
sampleNames <- "JBO019"
sfe <- read10xVisiumSFE(samples = sampleDir,
                        sample_id = sampleNames,
                        type = "sparse",
                        data = "filtered",
                        images = "lowres",
                        style = "W",
                        zero.policy = TRUE)

ground_truth <- read_table("./data/to_load/spotzonationGroup.txt")
```

3.4 Spot-level Quality Control

3.4.1 Calculating QC metrics

In this section we are effectively recapitlating the spot- and gene-level QC from practical 2 for this new dataset, in order that we can use it for the practical exercises in the next session.

```
is_mito <- grepl("(^MT-)|(^mt-)", rowData(sfe)$symbol)
sfe <- addPerLocQC(sfe, gTruth = ground_truth, assay = "counts", 2, subsets = list(mito
sfe <- addGeometries(sfe, samples = sampleDir, sample_id = sampleNames, res = "fullres"
sfe <- addPerGeneQC(sfe, assay = "counts", version = NULL, mirror = NULL)

colData(sfe)
```

```
## DataFrame with 1185 rows and 15 columns
##                     in_tissue array_row array_col   sample_id            Barcode
##                     <logical> <integer> <integer> <character>        <character>
## AAACAAGTATCTCCCA-1      TRUE        50       102      JBO019 AAACAAGTATCTCCCA-1
## AAACATTTCCCGGATT-1      TRUE        61        97      JBO019 AAACATTTCCCGGATT-1
## AAACCCGAACGAAATC-1      TRUE        45       115      JBO019 AAACCCGAACGAAATC-1
## AAACGAGACGGTTGAT-1      TRUE        35        79      JBO019 AAACGAGACGGTTGAT-1
## AAACTAACGTGGCGAC-1      TRUE         8       110      JBO019 AAACTAACGTGGCGAC-1
## ...                      ...       ...       ...         ...                ...
## TTGTAATCCGTACTCG-1      TRUE        35        55      JBO019 TTGTAATCCGTACTCG-1
## TTGTGAACCTAATCCG-1      TRUE        56        90      JBO019 TTGTGAACCTAATCCG-1
## TTGTGCAGCCACGTCA-1      TRUE        60        74      JBO019 TTGTGCAGCCACGTCA-1
## TTGTGTTTCCCGAAAG-1      TRUE        51        59      JBO019 TTGTGTTTCCCGAAAG-1
## TTGTTGTGTGTCAAGA-1      TRUE        31        77      JBO019 TTGTTGTGTGTCAAGA-1
##                      Capt_area  annotation       index  sparsity       sum
##                     <character> <character> <character> <numeric> <numeric>
## AAACAAGTATCTCCCA-1           1          NA      spot_1  0.910410     13443
## AAACATTTCCCGGATT-1           1          NA      spot_2  0.967805      2648
```

```
## AAACCCGAACGAAATC-1           1         Mid      spot_3  0.864958     27733
## AAACGAGACGGTTGAT-1           1     Central      spot_4  0.835818     32973
## AAACTAACGTGGCGAC-1           1          NA      spot_5  0.995418       400
## ...                        ...         ...         ...       ...       ...
## TTGTAATCCGTACTCG-1           1          NA   spot_1181  0.933716      7612
## TTGTGAACCTAATCCG-1           1          NA   spot_1182  0.955831      4299
## TTGTGCAGCCACGTCA-1           1          NA   spot_1183  0.978252      1452
## TTGTGTTTCCCGAAAG-1           1          NA   spot_1184  0.956778      3831
## TTGTTGTGTGTCAAGA-1           1         Mid   spot_1185  0.852160     27755
##                     detected subsets_mito_sum subsets_mito_detected
##                    <integer>        <numeric>             <integer>
## AAACAAGTATCTCCCA-1      2933             1021                    12
## AAACATTTCCCGGATT-1      1054              285                    12
## AAACCCGAACGAAATC-1      4421             2087                    12
## AAACGAGACGGTTGAT-1      5375              821                    12
## AAACTAACGTGGCGAC-1       150              182                    11
## ...                      ...              ...                   ...
## TTGTAATCCGTACTCG-1      2170              733                    11
## TTGTGAACCTAATCCG-1      1446              515                    12
## TTGTGCAGCCACGTCA-1       712               54                    10
## TTGTGTTTCCCGAAAG-1      1415              422                    11
## TTGTTGTGTGTCAAGA-1      4840              906                    12
##                    subsets_mito_percent     total
##                               <numeric> <numeric>
## AAACAAGTATCTCCCA-1              7.59503     13443
## AAACATTTCCCGGATT-1             10.76284      2648
## AAACCCGAACGAAATC-1              7.52533     27733
## AAACGAGACGGTTGAT-1              2.48992     32973
## AAACTAACGTGGCGAC-1             45.50000       400
## ...                                 ...       ...
## TTGTAATCCGTACTCG-1              9.62953      7612
## TTGTGAACCTAATCCG-1             11.97953      4299
## TTGTGCAGCCACGTCA-1              3.71901      1452
## TTGTGTTTCCCGAAAG-1             11.01540      3831
## TTGTTGTGTGTCAAGA-1              3.26428     27755
```

```
rowData(sfe)
```

```
## DataFrame with 32738 rows and 18 columns
##                     gene_name              id       mean  detected     total
##                   <character>     <character>  <numeric> <numeric> <numeric>
## ENSG00000243485   MIR1302-10 ENSG00000243485 0.00000000  0.000000         0
## ENSG00000237613      FAM138A ENSG00000237613 0.00000000  0.000000         0
## ENSG00000186092        OR4F5 ENSG00000186092 0.00000000  0.000000         0
## ENSG00000238009 RP11-34P13.7 ENSG00000238009 0.00590717  0.590717         7
## ENSG00000239945 RP11-34P13.8 ENSG00000239945 0.00000000  0.000000         0
```

```
## ...                     ...             ...         ...         ...         ...
## ENSG00000215635   AC145205.1 ENSG00000215635           0           0           0
## ENSG00000268590        BAGE5 ENSG00000268590           0           0           0
## ENSG00000251180   CU459201.1 ENSG00000251180           0           0           0
## ENSG00000215616   AC002321.2 ENSG00000215616           0           0           0
## ENSG00000215611   AC002321.1 ENSG00000215611           0           0           0
##                 JBO019.sparsity JBO019.total JBO019.nLocations JBO019.s_min
##                       <numeric>    <numeric>         <integer>    <numeric>
## ENSG00000243485        1.000000            0                 0          Inf
## ENSG00000237613        1.000000            0                 0          Inf
## ENSG00000186092        1.000000            0                 0          Inf
## ENSG00000238009        0.994093            7                 7            1
## ENSG00000239945        1.000000            0                 0          Inf
## ...                         ...          ...               ...          ...
## ENSG00000215635               1            0                 0          Inf
## ENSG00000268590               1            0                 0          Inf
## ENSG00000251180               1            0                 0          Inf
## ENSG00000215616               1            0                 0          Inf
## ENSG00000215611               1            0                 0          Inf
##                 JBO019.max JBO019.s_mean JBO019.s_median JBO019.s_SD
##                  <numeric>     <numeric>       <numeric>   <numeric>
## ENSG00000243485          0           NaN              NA          NA
## ENSG00000237613          0           NaN              NA          NA
## ENSG00000186092          0           NaN              NA          NA
## ENSG00000238009          1             1               1           0
## ENSG00000239945          0           NaN              NA          NA
## ...                    ...           ...             ...         ...
## ENSG00000215635          0           NaN              NA          NA
## ENSG00000268590          0           NaN              NA          NA
## ENSG00000251180          0           NaN              NA          NA
## ENSG00000215616          0           NaN              NA          NA
## ENSG00000215611          0           NaN              NA          NA
##                 JBO019.p_mean JBO019.p_median JBO019.p_SD JBO019.s_CV
##                     <numeric>       <numeric>   <numeric>   <numeric>
## ENSG00000243485   0.00000000               0   0.0000000          NA
## ENSG00000237613   0.00000000               0   0.0000000          NA
## ENSG00000186092   0.00000000               0   0.0000000          NA
## ENSG00000238009   0.00590717               0   0.0766631           0
## ENSG00000239945   0.00000000               0   0.0000000          NA
## ...                      ...             ...         ...         ...
## ENSG00000215635            0               0           0          NA
## ENSG00000268590            0               0           0          NA
## ENSG00000251180            0               0           0          NA
## ENSG00000215616            0               0           0          NA
## ENSG00000215611            0               0           0          NA
##                 JBO019.p_CV
```

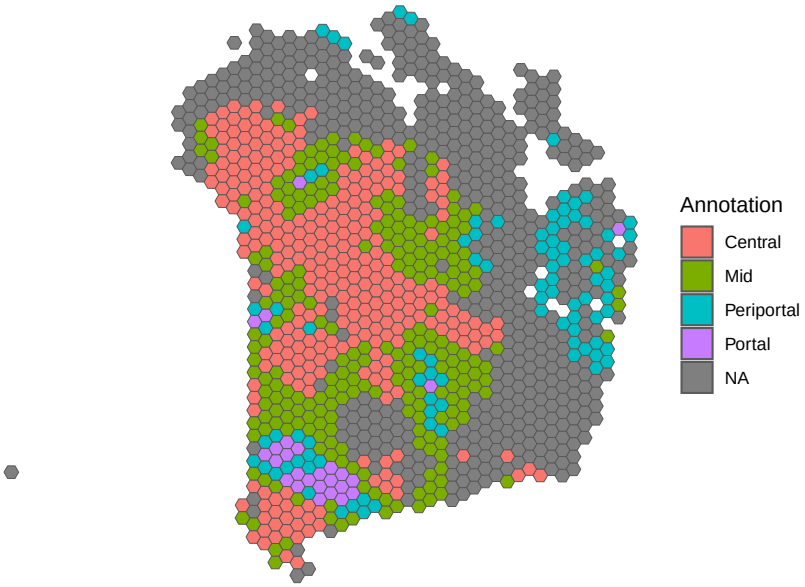
```
##                   <numeric>
## ENSG00000243485         NaN
## ENSG00000237613         NaN
## ENSG00000186092         NaN
## ENSG00000238009      1297.8
## ENSG00000239945         NaN
## ...                     ...
## ENSG00000215635         NaN
## ENSG00000268590         NaN
## ENSG00000251180         NaN
## ENSG00000215616         NaN
## ENSG00000215611         NaN
```

```
colGeometries(sfe)
```

```
## List of length 3
## names(3): spotPoly spotCntd spotHex
```

3.4.2 Plot manual annotation

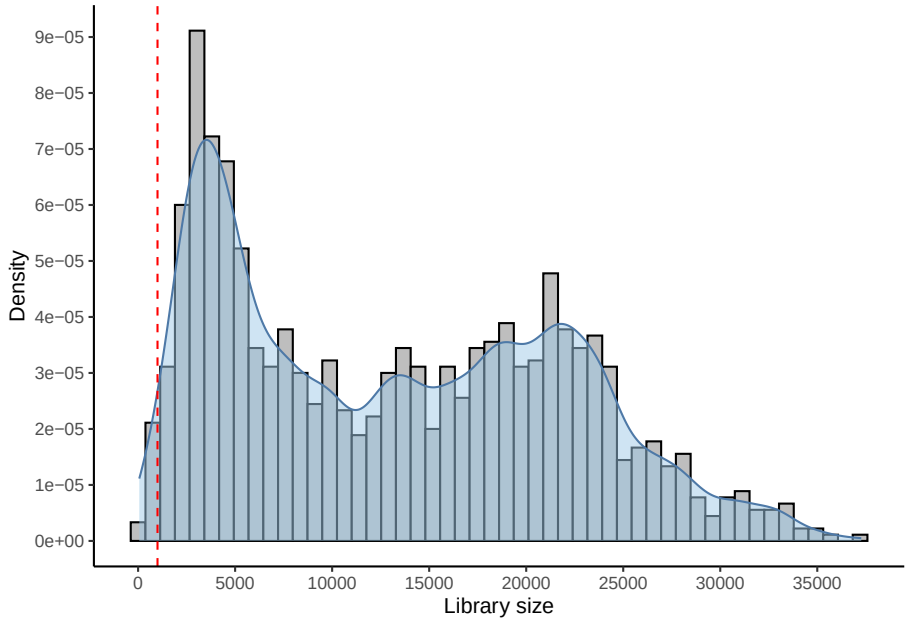
```
ggplot() +
  geom_sf(aes(geometry = colGeometries(sfe)$spotHex$geometry, fill = colData(sfe)$annotation)) +
  theme_void() +
  theme(legend.position = "right") +
  labs(fill = "Annotation")
```



3.4.3 Library size threshold

