# spicyPlayBook

Alex Qin

2024-11-11

# Table of contents

# Overview

## Welcome!

Recent advances in highly multiplexed cell imaging technologies such as *PhenoCycler, IMC, CosMx, Xenium, and MERFISH (and many more)* have fundamentally revolutionized our ability to observe complex cellular relationships in tissue. Where previous immunohistochemistry protocols only allowed the visualization of cells that could be characterized by two or three surface proteins, cutting-edge technologies characterize cells with upwards of 50 proteins or 1000s of RNA in situ. These technologies enable precise classification of cell sub-types and provide an unprecedented depiction of cellular heterogeneity in a tissue environment. These technical developments have necessitated the development of a variety of new analytical approaches that are required to harness these new imaging technologies. On this website we will demonstrate how packages in scdney can be used to provide new insights into complex biological systems and diseases.

## Packages

### MoleculeExperiment

MoleculeExperiment contains functions to create and work with objects from the new MoleculeExperiment class. We introduce this class for analysing molecule-based spatial transcriptomics data (e.g., Xenium by 10X, Cosmx SMI by Nanostring, and Merscope by Vizgen). This allows researchers to analyse spatial transcriptomics data at the molecule level, and to have standardised data formats accross vendors.

Peters Couto B, Robertson N, Patrick E, Ghazanfar S (2024). MoleculeExperiment: Prioritising a molecule-level storage of Spatial Transcriptomics Data. R package version 1.6.0.

### simpleSeg

Image segmentation is the process of identifying the borders of individual objects (in this case cells) within an image. This allows for the features of cells such as marker expression and morphology to be extracted, stored and analysed. simpleSeg provides functionality for user friendly, watershed based segmentation on multiplexed cellular images in R based on

the intensity of user specified protein marker channels. simpleSeg can also be used for the normalization of single cell data obtained from multiple images.

Canete N, Nicholls A, Patrick E (2024). simpleSeg: A package to perform simple cell segmentation. R package version 1.8.0.

## scMerge

Like all gene expression data, single-cell data suffers from batch effects and other unwanted variations that makes accurate biological interpretations difficult. The scMerge method leverages factor analysis, stably expressed genes (SEGs) and (pseudo-) replicates to remove unwanted variations and merge multiple single-cell data. This package contains all the necessary functions in the scMerge pipeline, including the identification of SEGs, replication-identification methods, and merging of single-cell data.

Lin Y, Ghazanfar S, Wang K, Gagnon-Bartsch J, Lo K, Su X, Han Z, Ormerod J, Speed T, Yang P, Yang J (2019). "scMerge leverages factor analysis, stable expression, and pseudoreplication to merge multiple single-cell RNA-seq datasets." Proceedings of the National Academy of Sciences. doi:10.1073/pnas.1820006116.

## FuseSOM

A correlation-based multiview self-organizing map for the characterization of cell types in highly multiplexed in situ imaging cytometry assays (`FuseSOM`) is a tool for unsupervised clustering. `FuseSOM` is robust and achieves high accuracy by combining a `Self Organizing Map` architecture and a `Multiview` integration of correlation based metrics. This allows FuseSOM to cluster highly multiplexed in situ imaging cytometry assays.

<0-length citation>

## treekoR

treekoR is a novel framework that aims to utilise the hierarchical nature of single cell cytometry data to find robust and interpretable associations between cell subsets and patient clinical end points. These associations are aimed to recapitulate the nested proportions prevalent in workflows inovlving manual gating, which are often overlooked in workflows using automatic clustering to identify cell populations. We developed treekoR to: Derive a hierarchical tree structure of cell clusters; quantify a cell types as a proportion relative to all cells in a sample (%total), and, as the proportion relative to a parent population (%parent); perform significance testing using the calculated proportions; and provide an interactive html visualisation to help highlight key results.

Chan A (2024). treekoR: Cytometry Cluster Hierarchy and Cellular-to-phenotype Associations. R package version 1.14.0.

## scFeatures

scFeatures constructs multi-view representations of single-cell and spatial data. scFeatures is a tool that generates multi-view representations of single-cell and spatial data through the construction of a total of 17 feature types. These features can then be used for a variety of analyses using other software in Biocondutor.

Cao,Y., Lin,Y., Patrick,E., Yang,P., Yang,J.Y.H. & (2022). "scFeatures: multi-view representations of single-cell and spatial data for disease outcome prediction." Bioinformatics, 38(20), 4745-4753. ISSN 1367-4803, doi:10.1093/bioinformatics/btac590.

## scHOT

Single cell Higher Order Testing (scHOT) is an R package that facilitates testing changes in higher order structure of gene expression along either a developmental trajectory or across space. scHOT is general and modular in nature, can be run in multiple data contexts such as along a continuous trajectory, between discrete groups, and over spatial orientations; as well as accommodate any higher order measurement such as variability or correlation. scHOT meaningfully adds to first order effect testing, such as differential expression, and provides a framework for interrogating higher order interactions from single cell data.

Ghazanfar S, Lin Y (2024). scHOT: single-cell higher order testing. R package version 1.18.0.

## spicyR

The spicyR package provides a framework for performing inference on changes in spatial relationships between pairs of cell types for cell-resolution spatial omics technologies. spicyR consists of three primary steps: (i) summarizing the degree of spatial localization between pairs of cell types for each image; (ii) modelling the variability in localization summary statistics as a function of cell counts and (iii) testing for changes in spatial localizations associated with a response variable.

Canete N, Iyengar S, Ormerod J, Baharlou H, Harman A, Patrick E (2022). "spicyR: spatial analysis of in situ cytometry data in R." Bioinformatics, 38(11), 3099–3105. doi:10.1093/bioinformatics/btac268.

## Statial

Statial is a suite of functions for identifying changes in cell state. The functionality provided by Statial provides robust quantification of cell type localisation which are invariant to changes in tissue structure. In addition to this Statial uncovers changes in marker expression associated with varying levels of localisation. These features can be used to explore how the structure and function of different cell types may be altered by the agents they are surrounded with.

Ameen F, Robertson N, Lin D, Ghazanfar S, Patrick E (2024). "Kontextual: Reframing analysis of spatial omics data reveals consistent cell relationships across images." bioRxiv. doi:10.1101/2024.09.03.611109.

## lisaClust

lisaClust provides a series of functions to identify and visualise regions of tissue where spatial associations between cell-types is similar. This package can be used to provide a high-level summary of cell-type colocalization in multiplexed imaging data that has been segmented at a single-cell resolution.

Patrick E, Canete N (2024). lisaClust: lisaClust: Clustering of Local Indicators of Spatial Association. R package version 1.14.4.

## ClassifyR

The software formalises a framework for classification and survival model evaluation in R. There are four stages; Data transformation, feature selection, model training, and prediction. The requirements of variable types and variable order are fixed, but specialised variables for functions can also be provided. The framework is wrapped in a driver loop that reproducibly carries out a number of cross-validation schemes. Functions for differential mean, differential variability, and differential distribution are included. Additional functions may be developed by the user, by creating an interface to the framework.

Strbenac D, Mann GJ, Ormerod JT, Yang JYH (2015). "ClassifyR: an R package for performance assessment of classification with applications to transcriptomics." Bioinformatics, 31(11), 1851-1853.

This guide presents a comprehensive workflow for analysing spatial omics data, featuring examples sorted by different technologies as described below. The workflow covers cell segmentation, data normalisation, various tests of proportion and spatial localisation, microenvironment estimation and patient prediction. We encourage focusing on the biological questions these methods can address rather than the specific technologies used.

| Disease | Technology | Title | Segmentation | Alignment | Clustering | Localisation | Microenvironments | Patient Classification |
|---|---|---|---|---|---|---|---|---|
| Breast cancer | MIBI-TOF | Keren_2018 | | | | X | X | X |
| Breast cancer | MIBI-TOF | Risom_2022 | X | X | X | X | X | X |
| Mouse organogenesis | seqFISH | Lohoff_2022 | | X | | X | | |

## Datasets

Through the course of this spicyWorkBook, we will take advantage of several different spatial datasets that are publicly available. These datasets are all accessible within our SpatialDatasets package on bioconductor. We will demonstrate several questions that could be answered or explored for each of these datasets using the available information.

### Spatial Proteomics - MIBITOF

MIBI-TOF (multiplexed ion beam imaging by time of flight) is an instrument that uses bright ion sources and orthogonal time-of-flight mass spectrometry to image metal-tagged antibodies at subcellular resolution in clinical tissue sections. It is capable of imaging approximately 40 labelled antibodies and image fields of about $1mm^2$ at resolutions down to $260nm$.

#### Triple Negative Breast Cancer - Keren_2018

This study profiles 36 proteins in tissue samples from 41 patients with triple-negative breast cancer using MIBI-TOF. We will use this dataset to demonstrate the functionality of our Statial package, which allows us to identify changes in cell state that are related to the spatial localisation of cells.

Keren et al. (2018). A Structured Tumor-Immune Microenvironment in Triple Negative Breast Cancer Revealed by Multiplexed Ion Beam Imaging. Cell, 174(6), 1373-1387.e1319. (DOI)

### Ductal carcinoma in situ - Risom_2022

This study uses MIBI-TOF to profile the spatial landscape of ductal carcinoma in situ (DCIS), a pre-invasive lesion believed to be a precursor to invasive breast cancer (IBC). A key conclusion of the manuscript is that spatial information about cells can be leveraged to predict disease progression in patients. We use the workflow described in this guide to reach a similar conclusion.

Risom et al. (2022). Transition to invasive breast cancer is associated with progressive changes in the structure and composition of tumor stroma. Cell, 185(2), 299-310.e18 (DOI)

## Spatial Proteomics - CODEX

CODEX (Co-detection by indexing) is a highly multiplexed tissue imaging technique that uses DNA-barcoded antibodies which are later revealed by fluorescent detector oligonucleotides. It can visualise up to 60 labelled antibodies at subcellular resolution.

### Colorectal cancer - Schurch_2020

A CODEX dataset which aimed to characterise the immune tumour microenvironment in advanced-stage colorectal cancer. The dataset consists of 35 advanced colorectal cancer patients, with 4 images per patient for a total of 140 images. Each image is marked with a 56-antibody panel to characterise a total of 24 distinct tumour and immune cell populations. Overall, the dataset contains 240,000 cells along with clinical information including patient tumour grade, tumour type, and patient survival.

Schürch et al. (2020). Coordinated Cellular Neighborhoods Orchestrate Antitumoral Immunity at the Colorectal Cancer Invasive Front et al. (2018). A Coordinated Cellular Neighborhoods Orchestrate Antitumoral Immunity at the Colorectal Cancer Invasive Front. Cell, 182(5), 1341-1359.e19. (DOI)

## Spatial Proteomics - IMC

IMC (Imaging Mass Cytometry) is an instrument that combines laser ablation with mass cytometry to image metal-tagged antibodies at subcellular resolution in clinical tissue sections. The datasets produced by IMC can image approximately 30–40 labeled antibodies, covering tissue areas of around $1mm^2$ with a resolution down to $1\mu m$.

### Breast cancer - Ali_2020

Also known as the METABRIC dataset, this 37-panel IMC dataset contains images of 456 primary invasive breast carcinoma patients obtained from 548 samples. Clinical variables in the dataset include age, chemotherapy (CT), radiotherapy (RT), hormone treatment (HT) indicators, estrogen receptor (ER) status, and gene expression markers (MKI67, EGFR, PGR, and ERBB2).

Ali et al. (2020). Imaging mass cytometry and multiplatform genomics define the phenogenomic landscape of breast cancer. Nature Cancer, 1, 163-175. (DOI)

### Head and neck squamous cell carcinoma - Ferguson_2020

This study uses IMC to map the immune landscape and identify differences between high-risk primary head and neck cancer (HNcSCC) tumors that did not progress and those that developed metastases (progressing tumours). The key conclusion of this manuscript (amongst others) is that spatial information about cells and the immune environment can be used to predict primary tumour progression or metastases in patients. We will use our spicyWorkflow to reach a similar conclusion.

Ferguson et al. (2022). High-Dimensional and Spatial Analysis Reveals Immune Landscape–Dependent Progression in Cutaneous Squamous Cell Carcinoma. Clinical Cancer Research, 28(21), 4677-4688. (DOI)

## Spatial Transcriptomics - seqFISH

SeqFISH (sequential Fluorescence In Situ Hybridization) is a technology that enables the identification of thousands of molecules like RNA, DNA, and proteins directly in single cells with their spatial context preserved. seqFISH can multiplex over 10,000 molecules and integrate multiple modalities.

### Mouse organogenesis - Lohoff_2022

This study uses seqFISH to spatially profile the expression of 387 genes in mouse embryos. A comprehensive spatially resolved map of gene expression was created by integrating the seqFISH data with existing scRNAseq data. This integration facilitated the exploration of cellular relationships across different regions of the embryo.

Lohoff et al. (2022). Integration of spatial and single-cell transcriptomic data elucidates mouse organogenesis. Nature Biotechnology 40, 74–85 (DOI).

# 1 Processing

In this section, we will describe how to read in and pre-process images obtained through various imaging technologies for downstream analysis.

Steps:

1. How to read in data with cytomapper
2. How to segment data with simpleSeg
3. How to segment data with BIDCell
4. How to read in spot-based data with MoleculeExperiment

## 1.1 Reading in images

```
library(cytomapper)
library(ggplot2)
library(simpleSeg)
```

It is convenient to set the number of cores for running code in parallel. Please choose a number that is appropriate for your resources. A minimum of 2 cores is suggested since running this workflow is rather computationally intensive.

If you would like to use parallel processing for the rest of the vignette, set the `use_mc` flag to `TRUE`.

```
use_mc <- TRUE

if (use_mc) {
  nCores <- max(parallel::detectCores()/2, 1)
} else {
  nCores <- 2
}
BPPARAM <- simpleSeg:::generateBPParam(nCores)

theme_set(theme_classic())
```

We will be using the Ferguson 2022 dataset to demonstrate how to perform pre-processing and cell segmentation. This dataset can be accessed through the `SpatialDatasets` package. The `loadImages()` function form the `cytomapper` package can be used to load all the TIFF images into a `CytoImageList` object and store the images as h5 file on-disk in a temporary directory using the `h5FilesPath = HDF5Array::getHDF5DumpDir()` parameter.

We will also assign the metadata columns of the `CytoImageList` object using the `mcols()` function.

```
pathToImages <- SpatialDatasets::Ferguson_Images()
```

```
see ?SpatialDatasets and browseVignettes('SpatialDatasets') for documentation
```

```
loading from cache
```

```
tmp <- tempfile()
unzip(pathToImages, exdir = tmp)

# Store images in a CytoImageList on_disk as h5 files to save memory.
images <- cytomapper::loadImages(
  tmp,
  single_channel = TRUE,
  on_disk = TRUE,
  h5FilesPath = HDF5Array::getHDF5DumpDir(),
  BPPARAM = BPPARAM
)

mcols(images) <- S4Vectors::DataFrame(imageID = names(images))
```

As we're reading the image channels directly from the names of the TIFF image, often these channel names will need to be cleaned for ease of downstream processing.

The channel names can be accessed from the `CytoImageList` object using the `channelNames()` function.

```
channelNames(images) <- channelNames(images) |>
                          # Remove preceding letters
                          sub(pattern = ".*_", replacement = "", x = _) |>
                          # Remove the .ome
                          sub(pattern = ".ome", replacement = "", x = _)
```

Similarly, the image names will be taken from the folder name containing the individual TIFF images for each channel. These will often also need to be cleaned.

```
split_names <- function(x) {
  sapply(strsplit(x, "_"), `[`, 3)
}

names(images) <- names(images) |> split_names()

mcols(images) <- S4Vectors::DataFrame(imageID = names(images))
```

## 1.2 Cell segmentation with simpleSeg

The simpleSeg package provides functionality to perform cell segmentation on multiplexed imaging data. The `simpleSeg()` function can be used to perform a simple cell segmentation process that traces out the nuclei using a specified channel.

In the particular example below, we have asked `simpleSeg` to do the following:

- `nucleus = c("HH3")`: trace out the nuclei signal in the images using the HH3 channel.
- `pca = TRUE`: segment out the nuclei mask using a principal component analysis of all channels and using the principal components most aligned with the nuclei channel, in this case, HH3.
- `cellBody = "dilate"`: use a dilation strategy of segmentation, expanding out from the nucleus by a specified `discSize`. In this case, `discSize = 3`, which means simpleSeg dilates out from the nucleus by 3 pixels.
- `sizeSelection = 20`: ensure that only cells with a size greater than 20 pixels will be used.
- `transform = "sqrt"`: perform square root transformation on each of the channels prior to segmentation.
- `tissue = c("panCK", "CD45", "HH3")`: use the specified tissue mask to filter out all background noise outside the tissue mask. This allows us to ignore background noise which happens outside of the tumour core.

There are many other parameters that can be specified in simpleSeg (`smooth`, `watershed`, `tolerance`, and `ext`), and we encourage the user to select the best parameters which suit their biological context.

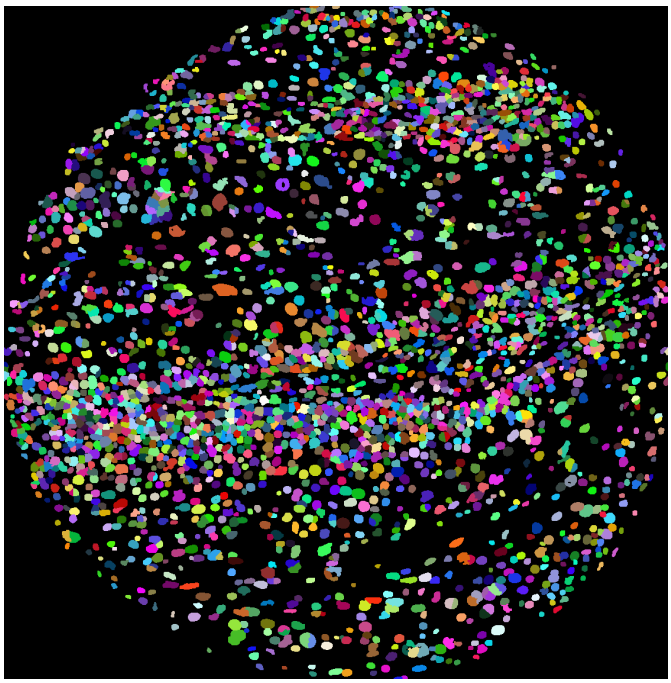
```
masks <- simpleSeg(images,
                   nucleus = c("HH3"),
                   pca = TRUE,
                   cellBody = "dilate",
```

```
                    discSize = 3,
                    sizeSelection = 20,
                    transform = "sqrt",
                    tissue = c("panCK", "CD45", "HH3"),
                    cores = nCores
                    )
```

### 1.2.1 Visualise separation

The `display()` and `colorLabels()` functions in the `EBImage` packagemake it very easy to examine the performance of the cell segmentation. If used in an interactive session, `display()` allows you to zoom in and out of the image.

```
EBImage::display(colorLabels(masks[[1]]))
```

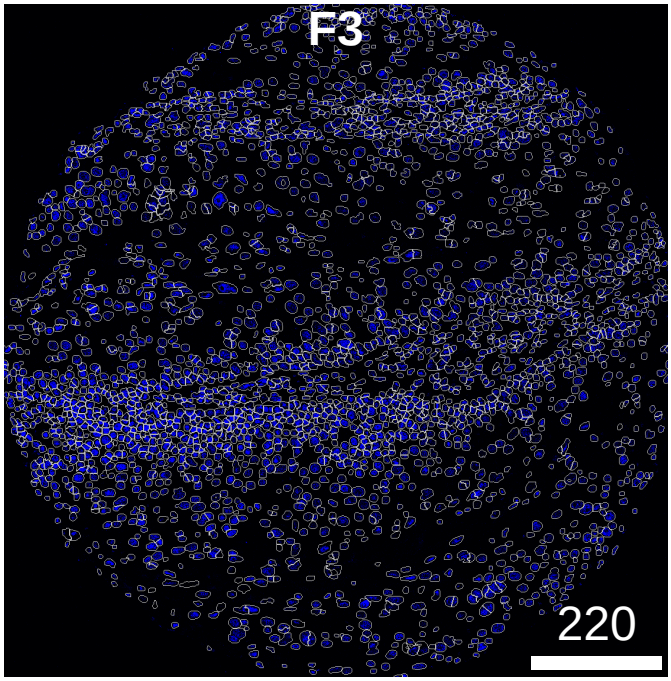


### 1.2.2 Visualise outlines

The `plotPixels` function in `cytomapper` makes it easy to overlay the mask on top of the nucleus intensity marker to see how well our segmentation process has performed. Here we can see that the segmentation appears to be performing reasonably.

If you see over or under-segmentation of your images, `discSize` is a key parameter in `simpleSeg()` for optimising the size of the dilation disc after segmenting out the nuclei.

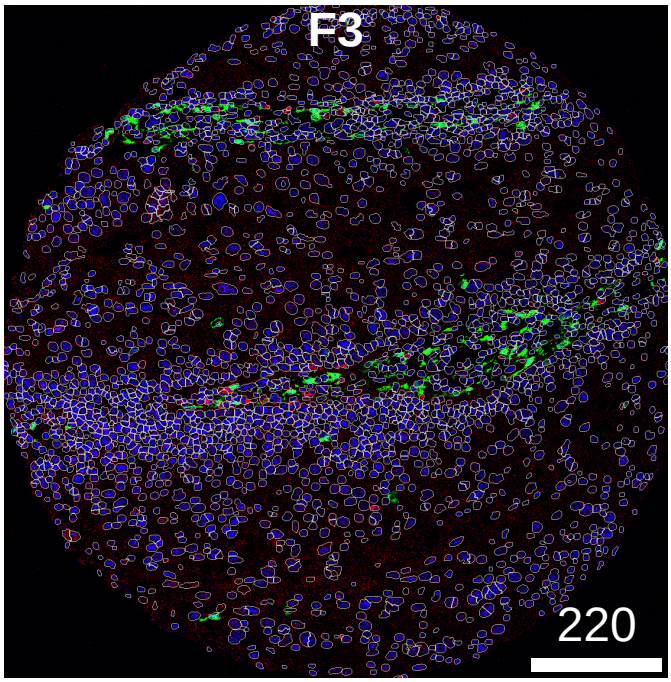
```
plotPixels(image = images["F3"],
           mask = masks["F3"],
           img_id = "imageID",
           colour_by = c("HH3"),
           display = "single",
           colour = list(HH3 = c("black","blue")),
           legend = NULL,
           bcg = list(
             HH3 = c(1, 1, 2)
           ))
```




We can also visualise multiple markers at once instead of just the HH3 marker to see how the segmentation mask performs. Below, we can see that our segmentation mask has done a good job of capturing the CD31 signal, but perhaps not such a good job of capturing the FXIIIA signal, which often lies outside of our dilated nuclear mask. This could suggest that we might need to increase the `discSize` or other parameters of `simpleSeg`.

```
plotPixels(image = images["F3"],
           mask = masks["F3"],
           img_id = "imageID",
```

```
            colour_by = c("HH3", "CD31", "FX111A"),
            display = "single",
            colour = list(HH3 = c("black","blue"),
                          CD31 = c("black", "red"),
                          FX111A = c("black", "green") ),
            legend = NULL,
            bcg = list(
              HH3 = c(1, 1, 2),
              CD31 = c(0, 1, 2),
              FX111A = c(0, 1, 1.5)
            ))
```




In particular, the `cellBody` and `watershed` parameters can strongly influence the way cells are segmented using `simpleSeg()`. We've provided further details on how the user may specify cell body identification and watershedding in the tables below.

#### 1.2.2.1 `cellBody` Parameters

| Method | Description |
|---|---|
| “distance” | Performs watershedding on a distance map of the thresholded nuclei signal. With a pixels distance being defined as the distance from the closest background signal. |
| “intensity” | Performs watershedding using the intensity of the nuclei marker. |
| “combine” | Combines the previous two methods by multiplying the distance map by the nuclei marker intensity. |

#### 1.2.2.2 `watershed` Parameters

| Method | Description |
|---|---|
| “dilation” | Dilates the nuclei by an amount defined by the user. The size of the dilatation in pixels may be specified with the `discDize` argument. |
| “discModel” | Uses all the markers to predict the presence of dilated ‘discs’ around the nuclei. The model therefore learns which markers are typically present in the cell cytoplasm and generates a mask based on this. |
| “marker” | The user may specify one or multiple dedicated cytoplasm markers to predict the cytoplasm. This can be done using `cellBody = "marker name"/"index"` |
| “None” | The nuclei mask is returned directly. |

## 1.3 Cell segmentation with BIDCell

## 1.4 Visual comparison of segmentations

plotPixels can plot multiple images <– use this to visualise multiple images at once after you have BIDCell ready.

## 1.5 Summarise cell features

In order to characterise the phenotypes of each of the segmented cells, `measureObjects()` from `cytomapper` will calculate the average intensity of each channel within each cell as well as a few morphological features. By default, the `measureObjects()` function will return a `SingleCellExperiment` object, where the channel intensities are stored in the `counts` assay and the spatial location of each cell is stored in `colData` in the `m.cx` and `m.cy` columns.

However, you can also specify `measureObjects()` to return a `SpatialExperiment` object by specifying `return_as = "spe"`. As a `SpatialExperiment` object, the spatial location of each cell is stored in the `spatialCoords` slot, as `m.cx` and `m.cy`, which simplifies plotting. In this demonstration, we will return a `SpatialExperiment` object.

```
# Summarise the expression of each marker in each cell
cells <- cytomapper::measureObjects(masks,
                                    images,
                                    img_id = "imageID",
                                    return_as = "spe",
                                    BPPARAM = BPPARAM)

spatialCoordsNames(cells) <- c("x", "y")
```

## 1.6 sessionInfo

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Debian GNU/Linux 12 (bookworm)

Matrix products: default
```

```
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.21.so;  LAPACK version 3

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: Australia/Sydney
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] SpatialDatasets_1.4.0       SpatialExperiment_1.16.0
 [3] ExperimentHub_2.14.0        AnnotationHub_3.14.0
 [5] BiocFileCache_2.14.0        dbplyr_2.5.0
 [7] simpleSeg_1.8.0             ggplot2_3.5.1
 [9] cytomapper_1.18.0           SingleCellExperiment_1.28.1
[11] SummarizedExperiment_1.36.0 Biobase_2.66.0
[13] GenomicRanges_1.58.0        GenomeInfoDb_1.42.0
[15] IRanges_2.40.0              S4Vectors_0.44.0
[17] BiocGenerics_0.52.0         MatrixGenerics_1.18.0
[19] matrixStats_1.4.1           EBImage_4.48.0

loaded via a namespace (and not attached):
  [1] DBI_1.2.3               bitops_1.0-9            deldir_2.0-4
  [4] gridExtra_2.3           rlang_1.1.4             magrittr_2.0.3
  [7] svgPanZoom_0.3.4        shinydashboard_0.7.2    RSQLite_2.3.7
 [10] compiler_4.4.1          spatstat.geom_3.3-3     png_0.1-8
 [13] systemfonts_1.1.0       fftwtools_0.9-11        vctrs_0.6.5
 [16] pkgconfig_2.0.3         crayon_1.5.3            fastmap_1.2.0
 [19] magick_2.8.5            XVector_0.46.0          utf8_1.2.4
 [22] promises_1.3.0          rmarkdown_2.29          UCSC.utils_1.2.0
 [25] ggbeeswarm_0.7.2        tinytex_0.54            purrr_1.0.2
 [28] bit_4.5.0               xfun_0.49               cachem_1.1.0
 [31] zlibbioc_1.52.0         jsonlite_1.8.9          blob_1.2.4
 [34] later_1.3.2             rhdf5filters_1.18.0     DelayedArray_0.32.0
 [37] spatstat.utils_3.1-1    Rhdf5lib_1.28.0         BiocParallel_1.40.0
 [40] jpeg_0.1-10             tiff_0.1-12             terra_1.7-83
```

```
 [43] parallel_4.4.1          R6_2.5.1                RColorBrewer_1.1-3
 [46] spatstat.data_3.1-2     spatstat.univar_3.1-1   Rcpp_1.0.13-1
 [49] knitr_1.49              httpuv_1.6.15           Matrix_1.7-1
 [52] nnls_1.6                tidyselect_1.2.1        yaml_2.3.10
 [55] rstudioapi_0.17.1       abind_1.4-8             viridis_0.6.5
 [58] codetools_0.2-20        curl_6.0.1              lattice_0.22-6
 [61] tibble_3.2.1            KEGGREST_1.46.0         shiny_1.9.1
 [64] withr_3.0.2             evaluate_1.0.1          polyclip_1.10-7
 [67] Biostrings_2.74.0       filelock_1.0.3          BiocManager_1.30.25
 [70] pillar_1.9.0            generics_0.1.3          sp_2.1-4
 [73] RCurl_1.98-1.16         BiocVersion_3.20.0      munsell_0.5.1
 [76] scales_1.3.0            xtable_1.8-4            glue_1.8.0
 [79] tools_4.4.1             locfit_1.5-9.10         rhdf5_2.50.0
 [82] grid_4.4.1              AnnotationDbi_1.68.0    colorspace_2.1-1
 [85] GenomeInfoDbData_1.2.13 raster_3.6-30           beeswarm_0.4.0
 [88] HDF5Array_1.34.0        vipor_0.4.7             cli_3.6.3
 [91] rappdirs_0.3.3          fansi_1.0.6             S4Arrays_1.6.0
 [94] viridisLite_0.4.2       svglite_2.1.3           dplyr_1.1.4
 [97] gtable_0.3.6            digest_0.6.37           SparseArray_1.6.0
[100] rjson_0.2.23            htmlwidgets_1.6.4       memoise_2.0.1
[103] htmltools_0.5.8.1       lifecycle_1.0.4         httr_1.4.7
[106] mime_0.12               bit64_4.5.2
```

# 2 Quality Control

Steps:

1. How to qc segmentation (simpleSeg/cellSPA?)
2. How to qc image batch-effect (simpleSeg::normalizeCells)
3. How to qc patient batch-effect (simpleSeg::normalizeCells)
4. How to qc batch effects (scMerge)

## 2.1 CellSPA: How do I determine segmentation quality?

## 2.2 simpleSeg: Do my images have a batch effect?

In many spatial imaging protocols, there tends to be a degree of variability in the intensity of each image. For example, in one image, the CD3 stain may be too strong, whereas in another image the CD3 staining is particularly weak. This variability is often times inevitable and can be hard to correct for in the imaging process. Hence, it is important that we identify when such variance occurs and correct it.

First, let's load in the images we previously segmented out in the last section. The `SpatialDatasets` package conveniently provides the segmented out images for the HNsCC dataset from Ferguson et al., 2022.