```
# ------------------------------------------------ #
## Density and histogram of library sizes
ggplot(data = as.data.frame(colData(sfe)),
       aes(x = sum)) +
    geom_histogram(aes(y = after_stat(density)),
                   colour = "black",
                   fill = "grey",
                   bins = 50) +
    geom_density(alpha = 0.5,
                 adjust = 0.5,
                 fill = "#A0CBE8",
                 colour = "#4E79A7") +
    geom_vline(xintercept = c(1000, NA),
               colour = "red",
               linetype = "dashed") +
    scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
    xlab("Library size") +
    ylab("Density") +
    theme_classic()
```

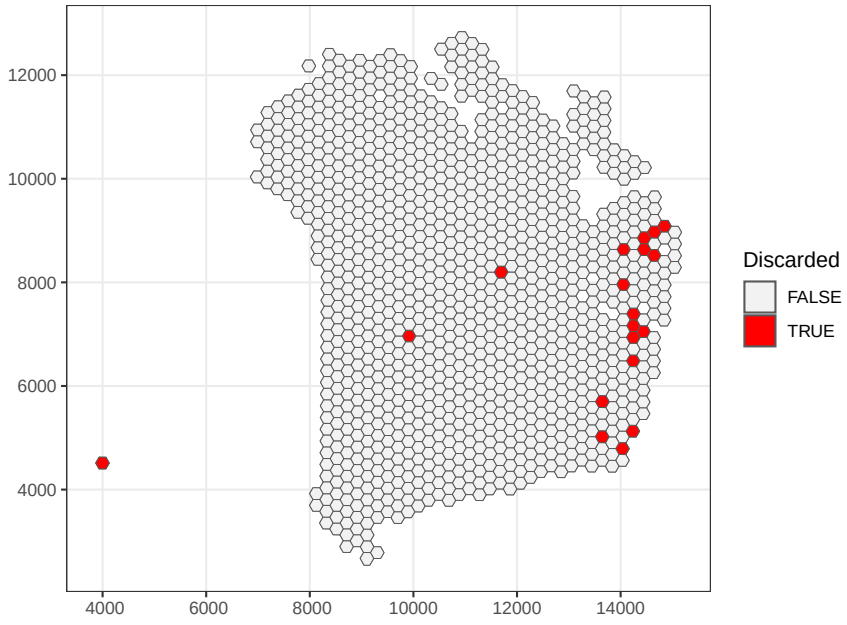
```
## Warning: Removed 1 rows containing missing values (`geom_vline()`).
```



```
## Select library size threshold
qc_lib_size <- colData(sfe)$sum < 1000 #| colData(sfe)$sum > 45000
## Check how many spots are filtered out
table(qc_lib_size)
```

```
## qc_lib_size
## FALSE  TRUE
##  1166    19
```

```
## Add threshold in colData
colData(sfe)$qc_lib_size <- qc_lib_size
## Check putative spatial patterns of removed spots
ggplot() +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry)) +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry, fill = colData(sfe)$qc_lib_size)) +
    scale_fill_manual(values = c("grey95", "red")) +
    labs(fill = "Discarded") +
    theme_bw()
```

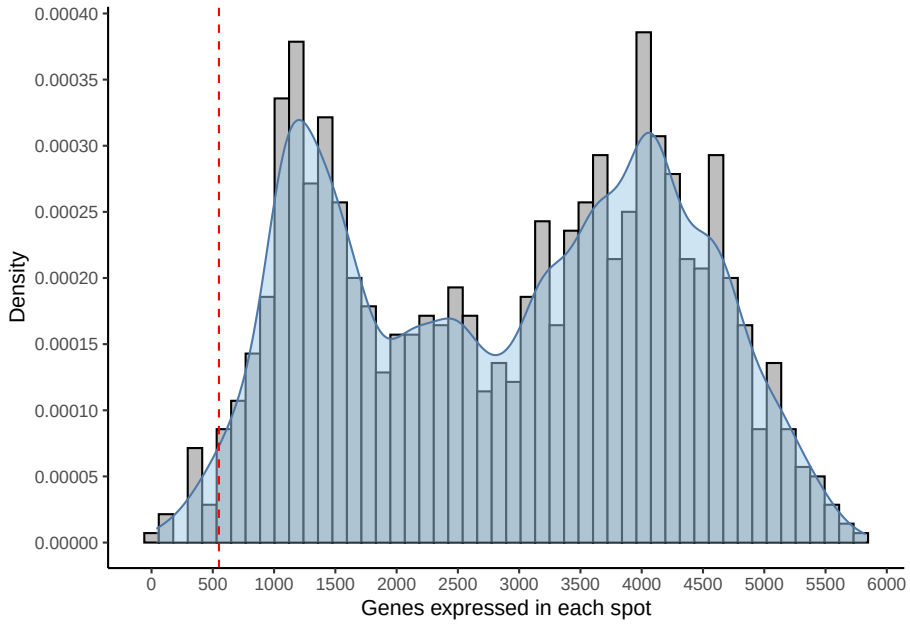


3.4.4 Number of expressed genes

```
# ------------------------------------------- #
## Density and histogram of expressed genes
```

```
ggplot(data = as.data.frame(colData(sfe)),
       aes(x = detected)) +
    geom_histogram(aes(y = after_stat(density)),
                   colour = "black",
                   fill = "grey",
                   bins = 50) +
    geom_density(alpha = 0.5,
                 adjust = 0.5,
                 fill = "#A0CBE8",
                 colour = "#4E79A7") +
    geom_vline(xintercept = c(550, NA),
               colour = "red",
               linetype = "dashed") +
    scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
    xlab("Genes expressed in each spot") +
    ylab("Density") +
    theme_classic()
```

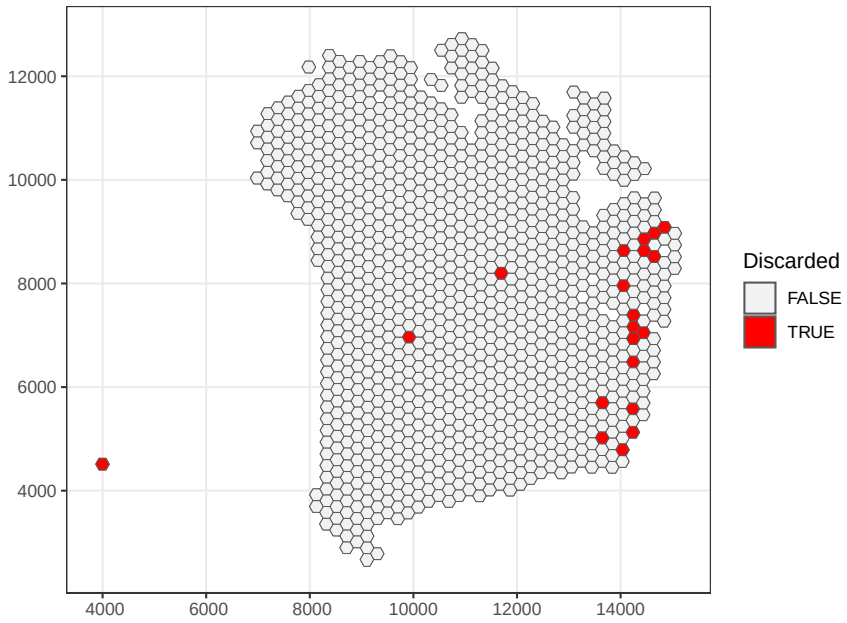
```
## Warning: Removed 1 rows containing missing values (`geom_vline()`).
```




```
## Select expressed genes threshold
qc_detected <- colData(sfe)$detected < 550 #| colData(sfe)$detected > 6000
## Check how many spots are filtered out
table(qc_detected)
```

```
## qc_detected
## FALSE  TRUE
##  1165    20
```

```
## Add threshold in colData
colData(sfe)$qc_detected <- qc_detected
## Check for putative spatial pattern of removed spots
ggplot() +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry)) +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry, fill = colData(sfe)$qc_detected)) +
    scale_fill_manual(values = c("grey95", "red")) +
    labs(fill = "Discarded") +
    theme_bw()
```

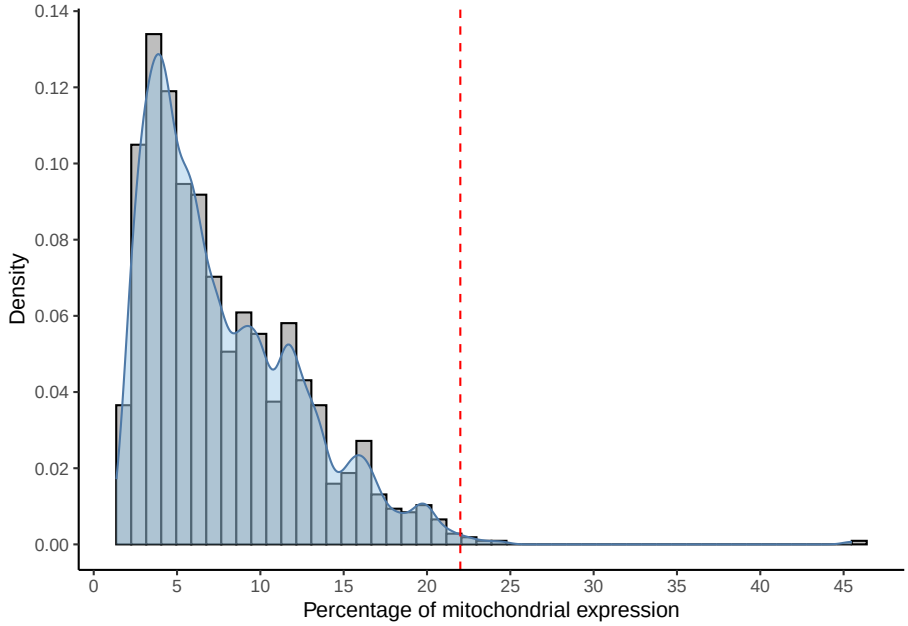


3.4.5 Percentage of mitochondrial expression

```
# ------------------------------------------------- #
## Density and histogram of percentage of mitochondrial expression
ggplot(data = as.data.frame(colData(sfe)),
       aes(x = subsets_mito_percent)) +
    geom_histogram(aes(y = after_stat(density)),
                   colour = "black",
                   fill = "grey",
```

```
                   bins = 50) +
    geom_density(alpha = 0.5,
                 adjust = 0.5,
                 fill = "#A0CBE8",
                 colour = "#4E79A7") +
    geom_vline(xintercept = c(22, NA),
               colour = "red",
               linetype = "dashed") +
    scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
    xlab("Percentage of mitochondrial expression") +
    ylab("Density") +
    theme_classic()
```

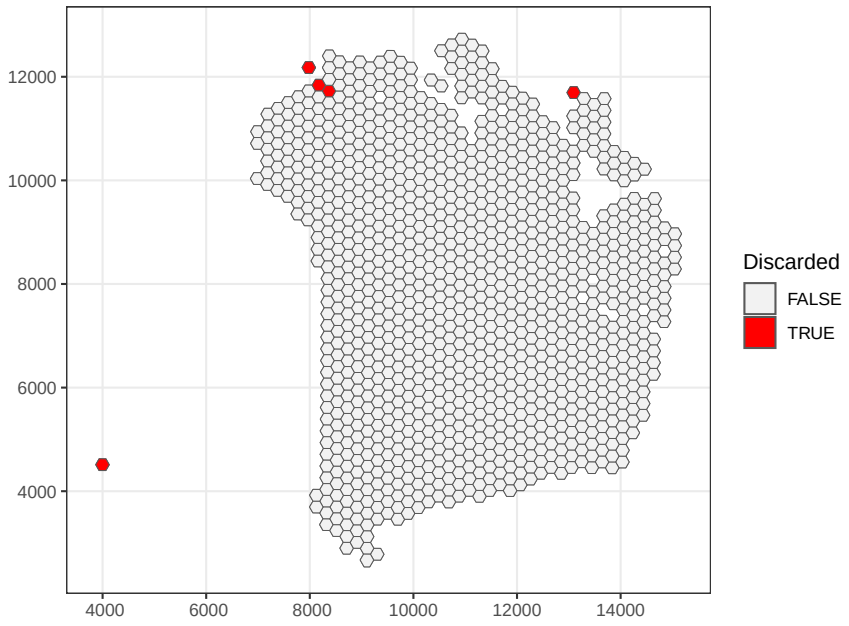
```
## Warning: Removed 1 rows containing missing values (`geom_vline()`).
```




```
## Select mitochondrial percentage threshold
qc_mito <- colData(sfe)$subsets_mito_percent > 22
## Check how many spots are filtered out
table(qc_mito)
```

```
## qc_mito
## FALSE  TRUE
##  1180     5
```

```
## Add threshold in colData
colData(sfe)$qc_mito <- qc_mito
## Check for putative spatial pattern of removed spots
ggplot() +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry)) +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry, fill = colData(sfe)$qc_mito)) +
    scale_fill_manual(values = c("grey95", "red")) +
    labs(fill = "Discarded") +
    theme_bw()
```



3.4.6 Remove low-quality spots

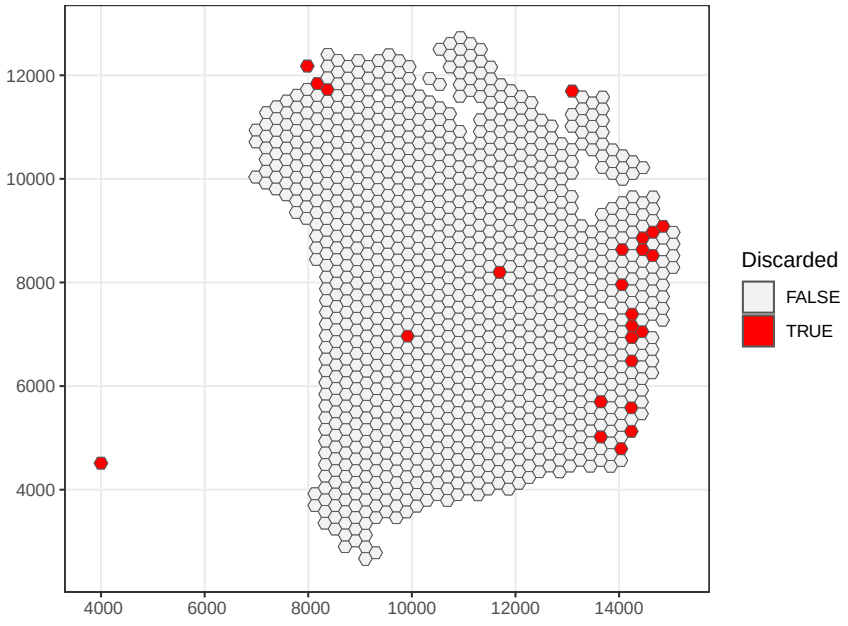
```
# ------------------------------------------------ #
## Check the number of discarded spots for each metric
apply(cbind(qc_lib_size, qc_detected, qc_mito), 2, sum)
```

```
## qc_lib_size qc_detected     qc_mito
##          19          20           5
```

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

```
## discard
## FALSE  TRUE
##  1161    24
```

```
## Store the set in the object
colData(sfe)$discard <- discard
## Check for putative spatial pattern of removed spots
ggplot() +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry)) +
    geom_sf(data = colGeometry(sfe, "spotHex"),
            aes(geometry = geometry, fill = colData(sfe)$discard)) +
    scale_fill_manual(values = c("grey95", "red")) +
    labels(fill = "Discarded") +
    theme_bw()
```




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

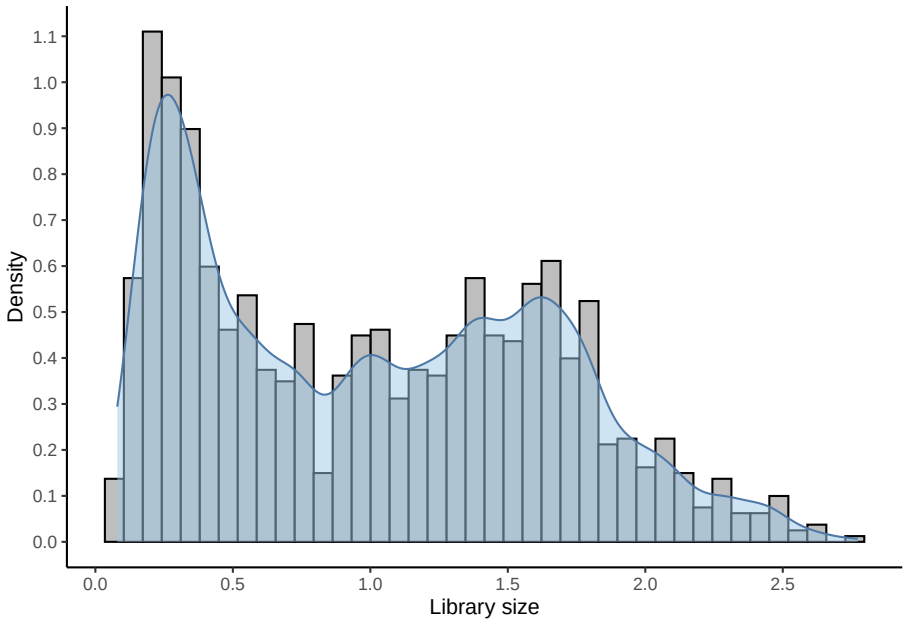
3.5 Normalisation of counts

```
## Calculate library size factors
sfe <- computeLibraryFactors(sfe)
## Have a look at the size factors
summary(sizeFactors(sfe))
```

3.5.1 Log-tranformation of counts

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.07961 0.36902 0.95469 1.00000 1.54936 2.77256
```

```
## Density and histogram of library sizes
ggplot(data = data.frame(sFact = sizeFactors(sfe)),
       aes(x = sFact)) +
    geom_histogram(aes(y = after_stat(density)),
                   colour = "black",
                   fill = "grey",
                   bins = 40) +
    geom_density(alpha = 0.5,
                 adjust = 0.5,
                 fill = "#A0CBE8",
                 colour = "#4E79A7") +
    scale_x_continuous(breaks = scales::pretty_breaks(n = 10)) +
    scale_y_continuous(breaks = scales::pretty_breaks(n = 10)) +
    xlab("Library size") +
    ylab("Density") +
    theme_classic()
```




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

3.6 Gene-level Quality Control

3.6.1 Calculating extra QC metrics

```
rowData(sfe)[["JBO019.s_logMean"]] <- rowSums(assay(sfe, "logcounts")) / rowData(sfe)[
```

3.6.2 Set and apply filters

```
is_zero <- rowData(sfe)$total == 0
is_logLow <- rowData(sfe)[["JBO019.s_logMean"]] <= 1
discard_gs <- is_zero | is_mito | is_logLow
table(discard_gs)
```

```
## discard_gs
## FALSE  TRUE
##  8535 24203
```

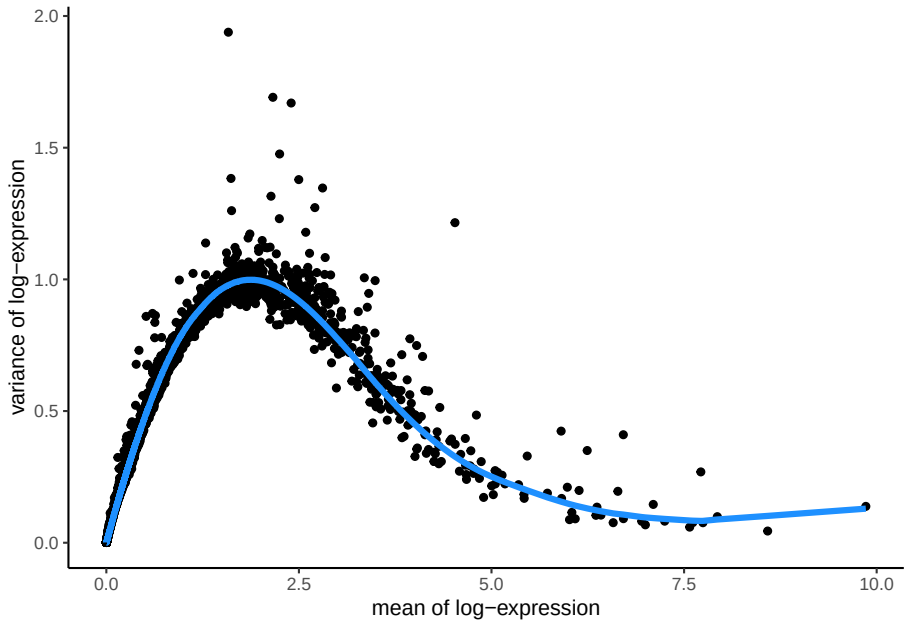
```
rowData(sfe)$discard <- discard_gs

## FEATURE SELECTION
## remove mitochondrial and other genes
```

```
sfe <- sfe[!rowData(sfe)$discard, ]
```

3.7 Selecting genes

3.7.1 Highly Variable Genes (HVGs)

```
## Fit mean-variance relationship
dec <- modelGeneVar(sfe,
                    assay.type = "logcounts")

## Visualize mean-variance relationship
fit <- metadata(dec)
fit_df <- data.frame(mean = fit$mean,
                     var = fit$var,
                     trend = fit$trend(fit$mean))

ggplot(data = fit_df,
       aes(x = mean, y = var)) +
    geom_point() +
    geom_line(aes(y = trend), colour = "dodgerblue", linewidth = 1.5) +
    labs(x = "mean of log-expression",
         y = "variance of log-expression") +
    theme_classic()
```



```
## Select top HVGs
top_hvgs <- getTopHVGs(dec,
                       var.field = "bio",
                       prop = 0.5,
                       var.threshold = 0,
                       fdr.threshold = 0.1)
```

3.7.2 Code for 3.3 to 3.7

```
## Import data
sampleDir <- "./data/spaceranger_outs/Human_Liver_Steatotic/JBO019_Results"
sampleNames <- "JBO019"
sfe <- read10xVisiumSFE(samples = sampleDir,
                        sample_id = sampleNames,
                        type = "sparse",
                        data = "filtered",
                        images = "lowres",
                        style = "W",
                        zero.policy = TRUE)
# ------------------------------------------------ #
ground_truth <- read_table("./data/to_load/spotzonationGroup.txt")
## Add QC metrics
is_mito <- grepl("(^MT-)|(^mt-)", rowData(sfe)$symbol)
sfe <- addPerLocQC(sfe, gTruth = ground_truth, assay = "counts", 2, subsets = list(mit
sfe <- addGeometries(sfe, samples = sampleDir, sample_id = sampleNames, res = "fullres
sfe <- addPerGeneQC(sfe, assay = "counts", version = NULL, mirror = NULL)
# ------------------------------------------------ #
## SPOT SELECTION
## Select library size threshold
qc_lib_size <- colData(sfe)$sum < 1000
## Add threshold in colData
colData(sfe)$qc_lib_size <- qc_lib_size
## Select expressed genes threshold
qc_detected <- colData(sfe)$detected < 550
## Add threshold in colData
colData(sfe)$qc_detected <- qc_detected
## Select mitochondrial percentage threshold
qc_mito <- colData(sfe)$subsets_mito_percent > 22
## Add threshold in colData
colData(sfe)$qc_mito <- qc_mito
## Combine together the set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito
## Store the set in the object
colData(sfe)$discard <- discard
## Remove combined set of low-quality spots
```

```
sfe <- sfe[, !colData(sfe)$discard]
# ------------------------------------------------ #
## FEATURE SELECTION
## Calculate library size factors
sfe <- computeLibraryFactors(sfe)
## Calculate logcounts using library size factors
sfe <- logNormCounts(sfe)
## Calculate log-counts sample mean
rowData(sfe)[["JBO019.s_logMean"]] <- rowSums(assay(sfe, "logcounts")) / rowData(sfe)[["JBO019.nI
## Set and apply filters
is_zero <- rowData(sfe)$total == 0
is_logLow <- rowData(sfe)[["JBO019.s_logMean"]] <= 1
discard_gs <- is_zero | is_mito | is_logLow
rowData(sfe)$discard <- discard_gs
## Remove mitochondrial and other genes
sfe <- sfe[!rowData(sfe)$discard, ]
## Fit mean-variance relationship
dec <- modelGeneVar(sfe,
                    assay.type = "logcounts")
## Select top HVGs
top_hvgs <- getTopHVGs(dec,
                       var.field = "bio",
                       prop = 0.5,
                       var.threshold = 0,
                       fdr.threshold = 0.05)
```

3.8 Neighbour graph and distance matrix

3.8.1 Adding spatial weights

The neighbour lists can be supplemented with spatial weights using the `nb2listw` and `nb2listwdist` function from `spdep` package for the chosen type and coding scheme style. There are 6 different coding scheme styles that can be used to weigh neighbour relationships:

1. **B**: is the basic binary coding (1 for neighbour, 0 for no neighbour).
2. **W**: is row standardised (sums over all links to n).
3. **C**: is globally standardised (sums over all links to n).
4. **U**: is equal to C divided by the number of neighbours (sums over all links to unity).
5. **S**: is the variance-stabilizing coding scheme (sums over all links to n).
6. **minmax**: divides the weights by the minimum of the maximum row sums and maximum column sums of the input weights; It is similar to the C and U styles.