```
library(tidySingleCellExperiment)
```

```
Loading required package: SingleCellExperiment
```

```
Loading required package: SummarizedExperiment
```

```
Loading required package: MatrixGenerics
```

```
Loading required package: matrixStats
```

```
Attaching package: 'MatrixGenerics'

The following objects are masked from 'package:matrixStats':

    colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
    colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
    colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
    colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
    colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
    colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
    colWeightedMeans, colWeightedMedians, colWeightedSds,
    colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
    rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
    rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
    rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
    rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
    rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
    rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
    rowWeightedSds, rowWeightedVars

Loading required package: GenomicRanges

Loading required package: stats4

Loading required package: BiocGenerics


Attaching package: 'BiocGenerics'

The following objects are masked from 'package:stats':

    IQR, mad, sd, var, xtabs

The following objects are masked from 'package:base':

    anyDuplicated, aperm, append, as.data.frame, basename, cbind,
    colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
    get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
    match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
    Position, rank, rbind, Reduce, rownames, sapply, saveRDS, setdiff,
    table, tapply, union, unique, unsplit, which.max, which.min
```

```
Loading required package: S4Vectors


Attaching package: 'S4Vectors'

The following object is masked from 'package:utils':

    findMatches

The following objects are masked from 'package:base':

    expand.grid, I, unname

Loading required package: IRanges

Loading required package: GenomeInfoDb

Loading required package: Biobase

Welcome to Bioconductor

    Vignettes contain introductory material; view with
    'browseVignettes()'. To cite Bioconductor, see
    'citation("Biobase")', and for packages 'citation("pkgname")'.


Attaching package: 'Biobase'

The following object is masked from 'package:MatrixGenerics':

    rowMedians

The following objects are masked from 'package:matrixStats':

    anyMissing, rowMedians
```

```r
library(simpleSeg)
# library(scMerge)
library(scater)
```

```
Loading required package: scuttle
```

```
library(ggplot2)
```

```
fergusonSPE <- SpatialDatasets::spe_Ferguson_2022()
```

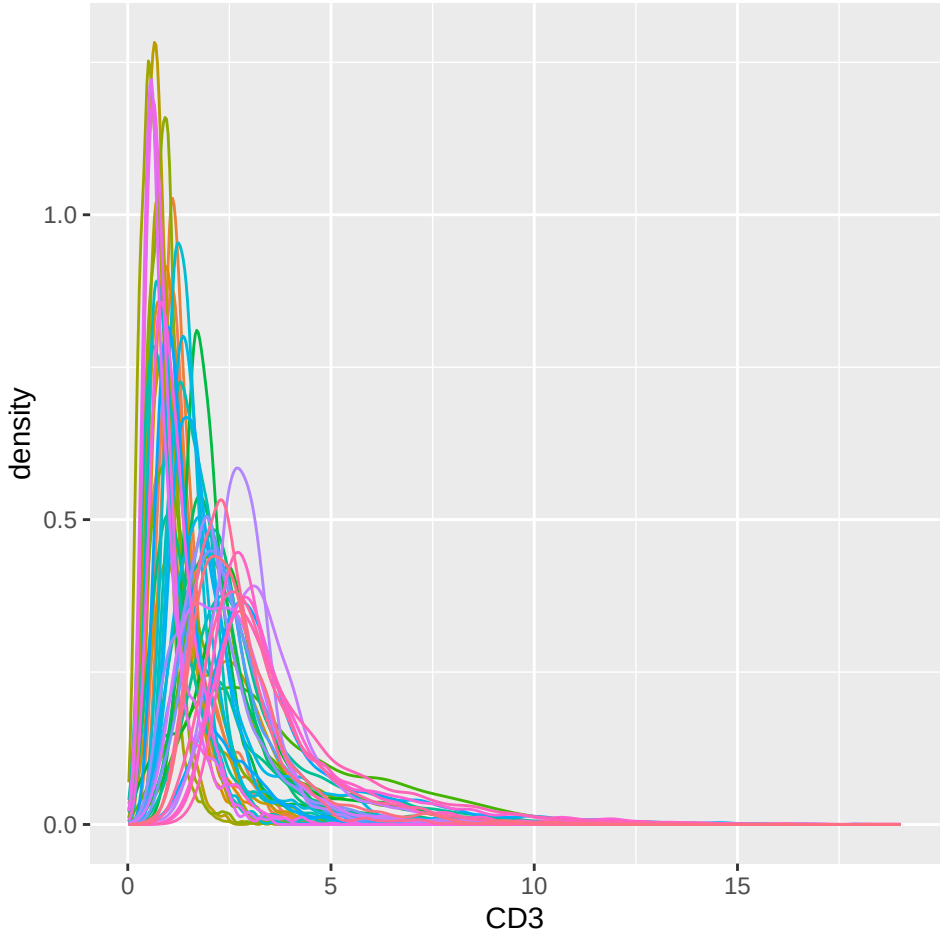
```
see ?SpatialDatasets and browseVignettes('SpatialDatasets') for documentation
```

```
loading from cache
```

Next, we can check if the marker intensities of each cell require some form of transformation or normalisation. The reason we do this is two-fold:
1) The intensities of images are often highly skewed, preventing any meaningful downstream analysis.
2) The intensities across different images are often different, meaning that what is considered "positive" can be different across images.

By transforming and normalising the data, we aim to reduce these two effects. Here we extract the intensities from the `counts` assay. Looking at CD3 which should be expressed in the majority of the T cells, the intensities are clearly very skewed, and it is difficult to see what is considered a CD3- cell, and what is a CD3+ cell. Further, we can clearly see some image-level batch effect, where across images, the intensity peaks differ drastically.

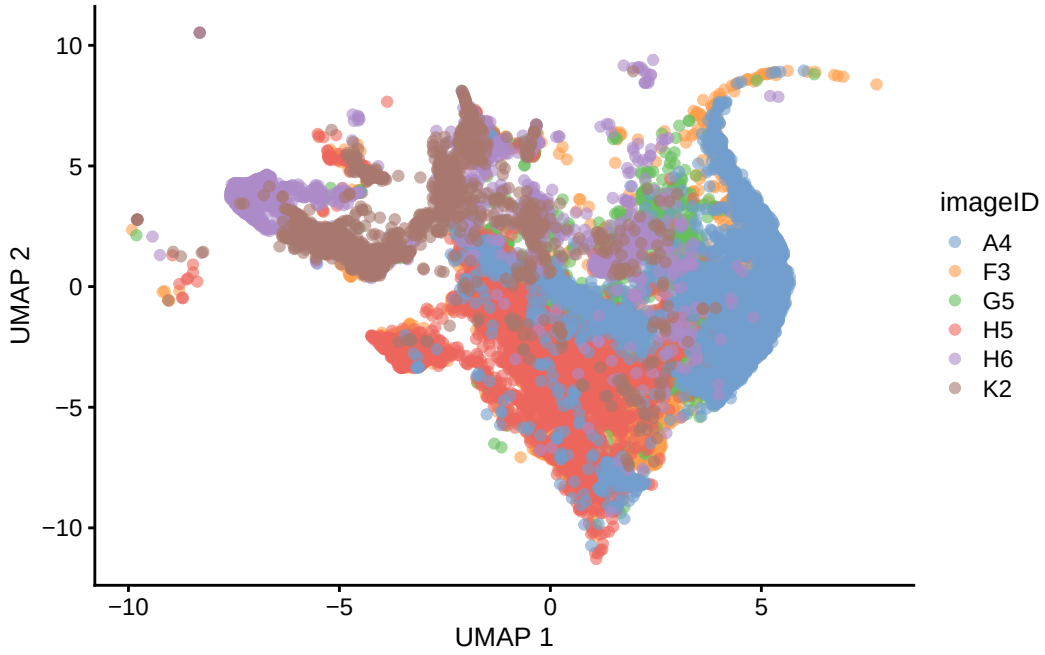
```
# Plot densities of CD3 for each image.
fergusonSPE |>
  join_features(features = rownames(fergusonSPE), shape = "wide", assay = "counts") |>
  ggplot(aes(x = CD3, colour = imageID)) +
  geom_density() +
  theme(legend.position = "none")
```




Another method of visualising batch effect is using a dimensionality reduction technique and visualising how the images separate out on a 2D plot. If no batch effect is expected, we should see the images largely overlap with each other.

```
# Usually we specify a subset of the original markers which are informative to separating ou
ct_markers <- c("podoplanin", "CD13", "CD31",
                "panCK", "CD3", "CD4", "CD8a",
                "CD20", "CD68", "CD16", "CD14",
                "HLADR", "CD66a")

set.seed(51773)
# Perform dimension reduction using UMAP.
fergusonSPE <- scater::runUMAP(
  fergusonSPE,
  subset_row = ct_markers,
  exprs_values = "counts"
```

```
)

# Select a subset of images to plot.
someImages <- unique(fergusonSPE$imageID)[c(1, 5, 10, 20, 30, 40)]

# UMAP by imageID.
scater::plotReducedDim(
  fergusonSPE[, fergusonSPE$imageID %in% someImages],
  dimred = "UMAP",
  colour_by = "imageID"
)
```




We can observe that from both our density plot and the UMAP, that there exists some level of batch effect in our dataset. `simpleSeg` also provides functionality for correcting image-level variability, using the `normalizeCells()` function.

In the `normalizeCells()` function, we specify the following parameters. `transformation` is an optional argument which specifies the function to be applied to the data. We do not apply an arcsinh transformation here, as we already apply a square root transform in the `simpleSeg()` function. `method = c("trim99", "mean", PC1")` is an optional argument which specifies the normalisation method/s to be performed. A comprehensive table of methods is provided below. `assayIn = "counts"` is a required argument which specifies what the assay you'll be taking the intensity data from is named. In our context, this is called `counts`.

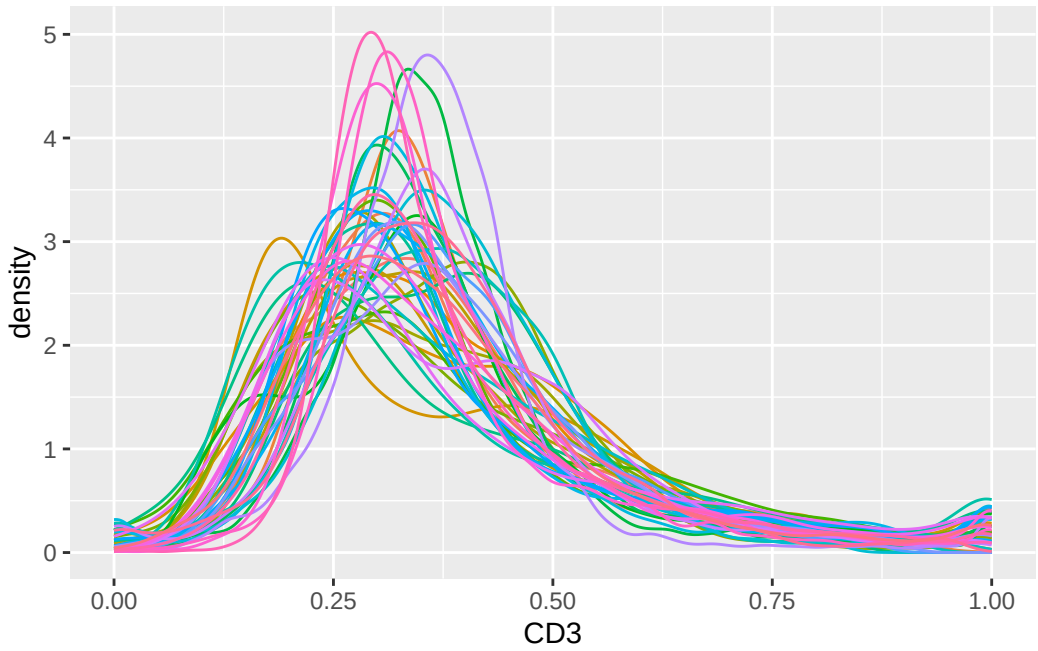
| Method | Description |
|---|---|
| "mean" | Divides the marker cellular marker intensities by their mean. |
| "minMax" | Subtracts the minimum value and scales markers between 0 and 1. |
| "trim99" | Sets the highest 1% of values to the value of the 99th percentile.' |
| "PC1" | Removes the 1st principal component) can be performed with one call of the function, in the order specified by the user. |

This modified data is then stored in the `norm` assay by default, but can be changed using the `assayOut` parameter.

```
# Leave out the nuclei markers from our normalisation process.
useMarkers <- rownames(fergusonSPE)[!rownames(fergusonSPE) %in% c("DNA1", "DNA2", "HH3")]

# Transform and normalise the marker expression of each cell type.
fergusonSPE <- normalizeCells(fergusonSPE,
                          markers = useMarkers,
                          transformation = NULL,
                          method = c("trim99", "mean", "PC1"),
                          assayIn = "counts",
                          cores = nCores
)
```

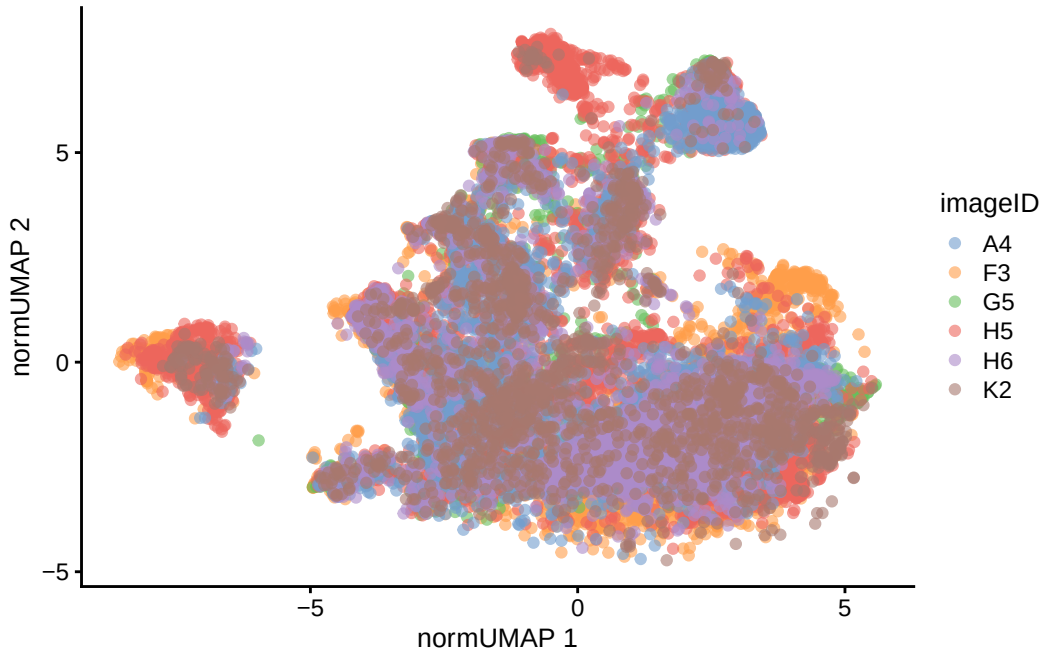
We can then plot the same density curve where we can see that this normalised data appears more bimodal where we can at least observe a clear CD3+ peak at 1.00, and a CD3- peak at around 0.3, and the images overlap much more strongly.

```
# Plot densities of CD3 for each image
fergusonSPE |>
  join_features(features = rownames(fergusonSPE), shape = "wide", assay = "norm") |>
  ggplot(aes(x = CD3, colour = imageID)) +
  geom_density() +
  theme(legend.position = "none")
```




We can also visualise the effect of normalisation on the UMAP, where our images now overlap much more strongly with each other.

```
set.seed(51773)
# Perform dimension reduction using UMAP.
fergusonSPE <- scater::runUMAP(
  fergusonSPE,
  subset_row = ct_markers,
  exprs_values = "norm",
  name = "normUMAP"
)

someImages <- unique(fergusonSPE$imageID)[c(1, 5, 10, 20, 30, 40)]

# UMAP by imageID.
scater::plotReducedDim(
  fergusonSPE[, fergusonSPE$imageID %in% someImages],
  dimred = "normUMAP",
  colour_by = "imageID"
)
```




## 2.3 scMerge: Combining multiple spatial datasets

A common question that pops up when analysing spatial datasets is:

Can I combine multiple spatial datasets?

## 2.4 sessionInfo

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Debian GNU/Linux 12 (bookworm)

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.21.so;  LAPACK version 3

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
```

```
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: Australia/Sydney
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] SpatialDatasets_1.4.0           SpatialExperiment_1.16.0
 [3] ExperimentHub_2.14.0            AnnotationHub_3.14.0
 [5] BiocFileCache_2.14.0            dbplyr_2.5.0
 [7] scater_1.34.0                   scuttle_1.16.0
 [9] simpleSeg_1.8.0                 ggplot2_3.5.1
[11] ttservice_0.4.1                 tidyr_1.3.1
[13] dplyr_1.1.4                     tidySingleCellExperiment_1.16.0
[15] SingleCellExperiment_1.28.1     SummarizedExperiment_1.36.0
[17] Biobase_2.66.0                  GenomicRanges_1.58.0
[19] GenomeInfoDb_1.42.0             IRanges_2.40.0
[21] S4Vectors_0.44.0                BiocGenerics_0.52.0
[23] MatrixGenerics_1.18.0           matrixStats_1.4.1

loaded via a namespace (and not attached):
  [1] RColorBrewer_1.1-3      rstudioapi_0.17.1       jsonlite_1.8.9
  [4] magrittr_2.0.3          spatstat.utils_3.1-1    ggbeeswarm_0.7.2
  [7] magick_2.8.5            farver_2.1.2            rmarkdown_2.29
 [10] zlibbioc_1.52.0         vctrs_0.6.5             memoise_2.0.1
 [13] RCurl_1.98-1.16         terra_1.7-83            svgPanZoom_0.3.4
 [16] tinytex_0.54            htmltools_0.5.8.1       S4Arrays_1.6.0
 [19] curl_6.0.1              BiocNeighbors_2.0.0     raster_3.6-30
 [22] Rhdf5lib_1.28.0         SparseArray_1.6.0       rhdf5_2.50.0
 [25] htmlwidgets_1.6.4       cachem_1.1.0            plotly_4.10.4
 [28] mime_0.12               lifecycle_1.0.4         pkgconfig_2.0.3
 [31] rsvd_1.0.5              Matrix_1.7-1            R6_2.5.1
 [34] fastmap_1.2.0           GenomeInfoDbData_1.2.13 shiny_1.9.1
 [37] digest_0.6.37           colorspace_2.1-1        AnnotationDbi_1.68.0
 [40] irlba_2.3.5.1           RSQLite_2.3.7           beachmat_2.22.0
 [43] labeling_0.4.3          filelock_1.0.3          fansi_1.0.6
 [46] nnls_1.6                httr_1.4.7              polyclip_1.10-7
 [49] abind_1.4-8             compiler_4.4.1          bit64_4.5.2
```

```
 [52] withr_3.0.2             tiff_0.1-12             BiocParallel_1.40.0
 [55] DBI_1.2.3               viridis_0.6.5           HDF5Array_1.34.0
 [58] cytomapper_1.18.0       rappdirs_0.3.3          DelayedArray_0.32.0
 [61] rjson_0.2.23            tools_4.4.1             vipor_0.4.7
 [64] beeswarm_0.4.0          httpuv_1.6.15           glue_1.8.0
 [67] EBImage_4.48.0          rhdf5filters_1.18.0     promises_1.3.0
 [70] grid_4.4.1              generics_0.1.3          gtable_0.3.6
 [73] spatstat.data_3.1-2     data.table_1.16.2       BiocSingular_1.22.0
 [76] ScaledMatrix_1.14.0     sp_2.1-4                utf8_1.2.4
 [79] XVector_0.46.0          RcppAnnoy_0.0.22        spatstat.geom_3.3-3
 [82] BiocVersion_3.20.0      ggrepel_0.9.6           pillar_1.9.0
 [85] stringr_1.5.1           later_1.3.2             lattice_0.22-6
 [88] bit_4.5.0               deldir_2.0-4            tidyselect_1.2.1
 [91] locfit_1.5-9.10         Biostrings_2.74.0       knitr_1.49
 [94] gridExtra_2.3           svglite_2.1.3           xfun_0.49
 [97] shinydashboard_0.7.2    stringi_1.8.4           UCSC.utils_1.2.0
[100] fftwtools_0.9-11        yaml_2.3.10             lazyeval_0.2.2
[103] evaluate_1.0.1          codetools_0.2-20        tibble_3.2.1
[106] BiocManager_1.30.25     cli_3.6.3               uwot_0.2.2
[109] xtable_1.8-4            systemfonts_1.1.0       munsell_0.5.1
[112] Rcpp_1.0.13-1           png_0.1-8               spatstat.univar_3.1-1
[115] parallel_4.4.1          ellipsis_0.3.2          blob_1.2.4
[118] jpeg_0.1-10             bitops_1.0-9            viridisLite_0.4.2
[121] scales_1.3.0            purrr_1.0.2             crayon_1.5.3
[124] rlang_1.1.4             cowplot_1.1.3           KEGGREST_1.46.0
```

# 3 Cell Annotation

Steps:

1. Rationale behind clustering vs annotation
2. How to cluster cells (FuseSOM)
3. How to annotate clusters (pheatmap)
4. How to annotate cells (scClassify)
5. How to select a reference dataset (scClassify)

## 3.1 Which package should I use? Clustering vs Annotation

Labeling the identity of your cells is a key step in any spatial processing protocol in order to determine differential cell type compositions and changes which occur in specific cell types during disease. However, the method by which this is done can differ from study to study. Here, we provide two packages capable of either cell type clustering (`FuseSOM`) or cell type annotation (`scClassify`).

Clustering is an unsupervised method of labelling cells. This means that an algorithm separates out clusters of cells based purely off their marker expression, and the subsequent labeling of these clusters must be done with some biological domain knowledge. Cell annotation is a supervised method which requires a separate, reference dataset. The algorithm then uses that reference dataset to determine the identity of each of your cell types, thereby labelling your cells in the process. There are advantages and disadvantages to both, and the choice of one or the other will be discussed in this chapter. First we'll walk through how to run both of these packages, and then we'll discuss how to choose between `FuseSOM` and `scClassify`

## 3.2 FuseSOM: Cell clustering

A correlation based multiview self organizing map for the characterization of cell types (`FuseSOM`) is a tool for unsupervised clustering. `FuseSOM` is robust and achieves high accuracy by combining a `Self Organizing Map` architecture and a `Multiview` integration of correlation based metrics to cluster highly multiplexed in situ imaging cytometry assays. The `FuseSOM` pipeline has been streamlined and accepts currently used data structures including `SingleCellExperiment` and `SpatialExperiment` objects as well as `DataFrames`.

### 3.2.1 FuseSOM Matrix Input

If you have a matrix containing expression data that was QCed and normalised by some other tool, the next step is to run the FuseSOM algorithm.This can be done by calling the runFuseSOM() function which takes in the matrix of interest where the columns are markers and the rows are observations, the makers of interest (if this is not provided, it is assumed that all columns are markers), and the number of clusters.

```
# load FuseSOM
library(FuseSOM)
```

Next we will load in the Risom et al dataset and run it through the FuseSOM pipeline. This dataset profiles the spatial landscape of ductal carcinoma in situ (DCIS), which is a pre-invasive lesion that is thought to be a precursor to invasive breast cancer (IBC). The key conclusion of this manuscript (amongst others) is that spatial information about cells can be used to predict disease progression in patients.We will also be using the markers used in the original study.

```
# load in the data
data("risom_dat")

# define the markers of interest
risomMarkers <- c('CD45','SMA','CK7','CK5','VIM','CD31','PanKRT','ECAD',
                  'Tryptase','MPO','CD20','CD3','CD8','CD4','CD14','CD68','FAP',
                  'CD36','CD11c','HLADRDPDQ','P63','CD44')

# we will be using the manual_gating_phenotype as the true cell type to gauge
# performance
names(risom_dat)[names(risom_dat) == 'manual_gating_phenotype'] <- 'CellType'
```

Now that we have loaded the data and define the markers of interest. We can run the FuseSOM algorithm. We have provided a function runFuseSOM that runs the pipeline from top to bottom and returns the cluster labels as well as the Self Organizing Map model.

```
risomRes <- runFuseSOM(data = risom_dat, markers = risomMarkers,
                       numClusters = 23)
```

```
You have provided a dataset of class data.frame

Everything looks good. Now running the FuseSOM algorithm

Now Generating the Self Organizing Map Grid
```

```
Optimal Grid Size is: 8

Now Running the Self Organizing Map Model

Now Clustering the Prototypes

Loading required namespace: fastcluster

Now Mapping Clusters to the Original Data

The Prototypes have been Clustered and Mapped Successfully

The FuseSOM algorithm has completed successfully
```

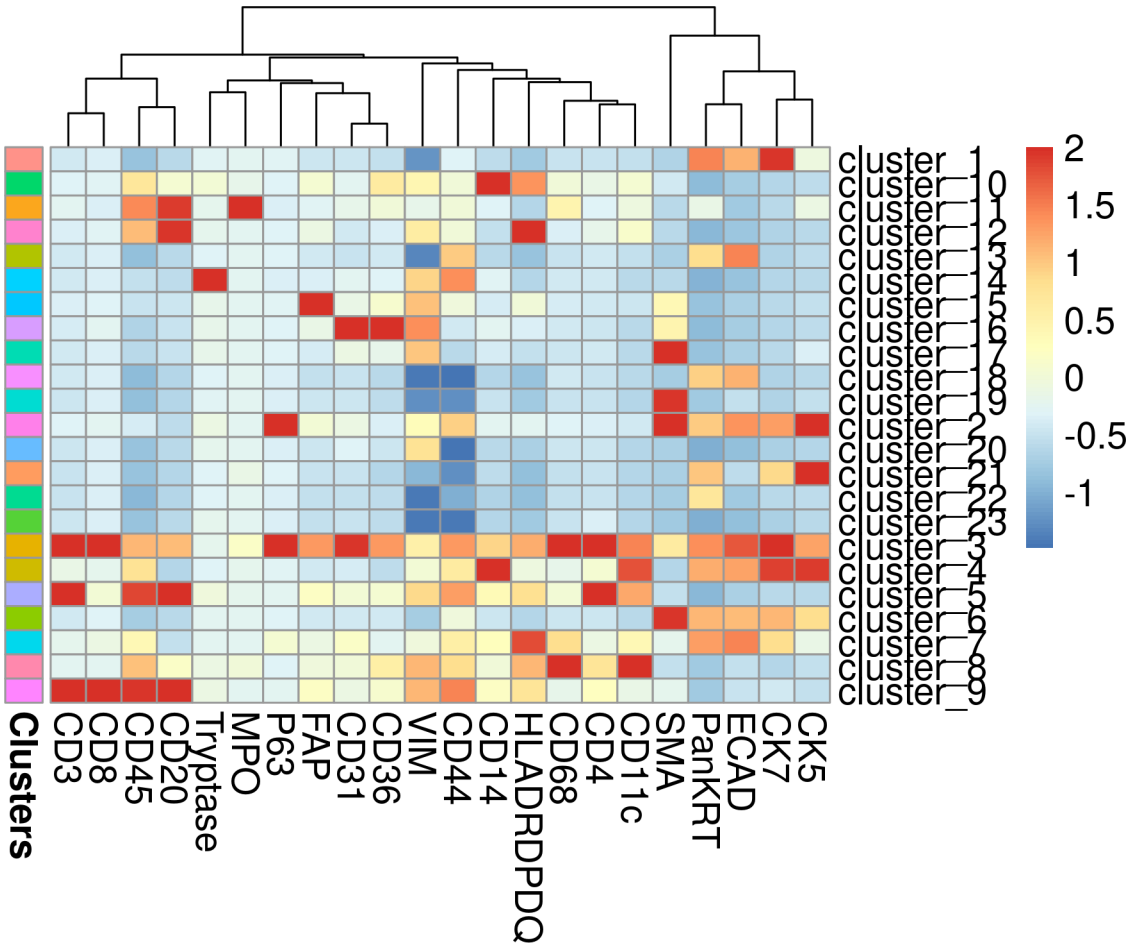
Lets look at the distribution of the clusters.

```
# get the distribution of the clusters
table(risomRes$clusters)/sum(table(risomRes$clusters))
```

```
  cluster_1  cluster_10  cluster_11  cluster_12  cluster_13  cluster_14
0.323602021 0.035968538 0.005439775 0.021443334 0.061100586 0.026596050
 cluster_15  cluster_16  cluster_17  cluster_18  cluster_19   cluster_2
0.020582156 0.032624297 0.024931106 0.076128143 0.015802618 0.014927087
 cluster_20  cluster_21  cluster_22  cluster_23   cluster_3   cluster_4
0.049962682 0.009185900 0.051771156 0.066913538 0.004923068 0.014108968
  cluster_5   cluster_6   cluster_7   cluster_8   cluster_9
0.040776783 0.064444827 0.020854863 0.010032725 0.007879780
```

Looks like `cluster_1` has about 32% of the cells which is interesting. Next, lets generate a heatmap of the marker expression for each cluster.

```
risomHeat <- FuseSOM::markerHeatmap(data = risom_dat, markers = risomMarkers,
                                    clusters = risomRes$clusters, clusterMarkers = TRUE)
```




### 3.2.2 Using `FuseSOM` to estimate the number of clusters

`FuseSOM` also provides functionality for estimating the number of clusters in a dataset using three classes of methods including:

1. Discriminant based method.
    - A method developed in house based on discriminant based maximum clusterability projection pursuit
2. Distance based methods which includes:
    - The Gap Statistic
    - The Jump Statistic
    - The Slope Statistic
    - The Within Cluster Dissimilarity Statistic
    - The Silhouette Statistic

We can estimate the number of clusters using the `estimateNumCluster`. Run `help(estimateNumCluster)` to see it's complete functionality.

```
# lets estimate the number of clusters using all the methods
# original clustering has 23 clusters so we will set kseq from 2:25
# we pass it the som model generated in the previous step
risomKest <- estimateNumCluster(data = risomRes$model, kSeq = 2:25,
                                method = c("Discriminant", "Distance"))
```

```
Now Computing the Number of Clusters using Discriminant Analysis
```

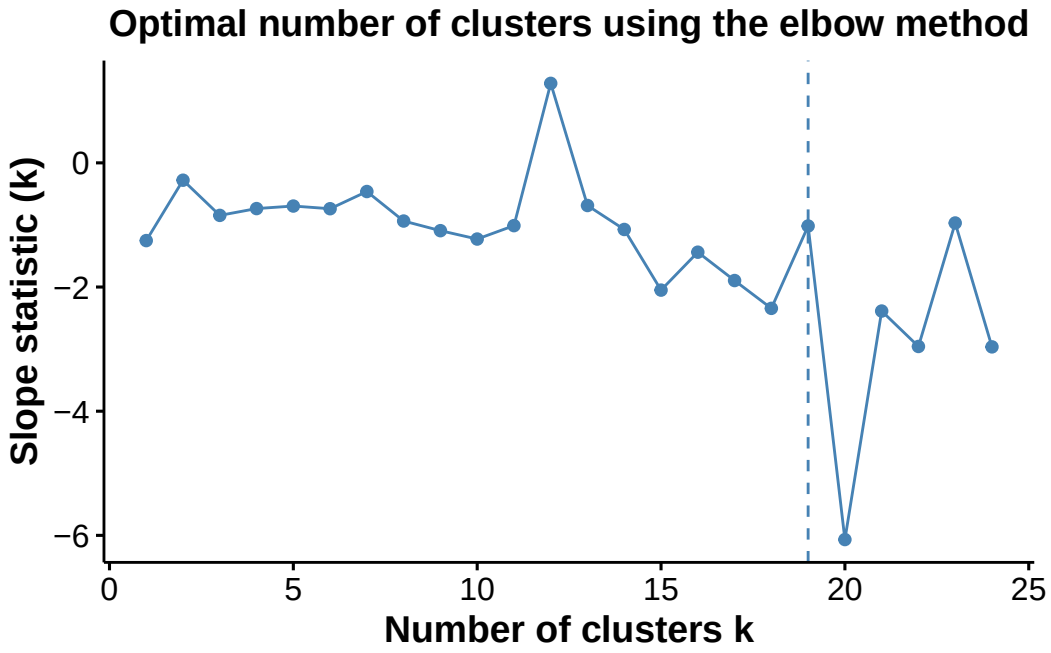
```
Now Computing The Number Of Clusters Using Distance Analysis
```

We can then use this result to determine the best number of clusters for this dataset based on the different metrics. The `FuseSOM` package provides a plotting function (`optiPlot`) which generates an elbow plot with the optimal value for the number of clusters for the distance based methods. See below

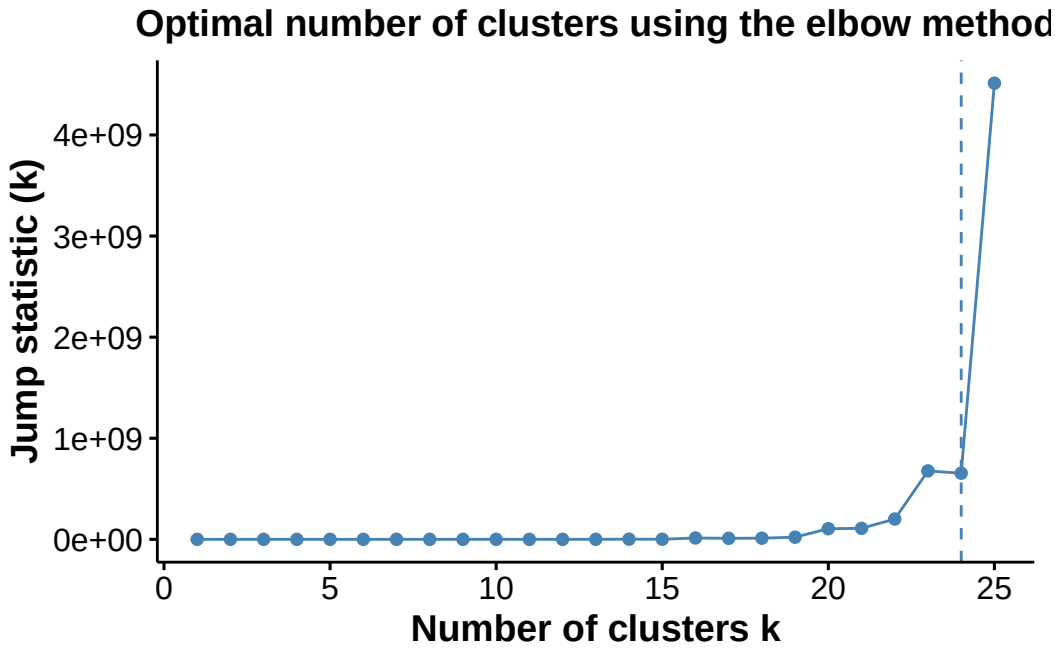
```
# what is the best number of clusters determined by the discriminant method?
# optimal number of clusters according to the discriminant method is 7
risomKest$Discriminant
```

```
[1] 10
```

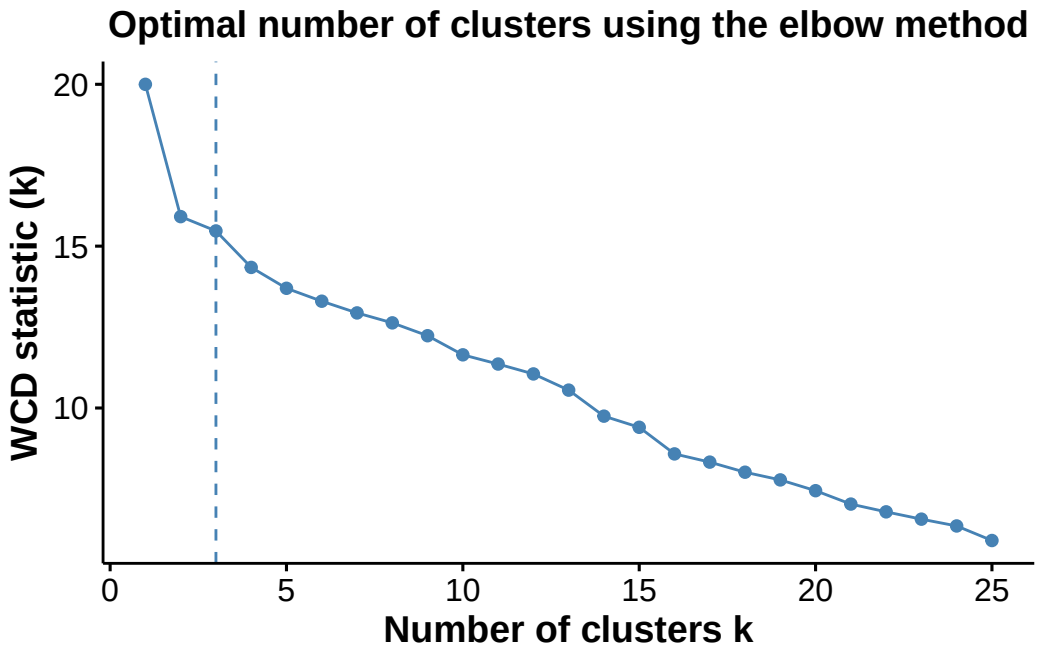
```
# we can plot the results using the optiplot function
pSlope <- optiPlot(risomKest, method = 'slope')
pSlope
```



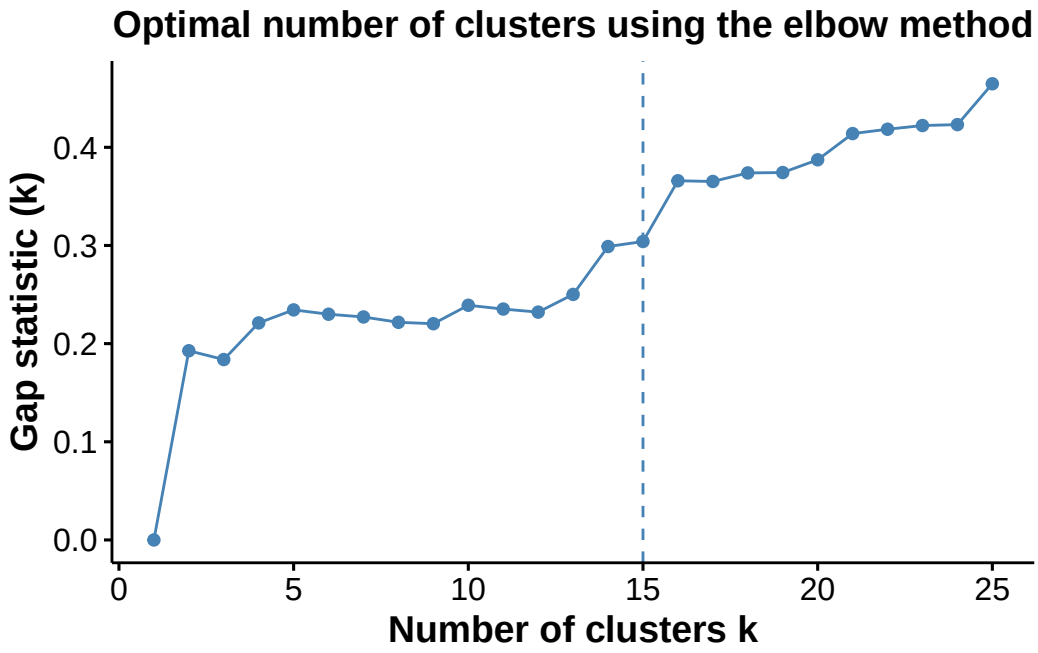

```
pJump <- optiPlot(risomKest, method = 'jump')
pJump
```



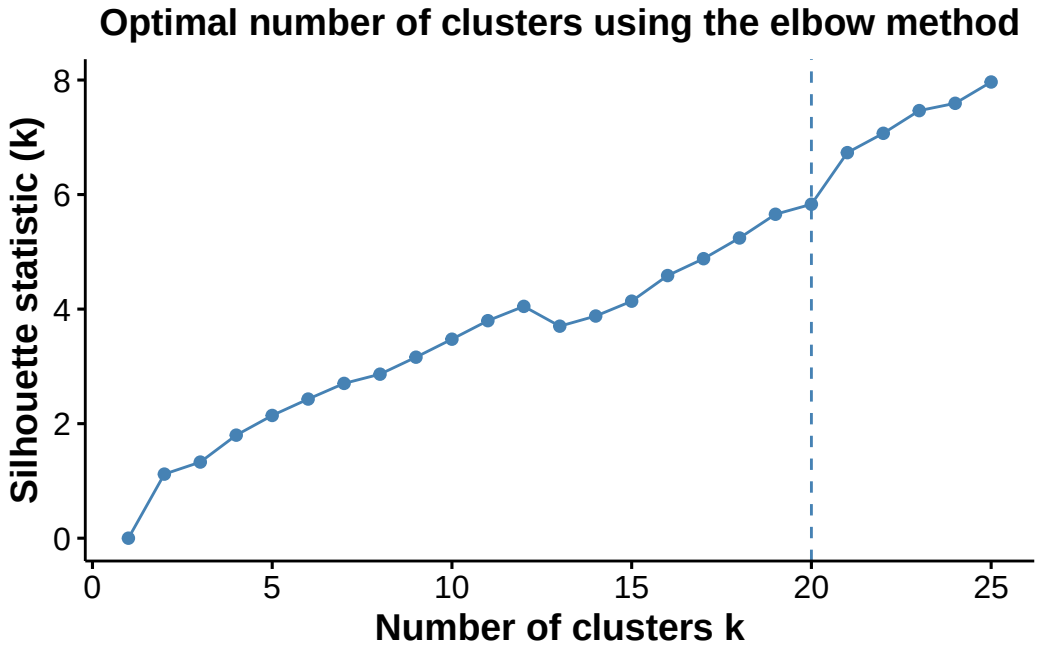
```
pWcd <- optiPlot(risomKest, method = 'wcd')
pWcd
```




```
pGap <- optiPlot(risomKest, method = 'gap')
pGap
```



```
pSil <- optiPlot(risomKest, method = 'silhouette')
pSil
```




From the plots, we see that the `Jump` statistics almost perfectly capture the number of clusters. The `Gap` method is a close second with 15 clusters. All the other methods significantly underestimates the number of clusters.

### 3.2.3 `FuseSOM` Single Cell Epxeriment object as input.

The `FuseSOM` algorithm is also equipped to take in a `SingleCellExperiment` object as input. The results of the pipeline will be written to either the metada or the colData fields. See below.

First we create a `SingleCellExperiment` object

```
library(SingleCellExperiment)

# create a singelcellexperiment object
colDat <- risom_dat[, setdiff(colnames(risom_dat), risomMarkers)]
sce <- SingleCellExperiment(assays = list(counts = t(risom_dat)),
                                  colData = colDat)

sce
```

```
class: SingleCellExperiment
dim: 23 69672
metadata(0):
assays(1): counts
rownames(23): CD45 SMA ... CD44 CellType
rowData names(0):
colnames: NULL
colData names(1): X
reducedDimNames(0):
mainExpName: NULL
altExpNames(0):
```

Next we pass it to the `runFuseSOM()` function. Here, we can provide the assay in which the data is stored and what name to store the clusters under in the colData section. Note that the `Self Organizing Map` that is generated will be stored in the metadata field.

```
risomRessce <- runFuseSOM(sce, markers = risomMarkers, assay = 'counts',
                      numClusters = 23, verbose = FALSE)
```

```
You have provided a dataset of class SingleCellExperiment

Everything looks good. Now running the FuseSOM algorithm

Now Generating the Self Organizing Map Grid

Optimal Grid Size is: 8

Now Running the Self Organizing Map Model

Now Clustering the Prototypes

Now Mapping Clusters to the Original Data

The Prototypes have been Clustered and Mapped Successfully

The FuseSOM algorithm has completed successfully
```

```
colnames(colData(risomRessce))
```

```
[1] "X"        "clusters"
```

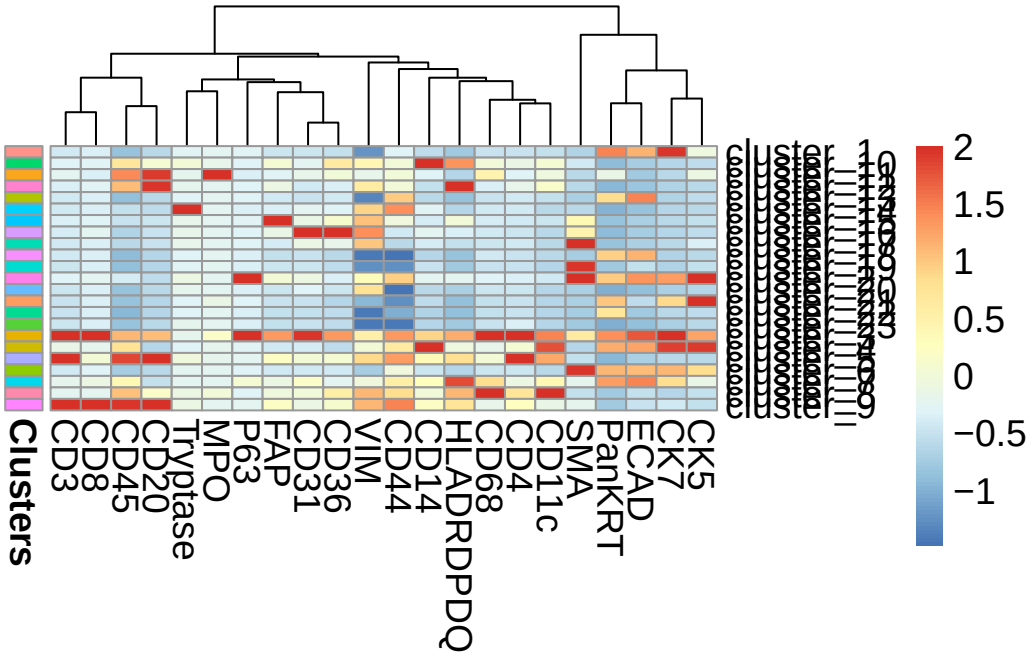
```
names(metadata(risomRessce))
```

```
[1] "SOM"
```

Notice how the there is now a clusters column in the colData and SOM field in the metadata. You can run this function again with a new set of cluster number. If you provide a new name for the clusters, it will be stored under that new column, else, it will overwrite the current clusters column. Running it again on the same object will overwrite the SOM field in the metadata.

Just like before, lets plot the heatmap of the resulting clusters across all markers.

```
data <- risom_dat[, risomMarkers] # get the original data used
clusters <- colData(risomRessce)$clusters # extract the clusters from the sce object
# generate the heatmap
risomHeatsce <- markerHeatmap(data = risom_dat, markers = risomMarkers,
                              clusters = clusters, clusterMarkers = TRUE)
```



### 3.2.4 Using `FuseSOM` to estimate the number of clusters for single cell experiment objects

Just like before, we can estimate the number of clusters

```
# lets estimate the number of clusters using all the methods
# original clustering has 23 clusters so we will set kseq from 2:25
# now we pass it a singlecellexperiment object instead of the som model as before
# this will return a singelcellexperiment object where the metatdata contains the
# cluster estimation information
risomRessce <- estimateNumCluster(data = risomRessce, kSeq = 2:25,
                                  method = c("Discriminant", "Distance"))
```

```
You have provided a dataset of class: SingleCellExperiment

Now Computing the Number of Clusters using Discriminant Analysis

Now Computing The Number Of Clusters Using Distance Analysis
```

```
names(metadata(risomRessce))
```

```
[1] "SOM"               "clusterEstimation"
```

Notice how the metadata now contains a `clusterEstimation` field which holds the results from the `estimateNumCluster()` function

We can assess the results in a similar fashion as before

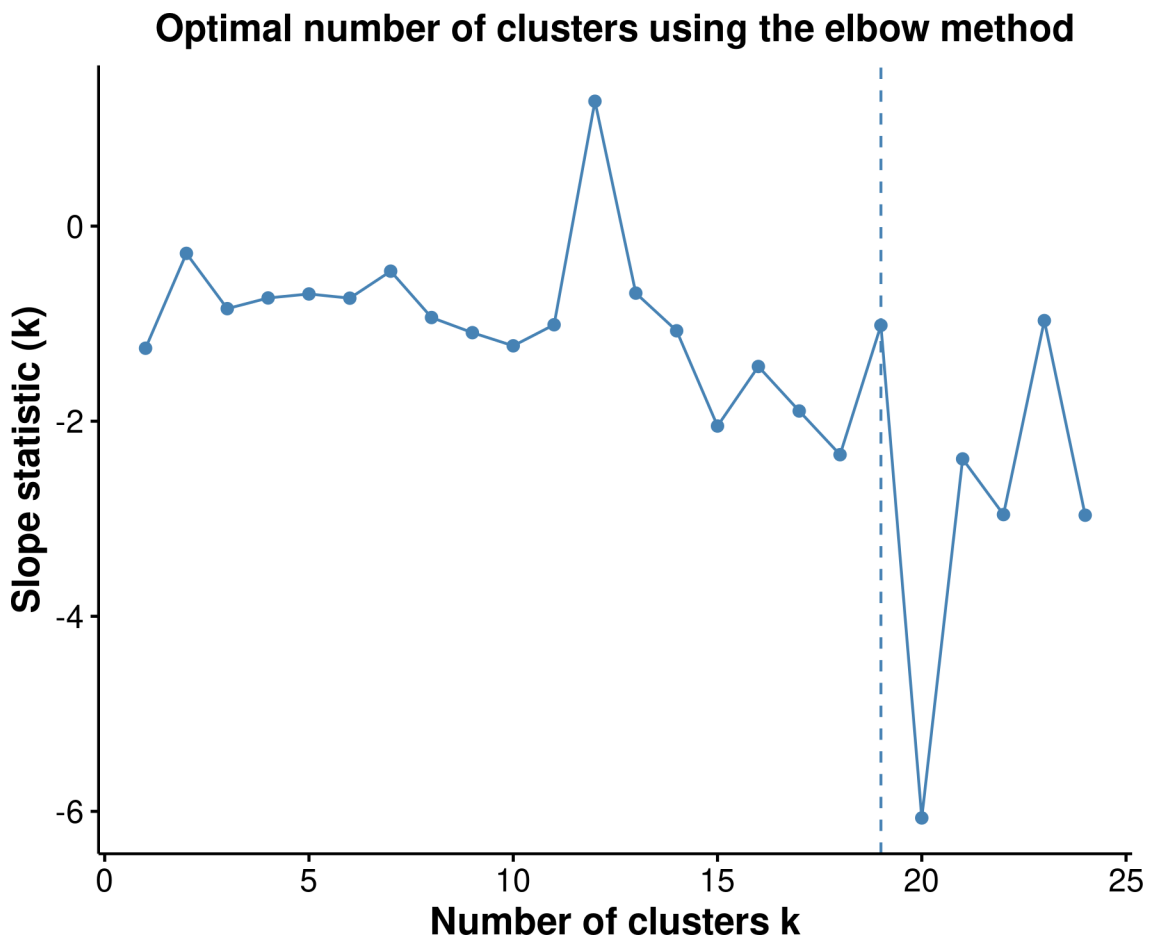
```
# what is the best number of clusters determined by the discriminant method?
# optimal number of clusters according to the discriminant method is 8
metadata(risomRessce)$clusterEstimation$Discriminant
```

```
[1] 7
```

```
# we can plot the results using the optiplot function
pSlope <- optiPlot(risomRessce, method = 'slope')
```

```
You have provided a dataset of class: SingleCellExperiment
```

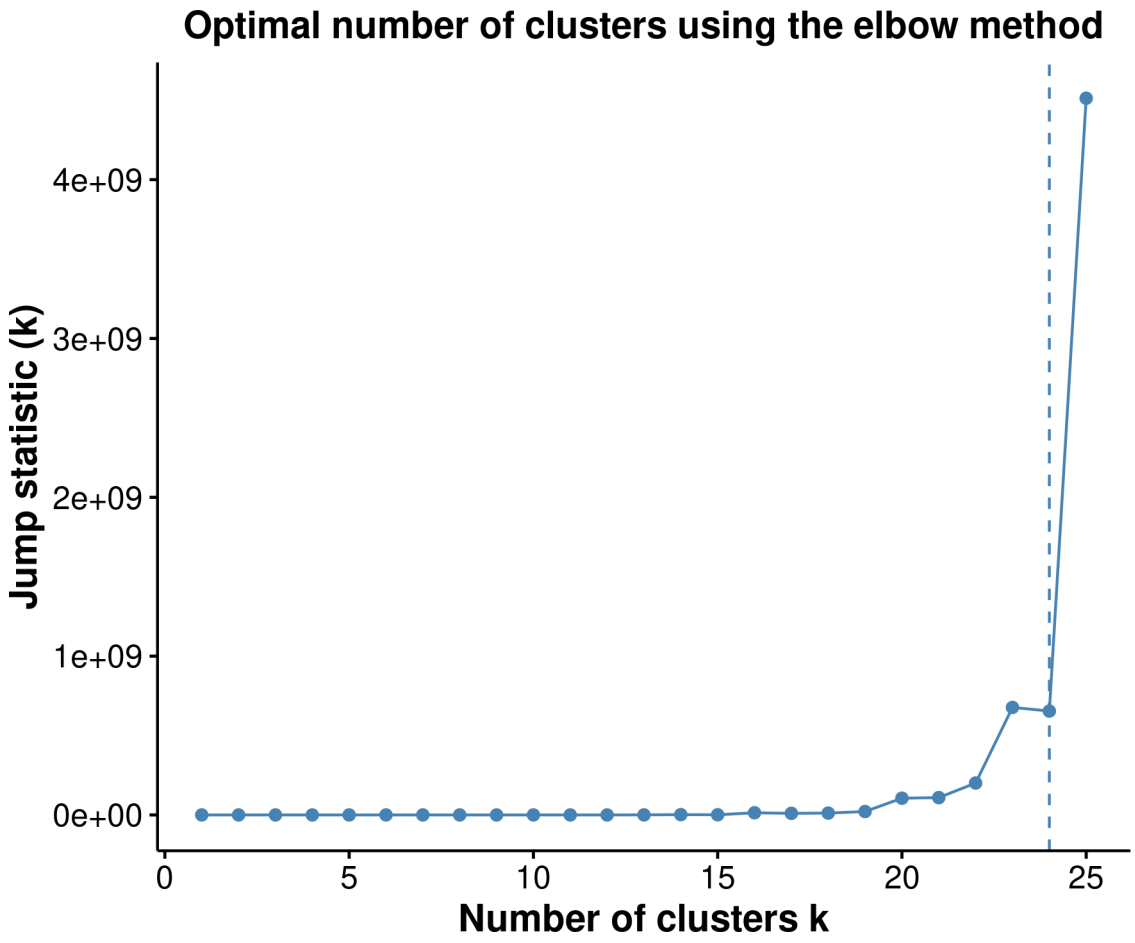
```
pSlope
```




```
pJump <- optiPlot(risomRessce, method = 'jump')
```

```
You have provided a dataset of class: SingleCellExperiment
```

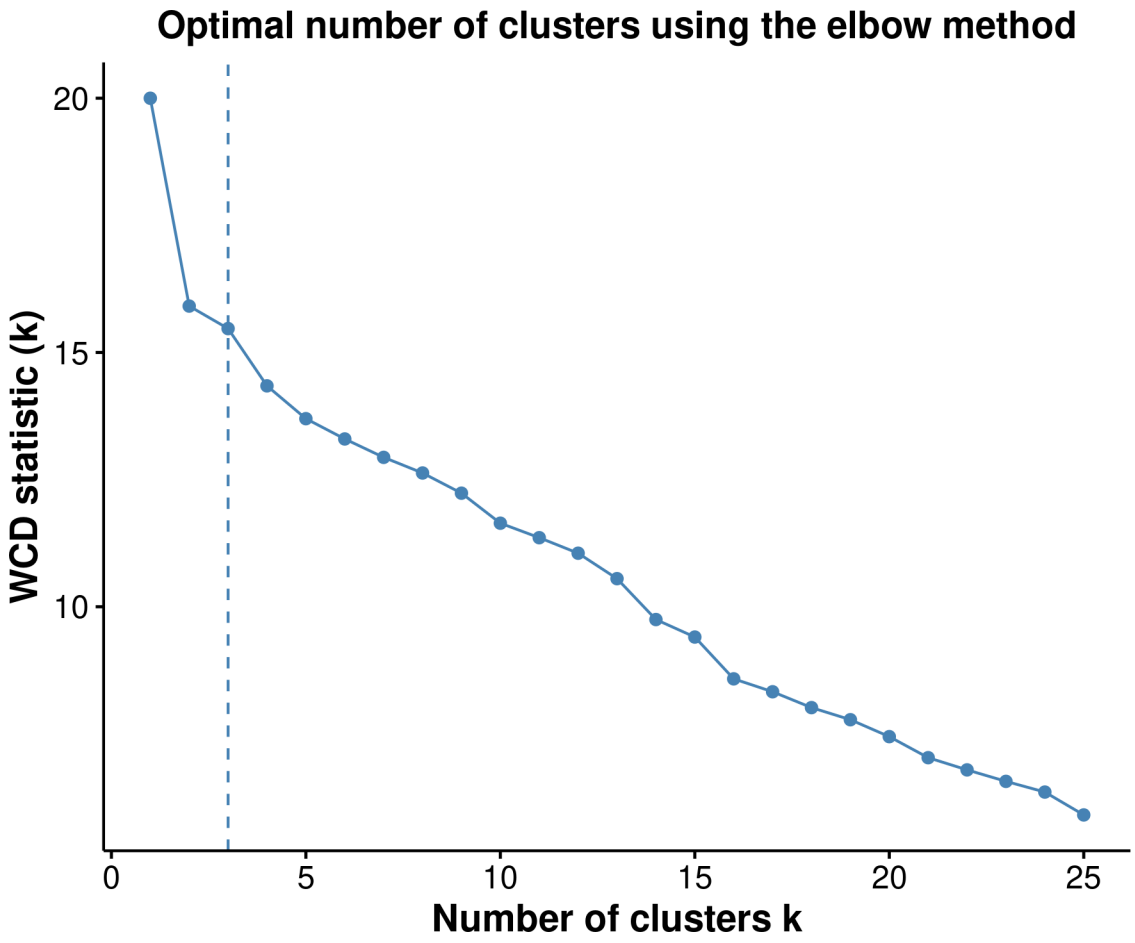
```
pJump
```




```
pWcd <- optiPlot(risomRessce, method = 'wcd')
```

```
You have provided a dataset of class: SingleCellExperiment
```

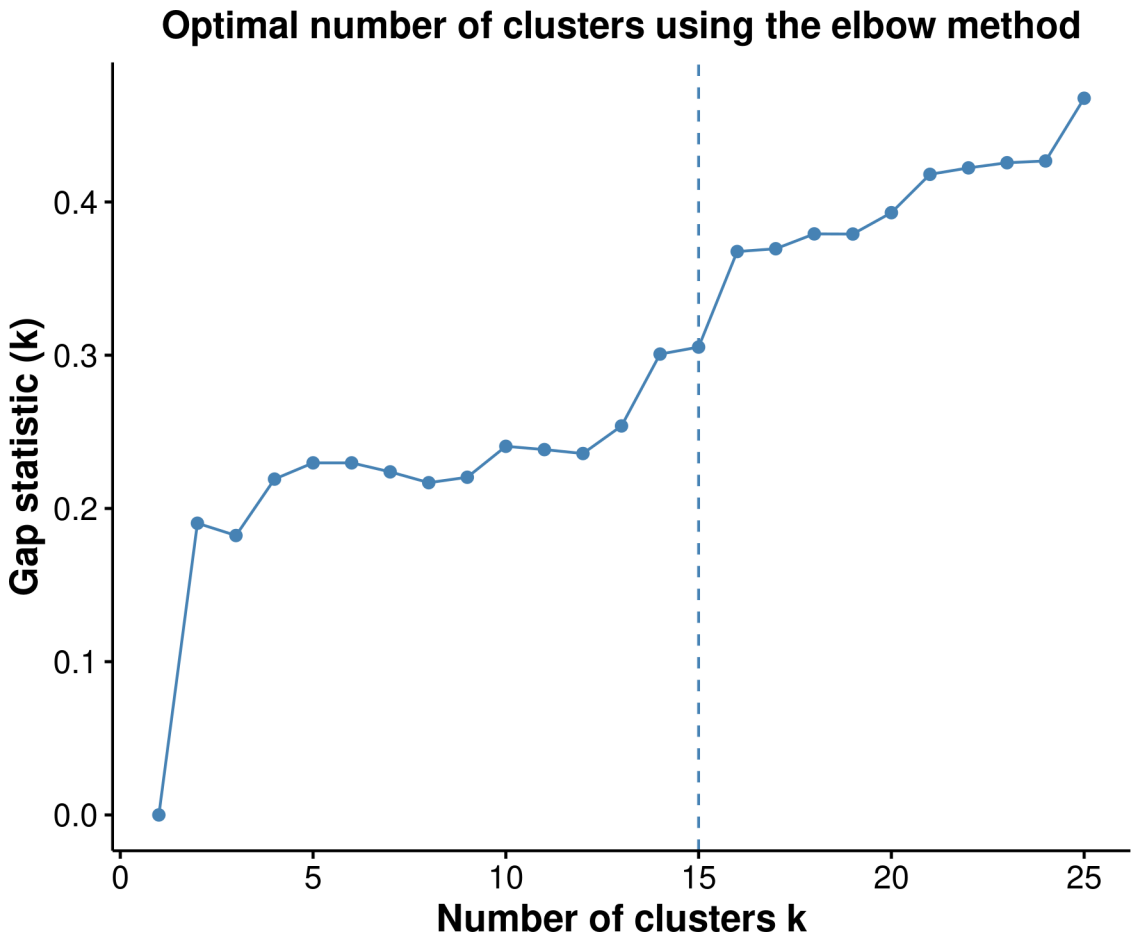
```
pWcd
```




```
pGap <- optiPlot(risomRessce, method = 'gap')
```

```
You have provided a dataset of class: SingleCellExperiment
```