The coding scheme style is practically the value each neighbour will get. For

example, in a binary coding scheme style (**B**) if a spot is a neighbour of the spot in focus then gets the value of **1**, else gets **0**. Another example, in a row standardised coding scheme style (**W**) if the spot in focus has a total of 10 neighbours and each neighbour has a weight of 1, then the sum of all neighbour weights is 10, and each neighbour will get a normalised weight of $1/10 = 0.1$. As a result, in the row standardised coding scheme, spots with many neighbours will have neighbours with lower weights and thus will not be over-emphasised.

Starting from a binary neighbours list, in which regions are either listed as neighbours or are absent (thus not in the set of neighbours for some definition), we can add a distance-based weights list. The `nb2listwdist` function supplements a neighbours list with spatial weights for the chosen types of distance modelling and coding scheme. While the offered coding schemes parallel those of the `nb2listw` function above, three distance-based types of weights are available: inverse distance weighting (IDW), double-power distance weights (DPD), and exponential distance decay (EXP). The three types of distance weight calculations are based on pairwise distances , all of which are controlled by parameter *"alpha"* (below):

1. **idw**: = − ,
2. **exp**: =exp(−),
3. **dpd**: =[1−(/ max)] ,

the latter of which leads to =0 for all > max. Note that *IDW* weights show extreme behaviour close to 0 and can take on the value infinity. In such cases, the infinite values are replaced by the largest finite weight present in the weights list.

3.8.2 Generate distance matrices

A distance matrix is a mirrored matrix that contains the distance between a spot and every other spot. This distance can be a simple Euclidean distance based on the coordinates of the spots or a weighted distance according to a bandwidth around each spot using a kernel that gives higher scores to distances between spots that are closer together compared to the ones that are farther away. These weighted distance matrices are later used to run geographically weighted (GW) models.

There are 6 different kernels that can be used to weight the distances between spots. The next two figures are from the `GWmodel` publication [Gollini et al., 2015] and illustrate the mathematical application of these kernals, and show graphically how they weight by distance.

In the below we choose one of the many possible ways of building a neighbour graph for the steatotic liver data set. In this example we are using a k-nearest neighbours approach with row-standardised distance-based weights.

```
## add a neighbour graph using a weighted distance matrix
sfe <- addSpatialNeighGraphs(sfe, "JBO019", type = "knearneigh", style = "W", distMod =
```

Global Model	$w_{ij} = 1$
Gaussian	$w_{ij} = \exp\left(-\frac{1}{2}\left(\frac{d_{ij}}{b}\right)^2\right)$
Exponential	$w_{ij} = \exp\left(-\frac{\lvert d_{ij}\rvert}{b}\right)$
Box-car	$w_{ij} = \begin{cases} 1 & \text{if } \lvert d_{ij}\rvert < b, \\ 0 & \text{otherwise} \end{cases}$
Bi-square	$w_{ij} = \begin{cases} (1-(d_{ij}/b)^2)^2 & \text{if } \lvert d_{ij}\rvert < b, \\ 0 & \text{otherwise} \end{cases}$
Tri-cube	$w_{ij} = \begin{cases} (1-(\lvert d_{ij}\rvert/b)^3)^3 & \text{if } \lvert d_{ij}\rvert < b, \\ 0 & \text{otherwise} \end{cases}$

Table 1: Six kernel functions; w_{ij} is the j-th element of the diagonal of the matrix of geographical weights $W(u_i, v_i)$, and d_{ij} is the distance between observations i and j, and b is the bandwidth.

Figure 3.2: The math equations that define the kernels.

```
colGraphs(sfe)
```

```
## $col
## Characteristics of weights list object:
## Neighbour list object:
## Number of regions: 1161
## Number of nonzero links: 6966
## Percentage nonzero weights: 0.5167959
## Average number of links: 6
## Non-symmetric neighbours list
##
## Weights style: W
## Weights constants summary:
##      n      nn   S0       S1       S2
## W 1161 1347921 1161 376.8333 4674.667
```

```
## Calculate a simple distance matrix
sfe <- addDistMat(sfe, p = 2)
```

We can use a `geom` from the `tidyterra` package (commonly used for map visualisations) to plot the neighbour graph we generated in the previous step.

```
## Retrieve the tissue image
sfei <- getImg(sfe, image_id = "lowres")
## Extract the spot locations
spot_coords <- spatialCoords(sfe) %>% as.data.frame()

## Set limits
xlim <- c(min(spot_coords$pxl_col_in_fullres) - 100,
          max(spot_coords$pxl_col_in_fullres) + 100)
```

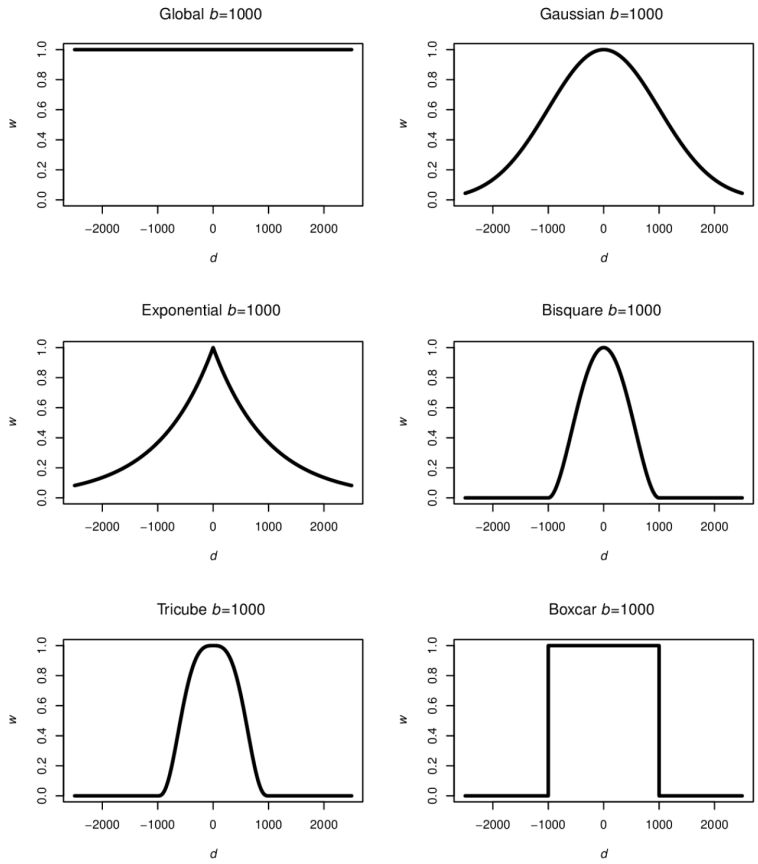
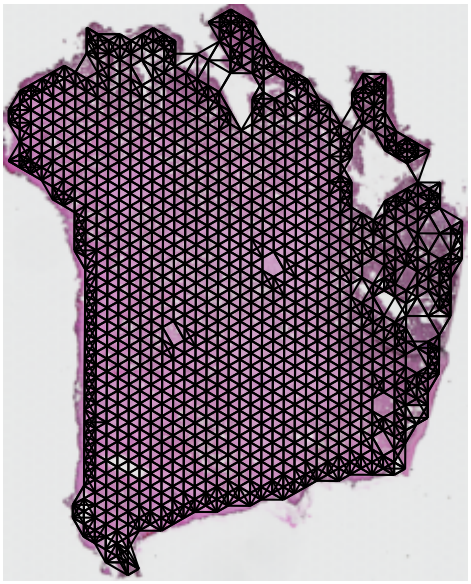


Figure 1: Plot of the six kernel functions, with the bandwidth $b = 1000$, and where w is the weight, and d is the distance between two observations.

Figure 3.3: Examples from using each kernel.

```
ylim <- c(min(spot_coords$pxl_row_in_fullres) - 100,
          max(spot_coords$pxl_row_in_fullres) + 100)
nbs <- colGraph(sfe)
ggplot() +
    geom_spatraster_rgb(data = imgRaster(sfei)) +
    geom_sf(data = as(nb2lines(nbs$neighbours, coords = spatialCoords(sfe)), "sf")) +
    lims(x = xlim, y = ylim) +
    coord_sf() +
    theme_void()
```

```
## Warning in CRS(proj4string): CRS: projargs should not be NULL; set to NA
```



Now that we have a fully QC-ed dataset with spatial weights and a neighbour graph applied, we have prepared our data fully for the application of geospatial methods - specifically in practical 4, geogrpahically weighted principal components analysis (GWPCA).

3.9 Putting it all together

The below code puts all these steps in order by selecting one of the options at each step.

```
## Import data
sampleDir <- "./data/spaceranger_outs/Human_Liver_Steatotic/JBO019_Results"
sampleNames <- "JBO019"
```

```
sfe <- read10xVisiumSFE(samples = sampleDir,
                        sample_id = sampleNames,
                        type = "sparse",
                        data = "filtered",
                        images = "lowres",
                        style = "W",
                        zero.policy = TRUE)
# ------------------------------------------------ #
ground_truth <- read_table("./data/to_load/spotzonationGroup.txt")
## Add QC metrics
is_mito <- grepl("(^MT-)|(^mt-)", rowData(sfe)$symbol)
sfe <- addPerLocQC(sfe, gTruth = ground_truth, assay = "counts", 2, subsets = list(mit
sfe <- addGeometries(sfe, samples = sampleDir, sample_id = sampleNames, res = "fullres
sfe <- addPerGeneQC(sfe, assay = "counts", version = NULL, mirror = NULL)
# ------------------------------------------------ #
## SPOT SELECTION
## Select library size threshold
qc_lib_size <- colData(sfe)$sum < 1000
## Add threshold in colData
colData(sfe)$qc_lib_size <- qc_lib_size
## Select expressed genes threshold
qc_detected <- colData(sfe)$detected < 550
## Add threshold in colData
colData(sfe)$qc_detected <- qc_detected
## Select mitochondrial percentage threshold
qc_mito <- colData(sfe)$subsets_mito_percent > 22
## Add threshold in colData
colData(sfe)$qc_mito <- qc_mito
## Combine together the set of discarded spots
discard <- qc_lib_size | qc_detected | qc_mito
## Store the set in the object
colData(sfe)$discard <- discard
## Remove combined set of low-quality spots
sfe <- sfe[, !colData(sfe)$discard]
# ------------------------------------------------ #
## FEATURE SELECTION
## Calculate library size factors
sfe <- computeLibraryFactors(sfe)
## Calculate logcounts using library size factors
sfe <- logNormCounts(sfe)
## Calculate log-counts sample mean
rowData(sfe)[["JBO019.s_logMean"]] <- rowSums(assay(sfe, "logcounts")) / rowData(sfe)[
## Set and apply filters
is_zero <- rowData(sfe)$total == 0
is_logLow <- rowData(sfe)[["JBO019.s_logMean"]] <= 1
```

```
discard_gs <- is_zero | is_mito | is_logLow
rowData(sfe)$discard <- discard_gs
## Remove mitochondrial and other genes
sfe <- sfe[!rowData(sfe)$discard, ]
## Fit mean-variance relationship
dec <- modelGeneVar(sfe,
                    assay.type = "logcounts")
## Select top HVGs
top_hvgs <- getTopHVGs(dec,
                       var.field = "bio",
                       prop = 0.5,
                       var.threshold = 0,
                       fdr.threshold = 0.05)
# ------------------------------------------------ #
## ADD GEOGRAPHY
## Add a neighbour graph using a weighted distance matrix
sfe <- addSpatialNeighGraphs(sfe, "JBO019", type = "knearneigh", style = "W", distMod = "raw", k
## Calculate a simple distance matrix
sfe <- addDistMat(sfe, p = 2)
```

Chapter 4

Practical session 4

In this session we will have a hands-on exploration of GW-PCA and its application to STx data. What can we learn from this novel technique?