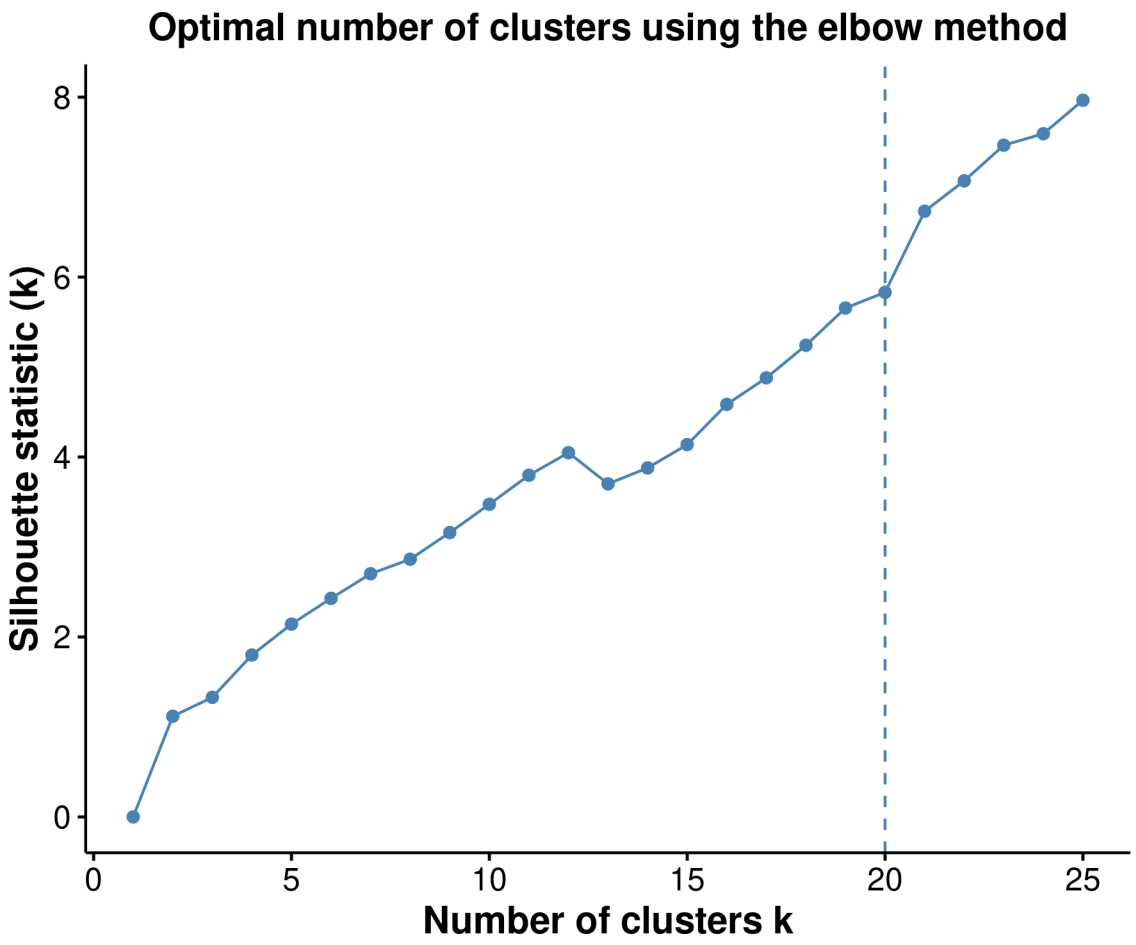
```
pGap
```




```
pSil <- optiPlot(risomRessce, method = 'silhouette')
```

```
You have provided a dataset of class: SingleCellExperiment
```

```
pSil
```




Again, we see that the `Jump` statistics almost perfectly capture the number of clusters. The `Gap` method is a close second with 15 clusters. All the other methods significantly underestimates the number of clusters.

## 3.3 scClassify: Cell annotation

## 3.4 Choosing between clustering and annotation

## 3.5 sessionInfo

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Debian GNU/Linux 12 (bookworm)

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.21.so;  LAPACK version 3

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: Australia/Sydney
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] SingleCellExperiment_1.28.1 SummarizedExperiment_1.36.0
 [3] Biobase_2.66.0              GenomicRanges_1.58.0
 [5] GenomeInfoDb_1.42.0         IRanges_2.40.0
 [7] S4Vectors_0.44.0            BiocGenerics_0.52.0
 [9] MatrixGenerics_1.18.0       matrixStats_1.4.1
[11] FuseSOM_1.8.0

loaded via a namespace (and not attached):
 [1] mnormt_2.1.1             permute_0.9-7            rlang_1.1.4
 [4] magrittr_2.0.3           compiler_4.4.1           mgcv_1.9-1
 [7] flexmix_2.3-19           analogue_0.17-7          vctrs_0.6.5
[10] stringr_1.5.1            pkgconfig_2.0.3          crayon_1.5.3
[13] fastmap_1.2.0            backports_1.5.0          magick_2.8.5
[16] XVector_0.46.0           labeling_0.4.3           utf8_1.2.4
[19] rmarkdown_2.29           UCSC.utils_1.2.0         tinytex_0.54
[22] purrr_1.0.2              coop_0.6-3               xfun_0.49
[25] modeltools_0.2-23        zlibbioc_1.52.0          jsonlite_1.8.9
[28] DelayedArray_0.32.0      fpc_2.2-13               psych_2.4.6.26
[31] broom_1.0.7              parallel_4.4.1           prabclus_2.3-4
[34] cluster_2.1.6            R6_2.5.1                 profileModel_0.6.1
[37] stringi_1.8.4            RColorBrewer_1.1-3       car_3.1-3
```

```
[40] diptest_0.77-1          Rcpp_1.0.13-1            knitr_1.49
[43] Matrix_1.7-1            splines_4.4.1            nnet_7.3-19
[46] tidyselect_1.2.1        rstudioapi_0.17.1        abind_1.4-8
[49] vegan_2.6-8             brglm_0.7.2              lattice_0.22-6
[52] tibble_3.2.1            withr_3.0.2              evaluate_1.0.1
[55] gridGraphics_0.5-1      proxy_0.4-27             kernlab_0.9-33
[58] mclust_6.1.1            pillar_1.9.0             ggpubr_0.6.0
[61] carData_3.0-5           generics_0.1.3           sp_2.1-4
[64] ggplot2_3.5.1           munsell_0.5.1            scales_1.3.0
[67] princurve_2.1.6         class_7.3-22             glue_1.8.0
[70] pheatmap_1.0.12         tools_4.4.1              robustbase_0.99-4-1
[73] ggsignif_0.6.4          fs_1.6.5                 fastcluster_1.2.6
[76] grid_4.4.1              tidyr_1.3.1              colorspace_2.1-1
[79] nlme_3.1-166            GenomeInfoDbData_1.2.13  Formula_1.2-5
[82] cli_3.6.3               DataVisualizations_1.3.2 FCPS_1.3.4
[85] fansi_1.0.6             S4Arrays_1.6.0           dplyr_1.1.4
[88] DEoptimR_1.1-3          gtable_0.3.6             rstatix_0.7.2
[91] yulab.utils_0.1.8       digest_0.6.37            SparseArray_1.6.0
[94] ggplotify_0.1.2         farver_2.1.2             htmltools_0.5.8.1
[97] lifecycle_1.0.4         httr_1.4.7               MASS_7.3-61
```

# 4 Cell localisation

Now that we've finished performing all of our upstream preprocessing (segmentation, quality control, and annotation), we can now begin dissecting out interesting findings for our datasets. One of the primary motivations behind pursuing a spatial technology as opposed to a space-agnostic technology such as single-cell RNA sequencing, is that we are more able to tease out whether changes are occurring spatially, i.e. are two cell types closer together in a disease state vs a non-disease state. Whilst these changes are usually visually obvious, to quantify localisation and dispersion relationships, more advanced statistical modelling is required.

```
# load required packages
library(spicyR)
library(Statial)
library(ggplot2)
library(SpatialExperiment)
library(SpatialDatasets)
library(imcRtools)
library(dplyr)
library(survival)
library(tibble)
library(treekoR)
library(ggsurvfit)

nCores <- 2
```

## 4.1 spicyR: Cell localisation

This section provides step-by-step instructions on assessing how the localisation of different cell types changes across different disease conditions.

We use the (**keren2018?**) breast cancer dataset to compare the spatial distribution of immune cells in individuals with different levels of tumour infiltration (cold and compartmentalised).

The data is stored as a `SpatialExperiment` object and contains single-cell spatial data from 41 images.

```
kerenSPE <- SpatialDatasets::spe_Keren_2018()
```

The cell types in this dataset includes 11 immune cell types (double negative CD3 T cells, CD4 T cells, B cells, monocytes, macrophages, CD8 T cells, neutrophils, natural killer cells, dendritic cells, regulatory T cells), 2 structural cell types (endothelial, mesenchymal), 2 tumour cell types (keratin+ tumour, tumour) and one unidentified category. Below we've provided detailed information on the statistical backend of `spicyR`.

### 4.1.1 Linear modelling

To investigate changes in localisation between two different cell types, we measure the level of localisation between two cell types by modelling with the L-function. The L-function is a variance-stabilised K-function given by the equation

$$\widehat{L_{ij}}(r) = \sqrt{\frac{\widehat{K_{ij}}(r)}{\pi}}$$

with $\widehat{K_{ij}}$ defined as

$$\widehat{K_{ij}}(r) = \frac{|W|}{n_i n_j} \sum_{n_i} \sum_{n_j} 1\{d_{ij} \leq r\} e_{ij}(r)$$

where $\widehat{K_{ij}}$ summarises the degree of co-localisation of cell type $j$ with cell type $i$, $n_i$ and $n_j$ are the number of cells of type $i$ and $j$, $|W|$ is the image area, $d_{ij}$ is the distance between two cells and $e_{ij}(r)$ is an edge correcting factor.

Specifically, the mean difference between the experimental function and the theoretical function is used as a measure for the level of localisation, defined as

$$u = \sum_{r'=r_{\min}}^{r_{\max}} \widehat{L}_{ij,\text{Experimental}}(r') - \widehat{L}_{ij,\text{Poisson}}(r')$$

where $u$ is the sum is taken over a discrete range of $r$ between $r_{\min}$ and $r_{\max}$. Differences of the statistic $u$ between two conditions is modelled using a weighted linear model.

### 4.1.2 Test for changes in localisation for a specific pair of cells

Firstly, we can test whether one cell type tends to be more localised with another cell type in one condition compared to the other. This can be done using the `spicy()` function, where we specify the `condition` parameter.

In this example, we want to see whether or not neutrophils (`to`) tend to be found around CD8 T cells (`from`) in compartmentalised tumours compared to cold tumours. Given that there are 3 conditions, we can specify the desired conditions by setting the order of our `condition` factor. `spicy` will choose the first level of the factor as the base condition and the second level as the comparison condition. `spicy` will also naturally coerce the `condition` column into a factor if it is not already a factor. The column containing cell type annotations can be specified using the `cellTypeCol` argument. By default, `spicy` uses the column named `cellType` in the `SpatialExperiment` object.

```
spicyTestPair <- spicy(
  kerenSPE,
  condition = "tumour_type",
  from = "CD8_T_cell",
  to = "Neutrophils"
)

topPairs(spicyTestPair)
```

```
                        intercept coefficient      p.value   adj.pvalue
CD8_T_cell__Neutrophils  -109.081    112.0185 2.166646e-05 2.166646e-05
                              from          to
CD8_T_cell__Neutrophils CD8_T_cell Neutrophils
```

We obtain a `spicy` object which details the results of the modelling performed. The `topPairs()` function can be used to obtain the associated coefficients and p-value.

As the `coefficient` in `spicyTestPair` is positive, we find that neutrophils are significantly more likely to be found near CD8 T cells in the compartmentalised tumours group compared to the cold tumour group.

### 4.1.3 Test for changes in localisation for all pairwise cell combinations

We can perform what we did above for all pairwise combinations of cell types by excluding the `from` and `to` parameters in `spicy()`.

```
spicyTest <- spicy(
  kerenSPE,
  condition = "tumour_type"
)

topPairs(spicyTest)
```

```
                             intercept coefficient      p.value   adj.pvalue
Macrophages__dn_T_CD3         56.446064   -50.08474 1.080273e-07 3.035568e-05
dn_T_CD3__Macrophages         54.987151   -48.38664 2.194018e-07 3.082595e-05
Macrophages__DC_or_Mono       73.239404   -59.90361 5.224660e-06 4.893765e-04
DC_or_Mono__Macrophages       71.777087   -58.46833 7.431172e-06 5.220399e-04
dn_T_CD3__dn_T_CD3           -63.786032   100.61010 2.878804e-05 1.208706e-03
Neutrophils__dn_T_CD3        -63.141840    69.64356 2.891872e-05 1.208706e-03
dn_T_CD3__Neutrophils        -63.133725    70.15508 3.011012e-05 1.208706e-03
DC__Macrophages               96.893239   -92.55112 1.801300e-04 5.758129e-03
Macrophages__DC               96.896215   -93.25194 1.844241e-04 5.758129e-03
CD4_T_cell__Keratin_Tumour    -4.845037   -22.14995 2.834659e-04 7.409016e-03
                                    from            to
Macrophages__dn_T_CD3       Macrophages      dn_T_CD3
dn_T_CD3__Macrophages          dn_T_CD3    Macrophages
Macrophages__DC_or_Mono     Macrophages     DC_or_Mono
DC_or_Mono__Macrophages      DC_or_Mono    Macrophages
dn_T_CD3__dn_T_CD3             dn_T_CD3      dn_T_CD3
Neutrophils__dn_T_CD3       Neutrophils      dn_T_CD3
dn_T_CD3__Neutrophils          dn_T_CD3    Neutrophils
DC__Macrophages                      DC    Macrophages
Macrophages__DC             Macrophages            DC
CD4_T_cell__Keratin_Tumour   CD4_T_cell Keratin_Tumour
```

Again, we obtain a `spicy` object which outlines the result of the linear models performed for each pairwise combination of cell types.

We can also examine the L-function metrics of individual images by using the convenient `bind()` function on our `spicyTest` results object.

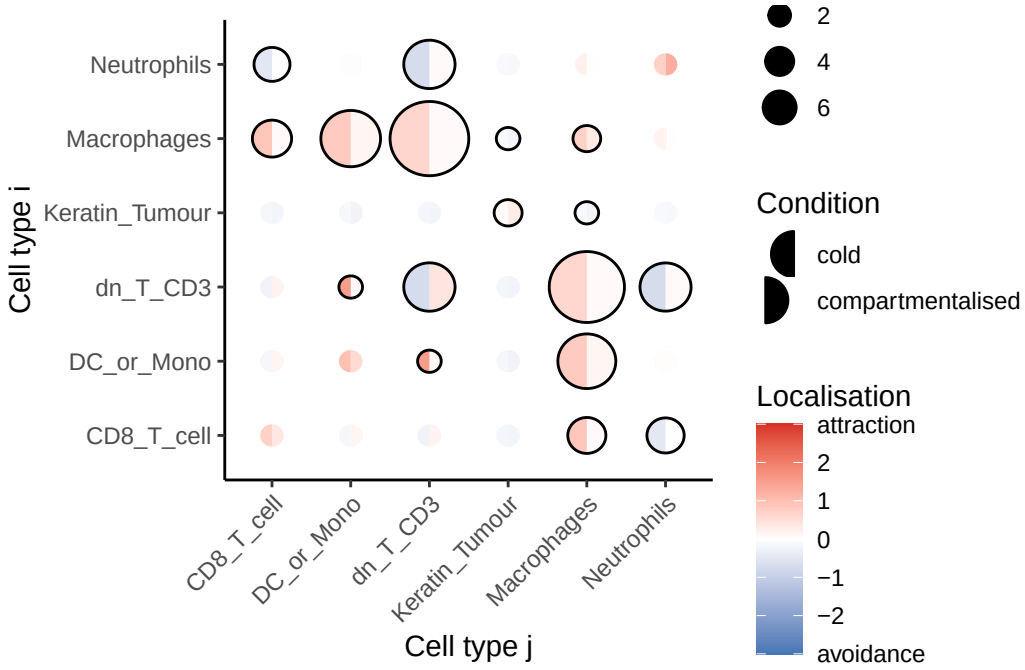
```
bind(spicyTest)[1:5, 1:5]
```

```
  imageID         condition Keratin_Tumour__Keratin_Tumour
1       1             mixed                      -2.300602
2       2             mixed                      -1.989699
```

```
3       3 compartmentalised                      11.373530
4       4 compartmentalised                      33.931133
5       5 compartmentalised                      17.922818
  dn_T_CD3__Keratin_Tumour B_cell__Keratin_Tumour
1                -5.298543             -20.827279
2               -16.020022               3.025815
3               -21.857447             -24.962913
4               -36.438476             -40.470221
5               -20.816783             -38.138076
```

The results can be represented as a bubble plot using the `signifPlot()` function.

```
signifPlot(
  spicyTest,
  breaks = c(-3, 3, 1),
  marksToPlot = c("Macrophages", "DC_or_Mono", "dn_T_CD3", "Neutrophils",
                  "CD8_T_cell", "Keratin_Tumour")
)
```

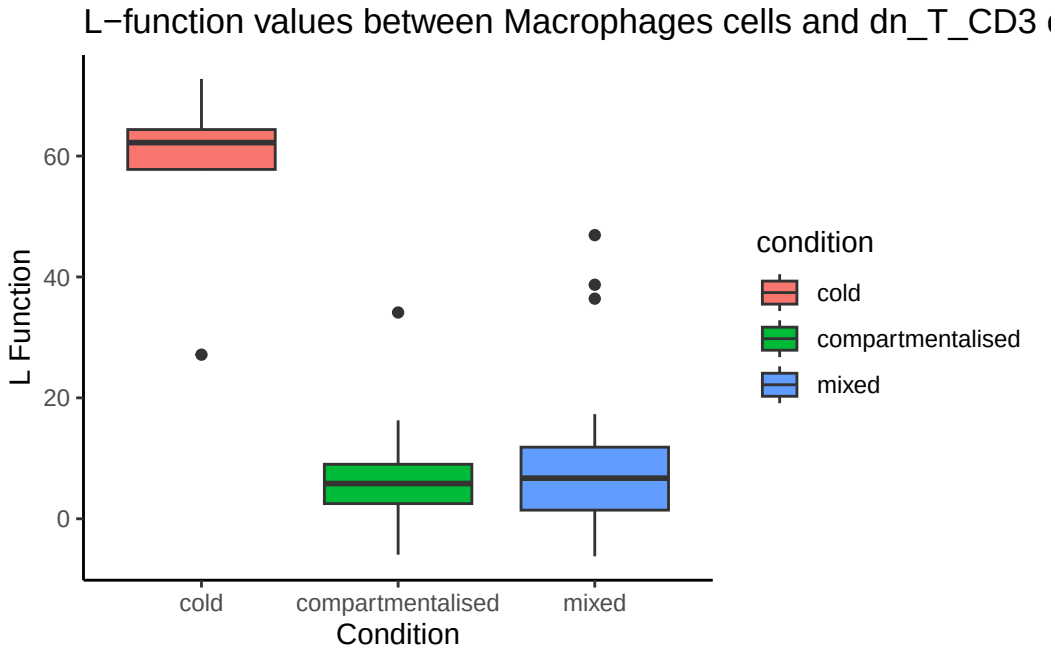



Here, we can observe that the most significant relationships occur between macrophages and double negative CD3 T cells, suggesting that the two cell types are far more dispersed in compartmentalised tumours compared to cold tumours.

To examine a specific cell type-cell type relationship in more detail, we can use `spicyBoxplot()` and specify either `from = "Macrophages"` and `to = "dn_T_CD3"` or `rank = 1`.

```
spicyBoxPlot(results = spicyTest,
             # from = "Macrophages",
             # to = "dn_T_CD3"
             rank = 1)
```

```
Warning: Removed 2 rows containing non-finite outside the scale range
(`stat_boxplot()`).
```




### 4.1.4 Linear modelling for custom metrics

`spicyR` can also be applied to custom distance or abundance metrics. A kNN interactions graph can be generated with the function `buildSpatialGraph` from the `imcRtools` package. This generates a `colPairs` object inside of the `SpatialExperiment` object.

`spicyR` provides the function `convPairs` for converting a `colPairs` object into an abundance matrix by calculating the average number of nearby cells types for every cell type for a given `k`. For example, if there exists on average 5 neutrophils for every macrophage in image 1, the column `Neutrophil__Macrophage` would have a value of 5 for image 1.

```
kerenSPE <- imcRtools::buildSpatialGraph(kerenSPE,
                                         img_id = "imageID",
                                         type = "knn", k = 20,
                                        coords = c("x", "y"))
```

```
'sample_id's are duplicated across 'SpatialExperiment' objects to cbind; appending sample inc
```

```
The returned object is ordered by the 'imageID' entry.
```

```
pairAbundances <- convPairs(kerenSPE,
                  colPair = "knn_interaction_graph")

head(pairAbundances["B_cell__B_cell"])
```

```
   B_cell__B_cell
1      12.7349608
10      0.2777778
11      0.0000000
12      1.3333333
13      1.2200957
14      0.0000000
```

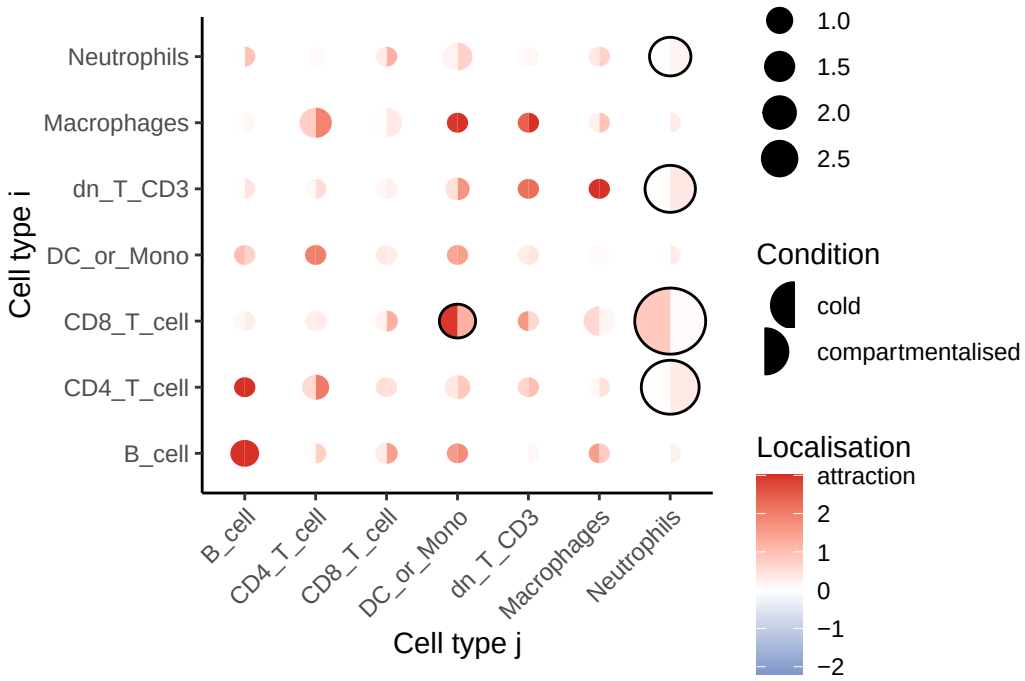
The custom distance or abundance metrics can then be included in the analysis with the `alternateResult` parameter. The `Statial` package contains other custom distance metrics which can be used with `spicy`.

```
spicyTestColPairs <- spicy(
  kerenSPE,
  condition = "tumour_type",
  alternateResult = pairAbundances,
  weights = FALSE
)

topPairs(spicyTestColPairs)
```

```
                          intercept coefficient     p.value adj.pvalue
CD8_T_cell__Neutrophils  0.833333333  -0.7592968 0.002645466  0.3291833
B_cell__Tumour           0.001937984   0.2602822 0.004872664  0.3291833
Other_Immune__NK         0.012698413   0.2612881 0.005673068  0.3291833
```

```
Unidentified__CD8_T_cell 0.106626794   0.6387339 0.005906526  0.3291833
dn_T_CD3__NK             0.004242424   0.2148797 0.006317829  0.3291833
CD4_T_cell__Neutrophils  0.036213602   0.2947696 0.007902670  0.3291833
Tregs__CD4_T_cell        0.128876212   0.5726201 0.010207087  0.3291833
Endothelial__DC          0.008771930   0.3008523 0.011189533  0.3291833
Tumour__Neutrophils      0.021638939   0.2529045 0.011388850  0.3291833
Mesenchymal__Neutrophils 0.004504505   0.2494301 0.012761315  0.3291833
                                 from          to
CD8_T_cell__Neutrophils    CD8_T_cell Neutrophils
B_cell__Tumour                B_cell      Tumour
Other_Immune__NK         Other_Immune          NK
Unidentified__CD8_T_cell Unidentified  CD8_T_cell
dn_T_CD3__NK                dn_T_CD3          NK
CD4_T_cell__Neutrophils    CD4_T_cell Neutrophils
Tregs__CD4_T_cell               Tregs  CD4_T_cell
Endothelial__DC           Endothelial          DC
Tumour__Neutrophils            Tumour Neutrophils
Mesenchymal__Neutrophils  Mesenchymal Neutrophils
```

```r
signifPlot(
  spicyTestColPairs,
  breaks = c(-3, 3, 1),
  marksToPlot = c("Macrophages", "dn_T_CD3", "CD4_T_cell",
                  "B_cell", "DC_or_Mono", "Neutrophils", "CD8_T_cell")
)
```




### 4.1.5 Performing survival analysis

`spicy` can also be used to perform survival analysis to asses whether changes in co-localisation between cell types are associated with survival probability. `spicy` requires the `SingleCellExperiment` object being used to contain a column called `survival` as a `Surv` object.

```
kerenSPE$event = 1 - kerenSPE$Censored
kerenSPE$survival = Surv(kerenSPE$`Survival_days_capped*`, kerenSPE$event)
```

We can then perform survival analysis using the `spicy` function by specifying `condition = "survival"`. We can then access the corresponding coefficients and p-values by accessing the `survivalResults` slot in the `spicy` results object.

```
# Running survival analysis
spicySurvival = spicy(kerenSPE,
                      condition = "survival")

# top 10 significant pairs
head(spicySurvival$survivalResults, 10)
```

```
# A tibble: 10 x 4
   test                        coef se.coef    p.value
   <chr>                      <dbl>   <dbl>      <dbl>
 1 Other_Immune__Tregs      0.0236  0.00866 0.00000893
 2 CD4_T_cell__Tregs        0.0177  0.00685 0.0000124
 3 Tregs__Other_Immune      0.0237  0.00873 0.0000126
 4 Tregs__CD4_T_cell        0.0171  0.00676 0.0000285
 5 CD8_T_cell__CD8_T_cell   0.00605 0.00272 0.000332
 6 Tumour__CD8_T_cell      -0.0305  0.0114  0.000617
 7 CD8_T_cell__Tumour      -0.0305  0.0116  0.000721
 8 CD4_T_cell__dn_T_CD3     0.00845 0.00353 0.000794
 9 dn_T_CD3__CD4_T_cell     0.00840 0.00353 0.000937
10 DC__Other_Immune        -0.0289  0.0123  0.00103
```

### 4.1.6 Accounting for tissue inhomogeneity

The `spicy` function can also account for tissue inhomogeneity to avoid false positives or negatives. This can be done by setting the `sigma =` parameter within the spicy function. By default, `sigma` is set to `NULL`, and `spicy` assumes a homogeneous tissue structure.

For example, when we examine the L-function for `Keratin_Tumour__Neutrophils` when `sigma = NULL` and `Rs = 100`, the value is positive, indicating attraction between the two cell types.

```
# filter SPE object to obtain image 24 data
kerenSubset = kerenSPE[, colData(kerenSPE)$imageID == "24"]

pairwiseAssoc = getPairwise(kerenSubset,
                            sigma = NULL,
                            Rs = 100) |>
  as.data.frame()

pairwiseAssoc[["Keratin_Tumour__Neutrophils"]]
```

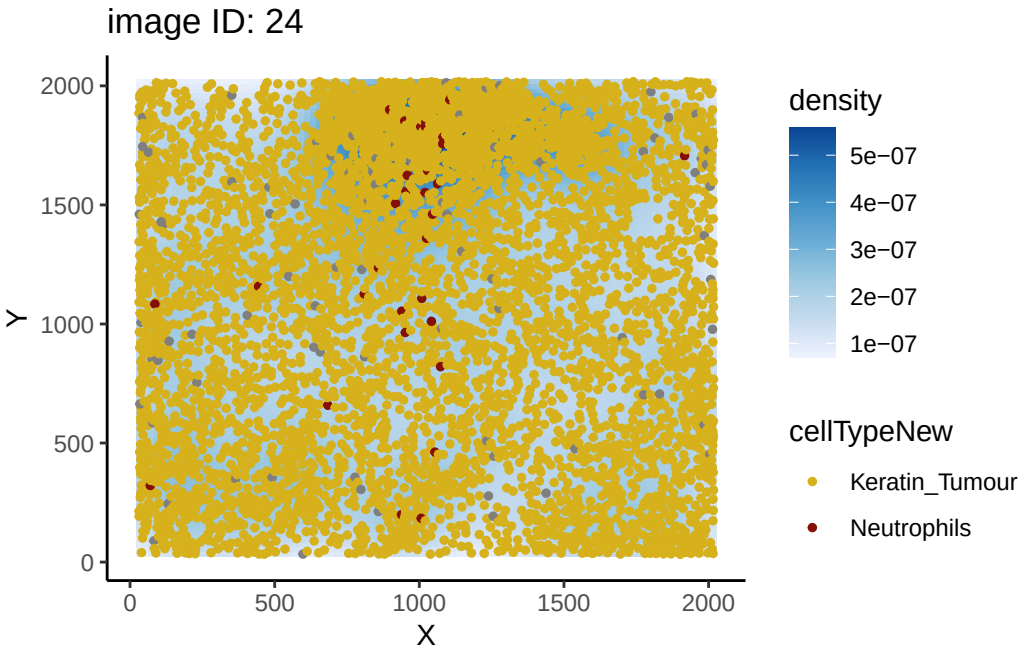
```
[1] 10.88892
```

When we specify `sigma = 20` and re-calculate the L-function, it indicates that there is no relationship between `Keratin_Tumour` and `Neutrophils`, i.e., there is no major attraction or dispersion, as it now takes into account tissue inhomogeneity.