4.1 Geographically Weighted Principal Components Analysis (GWPCA)

A standard PCA can pick out the key multivariate modes of variability in the data. Looking at outlying values of the principal components of these data gives us an idea of unusual sites (in terms of combinations of gene expression profiles - and to a certain extent of combinations of cell types in each spot). Next, geographically weighted PCA can be used to find spatial multivariate outliers. Sounds complicated, but really all this means is it identifies sites that have an unusual multi-way combination of gene expression in relation to their immediate geographical neighbours. It might be that the values observed at these sites as a combination is not uncommon in the tissue as a whole - but is very unusual in its locality.

To find such outliers the procedure is relatively simple - instead of doing a PCA on the tissue as a whole, for each sample we do a PCA on data falling into a window centred on the location of that spot. In that way we can check whether the spot is like its neighbours or not, from a multivariate viewpoint.

The procedure we will follow in this practical carries out a geographically weighted PCA. In short, it runs a 'windowed' PCA around each of the spots.

4.2 Load packages

4.3 Load Quality Controled and Normalised data

First of all, we need to load the data we prepared in the previous practical.

```
sfe <- readRDS(file = "./data/to_load/practical03_sfe.rds")
top_hvgs <- readRDS(file = "./data/to_load/practical03_topHVGs.rds")
```

4.4 Parameter prearation for GWPCA

The `gwpca` method uses `princomp` internally to run the PCAs - this function does not allow the number of variables (genes) to be greater than the number of samples (spots). This imposes a hard requirement on the data pre-processing. We have, however, already identified the highly variable genes in our sample, and for this case, there are fewer genes than spots.

Some other parameterisation is neccessary and these required parameters (as we have used for this dataset) are illustrated here:

```
## Get the gene names that are going to be evaluated
vars = top_hvgs
## Set a fixed bandwidth
bw = 6*sfe@metadata[["spotDiameter"]][["JBO019"]][["spot_diameter_fullres"]]
## Set the number of components to be retained
k = 20
## Set the kernel to be used
kernel = "gaussian"
## Set the Minkowski distance power: p = 2 --> Euclidean
p = 2
## Is the bandwidth adaptive?: No because spots are fixed
adaptive = FALSE
## Cross-Validate GWPCA?
cv = TRUE
## Calculate PCA scores?
scores = FALSE
## Run a robust GWPCA?
robust = FALSE
## Make a cluster for parallel computing (otherwise GWPCA is slow!)
my.cl <- parallel::makeCluster(parallelly::availableCores() - 1, type = 'FORK')
```

The bandwidth defines a radius around each spot - every spot that falls inside this radius is considered a neighbour. We can set bandwidth as a fixed value (as here) or we can select the bandwidth automatically. Without going into detail here, this is achieved by a form of cross validation, where each observation is

omitted, and it is attempted to reconstruct the values on the basis of principal components, derived from the other observations. The bandwidth achieving the optimal results is the one selected. For a complete explanation, see Harris et al. [2011]. The function `bw.gwpca` from `GWmodel` can be used to computes this.

- **NOTE**: Larger bandwidths imply bigger moving spatial windows, which in turn imply smoother spatially varying outputs.

4.5 Run GWPCA

Here we present the invocation to run GWPCA, however because this process is computationally intensive and time-consuming, we do not suggest running it on posit.cloud. We have pre-computed the result and provide it for you to load.

```
# DO NOT RUN THIS CHUNK

pcagw <- gwpcaSTE(sfe = sfe,
                  assay = "logcounts",
                  vars = vars,
                  p = p,
                  k = k,
                  bw = bw,
                  kernel = kernel,
                  adaptive = adaptive,
                  scores = scores,
                  robust = robust,
                  cv = cv,
                  future = FALSE,
                  strategy = "cluster",
                  workers = my.cl,
                  verbose = FALSE)
saveRDS(pcagw, file = "./data/to_load/practical04_pcagw.rds")
```

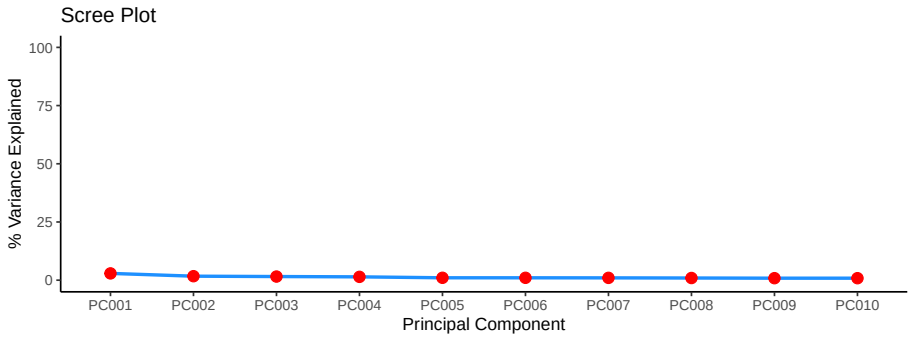
Because GWPCA can take some time to run, we ran it for you and below you can load the output:

```
pcagw <- readRDS(file = "./data/to_load/practical04_pcagw.rds")
```

4.6 Plot global PCA results

In the next steps we will take a look inside the output from the `gwpca` function and we are going to extract some basic information. Since GWPCA consists of multiple local PCAs, it is good to know how many PCs makes sense to look at. We can do so by running a global PCA and plotting a scree plot:

```
plotGWPCA_global(gwpca = pcagw,
                 comps = 1:10,
                 type = "scree",
                 point_args = list(size = 3, colour = "red"),
                 line_args = list(linewidth = 1, colour = "dodgerblue"))
```



In a Principal Component Analysis (PCA), the first three principal components may explain less than 15% of the variance in the data if the data is highly dispersed or if there is a large amount of noise in the data. This means that the first three principal components are not capturing a significant portion of the variability in the data. This could be due to a lack of clear structure in the data or a lack of meaningful patterns that can be captured by the PCA. Alternatively, it could be due to the presence of many irrelevant features or variables in the data that are not contributing to the overall variance. This is one more of the reasons why GWPCA is more appropriate for STx data. Because, it may be true that the global PCs are not strong but locally this can change.

4.7 Identify the leading genes in each location

The genes with the highest loading scores (where loading score = correlation between variable and component) at each location can be thought of as the "leading genes" - i.e. those with the most explanatory power with respect to the variability of gene expression at that location. These leading genes can be a local indicator of relevant biology.

Here we look at leading genes in 2 ways - (1) by finding the single gene with the highest loading at each location; (2) by finding sets of the top 4 genes by loading score, where the order of those genes does not matter (so the ordered set A,B,C,D is considered the same as D,B,A,C).

```
## Extract leading genes
pcagw <- gwpca_LeadingGene(gwpca = pcagw,
                           sfe = sfe,
                           pc_nos = 1:4,
```

```
                            type = "single",
                            names = "gene_names")
```

```
## 16  leading genes found for  PC1
## The leading genes in  PC1  are:
##     ADH1A        C7       CRP    CYP3A4      GLUL     GSTA2      HAMP      HBA2
##         2        11         4       365         7         1        13        33
##     IGLL5    MALAT1 MTRNR2L12  MTRNR2L8      NNMT     PTGDS      SAA1       SDS
##        87        39       153       181        23        73        36       133
## 21  leading genes found for  PC2
## The leading genes in  PC2  are:
##        C7       CAT     CFHR1       CRP    CYP3A4      GLUL      HBA2       HBB
##         3         6        38        39       149        83         2        37
##    IGFBP3    IGFBP7       IGJ     IGLL5    MALAT1 MTRNR2L10 MTRNR2L12  MTRNR2L8
##        49        39        34       246        80        10        78       124
##      NNMT      SAA1       SDS     TAGLN    UGT2B7
##        42        12        69        20         1
## 24  leading genes found for  PC3
## The leading genes in  PC3  are:
##     AEBP1        C7       CAT     CFHR1       CRP    CYP3A4      GLUL      HBA2
##         2         2        27        20         5        20        17        27
##       HBB    IGFBP3    IGFBP7       IGJ     IGLL5    MALAT1 MTRNR2L10 MTRNR2L12
##       150        41        77         6       399       136         6        61
##  MTRNR2L8      MYL9      NNMT      SAA1   SCGB3A1       SDS     TAGLN    UGT2B7
##        25         9        24         6        56        15        26         4
## 25  leading genes found for  PC4
## The leading genes in  PC4  are:
##     AEBP1       CAT     CFHR1       CRP     FXYD2      GLUL     GSTA2      HBA2
##         1        53        15         7         7        33         3         2
##       HBB    IGFBP3    IGFBP7       IGJ     IGLL5    MALAT1 MTRNR2L10 MTRNR2L12
##       181       100        51        60       281       201         5        16
##  MTRNR2L8      MYL9      NNMT      ORM2      SAA1       SDS    SPINK1     TAGLN
##        16         5        55         6         6        37        12         4
##    UGT2B7
##         4
```

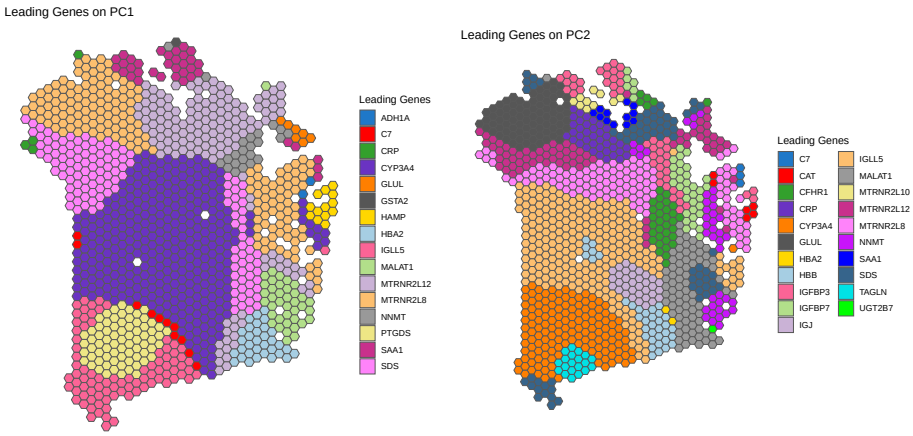
```
pcagw <- gwpca_LeadingGene(gwpca = pcagw,
                           sfe = sfe,
                           pc_nos = 1:4,
                           genes_n = 4,
                           type = "multi",
                           method = "membership",
                           names = "gene_names")
```

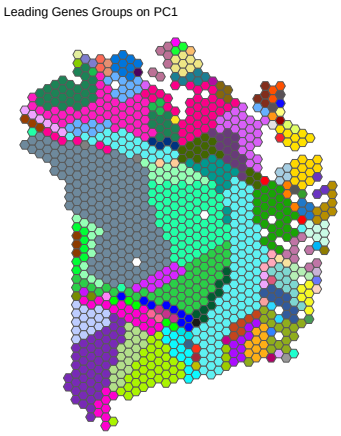
```
## The number of individual leading genes groups found for PC1 is: 110
## These groups are: Too many to print them!
```

```
## The number of individual leading genes groups found for PC2 is: 240
## These groups are: Too many to print them!
## The number of individual leading genes groups found for PC3 is: 310
## These groups are: Too many to print them!
## The number of individual leading genes groups found for PC4 is: 421
## These groups are: Too many to print them!
```

We can also plot these leading genes on the spot map - as each location by definition has (potentially) a different leading gene.

```
## Plot leading genes
plotGWPCA_leadingG(gwpca = pcagw,
                   comps = 1:2,
                   type = "single",
                   arrange = FALSE)

plotGWPCA_leadingG(gwpca = pcagw,
                   comps = 1,
                   type = "multi",
                   arrange = FALSE)
```