```
pairwiseAssoc = getPairwise(kerenSubset,
                            sigma = 20,
                            Rs = 100) |>
  as.data.frame()

pairwiseAssoc[["Keratin_Tumour__Neutrophils"]]
```

```
[1] 0.9024836
```

```
# obtain colData for image 24
cData = colData(kerenSPE) |> as.data.frame() |>
          dplyr::filter(imageID == "24")

# obtain cells present in image 24
coords = spatialCoords(kerenSPE) |> as.data.frame()
coords$cellID = rownames(coords)
coords = coords |> dplyr::filter(cellID %in% cData$CellID)

cData$X = coords$x
cData$Y = coords$y

cData = cData |>
  dplyr::mutate(cellTypeNew = ifelse(cellType %in% c("Keratin_Tumour", "Neutrophils"),
                                     cellType, "Other"))

 pal = setNames(c("#d6b11c", "#850f07"),
                c("Keratin_Tumour", "Neutrophils"))

ggplot() +
    stat_density_2d(data = cData, aes(x = X, y = Y, fill = after_stat(density)),
                    geom = "raster",
                    contour = FALSE) +
    geom_point(data = cData |> filter(cellType != "Other"),
               aes(x = X, y = Y, colour = cellTypeNew), size = 1) +
    scale_color_manual(values = pal) +
    scale_fill_distiller(palette = "Blues", direction = 1) +
    theme_classic() +
    labs(title = "image ID: 24")
```




Plotting image 24 shows that the supposed co-localisation occurs due to the dense cluster of cells near the top of the image.

### 4.1.7 Mixed effects modelling

`spicyR` supports mixed effects modelling when multiple images are obtained for each subject. In this case, `subject` is treated as a random effect and `condition` is treated as a fixed effect. To perform mixed effects modelling, we can specify the `subject` parameter in the `spicy` function.

```
spicyMixedTest <- spicy(
  diabetesData,
  condition = "stage",
  subject = "case"
)
```

## 4.2 Kontextual: Context aware cell localisation

`Kontextual` is a method for performing inference on cell localisation which explicitly defines the contexts in which spatial relationships between cells can be identified and interpreted. These contexts may represent landmarks, spatial domains, or groups of functionally similar cells which are consistent across regions. By modelling spatial relationships between cells

relative to these contexts, `Kontextual` produces robust spatial quantifications that are not confounded by biases such as the choice of region to image and the tissue structure present in the images.

In this example we demonstrate how cell type hierarchies can be used as a means to derive appropriate "contexts" for the evaluation of cell localisation. We then demonstrate the types of conclusions which `Kontextual` enables.

### 4.2.1 Using cell type hierarchies to define a "context"

A cell type hierarchy may be used to define the "context" in which cell type relationships are evaluated within. A cell type hierarchy defines how cell types are functionally related to one another. The bottom of the hierarchy represents homogeneous populations of a cell type (child), the cell populations at the nodes of the hierarchy represent broader parent populations with shared generalised function. For example CD4 T cells may be considered a child population to the Immune parent population.

There are two ways to define the cell type hierarchy. First, they can be defined based on our biological understanding of the cell types. We can represent this by creating a named list containing the names of each parent and the associated vector of child cell types.

*Note:* The `all` vector must be created to include cell types which do not have a parent e.g. the *undefined* cell type in this data set.

```
# Examine all cell types in image
unique(kerenSPE$cellType)
```

```
 [1] "Keratin_Tumour" "dn_T_CD3"       "B_cell"         "CD4_T_cell"
 [5] "DC_or_Mono"     "Unidentified"   "Macrophages"    "CD8_T_cell"
 [9] "Other_Immune"   "Endothelial"    "Mono_or_Neu"    "Mesenchymal"
[13] "Neutrophils"    "NK"             "Tumour"         "DC"
[17] "Tregs"
```

```
# Named list of parents and their child cell types
biologicalHierarchy = list(
  "tumour" = c("Keratin_Tumour", "Tumour"),
  "tcells" = c("dn_T_CD3", "CD4_T_cell", "CD8_T_cell", "Tregs"),
  "myeloid" = c("DC_or_Mono", "DC", "Mono_or_Neu", "Macrophages", "Neutrophils"),
  "tissue" = c("Endothelial", "Mesenchymal")
)

# Adding more broader immune parent populationse
```

```
biologicalHierarchy$immune = c(biologicalHierarchy$bcells,
                               biologicalHierarchy$tcells,
                               biologicalHierarchy$myeloid,
                               "NK", "Other_Immune", "B_cell")


# Creating a vector for all cellTypes
all <- unique(kerenSPE$cellType)
```

Alternatively, you can use the `treeKor` bioconductor package treekoR to define these hierarchies in a data driven way.

*Note:* These parent populations may not be accurate as we are using a small subset of the data.

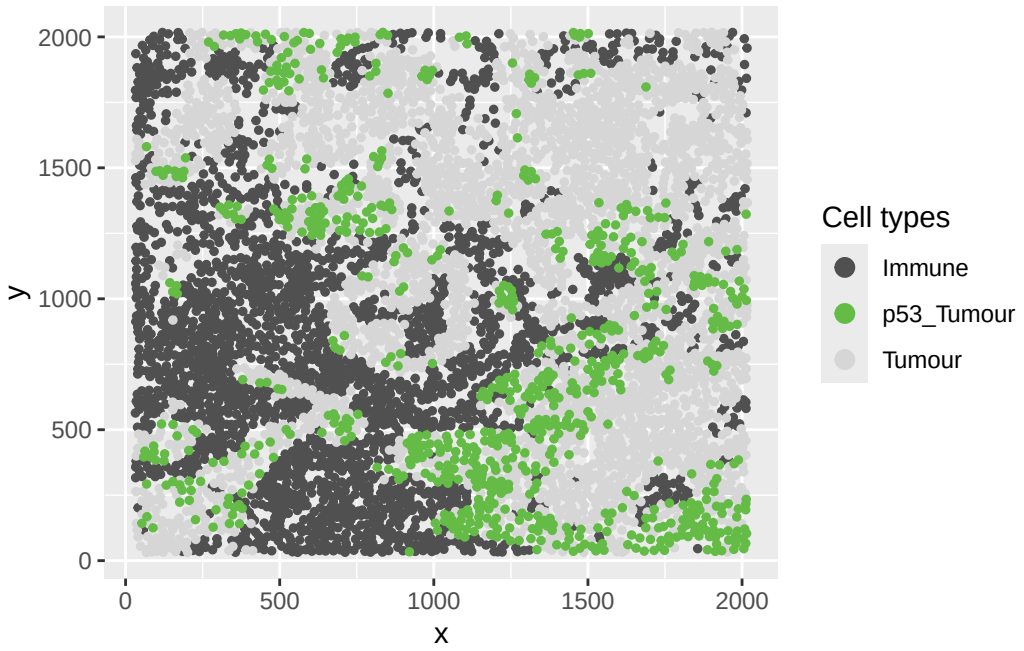
```
# Calculate hierarchy using treekoR
kerenTree <- treekoR::getClusterTree(t(assay(kerenSPE, "intensities")),
                            kerenSPE$cellType,
                            hierarchy_method="hopach",
                            hopach_K = 1)

# Convert treekoR result to a name list of parents and children.
treekorParents = getParentPhylo(kerenTree)

treekorParents
```

```
$parent_1
 [1] "Keratin_Tumour" "DC_or_Mono"     "Unidentified"   "Macrophages"
 [5] "Endothelial"    "Mono_or_Neu"    "Mesenchymal"    "Neutrophils"
 [9] "Tumour"         "DC"

$parent_2
[1] "dn_T_CD3"     "B_cell"       "CD4_T_cell"   "CD8_T_cell"   "Other_Immune"
[6] "NK"           "Tregs"
```

### 4.2.2 Application on triple negative breast cancer image.

Here we examine an image highlighted in the Keren et al. 2018 manuscript where accounting for context information enables new conclusions. In image 6 of the Keren et al. dataset, we can see that *p53+ tumour cells* and *immune cells* are dispersed i.e. these two cell types are not mixing. However we can also see that *p53+ tumour cells* appear much more localised to *immune cells* relative to the tumour context (*tumour cells* and *p53+ tumour cells*).

```r
# Lets define a new cell type vector
kerenSPE$cellTypeNew <- kerenSPE$cellType

# Select for all cells that express higher than baseline level of p53
p53Pos <- assay(kerenSPE)["p53", ] > -0.300460

# Find p53+ tumour cells
kerenSPE$cellTypeNew[kerenSPE$cellType %in% biologicalHierarchy$tumour] <- "Tumour"
kerenSPE$cellTypeNew[p53Pos & kerenSPE$cellType %in% biologicalHierarchy$tumour] <- "p53_Tumo

# Group all immune cells under the name "Immune"
kerenSPE$cellTypeNew[kerenSPE$cellType %in% biologicalHierarchy$immune] <- "Immune"

kerenSPE$x <- spatialCoords(kerenSPE)[,"x"]
kerenSPE$y <- spatialCoords(kerenSPE)[,"y"]

# Plot image 6
kerenSPE |>
  colData() |>
  as.data.frame() |>
  filter(imageID == "6") |>
  filter(cellTypeNew %in% c("Immune", "Tumour", "p53_Tumour")) |>
  arrange(cellTypeNew) |>
  ggplot(aes(x = x, y = y, color = cellTypeNew)) +
  geom_point(size = 1) +
  scale_colour_manual(values = c("Immune" = "#505050", "p53_Tumour" = "#64BC46", "Tumour" = "
  guides(colour = guide_legend(title = "Cell types", override.aes = list(size = 3)))
```




`Kontextual` accepts a `SingleCellExperiment` object, a single image, or list of images from a `SingleCellExperiment` object, which gets passed into the `cells` argument. The two cell types which will be evaluated are specified in the `to` and `from` arguments. A parent population must also be specified in the `parent` argument, note the parent cell population must include the `to` cell type. The argument `r` will specify the radius which the cell relationship will be evaluated on. `Kontextual` supports parallel processing, the number of cores can be specified using the `cores` argument. `Kontextual` can take a single value or multiple values for each argument and will test all combinations of the arguments specified.

We can calculate these relationships across all images for a single radius (r = 100). Small radii will examine local spatial relationships, whereas larger radii will examine global spatial relationships.

```
p53_Kontextual <- Kontextual(
  cells = kerenSPE,
  r = 100,
  from = "Immune",
  to = "p53_Tumour",
  parent = c("p53_Tumour", "Tumour"),
  cellType = "cellTypeNew"
)

p53_Kontextual
```

```
   imageID              test   original  kontextual   r inhomL
```

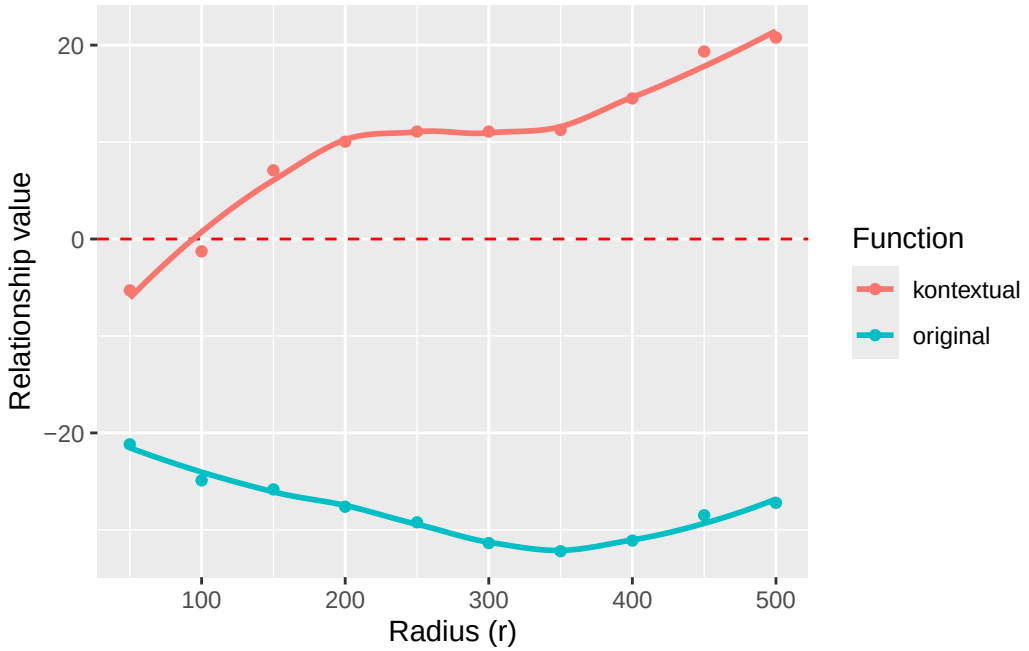
```
1        1 Immune__p53_Tumour -16.212016  -1.6815952 100  FALSE
2       10 Immune__p53_Tumour -14.715356  -1.7937407 100  FALSE
3       11 Immune__p53_Tumour -11.696597  -7.4615661 100  FALSE
4       12 Immune__p53_Tumour  -9.777271  -2.6287005 100  FALSE
5       13 Immune__p53_Tumour -15.613023  -3.9937364 100  FALSE
6       14 Immune__p53_Tumour -14.671281  -4.2879138 100  FALSE
7       15 Immune__p53_Tumour   8.369183  10.6710168 100  FALSE
8       16 Immune__p53_Tumour -41.081088 -20.9688333 100  FALSE
9       17 Immune__p53_Tumour  -6.331105   5.0017104 100  FALSE
10      18 Immune__p53_Tumour  -1.953366   0.5795853 100  FALSE
11      19 Immune__p53_Tumour -27.834450 -18.7433000 100  FALSE
12       2 Immune__p53_Tumour  -4.989150  -0.5330373 100  FALSE
13      20 Immune__p53_Tumour -20.580091  -9.2542544 100  FALSE
14      21 Immune__p53_Tumour -14.300802  -7.1425133 100  FALSE
15      22 Immune__p53_Tumour -13.673007 -12.9663547 100  FALSE
16      23 Immune__p53_Tumour  15.803493  37.3584378 100  FALSE
17      24 Immune__p53_Tumour -30.319961 -31.8146274 100  FALSE
18      25 Immune__p53_Tumour   6.262995   6.9429103 100  FALSE
19      26 Immune__p53_Tumour -38.190137 -24.6000029 100  FALSE
20      27 Immune__p53_Tumour  -2.373587   4.4044397 100  FALSE
21      28 Immune__p53_Tumour -70.058615 -33.4395839 100  FALSE
22      29 Immune__p53_Tumour -20.728463  -7.0172785 100  FALSE
23       3 Immune__p53_Tumour   1.719549  44.5060581 100  FALSE
24      31 Immune__p53_Tumour -12.306957  -5.2820792 100  FALSE
25      32 Immune__p53_Tumour -18.174569 -10.8972277 100  FALSE
26      33 Immune__p53_Tumour -19.750457 -19.6246151 100  FALSE
27      34 Immune__p53_Tumour -49.004947 -35.1320255 100  FALSE
28      35 Immune__p53_Tumour -75.980619 -66.2395276 100  FALSE
29      36 Immune__p53_Tumour -18.853398 -21.4398044 100  FALSE
30      37 Immune__p53_Tumour -43.624905 -27.7162991 100  FALSE
31      38 Immune__p53_Tumour -12.544687  -3.0415484 100  FALSE
32      39 Immune__p53_Tumour -19.293290  -4.5192485 100  FALSE
33       4 Immune__p53_Tumour         NA          NA 100  FALSE
34      40 Immune__p53_Tumour -37.744261 -27.9604962 100  FALSE
35      41 Immune__p53_Tumour -33.776940 -22.6113096 100  FALSE
36       5 Immune__p53_Tumour         NA          NA 100  FALSE
37       6 Immune__p53_Tumour -24.897348  -1.2724241 100  FALSE
38       7 Immune__p53_Tumour -13.068307   0.6361875 100  FALSE
39       8 Immune__p53_Tumour         NA          NA 100  FALSE
40       9 Immune__p53_Tumour -31.857501   1.0261067 100  FALSE
```

The `kontextCurve` function plots the L-function value and Kontextual values over a range of

radii. If the points lie above the red line (expected pattern) then localisation is indicated for that radius, if the points lie below the red line then dispersion is indicated.

As seen in the following plot the L-function produces negative values over a range of radii, indicating that *p53+ tumour cells* and *immune cells* are dispersed from one another. However by taking into account the tumour context, `Kontextual` shows positive values over some radii, indicating localisation between *p53+ tumour cells* and *immune cells*.

```
curves <- kontextCurve(
  cells = kerenSPE,
  from = "Immune",
  to = "p53_Tumour",
  parent = c("p53_Tumour", "Tumour"),
  rs = seq(50, 510, 50),
  image = "6",
  cellType = "cellTypeNew",
  cores = nCores
)

kontextPlot(curves)
```




Alternatively all pairwise cell relationships and their corresponding parent in the dataset can be tested. A data frame with all pairwise combinations can be creating using the `parentCombinations` function. This function takes in a vector of all the cells, as well as

the named list of parents and children created earlier in the `parentList` argument. As shown below the output is a data frame specifying the `to`, `from`, and `parent` arguments for `Kontextual`.

*Note:* the output of `getPhyloParent` may also be using the in the `parentList` argument, for example if you wanted to use the treekoR defined hierarchy instead.

```
# Get all relationships between cell types and their parents
parentDf <- parentCombinations(
  all = all,
  parentList = biologicalHierarchy
)
```

### 4.2.3 Calculating all pairwise relationships

Rather than specifying `to`, `from`, and `parent` in `Kontextual`, the output from `parentCombinations` can be inputed into `Kontextual` using the `parentDf` argument, to examine all pairwise relationships in the dataset. This chunk will take a significant amount of time to run, for demonstration the results have been saved and are loaded in.

```
# Running Kontextual on all relationships across all images.
kerenKontextual <- Kontextual(
  cells = kerenSPE,
  parentDf = parentDf,
  r = 100,
  cores = nCores
)
```

For every pairwise relationship (named accordingly: `from__to__parent`) `Kontextual` outputs the L-function values (original) and the Kontextual value. The relationships where the L-function and Kontextual disagree (e.g. one metric is positive and the other is negative) represent relationships where adding context information results in different conclusions on the spatial relationship between the two cell types.

### 4.2.4 Associating the relationships with survival outcomes.

To examine whether the features obtained from `Statial` are associated with patient outcomes or groupings, we can use the `spicy` function from the `spicyR` package.

In addition to this, the Kontextual results must be converted from a `data.frame` to a wide `matrix`, this can be done using `prepMatrix`. Note, to extract the original L-function values, specify `column = "original"` in `prepMatrix`.

```
# Converting Kontextual result into data matrix
kontextMat <- prepMatrix(kerenKontextual)

# Ensuring rownames of kontextMat match up with the image IDs of the SCE object
kontextMat <- kontextMat[kerenSPE$imageID |> unique(), ]

# Replace NAs with 0
kontextMat[is.na(kontextMat)] <- 0
```

Finally, both the `SingleCellExperiment` object and the Kontextual matrix are passed into the `spicy` function, with `condition = "survival"`. The resulting coefficients and p values can be obtained by accessing the `survivalResults` name.

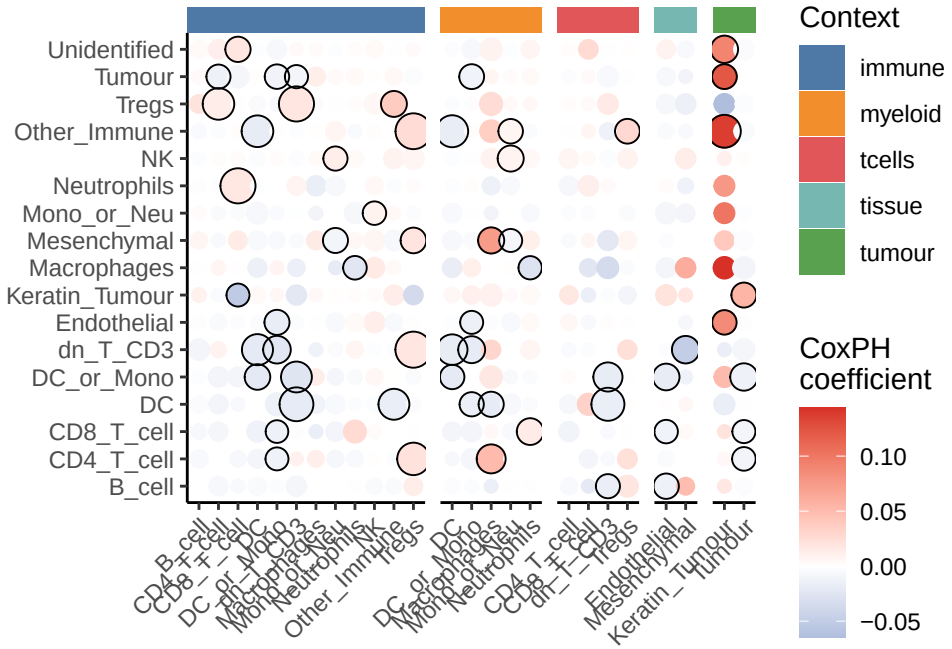
*Note:* You can specify additional covariates and include a subject id for mixed effects survival modelling, see spicyR for more information.

```
kerenSPE$event = 1 - kerenSPE$Censored
kerenSPE$survival = Surv(kerenSPE$`Survival_days_capped*`, kerenSPE$event)

# Running survival analysis
survivalResults = spicy(cells = kerenSPE,
                        alternateResult = kontextMat,
                        condition = "survival",
                        weights = TRUE)

head(survivalResults$survivalResults, 10)
```

```
# A tibble: 10 x 4
   test                                    coef se.coef     p.value
   <chr>                                  <dbl>   <dbl>       <dbl>
 1 dn_T_CD3__Tregs__immune               0.0187 0.00716 0.000000572
 2 Other_Immune__Tregs__immune           0.0255 0.00885 0.000000727
 3 Tregs__dn_T_CD3__immune               0.0189 0.00715 0.00000169
 4 Neutrophils__CD8_T_cell__immune       0.0178 0.00651 0.00000172
 5 DC__dn_T_CD3__immune                 -0.0195 0.00661 0.00000710
 6 DC__dn_T_CD3__tcells                 -0.0171 0.00758 0.0000167
 7 Other_Immune__Keratin_Tumour__tumour  0.137  0.0406  0.0000212
 8 CD4_T_cell__Tregs__immune             0.0216 0.00901 0.0000306
 9 Other_Immune__DC__myeloid            -0.0181 0.00664 0.0000492
10 Other_Immune__DC__immune             -0.0186 0.00669 0.0000656
```

The survival results can also be visualised using the `signifPlot` function.

```
signifPlot(survivalResults)
```




As we can see from the results `DC__NK__immune` is the one of the most significant pairwise relationship which contributes to patient survival. That is the relationship between dendritic cells and natural killer cells, relative to the parent population of immune cells. We can see that there is a negative coefficient associated with this relationship, which tells us an increase in localisation of these cell types relative to immune cells leads to better survival outcomes for patients.

The association between `DC__NK__immune` and survival can also be visualised on a Kaplan-Meier curve. First, we extract survival data from the `SingleCellExperiment` object and create a survival vector.

```
# Extracting survival data
survData <- kerenSPE |>
  colData() |>
  data.frame() |>
  select(imageID, survival) |>
  unique()

# Creating survival vector
kerenSurv <- survData$survival
names(kerenSurv) <- survData$imageID
```

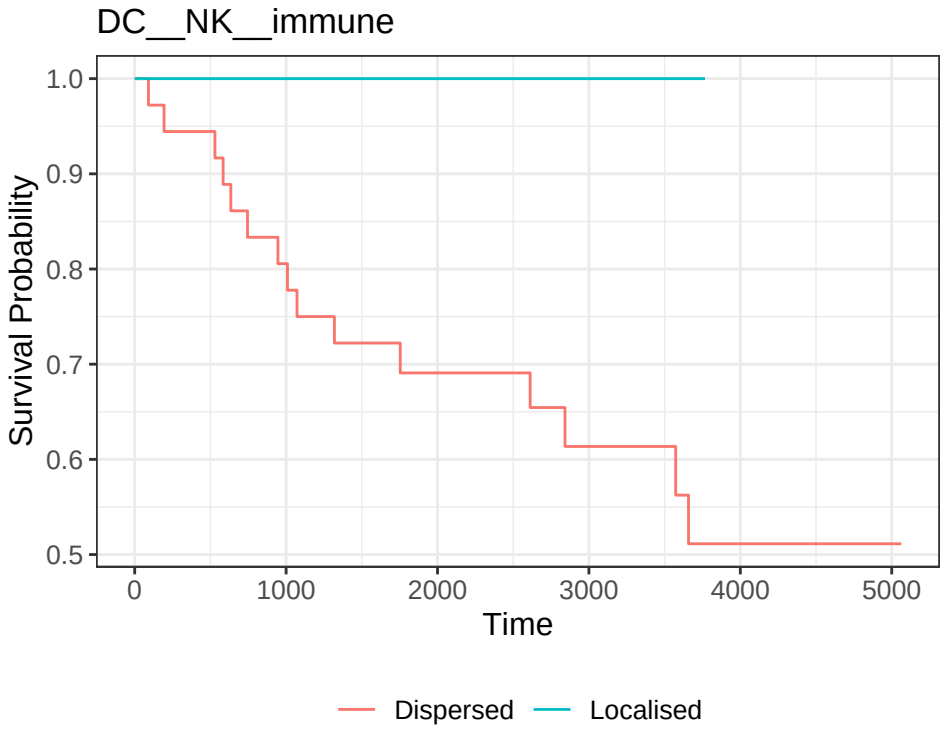
```
kerenSurv
```

```
    1     2     3     4     5     6     7     8     9    10    11    12    13
 2612   745 3130+ 2523+ 1683+ 2275+   584   946 3767+ 3822+ 3774+ 4353+  1072
   14    15    16    17    18    19    20    21    22    23    24    25    26
4145+  1754   530  2842 5063+ 3725+ 4761+   635   NA?    91   194 4785+ 4430+
   27    28    29    31    32    33    34    35    36    37    38    39    40
 3658 3767+  1319  1009 1568+ 1738+ 2832+ 2759+ 3063+ 2853+   NA? 2096+  3573
   41
3355+
```

Next, we extract the Kontextual values of this relationship across all images. We then determine if dendritic and natural killer cells are relatively attracted or avoiding in each image by comparing the Kontextual value in each image to the median Kontextual value.

Finally, we plot a Kaplan-Meier curve using the `ggsurvfit` package. As shown below, when dendritic and natural killer cells are more localised to one another relative to the immune cell population, patients tend to have better survival outcomes.