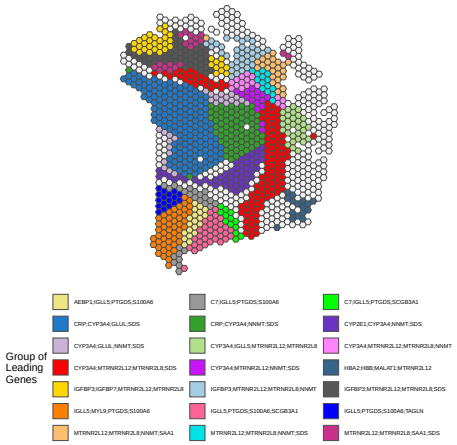




The "multi" plot here is problematic, because there are too many groups of genes to be able to print a legible legend. The alternative below is provided to highlight gene groups that are found in at least 12 spots.

```
### Plot multi type (alternative)
## The data
leadingGsMulti <- pcagw$leadingGeneMulti
## The Legend labels
spot_labels <- data.frame(table(leadingGsMulti[1])) %>%
    dplyr::rename(LeadingGs = colnames(leadingGsMulti)[1],
                  count = Freq) %>%
    dplyr::arrange(desc(count)) %>%
    mutate(show = ifelse(count > 12, TRUE, FALSE))

## The legend breaks:
spot_breaks <- spot_labels %>%
    dplyr::filter(show == TRUE) %>%
    dplyr::arrange(LeadingGs) %>%
    dplyr::select(LeadingGs) %>%
    .[["LeadingGs"]] %>%
    as.vector()

## The colours:
col_No <- sum(spot_labels$show)
colour_values <- getColours(col_No)
names(colour_values) <- spot_labels$LeadingGs[spot_labels$show]
pc <- "PC1"

## The Plot:
ggplot() +
    geom_sf(data = leadingGsMulti,
            aes(geometry = geometry$geometry,
```

```
                fill = .data[[pc]]),
            colour = "grey30",
            show.legend = TRUE) +
    scale_fill_manual(values = colour_values,
                      breaks = spot_breaks,
                      na.value = "gray95") +
    labs(title = NULL,
         fill = "Group of\nLeading\nGenes") +
    theme_void() +
    theme(legend.position = "bottom", legend.text = element_text(size=6)) +
    guides(fill = guide_legend(ncol = 3, byrow = TRUE))
```



4.8 Percentage of Total Variation (PTV)

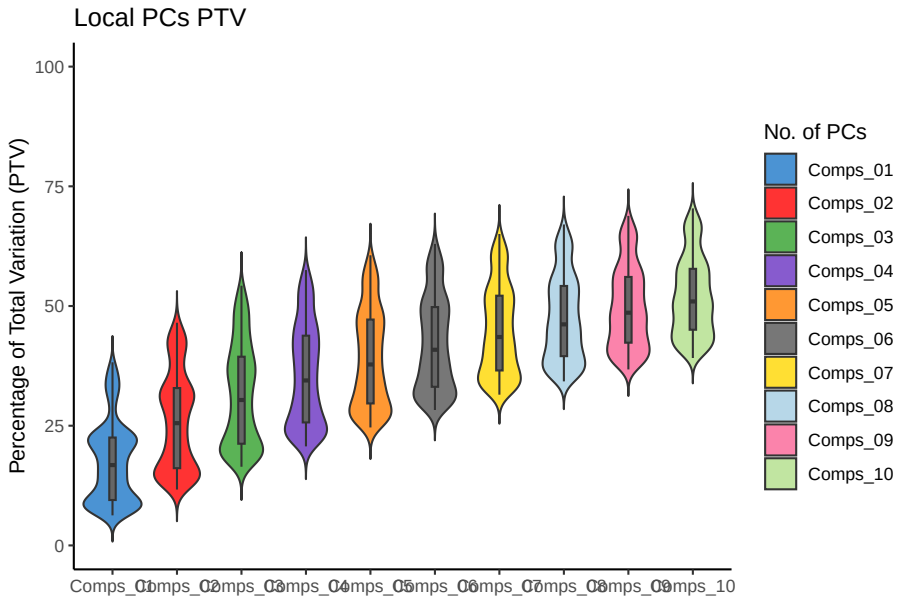
Another useful diagnostic for PCA is the percentage of variability in the data explained by each of the components. Locally, this can be achieved by looking at the `local.PV` component of `pcagw`; this is written as `pcagw$local.PV`. This is an 1161 by 20 matrix - where 1161 is the number of observations and 20 is the number of components (`k`). For each location, the 20 columns correspond to the percentage of the total variance explained by each of the principal components at that location. If, say, the first two components contributed 90% of the total variance, then it is reasonable to assume that much of the variability in the data can be seen by just looking at these two components. Because this is geographically weighted PCA, this quantity varies across the map.

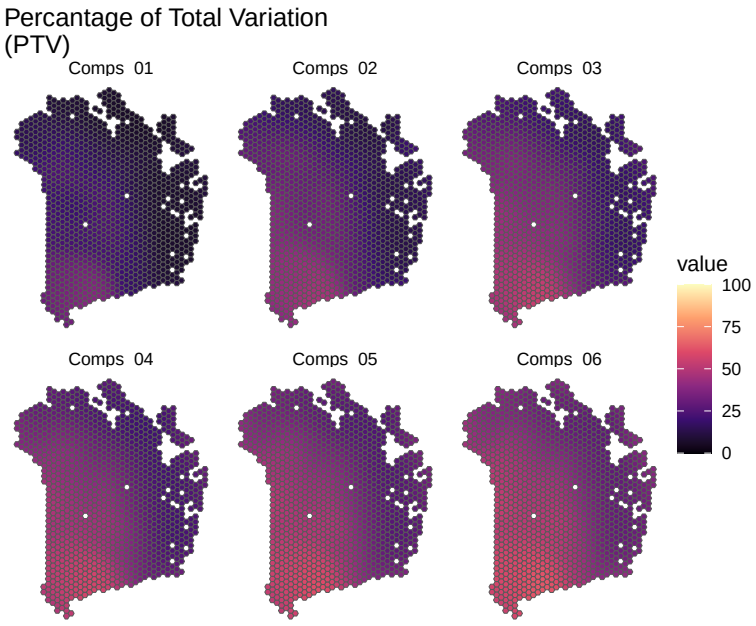
```
## Calculate the PTV for multiple Components
pcagw <- gwpca_PropVar(gwpca = pcagw, n_comp = 2:10, sfe = sfe)
```

```
##     Comps_01         Comps_02        Comps_03        Comps_04
##  Min.   : 6.279   Min.   :11.67   Min.   :16.43   Min.   :20.69
```

```
##  1st Qu.: 9.483   1st Qu.:16.13   1st Qu.:21.24   1st Qu.:25.69
##  Median :16.782   Median :25.54   Median :30.37   Median :34.46
##  Mean   :17.370   Mean   :25.92   Mean   :31.35   Mean   :35.49
##  3rd Qu.:22.534   3rd Qu.:32.87   3rd Qu.:39.42   3rd Qu.:43.81
##  Max.   :38.254   Max.   :46.50   Max.   :54.25   Max.   :57.51
##     Comps_05        Comps_06        Comps_07        Comps_08
##  Min.   :24.64   Min.   :28.28   Min.   :31.49   Min.   :34.26
##  1st Qu.:29.65   1st Qu.:33.13   1st Qu.:36.54   1st Qu.:39.53
##  Median :37.79   Median :40.86   Median :43.53   Median :46.17
##  Mean   :38.98   Mean   :42.07   Mean   :44.84   Mean   :47.38
##  3rd Qu.:47.17   3rd Qu.:49.78   3rd Qu.:52.16   3rd Qu.:54.19
##  Max.   :60.60   Max.   :62.97   Max.   :65.04   Max.   :67.03
##     Comps_09        Comps_10
##  Min.   :36.76   Min.   :39.15
##  1st Qu.:42.34   1st Qu.:45.05
##  Median :48.60   Median :50.96
##  Mean   :49.73   Mean   :51.91
##  3rd Qu.:56.07   3rd Qu.:57.77
##  Max.   :68.83   Max.   :70.39
```

```
## Plot PTV
plotGWPCA_ptv(gwpca = pcagw,
              comps = 1:10,
              type = "violin")

## Map PTV
plotGWPCA_ptv(gwpca = pcagw,
              comps = 1:6,
              type = "map")
```



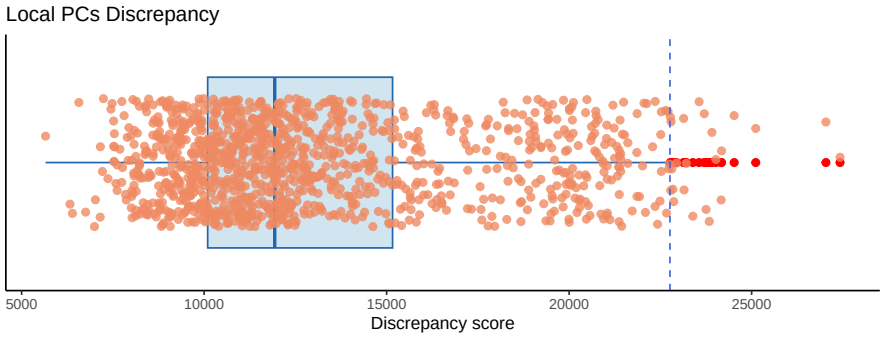


4.9 Identify discrepancies

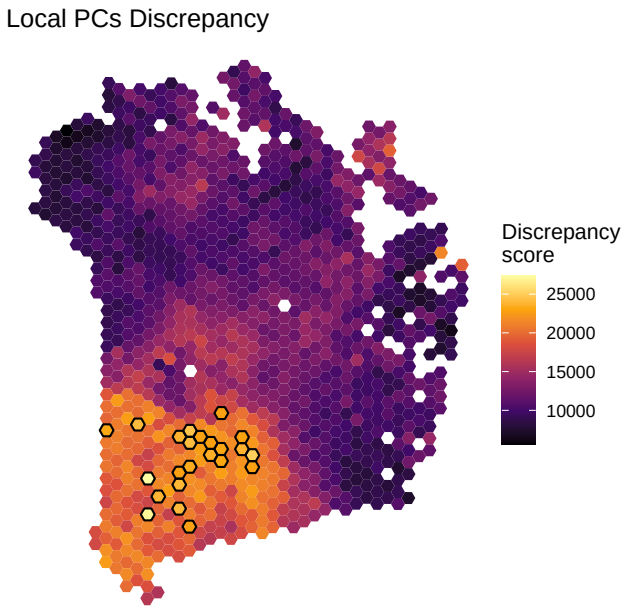
Global PCA can be used to identify multivariate outliers. Extending this, it is also possible to use local PCA (i.e., GWPCA) to identify local outliers. One

way of doing this links back to the cross-validation idea that can be used to select a bandwidth. Recall that this is based on a score of how well each observation can be reconstructed on the basis of local PCs. The score measures the total discrepancies of true data values from the reconstructed ones - and the bandwidth chosen is the one minimising this. However, the total discrepancy score is the sum of the individual discrepancies. A very large individual discrepancy associated with an observation suggests it is very different - in a multidimensional way, to the observations near to it.

```
## Plot the discrepancies as boxplot
plotGWPCA_discr(pcagw, type = "box")
```




```
## Plot the discrepancies map
plotGWPCA_discr(pcagw, type = "map")
```



```
## Get location data for the discrepancies
discrepancy_loc_dt <- getDiscrepancyLocData(sfe = sfe,
                                            gwpca = pcagw,
                                            sample_id = "JBO019")
```

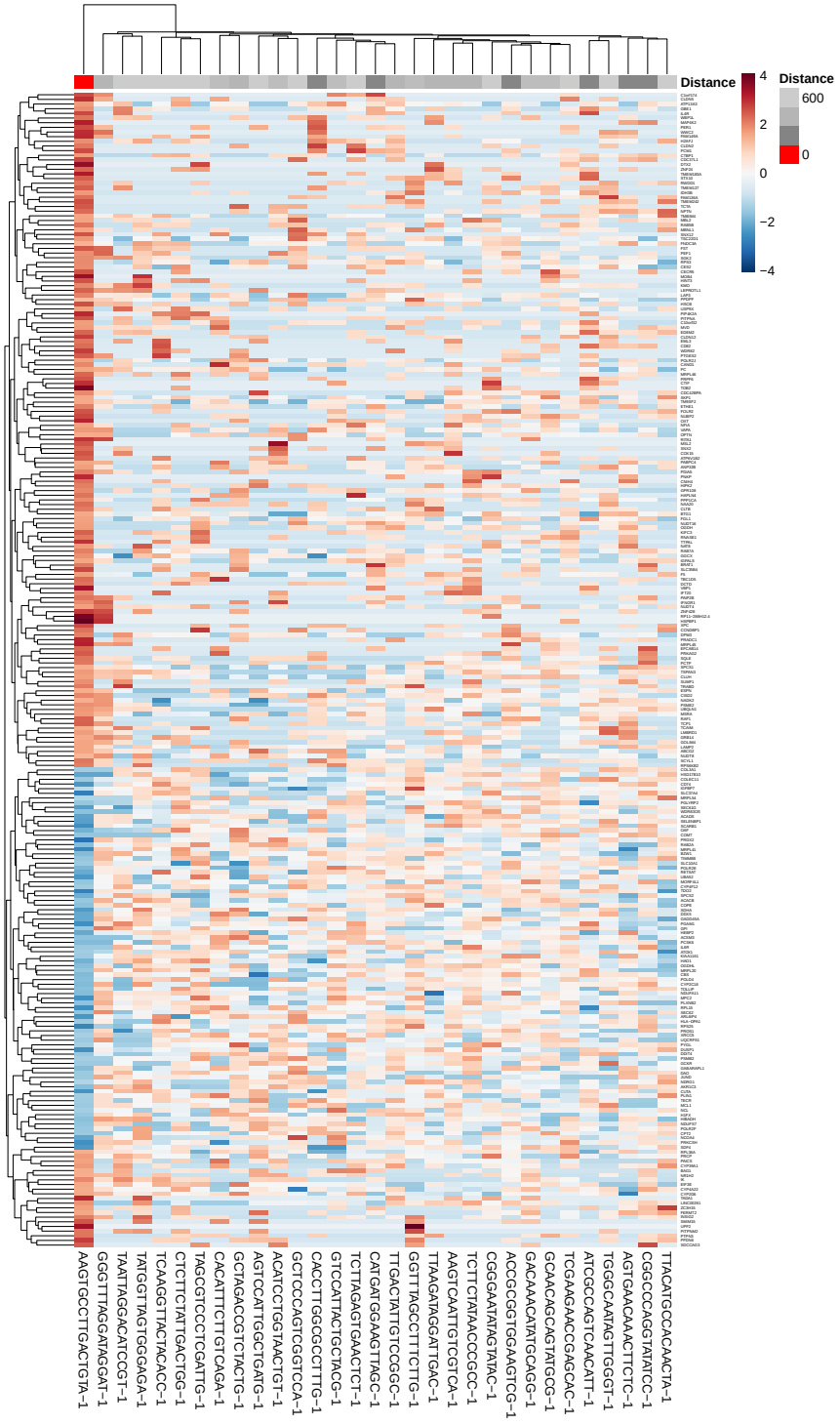
Another possibility to understand the nature of the outlier is a parallel coordinates heatmap. Here, each observation neighbouring the location that has been found to be an outlier is shown as a column with the genes in rows. Since here we are investigating local outliers, one particular observation is highlighted in red - the outlier - and the remaining ones in grey, but with the intensity of the grey fading according to their distance from the red observation. This enables you to see what characteristic the red observation has that means it as outlying from its neighbours. The plot can be created using `STExplorerDev::plotGWPCA_discrHeatmap`:

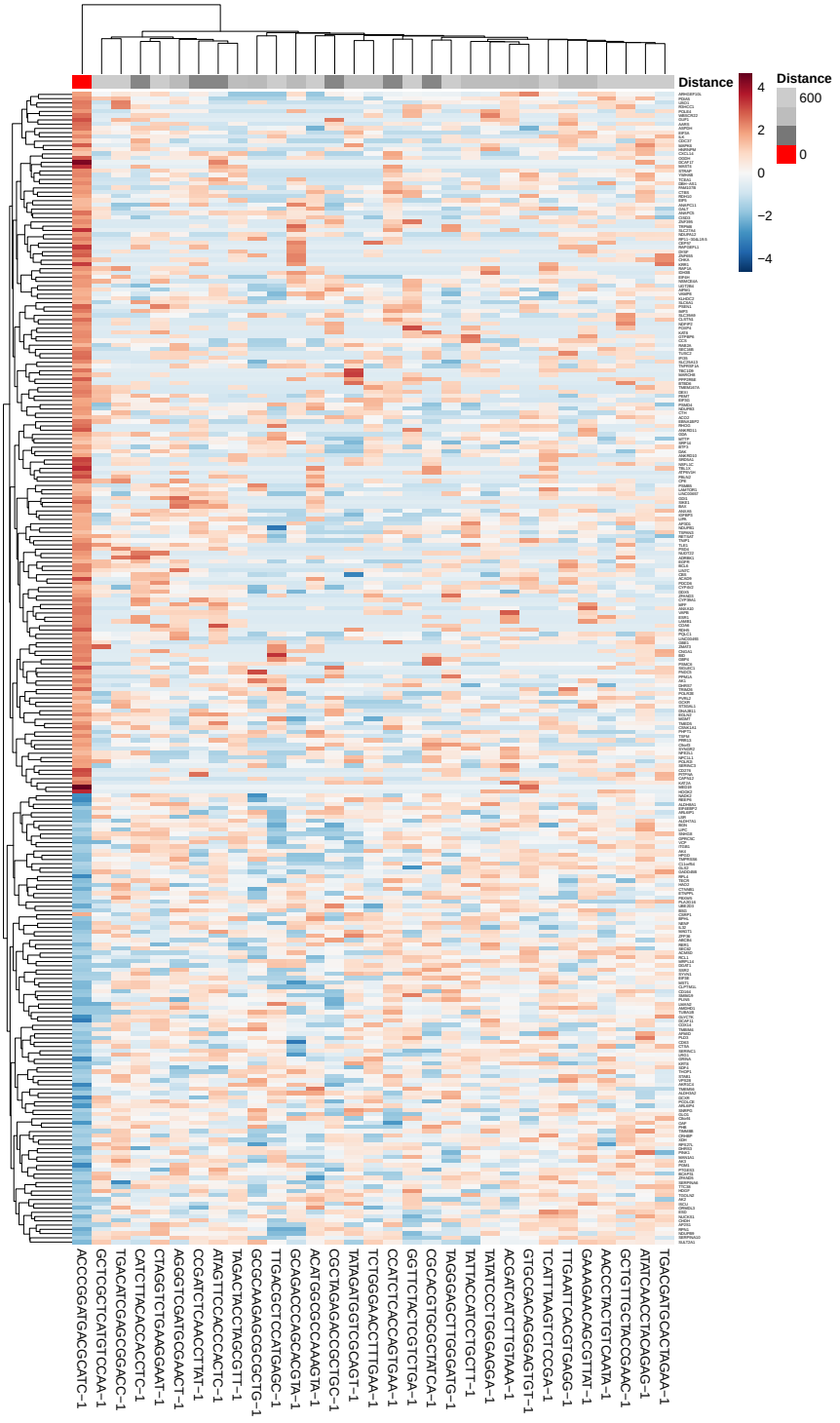
```
head(discrepancy_loc_dt)
```

```
##                                barcodes coords.pxl_col_in_fullres
## AAGTGCCTTGACTGTA-1 AAGTGCCTTGACTGTA-1                       11086
## ACCCGGATGACGCATC-1 ACCCGGATGACGCATC-1                        9908
## ACCTCCGTTATTCACC-1 ACCTCCGTTATTCACC-1                        9113
## AGATGATGGAGTCTGG-1 AGATGATGGAGTCTGG-1                        9117
## AGGTATAATTGATAGT-1 AGGTATAATTGATAGT-1                        9312
## AGTGAACAAACTTCTC-1 AGTGAACAAACTTCTC-1                       11088
##                    coords.pxl_row_in_fullres discScore
## AAGTGCCTTGACTGTA-1                      5148  22793.56
## ACCCGGATGACGCATC-1                      5607  24165.26
## ACCTCCGTTATTCACC-1                      4255  27035.77
## AGATGATGGAGTCTGG-1                      4933  27423.00
## AGGTATAATTGATAGT-1                      4593  23839.25
## AGTGAACAAACTTCTC-1                      5374  25111.79
##                                     geometry
## AAGTGCCTTGACTGTA-1 POLYGON ((11020.03 5034.788...
## ACCCGGATGACGCATC-1 POLYGON ((9841.741 5494.291...
## ACCTCCGTTATTCACC-1 POLYGON ((9046.741 4142.291...
## AGATGATGGAGTCTGG-1 POLYGON ((9050.741 4820.291...
## AGGTATAATTGATAGT-1 POLYGON ((9245.076 4480.294...
## AGTGAACAAACTTCTC-1 POLYGON ((11020.91 5261.585...
```

```
focus <- discrepancy_loc_dt$barcodes[1:2]
bw = 3*sfe@metadata[["spotDiameter"]][["JBO019"]][["spot_diameter_fullres"]]

# Plot the heatmap to visualise the genes that make this location an outlier
plotGWPCA_discrHeatmap(sfe = sfe,
                       assay = "logcounts",
                       vars = NULL,
```

```
                        focus = focus,
                        dMetric = "euclidean",
                        sample_id = "JBO019",
                        bw = bw,
                        mean.diff = 1,
                        show.vars = "top",
                        scale = "row",
                        gene.names = TRUE,
                        color = rev(colorRampPalette(brewer.pal(11, "RdBu"))(1000)),
                        fontsize_row = 3)
```


Distance
4
2
0
−2
−4
Distance
600
0
AAGTGCCTTGACTGTA−1
GGGTTTAGGATAGGAT−1
TAATTAGGACATCCGT−1
TATGGTTAGTGGGAGA−1
TCAAGGTTACTACACC−1
CTCTTCTATTGACTGG−1
TAGCGTCCCCTCGATTG−1
CACATTTCTTGTCAGA−1
GCTAGACCGTCTACTG−1
AGTCCATTGGCTGATG−1
ACATCCTGGTAACTGT−1
GCTCCCAGTCGGTCCA−1
CACCTTGGCGCCTTTG−1
GTCCATTACTGCTACG−1
TCTTAGAGTGAACTCT−1
CATGATGGAAGTTAGC−1
TTGACTATGTCCGGGC−1
GGTTTAGCCTTTCTTG−1
TTAAGATAGGATTGAC−1
AAGTCAATTGTCGTCA−1
TCTTCTATAACCCGCC−1
CGGGAATATAGTATAC−1
ACCGCGGTGGAAGTCG−1
GACAAACATATGCAGG−1
GCAACAGCAGTATGCG−1
TCGAAGAACCGAGCAC−1
ATCGCCAGTCAACATT−1
TGGGCAATAGTTGGGT−1
AGTGAACAAACTTCTC−1
CGGCCCAGGTATATCC−1
TTACATGCCACAACTA−1