```
# Selecting most significant relationship
survRelationship <- kontextMat[["DC__NK__immune"]]
survRelationship <- ifelse(survRelationship > median(survRelationship), "Localised", "Dispers

# Plotting Kaplan-Meier curve
survfit2(kerenSurv ~ survRelationship) |>
  ggsurvfit() +
  ggtitle("DC__NK__immune")
```




## 4.3 sessionInfo

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Debian GNU/Linux 12 (bookworm)

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.21.so;  LAPACK version 3

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: Australia/Sydney
```

```
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] ggsurvfit_1.1.0             treekoR_1.14.0
 [3] tibble_3.2.1                survival_3.7-0
 [5] dplyr_1.1.4                 imcRtools_1.12.0
 [7] SpatialDatasets_1.4.0       ExperimentHub_2.14.0
 [9] AnnotationHub_3.14.0        BiocFileCache_2.14.0
[11] dbplyr_2.5.0                SpatialExperiment_1.16.0
[13] SingleCellExperiment_1.28.1 SummarizedExperiment_1.36.0
[15] Biobase_2.66.0              GenomicRanges_1.58.0
[17] GenomeInfoDb_1.42.0         IRanges_2.40.0
[19] S4Vectors_0.44.0            BiocGenerics_0.52.0
[21] MatrixGenerics_1.18.0       matrixStats_1.4.1
[23] ggplot2_3.5.1               Statial_1.8.0
[25] spicyR_1.18.0

loaded via a namespace (and not attached):
  [1] vroom_1.6.5                 tiff_0.1-12
  [3] goftest_1.2-3               DT_0.33
  [5] Biostrings_2.74.0           HDF5Array_1.34.0
  [7] TH.data_1.1-2               vctrs_0.6.5
  [9] spatstat.random_3.3-2       digest_0.6.37
 [11] png_0.1-8                   shape_1.4.6.1
 [13] proxy_0.4-27                ggrepel_0.9.6
 [15] deldir_2.0-4                magick_2.8.5
 [17] MASS_7.3-61                 reshape2_1.4.4
 [19] httpuv_1.6.15               foreach_1.5.2
 [21] withr_3.0.2                 ggfun_0.1.7
 [23] xfun_0.49                   ggpubr_0.6.0
 [25] memoise_2.0.1               RTriangle_1.6-0.14
 [27] cytomapper_1.18.0           ggbeeswarm_0.7.2
 [29] RProtoBufLib_2.18.0         systemfonts_1.1.0
 [31] tidytree_0.4.6              zoo_1.8-12
 [33] GlobalOptions_0.1.2         Formula_1.2-5
 [35] KEGGREST_1.46.0             promises_1.3.0
 [37] httr_1.4.7                  rstatix_0.7.2
 [39] rhdf5filters_1.18.0         rhdf5_2.50.0
 [41] rstudioapi_0.17.1           UCSC.utils_1.2.0
```

```
 [43] units_0.8-5                 generics_0.1.3
 [45] concaveman_1.1.0            curl_6.0.1
 [47] zlibbioc_1.52.0             ggraph_2.2.1
 [49] polyclip_1.10-7             GenomeInfoDbData_1.2.13
 [51] SparseArray_1.6.0           fftwtools_0.9-11
 [53] xtable_1.8-4                stringr_1.5.1
 [55] doParallel_1.0.17           evaluate_1.0.1
 [57] S4Arrays_1.6.0              hms_1.1.3
 [59] colorspace_2.1-1            filelock_1.0.3
 [61] spatstat.data_3.1-2         magrittr_2.0.3
 [63] readr_2.1.5                 later_1.3.2
 [65] viridis_0.6.5               ggtree_3.14.0
 [67] lattice_0.22-6              spatstat.geom_3.3-3
 [69] XML_3.99-0.17               scuttle_1.16.0
 [71] ggupset_0.4.0               class_7.3-22
 [73] svgPanZoom_0.3.4            pillar_1.9.0
 [75] nlme_3.1-166                iterators_1.0.14
 [77] EBImage_4.48.0              compiler_4.4.1
 [79] beachmat_2.22.0             stringi_1.8.4
 [81] sf_1.0-19                   tensor_1.5
 [83] minqa_1.2.8                 ClassifyR_3.10.0
 [85] plyr_1.8.9                  crayon_1.5.3
 [87] abind_1.4-8                 gridGraphics_0.5-1
 [89] locfit_1.5-9.10             sp_2.1-4
 [91] graphlayouts_1.2.0          bit_4.5.0
 [93] terra_1.7-83                sandwich_3.1-1
 [95] codetools_0.2-20            multcomp_1.4-26
 [97] e1071_1.7-16                GetoptLong_1.0.5
 [99] plotly_4.10.4               mime_0.12
[101] MultiAssayExperiment_1.32.0 splines_4.4.1
[103] circlize_0.4.16             Rcpp_1.0.13-1
[105] knitr_1.49                  blob_1.2.4
[107] utf8_1.2.4                  clue_0.3-66
[109] BiocVersion_3.20.0          lme4_1.1-35.5
[111] fs_1.6.5                    nnls_1.6
[113] ggplotify_0.1.2             ggsignif_0.6.4
[115] Matrix_1.7-1                scam_1.2-17
[117] statmod_1.5.0               tzdb_0.4.0
[119] svglite_2.1.3               tweenr_2.0.3
[121] pkgconfig_2.0.3             pheatmap_1.0.12
[123] tools_4.4.1                 cachem_1.1.0
[125] RSQLite_2.3.7               viridisLite_0.4.2
[127] DBI_1.2.3                   numDeriv_2016.8-1.1
```

```
[129] fastmap_1.2.0                rmarkdown_2.29
[131] scales_1.3.0                 grid_4.4.1
[133] shinydashboard_0.7.2         broom_1.0.7
[135] patchwork_1.3.0              BiocManager_1.30.25
[137] carData_3.0-5                farver_2.1.2
[139] tidygraph_1.3.1              mgcv_1.9-1
[141] yaml_2.3.10                  ggthemes_5.1.0
[143] cli_3.6.3                    purrr_1.0.2
[145] hopach_2.66.0                lifecycle_1.0.4
[147] mvtnorm_1.3-2                backports_1.5.0
[149] BiocParallel_1.40.0          cytolib_2.18.0
[151] gtable_0.3.6                 rjson_0.2.23
[153] parallel_4.4.1               ape_5.8
[155] limma_3.62.1                 jsonlite_1.8.9
[157] edgeR_4.4.0                  bitops_1.0-9
[159] bit64_4.5.2                  Rtsne_0.17
[161] FlowSOM_2.14.0               yulab.utils_0.1.8
[163] spatstat.utils_3.1-1         BiocNeighbors_2.0.0
[165] ranger_0.17.0                flowCore_2.18.0
[167] bdsmatrix_1.3-7              spatstat.univar_3.1-1
[169] lazyeval_0.2.2               shiny_1.9.1
[171] ConsensusClusterPlus_1.70.0 htmltools_0.5.8.1
[173] diffcyt_1.26.0               rappdirs_0.3.3
[175] tinytex_0.54                 distances_0.1.11
[177] glue_1.8.0                   XVector_0.46.0
[179] RCurl_1.98-1.16              treeio_1.30.0
[181] classInt_0.4-10              coxme_2.2-22
[183] jpeg_0.1-10                  gridExtra_2.3
[185] boot_1.3-31                  igraph_2.1.1
[187] R6_2.5.1                     tidyr_1.3.1
[189] ggiraph_0.8.10               labeling_0.4.3
[191] ggh4x_0.2.8                  cluster_2.1.6
[193] Rhdf5lib_1.28.0              aplot_0.2.3
[195] nloptr_2.1.1                 DelayedArray_0.32.0
[197] tidyselect_1.2.1             vipor_0.4.7
[199] ggforce_0.4.2                raster_3.6-30
[201] car_3.1-3                    AnnotationDbi_1.68.0
[203] munsell_0.5.1                KernSmooth_2.23-24
[205] data.table_1.16.2            htmlwidgets_1.6.4
[207] ComplexHeatmap_2.22.0        RColorBrewer_1.1-3
[209] rlang_1.1.4                  spatstat.sparse_3.1-0
[211] spatstat.explore_3.3-3       lmerTest_3.1-3
[213] uuid_1.2-1                   colorRamps_2.3.4
```

```
[215] ggnewscale_0.5.0            fansi_1.0.6
[217] beeswarm_0.4.0
```

# 5 Spatial domains

## 5.1 Why look at spatial domains?

Beyond spatial relationships between cell types, imaging datasets also contain another source of rich information - spatial domains. To give an idea of what spatial domains might visually look like, we've provided an image on the right, where we can clearly map out our healthy epithelial tissue spatial domain on the left of the image, and our immune and tumour domains on the right of the image.

However, spatial domains tend to be highly dependent on the biological question being answered. For example, when your primary tissue of interest are solid tumours, spatial domain analysis can provide insights into proportion of tumour domain vs immune domains, or how tumour domains differ between progressive and non-progressive cancers. Alternatively, if your primary tissue of interest is diabetes, spatial domains can provide insights into marker or cell type differences in your pancreatic islets.

In this section, we'll be exploring the use of `lisaClust` on two different datasets to help inform

```
# load required packages
library(lisaClust)
library(spicyR)
library(ggplot2)
library(SingleCellExperiment)
library(SpatialDatasets)
```

## 5.2 lisaClust

Clustering local indicators of spatial association (LISA) functions is a methodology for identifying consistent spatial organisation of multiple cell-types in an unsupervised way. This can be used to enable the characterization of interactions between multiple cell-types simultaneously and can complement traditional pairwise analysis. In our implementation our LISA curves are a localised summary of an L-function from a Poisson point process model. Our

framework `lisaClust` can be used to provide a high-level summary of cell-type colocalization in high-parameter spatial cytometry data, facilitating the identification of distinct tissue compartments or identification of complex cellular microenvironments.

### 5.2.1 How lisaClust works

The workflow that lisaClust uses to identify regions of tissue with similar localisation patterns of cells contains multiple key steps. First, cells are treated as objects and assigned coordinates in an x-y space. Second, distances between all cells are calculated and then, by modeling the cells as a multi-type Poisson point process, the distances are used to calculate local indicators of spatial association (LISA). These LISA curves summarize the spatial association between each cell and a specific cell type over a range of radii, r. The LISA curves are calculated for each cell and cell type and then clustered to assign a region label for each cell.

### 5.2.2 Case study: Keren

We will start by reading in the data from the `SpatialDatasets` package as a `SingleCellExperiment` object. Here the data is in a format consistent with that outputted by CellProfiler.

```
kerenSPE <- SpatialDatasets::spe_Keren_2018()
```

```
see ?SpatialDatasets and browseVignettes('SpatialDatasets') for documentation
```

```
loading from cache
```

#### 5.2.2.1 Generate LISA curves

This data includes annotation of the cell-types of each cell. Hence, we can move directly to performing k-means clustering on the local indicators of spatial association (LISA) functions using the `lisaClust` function, remembering to specify the `imageID`, `cellType`, and `spatialCoords` columns in `colData`. For the purpose of demonstration, we will be using only images 5 and 6 of the `kerenSPE` dataset.

```
kerenSPE <- kerenSPE[,kerenSPE$imageID %in% c("5", "6")]

kerenSPE <- lisaClust(kerenSPE,
  k = 5
)
```

```
Generating local L-curves.
```

These regions are stored in `colData` and can be extracted.

```
colData(kerenSPE)[, c("imageID", "region")] |>
  head(20)
```

```
DataFrame with 20 rows and 2 columns
          imageID      region
      <character> <character>
21154           5    region_5
21155           5    region_5
21156           5    region_5
21157           5    region_2
21158           5    region_2
...           ...         ...
21169           5    region_2
21170           5    region_2
21171           5    region_3
21172           5    region_2
21173           5    region_3
```

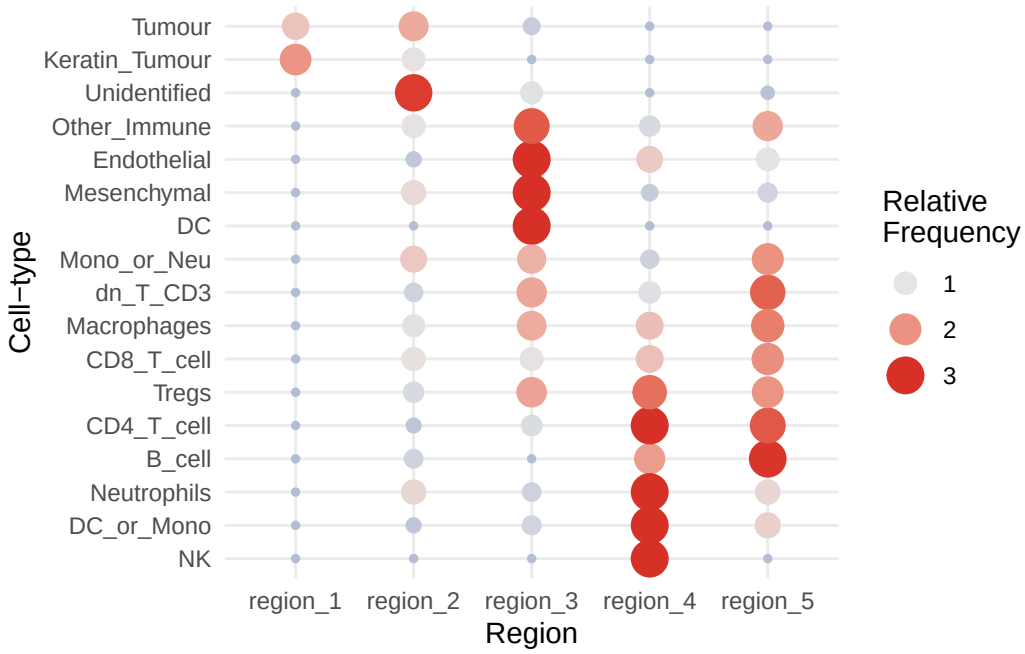
#### 5.2.2.2 Examine cell type enrichment

`lisaClust` also provides a convenient function, `regionMap`, for examining which cell types are located in which regions. In this example, we use this to check which cell types appear more frequently in each region than expected by chance.

Here, we clearly see that healthy epithelial and mesenchymal tissue are highly concentrated in region 1, immune cells are concentrated in regions 2 and 4, whilst tumour cells are concentrated in region 3.

We can further segregate these cells by increasing the number of clusters, i.e., increasing the parameter `k =` in the `lisaClust()` function. For the purposes of demonstration, let's take a look at the `hatchingPlot` of these regions.

```
regionMap(kerenSPE,
  type = "bubble"
)
```

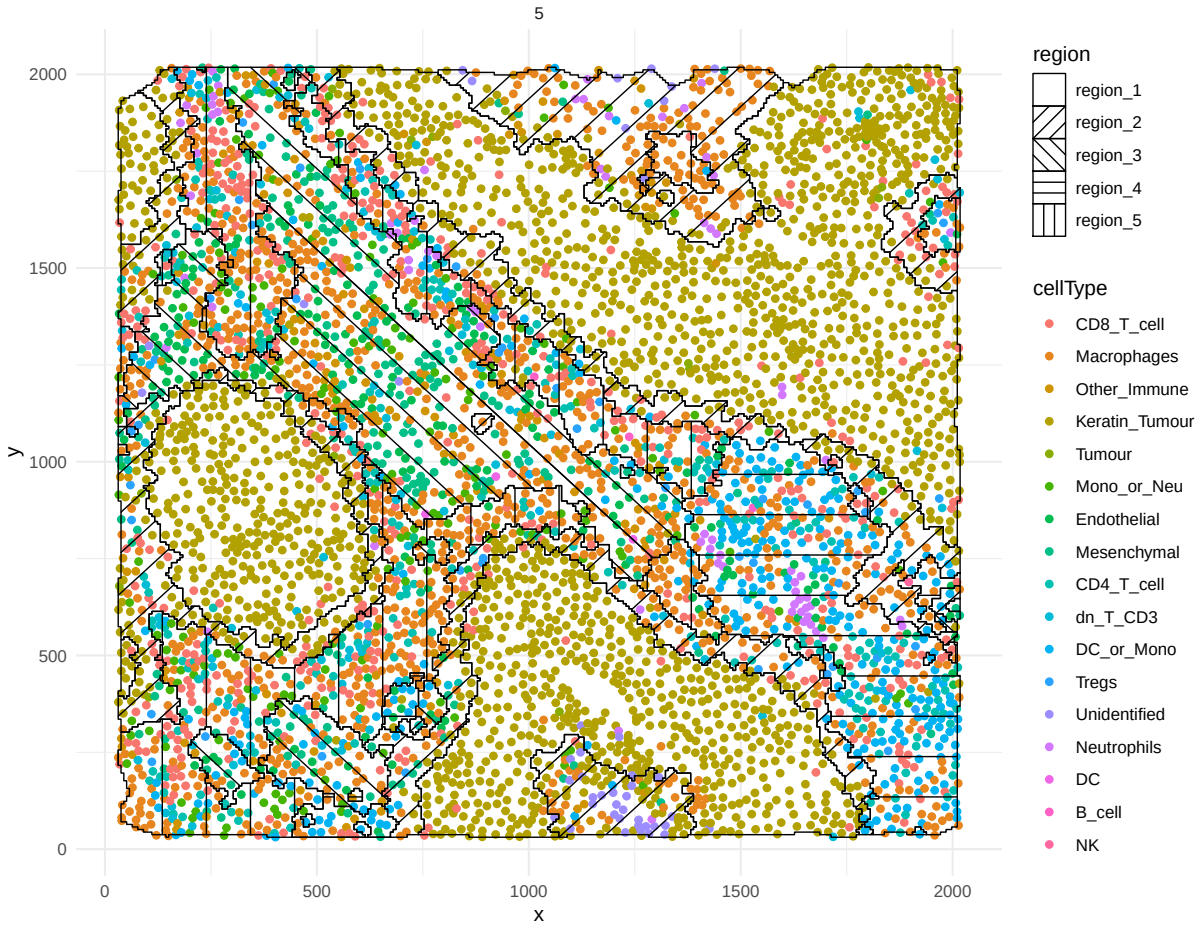



#### 5.2.2.3 Plot identified regions

Finally, we can use `hatchingPlot` to construct a `ggplot` object where the regions are marked by different hatching patterns. This allows us to visualize the 5 regions and 17 cell-types simultaneously.

```
hatchingPlot(kerenSPE, nbp = 300)
```

```
Concave windows are temperamental. Try choosing values of window.length > and < 1 if you have
```




## 5.3 sessionInfo

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Debian GNU/Linux 12 (bookworm)

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.21.so;  LAPACK version 3

locale:
```

```
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: Australia/Sydney
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] SpatialDatasets_1.4.0       SpatialExperiment_1.16.0
 [3] ExperimentHub_2.14.0        AnnotationHub_3.14.0
 [5] BiocFileCache_2.14.0        dbplyr_2.5.0
 [7] SingleCellExperiment_1.28.1 SummarizedExperiment_1.36.0
 [9] Biobase_2.66.0              GenomicRanges_1.58.0
[11] GenomeInfoDb_1.42.0         IRanges_2.40.0
[13] S4Vectors_0.44.0            BiocGenerics_0.52.0
[15] MatrixGenerics_1.18.0       matrixStats_1.4.1
[17] ggplot2_3.5.1               spicyR_1.18.0
[19] lisaClust_1.14.4

loaded via a namespace (and not attached):
  [1] RColorBrewer_1.1-3          rstudioapi_0.17.1
  [3] jsonlite_1.8.9              MultiAssayExperiment_1.32.0
  [5] magrittr_2.0.3              spatstat.utils_3.1-1
  [7] magick_2.8.5                farver_2.1.2
  [9] nloptr_2.1.1                rmarkdown_2.29
 [11] zlibbioc_1.52.0             vctrs_0.6.5
 [13] memoise_2.0.1               minqa_1.2.8
 [15] spatstat.explore_3.3-3      tinytex_0.54
 [17] rstatix_0.7.2               htmltools_0.5.8.1
 [19] S4Arrays_1.6.0              curl_6.0.1
 [21] broom_1.0.7                 SparseArray_1.6.0
 [23] Formula_1.2-5               plyr_1.8.9
 [25] cachem_1.1.0                mime_0.12
 [27] lifecycle_1.0.4             pkgconfig_2.0.3
 [29] Matrix_1.7-1                R6_2.5.1
 [31] fastmap_1.2.0               GenomeInfoDbData_1.2.13
 [33] digest_0.6.37               numDeriv_2016.8-1.1
 [35] colorspace_2.1-1            AnnotationDbi_1.68.0
```

```
 [37] tensor_1.5                 RSQLite_2.3.7
 [39] ggpubr_0.6.0               labeling_0.4.3
 [41] filelock_1.0.3             fansi_1.0.6
 [43] spatstat.sparse_3.1-0      httr_1.4.7
 [45] polyclip_1.10-7            abind_1.4-8
 [47] mgcv_1.9-1                 compiler_4.4.1
 [49] bit64_4.5.2                withr_3.0.2
 [51] backports_1.5.0            BiocParallel_1.40.0
 [53] carData_3.0-5              DBI_1.2.3
 [55] ggupset_0.4.0              ggforce_0.4.2
 [57] coxme_2.2-22               ggsignif_0.6.4
 [59] MASS_7.3-61                concaveman_1.1.0
 [61] rappdirs_0.3.3             DelayedArray_0.32.0
 [63] rjson_0.2.23               tools_4.4.1
 [65] goftest_1.2-3              glue_1.8.0
 [67] nlme_3.1-166               grid_4.4.1
 [69] ClassifyR_3.10.0           reshape2_1.4.4
 [71] generics_0.1.3             gtable_0.3.6
 [73] spatstat.data_3.1-2        class_7.3-22
 [75] tidyr_1.3.1                data.table_1.16.2
 [77] car_3.1-3                  utf8_1.2.4
 [79] XVector_0.46.0             spatstat.geom_3.3-3
 [81] BiocVersion_3.20.0         pillar_1.9.0
 [83] stringr_1.5.1              splines_4.4.1
 [85] dplyr_1.1.4                tweenr_2.0.3
 [87] lattice_0.22-6             survival_3.7-0
 [89] bit_4.5.0                  deldir_2.0-4
 [91] tidyselect_1.2.1           Biostrings_2.74.0
 [93] knitr_1.49                 V8_6.0.0
 [95] xfun_0.49                  pheatmap_1.0.12
 [97] fftwtools_0.9-11           scam_1.2-17
 [99] stringi_1.8.4              UCSC.utils_1.2.0
[101] yaml_2.3.10                boot_1.3-31
[103] evaluate_1.0.1             codetools_0.2-20
[105] tibble_3.2.1               BiocManager_1.30.25
[107] cli_3.6.3                  munsell_0.5.1
[109] Rcpp_1.0.13-1              spatstat.random_3.3-2
[111] png_0.1-8                  bdsmatrix_1.3-7
[113] spatstat.univar_3.1-1      parallel_4.4.1
[115] ggh4x_0.2.8                blob_1.2.4
[117] lme4_1.1-35.5              ggthemes_5.1.0
[119] lmerTest_3.1-3             scales_1.3.0
[121] purrr_1.0.2                crayon_1.5.3
```

```
[123] rlang_1.1.4                KEGGREST_1.46.0
```

# 6 Marker expression

```
# Loading required packages
library(Statial)
library(spicyR)
library(ClassifyR)
library(lisaClust)
library(dplyr)
library(SingleCellExperiment)
library(ggplot2)
library(ggsurvfit)
library(survival)
library(tibble)
library(treekoR)
devtools::load_all("~/Statial")

theme_set(theme_classic())
nCores <- 8
```

## 6.1 Statial: Marker means

One of the easiest things to quantify in terms of markers is a marker mean. For a given image, we assess the total marker mean across all cells within an image, and compare across disease states. We can do this on an image level, a cell type level, a region level, and a cell type within regions level. For example, if your question is: "How does the expression of CD163 in infiltrating macrophages within the tumour spatial domain differ across my 2 treatment groups?", you'll want to look at the marker mean of macrophages within specifically the tumour domain.

`Statial` provides functionality to identify the average marker expression of a given cell type in a given region, using the `getMarkerMeans` function. Similar to the analysis above, these features can also be used for survival analysis.

```r
cellTypeRegionMeans <- getMarkerMeans(kerenSPE,
  imageID = "imageID",
  cellType = "cellType",
  region = "region"
)

survivalResults <- colTest(cellTypeRegionMeans[names(kerenSurv), ], kerenSurv, type = "survi

head(survivalResults)
```

```
                                    coef se.coef    pval adjPval
B7H3__CD4_T_cell__region_4         270.0   76.00 0.00038    0.43
CD163__CD4_T_cell__region_4         67.0   19.00 0.00038    0.43
FoxP3__CD4_T_cell__region_4         25.0    7.20 0.00052    0.43
Si__Unidentified__region_2          -3.1    0.89 0.00053    0.43
CD56__CD4_T_cell__region_4          28.0    8.10 0.00067    0.43
Keratin6__Keratin_Tumour__region_5   1.6    0.47 0.00073    0.43
                                                          cluster
B7H3__CD4_T_cell__region_4                 B7H3__CD4_T_cell__region_4
CD163__CD4_T_cell__region_4               CD163__CD4_T_cell__region_4
FoxP3__CD4_T_cell__region_4               FoxP3__CD4_T_cell__region_4
Si__Unidentified__region_2                 Si__Unidentified__region_2
CD56__CD4_T_cell__region_4                 CD56__CD4_T_cell__region_4
Keratin6__Keratin_Tumour__region_5 Keratin6__Keratin_Tumour__region_5
```

## 6.2 SpatioMark: Identifying continuous changes in cell state

Changes in cell states can be analytically framed as the change in abundance of a gene or protein within a particular cell type. We can use marker expression to identify and quantify evidence of cell interactions that catalyse cell state changes. This approach measures how protein markers in a cell change with spatial proximity and abundance to other cell types. The methods utilised here will thereby provide a framework to explore how the dynamic behaviour of cells are altered by the agents they are surrounded by.

### 6.2.1 Continuous cell state changes within a single image

The first step in analysing these changes is to calculate the spatial proximity (`getDistances`) and abundance (`getAbundances`) of each cell to every cell type. These values will then be stored in the `reducedDims` slot of the `SingleCellExperiment` object under the names `distances` and `abundances` respectively.

```
kerenSPE <- getDistances(kerenSPE,
  maxDist = 200,
  nCores = 1
)

kerenSPE <- getAbundances(kerenSPE,
  r = 200,
  nCores = 1
)
```

First, let's examine the same effect observed earlier with Kontextual - the localisation between p53-positive keratin/tumour cells and macrophages in the context of total keratin/tumour cells for image 6 of the Keren et al. dataset.

Statial provides two main functions to assess this relationship - `calcStateChanges` and `plotStateChanges`. We can use `calcStateChanges` to examine the relationship between 2 cell types for 1 marker in a specific image. In this case, we're examining the relationship between keratin/tumour cells (`from = Keratin_Tumour`) and macrophages (`to = "Macrophages"`) for the marker p53 (`marker = "p53"`) in `image = "6"`. We can appreciate that the `fdr` statistic for this relationship is significant, with a negative tvalue, indicating that the expression of p53 in keratin/tumour cells decreases as distance from macrophages increases.

```
stateChanges <- calcStateChanges(
  cells = kerenSPE,
  type = "distances",
  image = "6",
  from = "Keratin_Tumour",
  to = "Macrophages",
  marker = "p53",
  nCores = 1
)

stateChanges
```

```
  imageID primaryCellType otherCellType marker         coef      tval
1       6  Keratin_Tumour   Macrophages    p53 -0.001402178 -7.010113
          pval          fdr
1 2.868257e-12 2.868257e-12
```

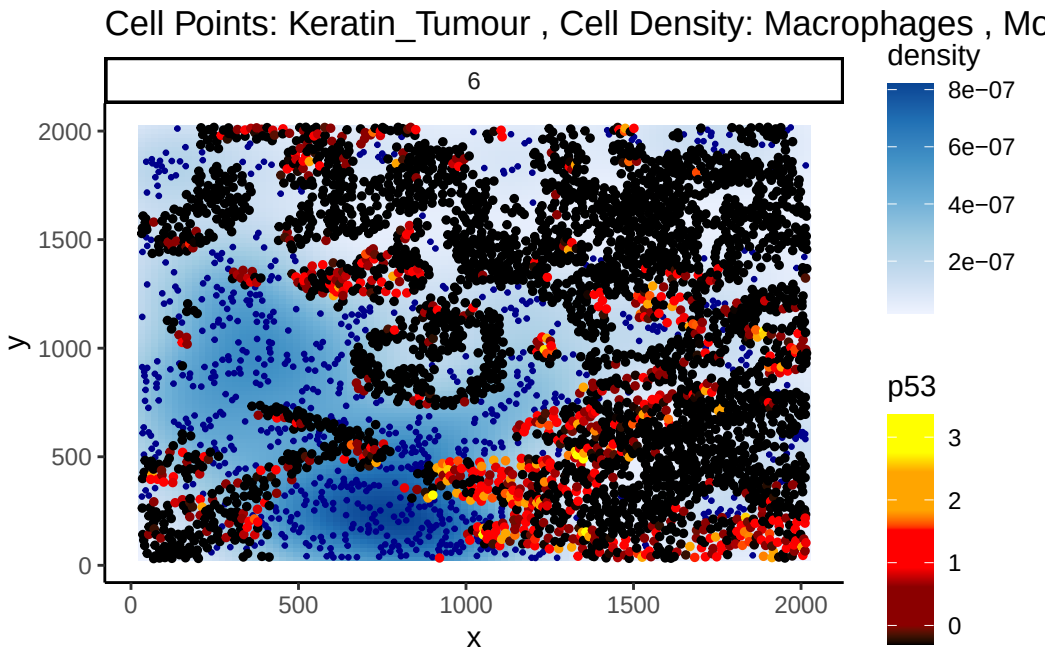
Statial also provides a convenient function for visualising this interaction - `plotStateChanges`. Here, again we can specify `image = 6` and our main cell types of interest, keratin/tumour cells and macrophages, and our marker p53, in the same format as `calcStateChanges`.

Through this analysis, we can observe that keratin/tumour cells closer to a group of macrophages tend to have higher expression of p53, as observed in the first graph. This relationship is quantified with the second graph, showing an overall decrease of p53 expression in keratin/tumour cells as distance to macrophages increase.

These results allow us to essentially arrive at the same result as Kontextual, which calculated a localisation between p53+ keratin/tumour cells and macrophages in the wider context of keratin/tumour cells.

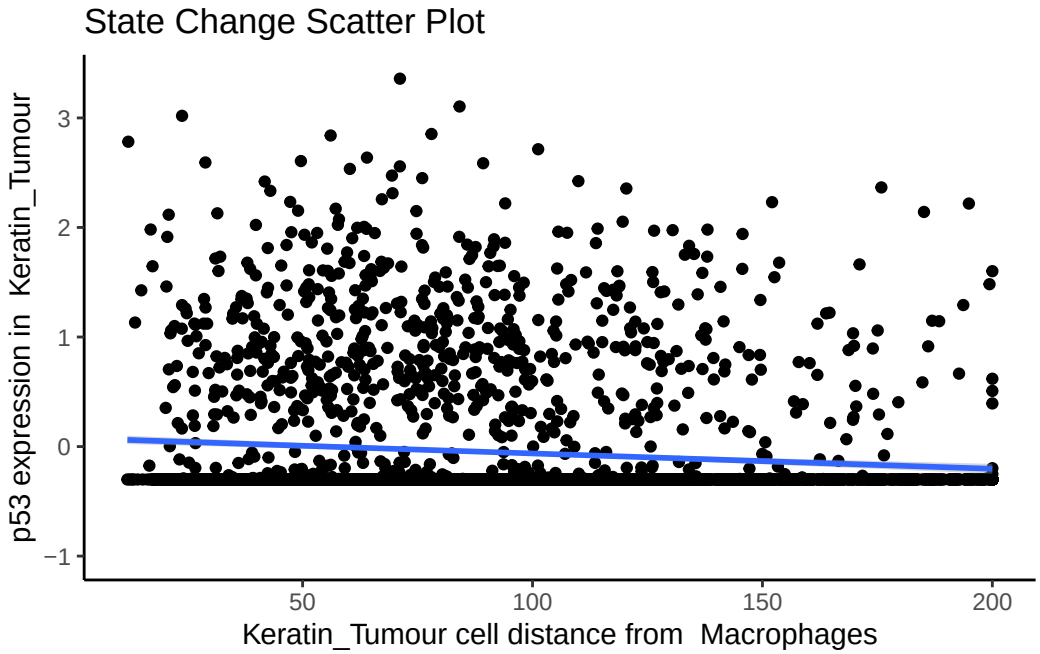
```
p <- plotStateChanges(
  cells = kerenSPE,
  type = "distances",
  image = "6",
  from = "Keratin_Tumour",
  to = "Macrophages",
  marker = "p53",
  size = 1,
  shape = 19,
  interactive = FALSE,
  plotModelFit = FALSE,
  method = "lm"
)

p
```

```
$image
```




```
$scatter
```

```
`geom_smooth()` using formula = 'y ~ x'
```



### 6.2.2 Continuous cell state changes across all images

Beyond looking at single cell-to-cell interactions for a single image, we can also look at all interactions across all images. The `calcStateChanges` function provided by Statial can be expanded for this exact purpose - by not specifying cell types, a marker, or an image, `calcStateChanges` will examine the most significant correlations between distance and marker expression across the entire dataset. Here, we've filtered out the most significant interactions to only include those found within image 6 of the Keren et al. dataset.

```
stateChanges <- calcStateChanges(
  cells = kerenSPE,
  type = "distances",
  nCores = 1,
  minCells = 100
)

stateChanges |>
  filter(imageID == 6) |>
  head(n = 10)
```

```
   imageID primaryCellType otherCellType       marker         coef      tval
1        6  Keratin_Tumour  Unidentified           Na  0.004218419  25.03039
2        6  Keratin_Tumour   Macrophages  HLA_Class_1 -0.003823497 -24.69629
3        6  Keratin_Tumour    CD4_T_cell  HLA_Class_1 -0.003582774 -23.87797
4        6  Keratin_Tumour  Unidentified Beta.catenin  0.005893120  23.41953
5        6  Keratin_Tumour    CD8_T_cell  HLA_Class_1 -0.003154544 -23.13804
6        6  Keratin_Tumour    DC_or_Mono  HLA_Class_1 -0.003353834 -22.98944
7        6  Keratin_Tumour      dn_T_CD3  HLA_Class_1 -0.003123446 -22.63197
8        6  Keratin_Tumour        Tumour  HLA_Class_1  0.003684079  21.94265
9        6  Keratin_Tumour    CD4_T_cell           Fe -0.003457338 -21.43550
10       6  Keratin_Tumour    CD4_T_cell   phospho.S6 -0.002892457 -20.50767
              pval           fdr
1  6.971648e-127 3.382534e-123
2  7.814253e-124 3.430271e-120
3  1.745242e-116 5.362839e-113
4  1.917245e-112 5.523165e-109
5  5.444541e-110 1.434014e-106
6  1.053130e-108 2.696743e-105
7  1.237988e-105 2.889213e-102
8  8.188258e-100  1.640945e-96
9   1.287478e-95  2.260688e-92
10  3.928912e-88  5.748996e-85
```

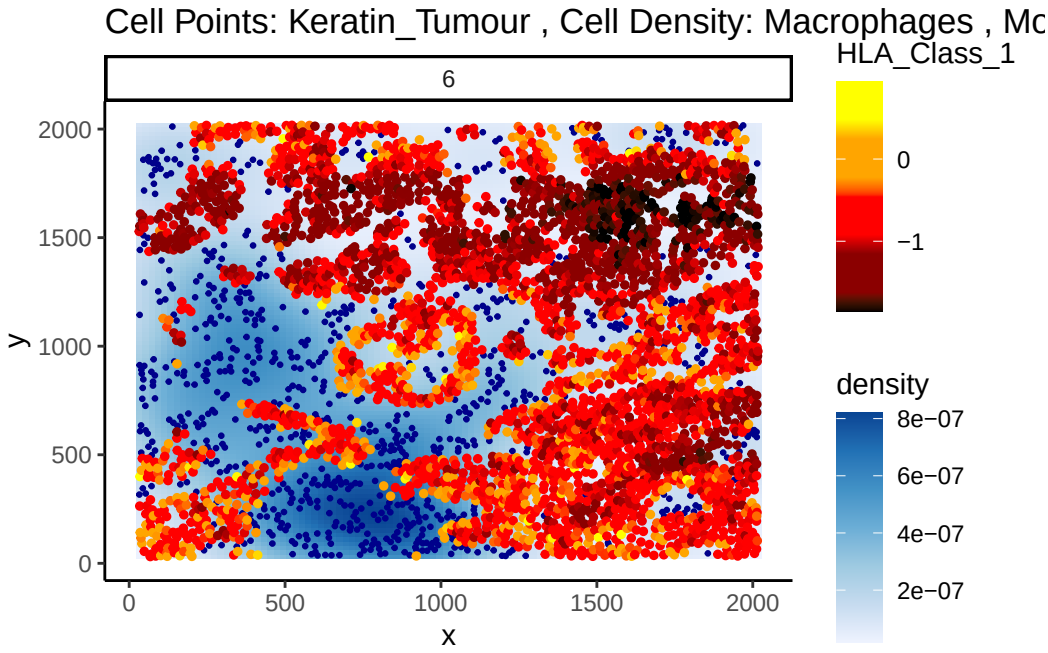
In image 6, the majority of the top 10 most significant interactions occur between keratin/tumour cells and an immune population, and many of these interactions appear to involve the HLA class I ligand.

We can examine some of these interactions further with the `plotStateChanges` function. Taking a closer examination of the relationship between macrophages and keratin/tumour HLA class I expression, the plot below shows us a clear visual correlation - as macrophage density increases, keratin/tumour cells increase their expression HLA class I.

Biologically, HLA Class I is a ligand which exists on all nucleated cells, tasked with presenting internal cell antigens for recognition by the immune system, marking aberrant cells for destruction by either CD8+ T cells or NK cells.

```
p <- plotStateChanges(
  cells = kerenSPE,
  type = "distances",
  image = "6",
  from = "Keratin_Tumour",
  to = "Macrophages",
  marker = "HLA_Class_1",
  size = 1,
  shape = 19,
  interactive = FALSE,
  plotModelFit = FALSE,
  method = "lm"
)

p
```

```
$image
```

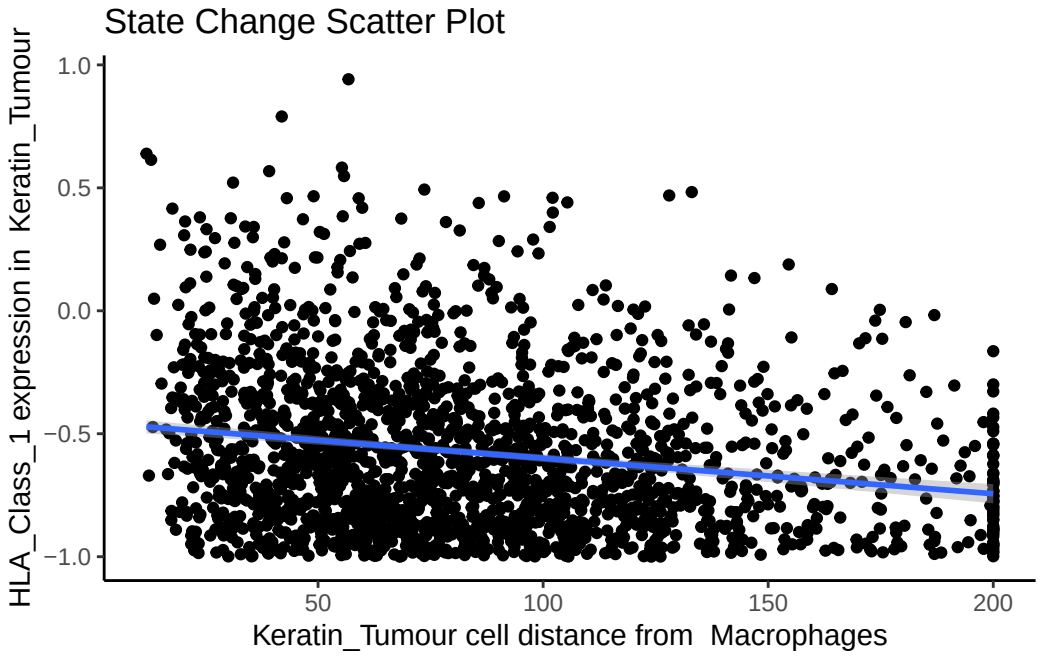



```
$scatter
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 1359 rows containing non-finite outside the scale range
(`stat_smooth()`).
```

```
Warning: Removed 1359 rows containing missing values or values outside the scale range
(`geom_point()`).
```




Next, let's take a look at the top 10 most significant results across all images.

```
stateChanges |> head(n = 10)
```

```
       imageID primaryCellType otherCellType     marker         coef
69468       37     Endothelial        Tumour       Lag3 -0.001621517
153135      11     Neutrophils            NK       CD56 -0.059936866
16402       35      CD4_T_cell        B_cell       CD20 -0.029185750
16498       35      CD4_T_cell    DC_or_Mono       CD20  0.019125946
4891        35          B_cell    DC_or_Mono phospho.S6  0.005282065
16507       35      CD4_T_cell    DC_or_Mono phospho.S6  0.004033218
4885        35          B_cell    DC_or_Mono     HLA.DR  0.011120703
5043        35          B_cell  Other_Immune          P  0.011182182
16354       35      CD4_T_cell      dn_T_CD3       CD20  0.016349492
4888        35          B_cell    DC_or_Mono     H3K9ac  0.005096632
                tval          pval           fdr
69468  -4.916884e+14  0.000000e+00  0.000000e+00
153135 -2.172437e+15  0.000000e+00  0.000000e+00
16402  -4.057355e+01 7.019343e-282 4.313854e-277
16498   4.053436e+01 1.891267e-281 8.717324e-277
4891    4.041385e+01 5.306590e-278 1.956752e-273
16507   3.472882e+01 4.519947e-219 1.388904e-214
4885    3.415344e+01 8.401034e-212 2.212712e-207
5043    3.414375e+01 1.056403e-211 2.434613e-207
```

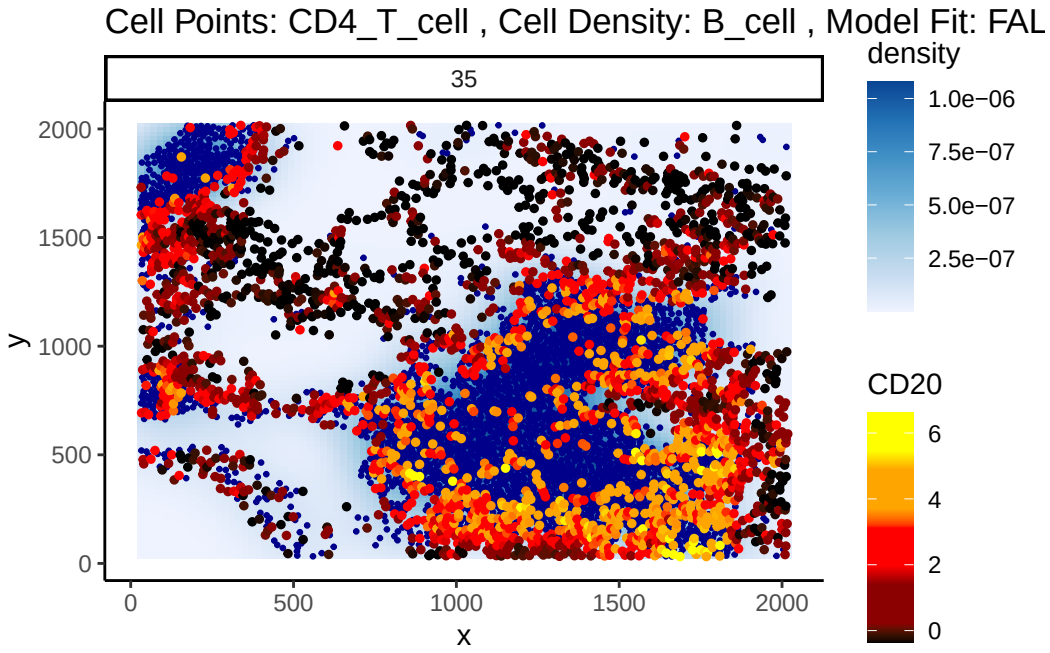
```
16354    3.391901e+01 1.219488e-210 2.498188e-206
4888     3.399856e+01 3.266533e-210 6.022506e-206
```

Immediately, we can appreciate that a couple of these interactions are not biologically plausible. One of the most significant interactions occurs between B cells and CD4 T cells in image 35, where CD4 T cells are found to increase in CD20 expression when in close proximity to B cells. Biologically, CD20 is a highly specific ligand for B cells, and under healthy circumstances are usually not expressed in T cells.

Could this potentially be an artefact of `calcStateChanges`? We can examine the image through the `plotStateChanges` function, where we indeed observe a strong increase in CD20 expression in T cells nearby B cell populations.

```
p <- plotStateChanges(
  cells = kerenSPE,
  type = "distances",
  image = "35",
  from = "CD4_T_cell",
  to = "B_cell",
  marker = "CD20",
  size = 1,
  shape = 19,
  interactive = FALSE,
  plotModelFit = FALSE,
  method = "lm"
)

p
```

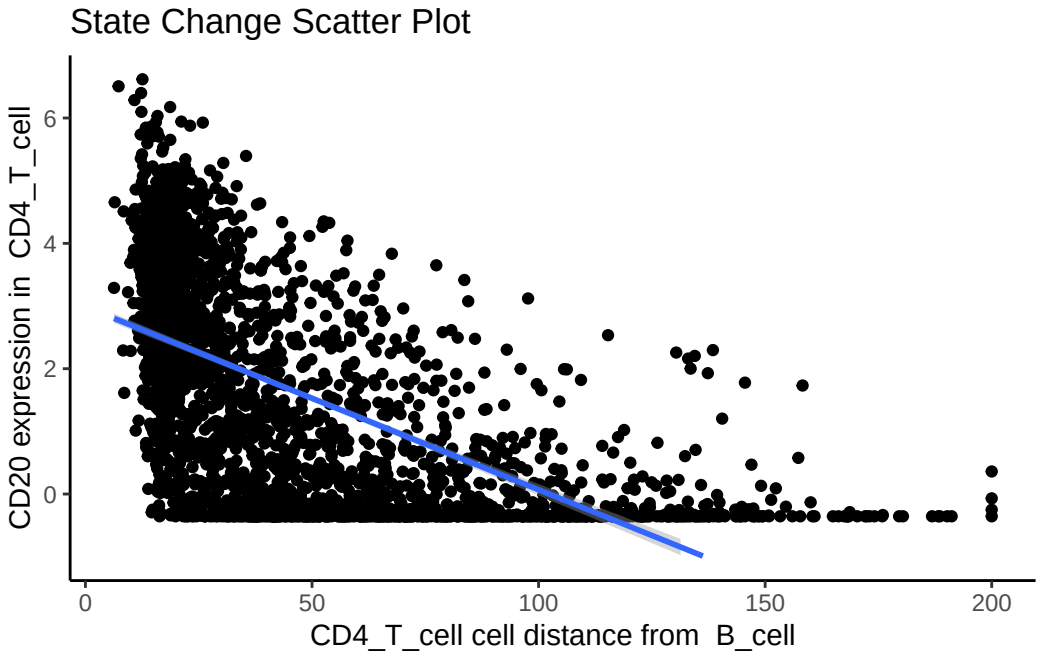
```
$image
```




```
$scatter
```

```
`geom_smooth()` using formula = 'y ~ x'
```

```
Warning: Removed 26 rows containing missing values or values outside the scale range
(`geom_smooth()`).
```




So why are T cells expressing CD20? This brings us to a key problem of cell segmentation - contamination.

### 6.2.3 Contamination (Lateral marker spill over)

Contamination, or lateral marker spill over is an issue that results in a cell's marker expressions being wrongly attributed to another adjacent cell. This issue arises from incorrect segmentation where components of one cell are wrongly determined as belonging to another cell. Alternatively, this issue can arise when antibodies used to tag and measure marker expressions don't latch on properly to a cell of interest, thereby resulting in residual markers being wrongly assigned as belonging to a cell near the intended target cell. It is important that we either correct or account for this incorrect attribution of markers in our modelling process. This is critical in understanding whether significant cell-cell interactions detected are an artefact of technical measurement errors driven by spill over or are real biological changes that represent a shift in a cell's state.

To circumvent this problem, Statial provides a function that predicts the probability that a cell is any particular cell type - `calcContamination`. `calcContamination` returns a dataframe of probabilities demarcating the chance of a cell being any particular cell type. This dataframe is stored under `contaminations` in the `reducedDim` slot of the `SingleCellExperiment` object. It also provides the `rfMainCellProb` column, which provides the probability that a cell is indeed the cell type it has been designated. E.g. For a cell designated as CD8, rfMainCellProb could give a 80% chance that the cell is indeed CD8, due to contamination.

We can then introduce these probabilities as covariates into our linear model by setting `contamination = TRUE` as a parameter in our `calcStateChanges` function. However, this is not a perfect solution for the issue of contamination. As we can see, despite factoring in contamination into our linear model, the correlation between B cell density and CD20 expression in CD4 T cells remains one of the most significant interactions in our model.

```
kerenSPE <- calcContamination(kerenSPE)
```

```
Growing trees.. Progress: 30%. Estimated remaining time: 1 minute, 12 seconds.
Growing trees.. Progress: 62%. Estimated remaining time: 39 seconds.
Growing trees.. Progress: 93%. Estimated remaining time: 7 seconds.
```

```
stateChangesCorrected <- calcStateChanges(
  cells = kerenSPE,
  type = "distances",
  nCores = 1,
  minCells = 100,
  contamination = TRUE
)

stateChangesCorrected |> head(n = 20)
```

```
       imageID primaryCellType otherCellType      marker         coef
69468       37     Endothelial        Tumour        Lag3 -0.001621517
153135      11     Neutrophils            NK        CD56 -0.059936866
16402       35      CD4_T_cell        B_cell        CD20 -0.025155997
16498       35      CD4_T_cell    DC_or_Mono        CD20  0.016317938
16507       35      CD4_T_cell    DC_or_Mono  phospho.S6  0.003619727
16354       35      CD4_T_cell      dn_T_CD3        CD20  0.013869372
4891        35          B_cell    DC_or_Mono  phospho.S6  0.004279319
16357       35      CD4_T_cell      dn_T_CD3      HLA.DR  0.010366047
89188        3  Keratin_Tumour            DC          Ca -0.013725032
3697        28          B_cell            NK          Na -0.004437050
82222       20  Keratin_Tumour        Tumour HLA_Class_1  0.002922784
4741        35          B_cell      dn_T_CD3      HLA.DR  0.008880493
83998       23  Keratin_Tumour  Unidentified HLA_Class_1  0.003104015
16491       35      CD4_T_cell    DC_or_Mono      CSF.1R  0.008618601
82985       21  Keratin_Tumour            DC Pan.Keratin -0.005990784
16363       35      CD4_T_cell      dn_T_CD3  phospho.S6  0.002970004
99073        6  Keratin_Tumour  Unidentified          Na  0.004210683
4885        35          B_cell    DC_or_Mono      HLA.DR  0.008659992
```

```
82177       20  Keratin_Tumour        Tumour          Na  0.002528292
5083        35          B_cell  Other_Immune  phospho.S6  0.004549575
                tval          pval           fdr
69468  -3.668163e+14  0.000000e+00  0.000000e+00
153135 -1.469129e+17  0.000000e+00  0.000000e+00
16402  -3.503922e+01 4.151258e-222 2.551225e-217
16498   3.451351e+01 1.286531e-216 5.929942e-212
16507   2.995929e+01 1.502564e-170 5.540553e-166
16354   2.994028e+01 2.305225e-170 7.083573e-166
4891    2.953140e+01 1.404125e-165 3.698265e-161
16357   2.920130e+01 3.488396e-163 8.039445e-159
89188  -2.940905e+01 6.078033e-161 1.245119e-156
3697   -2.890578e+01 2.110059e-156 3.890316e-152
82222   2.563648e+01 1.248985e-132 2.093412e-128
4741    2.580213e+01 6.315062e-131 9.702567e-127
83998   2.513875e+01 2.869643e-128 4.069816e-124
16491   2.539208e+01 9.301666e-128 1.224963e-123
82985  -2.470505e+01 8.193691e-127 1.007114e-122
16363   2.510997e+01 2.992332e-125 3.448101e-121
99073   2.468985e+01 9.949871e-124 1.079093e-119
4885    2.465260e+01 8.651171e-121 8.861202e-117
82177   2.424891e+01 6.517655e-120 6.324526e-116
5083    2.425395e+01 2.432632e-117 2.242522e-113
```

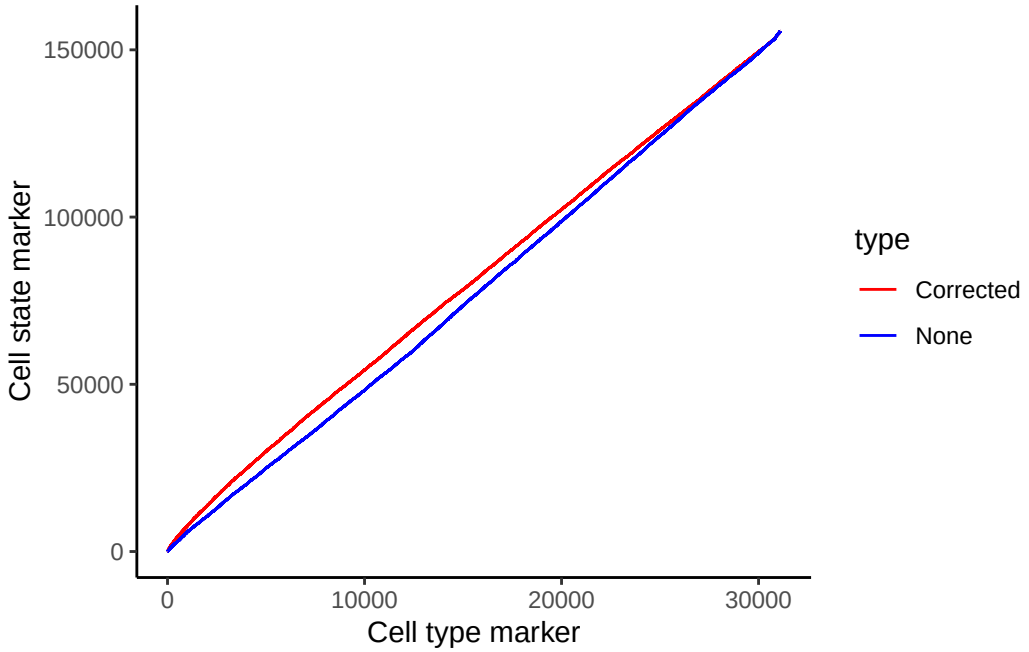
However, this does not mean factoring in contamination into our linear model was ineffective.

Whilst our correction attempts do not rectify every relationship which arises due to contamination, we show that a significant portion of these relationships are rectified. We can show this by plotting a ROC curve of true positives against false positives. In general, cell type specific markers such as CD4, CD8, and CD20 should not change in cells they are not specific to. Therefore, relationships detected to be significant involving these cell type markers are likely false positives and will be treated as such for the purposes of evaluation. Meanwhile, cell state markers are predominantly likely to be true positives.

Plotting the relationship between false positives and true positives, we'd expect the contamination correction to be greatest in the relationships with the top 100 lowest p values, where we indeed see more true positives than false positives with contamination correction.

```
cellTypeMarkers <- c("CD3", "CD4", "CD8", "CD56", "CD11c", "CD68", "CD45", "CD20")

values <- c("blue", "red")
names(values) <- c("None", "Corrected")
```

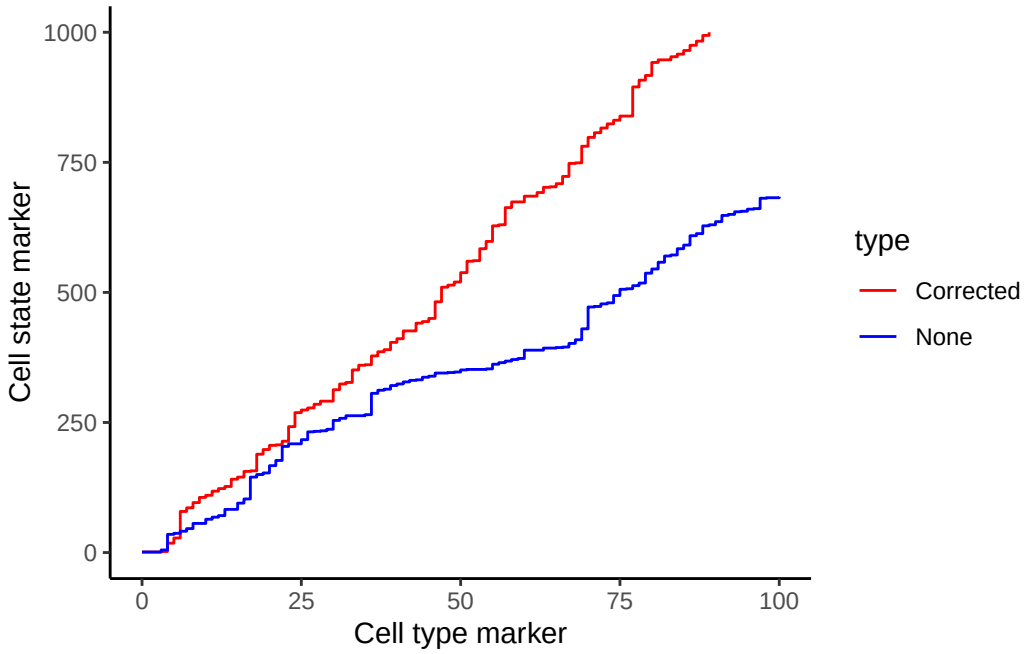
```r
df <- rbind(
  data.frame(TP = cumsum(stateChanges$marker %in% cellTypeMarkers), FP = cumsum(!stateChanges
  data.frame(TP = cumsum(stateChangesCorrected$marker %in% cellTypeMarkers), FP = cumsum(!sta
)

ggplot(df, aes(x = TP, y = FP, colour = type)) +
  geom_line() +
  labs(y = "Cell state marker", x = "Cell type marker") +
  scale_colour_manual(values = values)
```




Here, we zoom in on the ROC curve where the top 100 lowest p values occur, where we indeed see more true positives than false positives with contamination correction.

```r
ggplot(df, aes(x = TP, y = FP, colour = type)) +
  geom_line() +
  xlim(0, 100) +
  ylim(0, 1000) +
  labs(y = "Cell state marker", x = "Cell type marker") +
  scale_colour_manual(values = values)
```

```
Warning: Removed 371471 rows containing missing values or values outside the scale range
(`geom_line()`).
```




### 6.2.4 Associate continuous state changes with survival outcomes

Similiar to `Kontextual`, we can run a similar survival analysis using our state changes results. Here, `prepMatrix` extracts the coefficients, or the `coef` column of `stateChanges` by default. To use the t values instead, specify `column = "tval"` in the `prepMatrix` function.

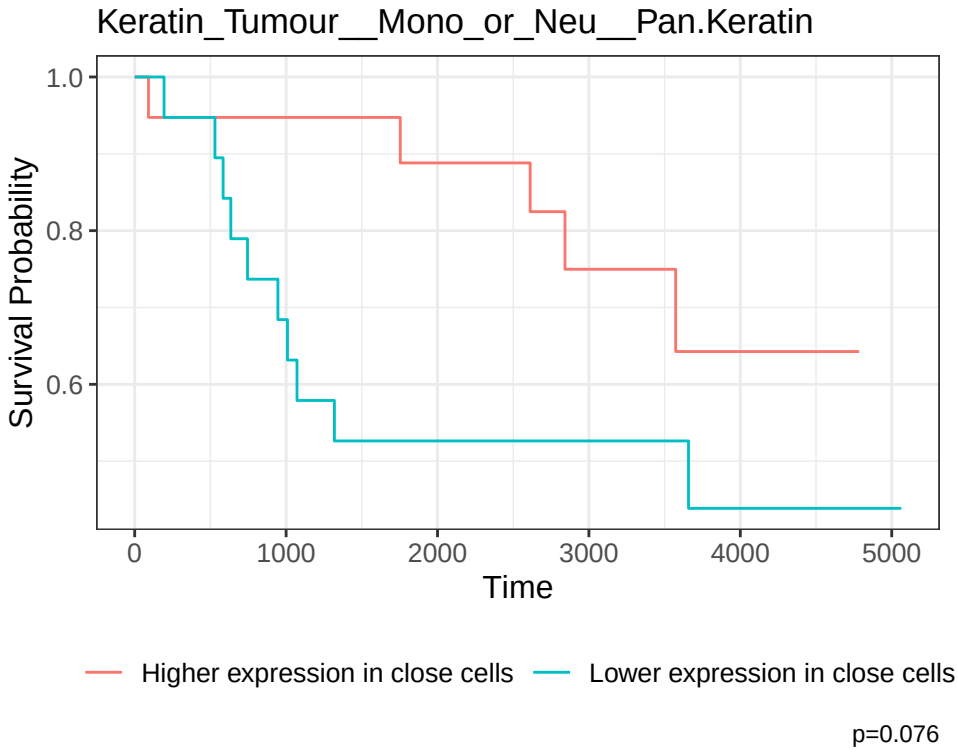
```
# Preparing features for Statial
stateMat <- prepMatrix(stateChanges)

# Ensuring rownames of stateMat match up with rownames of the survival vector
stateMat <- stateMat[names(kerenSurv), ]

# Remove some very small values
stateMat <- stateMat[, colMeans(abs(stateMat) > 0.0001) > .8]

survivalResults <- colTest(stateMat, kerenSurv, type = "survival")

head(survivalResults)
```

```
                                       coef se.coef   pval adjPval
Keratin_Tumour__Mono_or_Neu__Pan.Keratin -280      89 0.0018    0.63
Macrophages__Keratin_Tumour__HLA_Class_1  220      75 0.0034    0.63
```

```
Keratin_Tumour__CD8_T_cell__Keratin6     -220      77 0.0036    0.63
Macrophages__Other_Immune__HLA_Class_1   -480     170 0.0057    0.75
Keratin_Tumour__Mesenchymal__dsDNA       -810     310 0.0094    0.80
Keratin_Tumour__Unidentified__H3K27me3    490     190 0.0100    0.80
                                                                    cluster
Keratin_Tumour__Mono_or_Neu__Pan.Keratin Keratin_Tumour__Mono_or_Neu__Pan.Keratin
Macrophages__Keratin_Tumour__HLA_Class_1 Macrophages__Keratin_Tumour__HLA_Class_1
Keratin_Tumour__CD8_T_cell__Keratin6         Keratin_Tumour__CD8_T_cell__Keratin6
Macrophages__Other_Immune__HLA_Class_1     Macrophages__Other_Immune__HLA_Class_1
Keratin_Tumour__Mesenchymal__dsDNA             Keratin_Tumour__Mesenchymal__dsDNA
Keratin_Tumour__Unidentified__H3K27me3     Keratin_Tumour__Unidentified__H3K27me3
```

For our state changes results, `Keratin_Tumour__CD4_Cell__Keratin6` is the most significant pairwise relationship which contributes to patient survival. That is, the relationship between HLA class I expression in keratin/tumour cells and their spatial proximity to mesenchymal cells. As there is a negative coeffcient associated with this relationship, which tells us that higher HLA class I expression in keratin/tumour cells nearby mesenchymal cell populations lead to poorer survival outcomes for patients.