Distance
4
2
0
−2
−4
Distance
600
0
ACCCGGATGACGCATC−1
GCTCGGCTCATGTCCAA−1
TGACATCGAGCGGACC−1
CATCTTACACCACCTC−1
CTAGGTCTGAAGGAAT−1
AGGGTCGATGCGAACT−1
CCGATCTCAACCTTAT−1
ATAGTTCCACCCACTC−1
TAGACTACCTAGCGTT−1
GCGCAAGAGCGCGCTG−1
TTGACGCTCCATGAGC−1
GCAGACCCAGCACGTA−1
ACATGGCGCGCCAAAGTA−1
CGCTAGAGACCGCTGC−1
TATAGATGGTCGCAGT−1
TCTGGGAACCTTTGAA−1
CCATCTCACCAGTGAA−1
GGTTCTACTCGTCTGA−1
CGCACGTGCGCTATCA−1
TAGGGAGCTTGGGATG−1
TATTACCATCCTGCTT−1
TATATCCCCTGGGAGGA−1
ACGATCATCTTGTAAA−1
GTGCGACAGGGAGTGT−1
TCATTTAAGTCTCCGA−1
TTGAATTCACGTGAGG−1
GAAAGAACAGCGTTAT−1
AACCCTACTGTCAATA−1
GCTGTTGCTACCGAAC−1
ATATCAACCTACAGAG−1
TGACGATGCACTAGAA−1

```
discrepancy_gene_dt <- getDiscrepancyGeneData(sfe = sfe,
                                              assay = "logcounts",
                                              vars = NULL,
                                              focus = focus[2],
                                              dMetric = "euclidean",
                                              sample_id = "JBO019",
                                              bw = bw,
                                              mean.diff = 1,
                                              show.vars = "top",
                                              exportExpression = TRUE)
head(discrepancy_gene_dt)
```

```
##                 AACCCTACTGTCAATA-1 ACATGGCGCCAAAGTA-1 ACCCGGATGACGCATC-1
## ENSG00000078808          1.9800764          1.7845077           0.000000
## ENSG00000157916          1.1230410          2.3434121           0.000000
## ENSG00000171603          0.6681445          0.8599719           1.667949
## ENSG00000162496          1.1230410          0.8599719           0.000000
## ENSG00000074964          1.4683664          0.0000000           1.964768
## ENSG00000158828          1.1230410          0.8599719           0.000000
##                 ACGATCATCTTGTAAA-1 AGGGTCGATGCGAACT-1 ATAGTTCCACCCACTC-1
## ENSG00000078808           1.434549          0.0000000          0.0000000
## ENSG00000157916           1.094451          1.1491034          0.7197148
## ENSG00000171603           0.000000          0.0000000          0.0000000
## ENSG00000162496           1.094451          2.2182483          1.5560955
## ENSG00000074964           1.709571          0.6860569          0.7197148
## ENSG00000158828           1.434549          2.0159260          1.1976848
##                 ATATCAACCTACAGAG-1 CATCTTACACCACCTC-1 CCATCTCACCAGTGAA-1
## ENSG00000078808           1.848204           2.383137          0.7918769
## ENSG00000157916           1.356318           1.489110          2.2192853
## ENSG00000171603           0.000000           0.000000          1.3002121
## ENSG00000162496           2.214350           0.000000          2.2192853
## ENSG00000074964           1.028715           1.140629          0.7918769
## ENSG00000158828           2.632509           1.140629          0.7918769
##                 CCGATCTCAACCTTAT-1 CGCACGTGCGCTATCA-1 CGCTAGAGACCGCTGC-1
## ENSG00000078808          1.1456336          2.0109592           1.070644
## ENSG00000157916          1.7760473          2.4713829           1.070644
## ENSG00000171603          0.0000000          0.0000000           0.000000
## ENSG00000162496          0.6836662          1.7105588           1.678406
## ENSG00000074964          1.1456336          0.8136275           1.070644
## ENSG00000158828          0.6836662          1.3307021           1.070644
##                 CTAGGTCTGAAGGAAT-1 GAAAGAACAGCGTTAT-1 GCAGACCCAGCACGTA-1
## ENSG00000078808          1.1275395          2.0533150          2.1052992
## ENSG00000157916          0.0000000          1.5312450          0.8684635
## ENSG00000171603          0.6712291          0.7049757          0.0000000
## ENSG00000162496          1.1275395          2.5947724          1.4067734
```

```
## ENSG00000074964          1.4736763          1.5312450          1.4067734
## ENSG00000158828          1.1275395          1.5312450          0.8684635
##                 GCGCAAGAGCGCGCTG-1 GCTCGCTCATGTCCAA-1 GCTGTTGCTACCGAAC-1
## ENSG00000078808           2.505794          1.4756527           2.004239
## ENSG00000157916           1.117631          1.8764253           1.588731
## ENSG00000171603           0.000000          0.0000000           1.325320
## ENSG00000162496           2.173035          2.1897262           1.588731
## ENSG00000074964           0.000000          0.9188039           1.002827
## ENSG00000158828           1.739765          0.9188039           1.811392
##                 GGTTCTACTCGTCTGA-1 GTGCGACAGGGAGTGT-1 TAGACTACCTAGCGTT-1
## ENSG00000078808          1.7977169           1.633005           1.421361
## ENSG00000157916          1.4066078           1.633005           0.000000
## ENSG00000171603          0.8683433           0.000000           0.000000
## ENSG00000162496          2.3583475           1.263669           2.123256
## ENSG00000074964          0.8683433           2.170771           0.000000
## ENSG00000158828          1.4066078           1.633005           1.695032
##                 TAGGGAGCTTGGGATG-1 TATAGATGGTCGCAGT-1 TATATCCCTGGGAGGA-1
## ENSG00000078808          2.2316954          0.0000000          0.7102708
## ENSG00000157916          0.7985659          1.9335702          2.6061791
## ENSG00000171603          0.0000000          0.0000000          0.0000000
## ENSG00000162496          0.7985659          1.9335702          1.1841040
## ENSG00000074964          0.7985659          0.9560551          0.7102708
## ENSG00000158828          0.0000000          1.5260667          1.1841040
##                 TATTACCATCCTGCTT-1 TCATTTAAGTCTCCGA-1 TCTGGGAACCTTTGAA-1
## ENSG00000078808          1.9382158          2.1150669          2.3926955
## ENSG00000157916          1.0928336          1.2220876          1.4967274
## ENSG00000171603          0.6475105          0.0000000          0.6846746
## ENSG00000162496          2.4677235          1.5846069          2.0131754
## ENSG00000074964          0.6475105          1.2220876          0.6846746
## ENSG00000158828          1.0928336          0.7367522          1.4967274
##                 TGACATCGAGCGGACC-1 TGACGATGCACTAGAA-1 TTGAATTCACGTGAGG-1
## ENSG00000078808          1.9076262          2.4843308          1.8133018
## ENSG00000157916          1.2485083          1.3402294          2.0507006
## ENSG00000171603          0.7552962          1.0152672          0.0000000
## ENSG00000162496          2.7012885          1.3402294          2.0507006
## ENSG00000074964          1.2485083          0.5951586          0.7036449
## ENSG00000158828          1.6153854          0.0000000          1.1745524
##                 TTGACGCTCCATGAGC-1 gene_name
## ENSG00000078808           2.296259      SDF4
## ENSG00000157916           1.851153      RER1
## ENSG00000171603           0.000000    CLSTN1
## ENSG00000162496           1.851153     DHRS3
## ENSG00000074964           0.000000 ARHGEF10L
## ENSG00000158828           1.851153     PINK1
```

4.10 Final Summary

In this practical we have shown the utility of a geospatial method, GWPCA, to explore the variability of an STx dataset at the local level. By assessing features of the output of this method, we can learn things about the spatial distribution of biologically relevant gene expression.

Hopefully this, alongside the other practicals today, have given you a basic grounding in how to work with STx data and some of the practical considerations of doing so. Although we have demonstrated all of these methods with 10X Genomics Visium data, there is no reason why they are not applicable to any STx method, such as Slide-Seq or Stereo-Seq.

You can learn more about the application of geospatial methods to this liver dataset by coming to see our poster: **B-122** in Poster Session B - Tuesday, July 25, between 18:00 CEST and 19:00 CEST.

Bibliography

Notes on Continuous Stochastic Phenomena on JSTOR, June 1950. URL https://www.jstor.org/stable/2332142. [Online; accessed 10. Apr. 2023].

The Contiguity Ratio and Statistical Mapping on JSTOR, November 1954. URL https://www.jstor.org/stable/2986645. [Online; accessed 10. Apr. 2023].

Robert A. Amezquita, Aaron T. L. Lun, Etienne Becht, Vince J. Carey, Lindsay N. Carpp, Ludwig Geistlinger, Federico Marini, Kevin Rue-Albrecht, Davide Risso, Charlotte Soneson, Levi Waldron, Hervé Pagès, Mike L. Smith, Wolfgang Huber, Martin Morgan, Raphael Gottardo, and Stephanie C. Hicks. Orchestrating single-cell analysis with Bioconductor. *Nat Methods*, 17:137–145, February 2020. ISSN 1548-7105. doi: 10.1038/s41592-019-0654-x.

Isabella Gollini, Binbin Lu, Martin Charlton, Christopher Brunsdon, and Paul Harris. GWmodel: An R Package for Exploring Spatial Heterogeneity Using Geographically Weighted Models. *J Stat Soft*, 63:1–50, February 2015. ISSN 1548-7660. doi: 10.18637/jss.v063.i17.

Martin Guilliams, Johnny Bonnardel, Birthe Haest, Bart Vanderborght, Camille Wagner, Anneleen Remmerie, Anna Bujko, Liesbet Martens, Tinne Thoné, Robin Browaeys, Federico F. De Ponti, Bavo Vanneste, Christian Zwicker, Freya R. Svedberg, Tineke Vanhalewyn, Amanda Gonçalves, Saskia Lippens, Bert Devriendt, Eric Cox, Giuliano Ferrero, Valerie Wittamer, Andy Willaert, Suzanne J.F. Kaptein, Johan Neyts, Kai Dallmeier, Peter Geldhof, Stijn Casaert, Bart Deplancke, Peter ten Dijke, Anne Hoorens, Aude Vanlander, Frederik Berrevoet, Yves Van Nieuwenhove, Yvan Saeys, Wouter Saelens, Hans Van Vlierberghe, Lindsey Devisscher, and Charlotte L. Scott. Spatial proteogenomics reveals distinct and evolutionarily conserved hepatic macrophage niches. *Cell*, 185(2):379–396.e38, 2022. ISSN 0092-8674. doi: https://doi.org/10.1016/j.cell.2021.12.018. URL https://www.sciencedirect.com/science/article/pii/S0092867421014811.

Yuhan Hao, Stephanie Hao, Erica Andersen-Nissen, William M. Mauck, Shiwei Zheng, Andrew Butler, Maddie J. Lee, Aaron J. Wilk, Charlotte Darby, Michael Zager, Paul Hoffman, Marlon Stoeckius, Efthymia Papalexi, Eleni P. Mimitou, Jaison Jain, Avi Srivastava, Tim Stuart, Lamar M. Fleming,

Bertrand Yeung, Angela J. Rogers, Juliana M. McElrath, Catherine A. Blish, Raphael Gottardo, Peter Smibert, and Rahul Satija. Integrated analysis of multimodal single-cell data. *Cell*, 184(13):3573–3587.e29, June 2021. ISSN 0092-8674. doi: 10.1016/j.cell.2021.04.048.

Paul Harris, Chris Brunsdon, and Martin Charlton. Geographically weighted principal components analysis. *International Journal of Geographical Information Science*, 25(10):1717–1736, October 2011. ISSN 1365-8816. doi: 10.1080/13658816.2011.554838.

Eamonn Keogh and Abdullah Mueen. Curse of Dimensionality. In *Encyclopedia of Machine Learning and Data Mining*, pages 314–315. Springer, Boston, MA, Boston, MA, USA, April 2017. doi: 10.1007/978-1-4899-7687-1_192.

Yijun Li, Stefan Stanojevic, Bing He, Zheng Jing, Qianhui Huang, Jian Kang, and Lana X. Garmire. Benchmarking Computational Integration Methods for Spatial Transcriptomics Data. *bioRxiv*, page 2021.08.27.457741, January 2022. URL https://doi.org/10.1101/2021.08.27.457741.

Aaron T. L. Lun, Davis J. McCarthy, and John C. Marioni. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research*, 5(2122):2122, October 2016. doi: 10.12688/f1000research.9501.2.

Kristen R. Maynard, Leonardo Collado-Torres, Lukas M. Weber, Cedric Uytingco, Brianna K. Barry, Stephen R. Williams, Joseph L. Catallini, Matthew N. Tran, Zachary Besich, Madhavi Tippani, Jennifer Chew, Yifeng Yin, Joel E. Kleinman, Thomas M. Hyde, Nikhil Rao, Stephanie C. Hicks, Keri Martinowich, and Andrew E. Jaffe. Transcriptome-scale spatial gene expression in the human dorsolateral prefrontal cortex. *Nat Neurosci*, 24:425–436, March 2021. ISSN 1546-1726. doi: 10.1038/s41593-020-00787-0.

Davis J. McCarthy, Kieran R. Campbell, Aaron T. L. Lun, and Quin F. Wills. Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics*, 33(8):1179–1186, April 2017. ISSN 1367-4803. doi: 10.1093/bioinformatics/btw777.

Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv*, February 2018. doi: 10.48550/arXiv.1802.03426.

Dario Righelli, Lukas M. Weber, Helena L. Crowell, Brenda Pardo, Leonardo Collado-Torres, Shila Ghazanfar, Aaron T. L. Lun, Stephanie C. Hicks, and Davide Risso. SpatialExperiment: infrastructure for spatially-resolved transcriptomics data in R using Bioconductor. *Bioinformatics*, 38(11):3128–3131, June 2022. ISSN 1367-4803. doi: 10.1093/bioinformatics/btac299.

Shiquan Sun, Jiaqiang Zhu, and Xiang Zhou. Statistical analysis of spatial expression patterns for spatially resolved transcriptomic studies. *Nat Methods*, 17:193–200, February 2020. ISSN 1548-7105. doi: 10.1038/s41592-019-0701-7.

Valentine Svensson, Sarah A. Teichmann, and Oliver Stegle. SpatialDE: identification of spatially variable genes. *Nat Methods*, 15:343–346, May 2018. ISSN 1548-7105. doi: 10.1038/nmeth.4636.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(86):2579–2605, 2008. URL http://jmlr.org/papers/v9/vandermaaten08a.html.

Lukas M. Weber and Helena L. Crowell. *ggspavis: Visualization functions for spatially resolved transcriptomics data*, 2022. URL https://github.com/lmweber/ggspavis. R package version 1.4.0.

Jiaqiang Zhu, Shiquan Sun, and Xiang Zhou. SPARK-X: non-parametric modeling enables scalable and robust detection of spatial expression patterns for large spatial transcriptomic studies. *Genome Biol*, 22(1):1–25, December 2021. ISSN 1474-760X. doi: 10.1186/s13059-021-02404-0.