```r
# Selecting the most significant relationship
survRelationship <- stateMat[["Keratin_Tumour__Mono_or_Neu__Pan.Keratin"]]
survRelationship <- ifelse(survRelationship > median(survRelationship), "Higher expression i

# Plotting Kaplan-Meier curve
survfit2(kerenSurv ~ survRelationship) |>
  ggsurvfit() +
  add_pvalue() +
  ggtitle("Keratin_Tumour__Mono_or_Neu__Pan.Keratin")
```




## 6.3 scFeatures: Moran's I

## 6.4 sessionInfo

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Debian GNU/Linux 12 (bookworm)

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.21.so;  LAPACK version 3

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
```

```
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: Australia/Sydney
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] SpatialDatasets_1.4.0       SpatialExperiment_1.16.0
 [3] ExperimentHub_2.14.0        AnnotationHub_3.14.0
 [5] BiocFileCache_2.14.0        dbplyr_2.5.0
 [7] Statial_1.7.4               testthat_3.2.1.1
 [9] treekoR_1.14.0              tibble_3.2.1
[11] ggsurvfit_1.1.0             ggplot2_3.5.1
[13] SingleCellExperiment_1.28.1 dplyr_1.1.4
[15] lisaClust_1.14.4            ClassifyR_3.10.0
[17] survival_3.7-0              BiocParallel_1.40.0
[19] MultiAssayExperiment_1.32.0 SummarizedExperiment_1.36.0
[21] Biobase_2.66.0              GenomicRanges_1.58.0
[23] GenomeInfoDb_1.42.0         IRanges_2.40.0
[25] MatrixGenerics_1.18.0       matrixStats_1.4.1
[27] S4Vectors_0.44.0            BiocGenerics_0.52.0
[29] generics_0.1.3              spicyR_1.18.0

loaded via a namespace (and not attached):
  [1] fs_1.6.5                    spatstat.sparse_3.1-0
  [3] devtools_2.4.5              httr_1.4.7
  [5] hopach_2.66.0               RColorBrewer_1.1-3
  [7] doParallel_1.0.17           numDeriv_2016.8-1.1
  [9] profvis_0.4.0               tools_4.4.1
 [11] backports_1.5.0             utf8_1.2.4
 [13] R6_2.5.1                    lazyeval_0.2.2
 [15] mgcv_1.9-1                  GetoptLong_1.0.5
 [17] urlchecker_1.0.1            withr_3.0.2
 [19] coxme_2.2-22                cli_3.6.3
 [21] spatstat.explore_3.3-3      sandwich_3.1-1
 [23] labeling_0.4.3              mvtnorm_1.3-2
 [25] spatstat.data_3.1-2         systemfonts_1.1.0
 [27] yulab.utils_0.1.8           ggupset_0.4.0
 [29] colorRamps_2.3.4            sessioninfo_1.2.2
```

```
 [31] limma_3.62.1               RSQLite_2.3.7
 [33] flowCore_2.18.0            rstudioapi_0.17.1
 [35] gridGraphics_0.5-1         shape_1.4.6.1
 [37] spatstat.random_3.3-2      car_3.1-3
 [39] scam_1.2-17                Matrix_1.7-1
 [41] RProtoBufLib_2.18.0        fansi_1.0.6
 [43] abind_1.4-8                lifecycle_1.0.4
 [45] yaml_2.3.10                multcomp_1.4-26
 [47] edgeR_4.4.0                carData_3.0-5
 [49] SparseArray_1.6.0          Rtsne_0.17
 [51] blob_1.2.4                 grid_4.4.1
 [53] promises_1.3.0             crayon_1.5.3
 [55] bdsmatrix_1.3-7            miniUI_0.1.1.1
 [57] lattice_0.22-6             KEGGREST_1.46.0
 [59] magick_2.8.5               pillar_1.9.0
 [61] knitr_1.49                 ComplexHeatmap_2.22.0
 [63] rjson_0.2.23               boot_1.3-31
 [65] codetools_0.2-20           glue_1.8.0
 [67] ggiraph_0.8.10             ggfun_0.1.7
 [69] spatstat.univar_3.1-1      data.table_1.16.2
 [71] remotes_2.5.0              vctrs_0.6.5
 [73] png_0.1-8                  treeio_1.30.0
 [75] gtable_0.3.6               cachem_1.1.0
 [77] xfun_0.49                  S4Arrays_1.6.0
 [79] mime_0.12                  ConsensusClusterPlus_1.70.0
 [81] pheatmap_1.0.12            iterators_1.0.14
 [83] tinytex_0.54               cytolib_2.18.0
 [85] statmod_1.5.0              ellipsis_0.3.2
 [87] TH.data_1.1-2              nlme_3.1-166
 [89] usethis_3.0.0              ggtree_3.14.0
 [91] bit64_4.5.2                filelock_1.0.3
 [93] rprojroot_2.0.4            DBI_1.2.3
 [95] colorspace_2.1-1           tidyselect_1.2.1
 [97] curl_6.0.1                 bit_4.5.0
 [99] compiler_4.4.1             diffcyt_1.26.0
[101] desc_1.4.3                 DelayedArray_0.32.0
[103] plotly_4.10.4              scales_1.3.0
[105] rappdirs_0.3.3             stringr_1.5.1
[107] digest_0.6.37              goftest_1.2-3
[109] fftwtools_0.9-11           spatstat.utils_3.1-1
[111] minqa_1.2.8                rmarkdown_2.29
[113] XVector_0.46.0             htmltools_0.5.8.1
[115] pkgconfig_2.0.3            lme4_1.1-35.5
```

```
[117] fastmap_1.2.0              rlang_1.1.4
[119] GlobalOptions_0.1.2        htmlwidgets_1.6.4
[121] ggthemes_5.1.0             UCSC.utils_1.2.0
[123] shiny_1.9.1                ggh4x_0.2.8
[125] farver_2.1.2               zoo_1.8-12
[127] jsonlite_1.8.9             magrittr_2.0.3
[129] Formula_1.2-5              GenomeInfoDbData_1.2.13
[131] ggplotify_0.1.2            patchwork_1.3.0
[133] munsell_0.5.1              Rcpp_1.0.13-1
[135] ape_5.8                    ggnewscale_0.5.0
[137] stringi_1.8.4              brio_1.1.5
[139] zlibbioc_1.52.0            MASS_7.3-61
[141] plyr_1.8.9                 pkgbuild_1.4.5
[143] parallel_4.4.1             deldir_2.0-4
[145] Biostrings_2.74.0          splines_4.4.1
[147] tensor_1.5                 circlize_0.4.16
[149] locfit_1.5-9.10            igraph_2.1.1
[151] ggpubr_0.6.0               uuid_1.2-1
[153] ranger_0.17.0              spatstat.geom_3.3-3
[155] ggsignif_0.6.4             reshape2_1.4.4
[157] pkgload_1.4.0              BiocVersion_3.20.0
[159] XML_3.99-0.17              evaluate_1.0.1
[161] BiocManager_1.30.25        nloptr_2.1.1
[163] foreach_1.5.2              tweenr_2.0.3
[165] httpuv_1.6.15              tidyr_1.3.1
[167] purrr_1.0.2                polyclip_1.10-7
[169] clue_0.3-66                ggforce_0.4.2
[171] broom_1.0.7                xtable_1.8-4
[173] tidytree_0.4.6             rstatix_0.7.2
[175] later_1.3.2                viridisLite_0.4.2
[177] class_7.3-22               lmerTest_3.1-3
[179] aplot_0.2.3                AnnotationDbi_1.68.0
[181] memoise_2.0.1              FlowSOM_2.14.0
[183] cluster_2.1.6              concaveman_1.1.0
```

# 7 Classification

Steps:

1. Motivation of classification (prediction)
2. Classification of patients by condition
3. Classification of patients by survival
4. Easy and Hard to classify patients (samplesMetricMap)
5. Maximising accuracy during classification (parameter tuning for crossValidate)

## 7.1 Why classify?

When it comes to biological datasets, the end goal is either mechanistic or translational. For example, if we had a mechanistic end goal, we might want to find what genes are differentially expressed between two conditions, and further aim to characterise the pathways which lead to this differential expression. Alternatively, if the end goal is translational, we might want to use a biological dataset that can be relatively cheaply obtained (e.g. IMC) to predict whether a patient's disease will progress or not progress (e.g. metastasize in cancer).

Regardless of the end goal, classification is an important end-point for any biological dataset, as it provides us information on:

1. Important features
2. Predictability of the full dataset, or a subset of the dataset

`ClassifyR` simplifies the process of classification, and provides quick and simple ways to analyse datasets and determine how well biological datasets can be classified into disease states using particular feature generation techniques. In this chapter, we will provide a guide on how to use `ClassifyR` on biological datasets, and typical workflows for classification.

## 7.2 Cross validation

Cross validation is an essential part of any classification process, and is necessary to robustly determine how well your model has performed. `ClassifyR` makes this extremely easy to implement with the `crossValidate` function.

```
library(ClassifyR)
library(lisaClust)

BPPARAM <- simpleSeg:::generateBPParam(cores = 40)
```

Using all the features we've generated in the previous chapters, we can build a list of features matrices.

```
kerenSPE <- SpatialDatasets::spe_Keren_2018()
```

```
see ?SpatialDatasets and browseVignettes('SpatialDatasets') for documentation
```

```
loading from cache
```

# 8 Case Study: Head and Neck Squamous Cell Carcinoma (Ferguson et al., 2022)

## 8.1 Load libraries

```
library(cytomapper)
library(dplyr)
library(ggplot2)
library(simpleSeg)
library(FuseSOM)
library(ggpubr)
library(scater)
library(spicyR)
library(ClassifyR)
library(lisaClust)
library(Statial)
library(tidySingleCellExperiment)
library(SpatialExperiment)
library(SpatialDatasets)
```

## 8.2 Global parameters

It is convenient to set the number of cores for running code in parallel. Please choose a number that is appropriate for your resources. Set the `use_mc` flag to `TRUE` if you would like to use parallel processing for the rest of the vignette. A minimum of 2 cores is suggested since running this workflow is rather computationally intensive.

```
use_mc <- TRUE

if (use_mc) {
  nCores <- max(parallel::detectCores()/2, 1)
} else {
  nCores <- 2
```

```
}
BPPARAM <- simpleSeg:::generateBPParam(nCores)

theme_set(theme_classic())
```

## 8.3 Context

In the following we will re-analyse some IMC data (Ferguson et al, 2022) profiling the spatial landscape of head and neck cutaneous squamous cell carcinomas (HNcSCC), the second most common type of skin cancer. The majority of HNcSCC can be treated with surgery and good local control, but a subset of large tumours infiltrate subcutaneous tissue and are considered high risk for local recurrence and metastases. The key conclusion of this manuscript (amongst others) is that spatial information about cells and the immune environment can be used to predict primary tumour progression or metastases in patients. We will use our spicy workflow to reach a similar conclusion.

The R code for this analysis is available on github https://github.com/SydneyBioX/spicyWorkflow.

## 8.4 Read in images

Once the spicyWorkflow package is installed, these images will be located within the `spicyWorkflow` folder where the `spicyWorkflow` package is installed, under `inst/extdata/images`. Here we use `loadImages()` from the `cytomapper` package to load all the tiff images into a `CytoImageList` object and store the images as h5 file on-disk in a temporary directory using the `h5FilesPath = HDF5Array::getHDF5DumpDir()` parameter.

We will also assign the metadata columns of the `CytoImageList` object using the `mcols()` function.

```
pathToImages <- SpatialDatasets::Ferguson_Images()
```

```
see ?SpatialDatasets and browseVignettes('SpatialDatasets') for documentation
```

```
loading from cache
```

```
tmp <- tempfile()
unzip(pathToImages, exdir = tmp)

# Store images in a CytoImageList on_disk as h5 files to save memory.
images <- cytomapper::loadImages(
  tmp,
  single_channel = TRUE,
  on_disk = TRUE,
  h5FilesPath = HDF5Array::getHDF5DumpDir(),
  BPPARAM = BPPARAM
)

mcols(images) <- S4Vectors::DataFrame(imageID = names(images))
```

### 8.4.1 Clean channel names

As we're reading the image channels directly from the names of the TIFF image, often these channel names will need to be cleaned for ease of downstream processing.

The channel names can be accessed from the `CytoImageList` object using the `channelNames()` function.

```
cn <- channelNames(images) # Read in channel names
head(cn)
```

```
[1] "139La_139La_panCK.ome"      "141Pr_141Pr_CD20.ome"
[3] "142Nd_142Nd_HH3.ome"        "143Nd_143Nd_CD45RA.ome"
[5] "146Nd_146Nd_CD8a.ome"       "147Sm_147Sm_podoplanin.ome"
```

```
cn <- sub(".*_", "", cn) # Remove preceding letters
cn <- sub(".ome", "", cn) # Remove the .ome
head(cn)
```

```
[1] "panCK"      "CD20"       "HH3"        "CD45RA"     "CD8a"
[6] "podoplanin"
```

```
channelNames(images) <- cn # Reassign channel names
```

### 8.4.2 Clean image names

Similarly, the image names will be taken from the folder name containing the individual TIFF images for each channel. These will often also need to be cleaned.

```
head(names(images))
```

```
[1] "ROI001_ROI 01_F3_SP16-001550_1E" "ROI002_ROI 02_F4_SP16-001550_1E"
[3] "ROI003_ROI 03_F5_SP16-001550_1E" "ROI005_ROI 05_G4_SP17-002069_1F"
[5] "ROI006_ROI 06_G5_SP17-002069_1F" "ROI007_ROI 07_G6_SP17-005715_1B"
```

```
nam <- sapply(strsplit(names(images), "_"), `[`, 3)
head(nam)
```

```
[1] "F3" "F4" "F5" "G4" "G5" "G6"
```

```
names(images) <- nam # Reassigning image names
mcols(images)[["imageID"]] <- nam # Reassigning image names
```

## 8.5 SimpleSeg: Segment the cells in the images

Our simpleSeg R package on https://github.com/SydneyBioX/simpleSeg provides a series of functions to generate simple segmentation masks of images. These functions leverage the functionality of the EBImage package on Bioconductor. For more flexibility when performing your segmentation in R we recommend learning to use the EBimage package. A key strength of the simpleSeg package is that we have coded multiple ways to perform some simple segmentation operations as well as incorporating multiple automatic procedures to optimise some key parameters when these aren't specified.

### 8.5.1 Run simpleSeg

If your images are stored in a `list` or `CytoImageList` they can be segmented with a simple call to `simpleSeg()`. To summarise, `simpleSeg()` is an R implementation of a simple segmentation technique which traces out the nuclei using a specified channel using `nucleus` then dilates around the traced nuclei by a specified amount using `discSize`. The nucleus can be traced out using either one specified channel, or by using the principal components of all channels most correlated to the specified nuclear channel by setting `pca = TRUE`.

In the particular example below, we have asked `simpleSeg` to do the following:

By setting `nucleus = c("HH3")`, we've asked simpleSeg to trace out the nuclei signal in the images using the HH3 channel. By setting `pca = TRUE`, simpleSeg segments out the nuclei mask using a principal component analysis of all channels and using the principal components most aligned with the nuclei channel, in this case, HH3. By setting `cellBody = "dilate"`, simpleSeg uses a dilation strategy of segmentation, expanding out from the nucleus by a specified `discSize`. By setting `discSize = 3`, simpleSeg dilates out from the nucleus by 3 pixels. By setting `sizeSelection = 20`, simpleSeg ensures that only cells with a size greater than 20 pixels will be used. By setting `transform = "sqrt"`, simpleSeg square root transforms each of the channels prior to segmentation. By setting `tissue = c("panCK", "CD45", "HH3")`, we specify a tissue mask which simpleSeg uses, filtering out all background noise outside the tissue mask. This is important as these are tumour cores, wand hence circular, so we'd want to ignore background noise which happens outside of the tumour core.

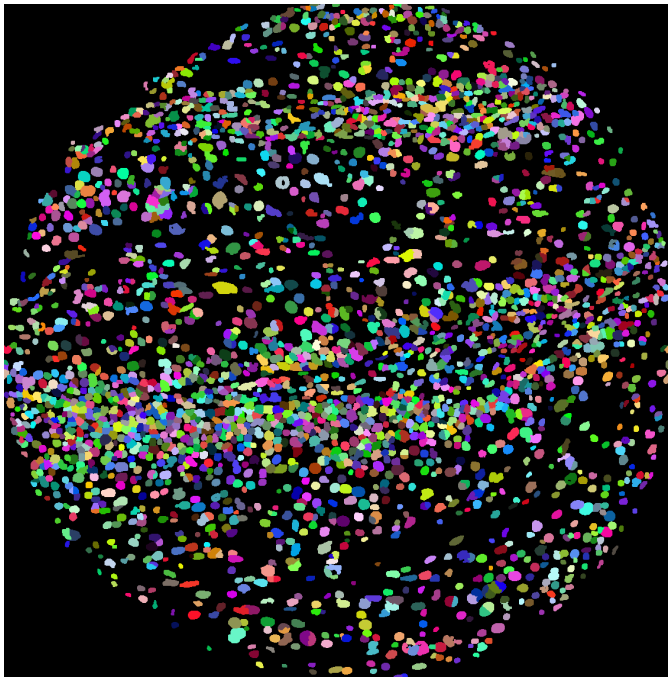
There are many other parameters that can be specified in simpleSeg (`smooth`, `watershed`, `tolerance`, and `ext`), and we encourage the user to select the best parameters which suit their biological context.

```
masks <- simpleSeg(images,
                   nucleus = c("HH3"),
                   pca = TRUE,
                   cellBody = "dilate",
                   discSize = 3,
                   sizeSelection = 20,
                   transform = "sqrt",
                   tissue = c("panCK", "CD45", "HH3"),
                   cores = nCores
                   )
```

### 8.5.2 Visualise separation

The `display` and `colorLabels` functions in `EBImage` make it very easy to examine the performance of the cell segmentation. The great thing about `display` is that if used in an interactive session it is very easy to zoom in and out of the image.

```
EBImage::display(colorLabels(masks[[1]]))
```

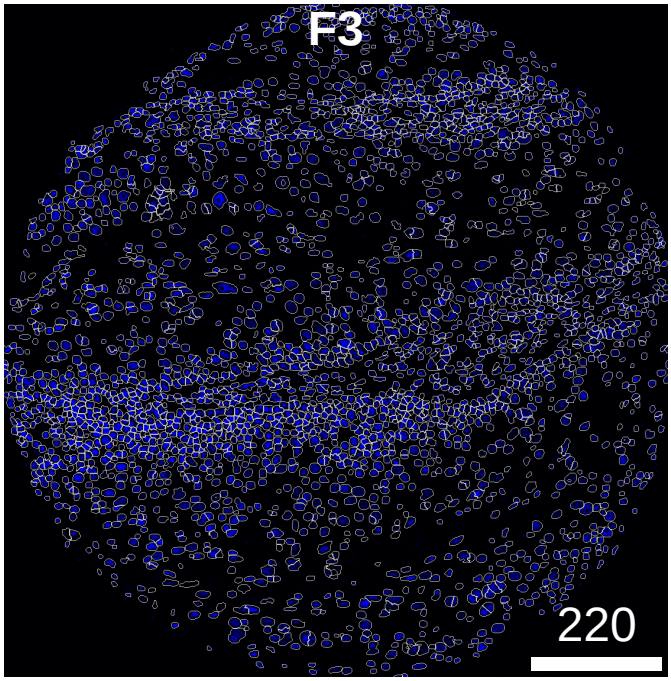


### 8.5.3 Visualise outlines

The `plotPixels` function in `cytomapper` makes it easy to overlay the mask on top of the nucleus intensity marker to see how well our segmentation process has performed. Here we can see that the segmentation appears to be performing reasonably.

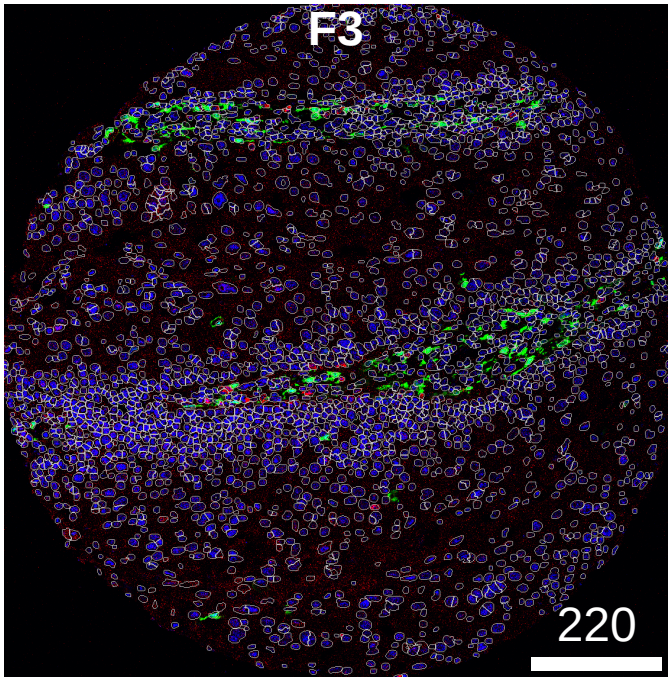
If you see over or under-segmentation of your images, `discSize` is a key parameter in `simpleSeg()` for optimising the size of the dilation disc after segmenting out the nuclei.

```
plotPixels(image = images["F3"],
           mask = masks["F3"],
           img_id = "imageID",
           colour_by = c("HH3"),
           display = "single",
           colour = list(HH3 = c("black","blue")),
           legend = NULL,
           bcg = list(
             HH3 = c(1, 1, 2)
           ))
```




If you wish to visualise multiple markers instead of just the HH3 marker and see how the segmentation mask performs, this can also be done. Here, we can see that our segmentation mask has done a good job of capturing the CD31 signal, but perhaps not such a good job of capturing the FXIIIA signal, which often lies outside of our dilated nuclear mask. This could suggest that we might need to increase the `discSize` of our dilation.

```
plotPixels(image = images["F3"],
           mask = masks["F3"],
           img_id = "imageID",
           colour_by = c("HH3", "CD31", "FX111A"),
           display = "single",
           colour = list(HH3 = c("black","blue"),
                         CD31 = c("black", "red"),
                         FX111A = c("black", "green") ),
           legend = NULL,
           bcg = list(
             HH3 = c(1, 1, 2),
             CD31 = c(0, 1, 2),
             FX111A = c(0, 1, 1.5)
           ))
```




## 8.6 Summarise cell features.

In order to characterise the phenotypes of each of the segmented cells, `measureObjects()` from `cytomapper` will calculate the average intensity of each channel within each cell as well as a few morphological features. By default, the `measureObjects()` function will return a `SingleCellExperiment` object, where the channel intensities are stored in the `counts` assay and the spatial location of each cell is stored in `colData` in the `m.cx` and `m.cy` columns.

However, you can also specify `measureObjects()` to return a `SpatialExperiment` object by specifying `return_as = "spe"`. As a `SpatialExperiment` object, the spatial location of each cell is stored in the `spatialCoords` slot, as `m.cx` and `m.cy`, which simplifies plotting. In this demonstration, we will return a `SpatialExperiment` object.

```
# Summarise the expression of each marker in each cell
cells <- cytomapper::measureObjects(masks,
                                    images,
                                    img_id = "imageID",
                                    return_as = "spe",
                                    BPPARAM = BPPARAM)

spatialCoordsNames(cells) <- c("x", "y")
```

## 8.7 Load the clinical data

To associate features in our image with disease progression, it is important to read in information which links image identifiers to their progression status. We will do this here, making sure that our `imageID` match.

### 8.7.1 Read the clinical data

```
clinical <- read.csv(
  system.file(
    "extdata/clinicalData_TMA1_2021_AF.csv",
    package = "spicyWorkflow"
  )
)

rownames(clinical) <- clinical$imageID
clinical <- clinical[names(images), ]
```

### 8.7.2 Put the clinical data into the colData of SingleCellExperiment

```
colData(cells) <- cbind(colData(cells), clinical[cells$imageID, ])
```

```
save(cells, file = "spe_Ferguson_2022.rda")
```

In case you already have your SCE object, you may only be interested in our downstream workflow. For the sake of convenience, we've provided capability to directly load in the SpatialExperiment (SPE) object that we've generated.

```
load(system.file("extdata/cells.rda", package = "spicyWorkflow"))
```

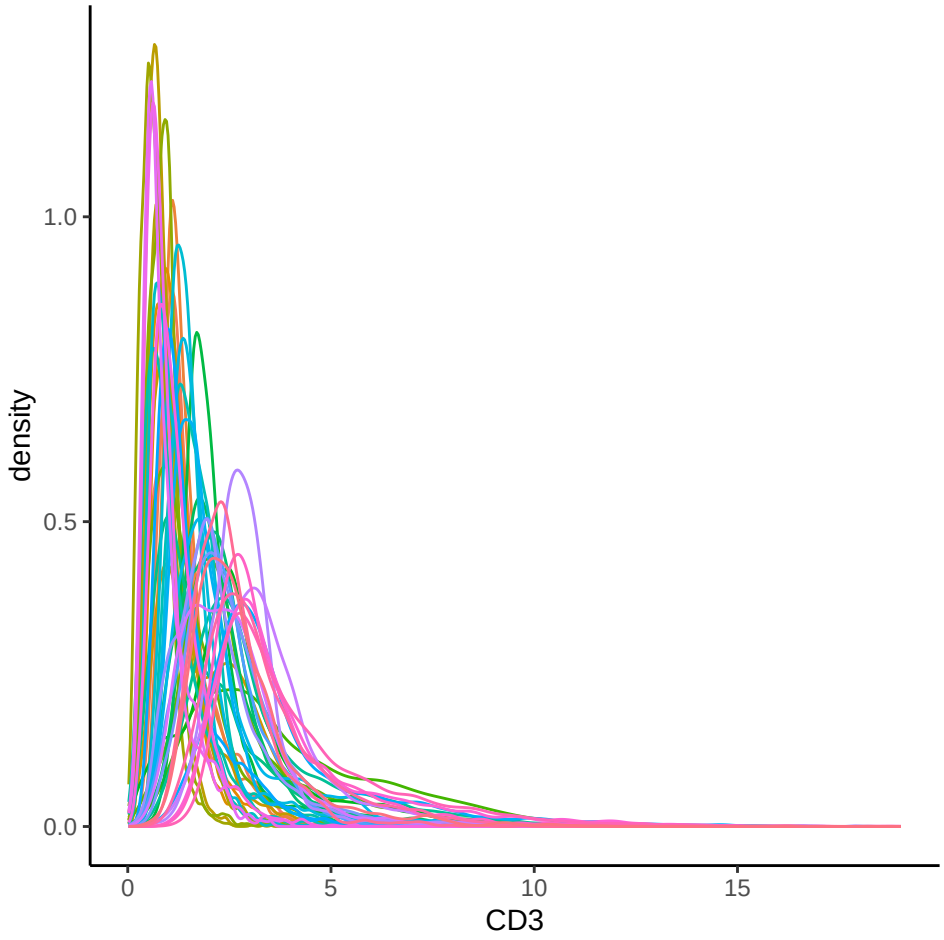
## 8.8 Normalise the data

We should check to see if the marker intensities of each cell require some form of transformation or normalisation. The reason we do this is two-fold:
1) The intensities of images are often highly skewed, preventing any meaningful downstream analysis.

2) The intensities across different images are often different, meaning that what is considered "positive" can be different across images.

By transforming and normalising the data, we aim to reduce these two effects. Here we extract the intensities from the `counts` assay. Looking at CD3 which should be expressed in the majority of the T cells, the intensities are clearly very skewed, and it is difficult to see what is considered a CD3- cell, and what is a CD3+ cell.

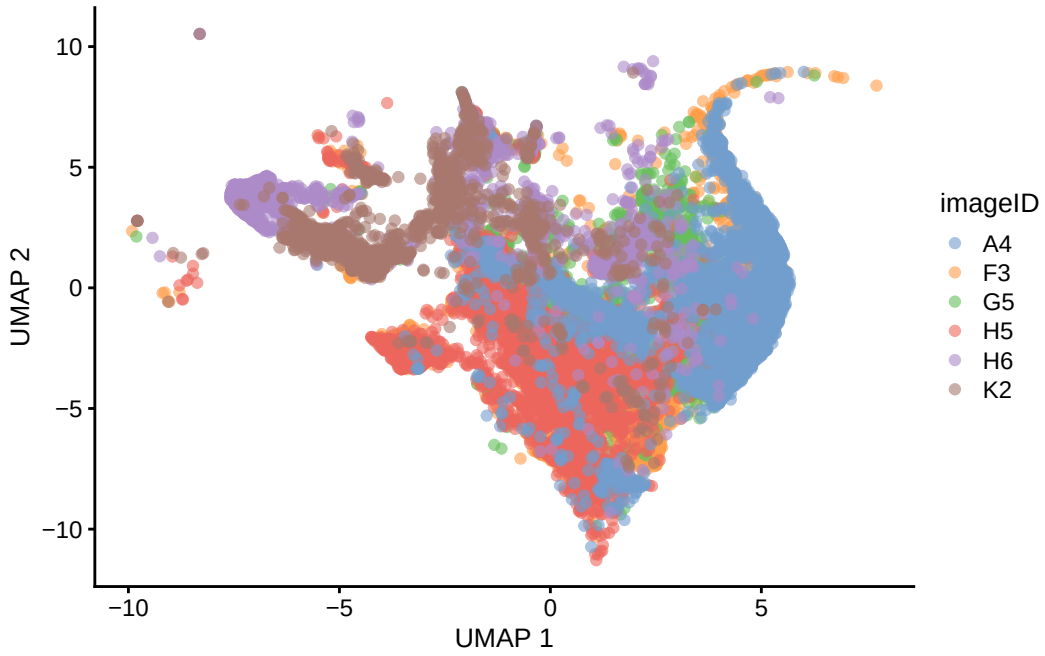
```
# Plot densities of CD3 for each image.
cells |>
  join_features(features = rownames(cells), shape = "wide", assay = "counts") |>
  ggplot(aes(x = CD3, colour = imageID)) +
  geom_density() +
  theme(legend.position = "none")
```



### 8.8.1 Dimension reduction and visualisation

As our data is stored in a `SpatialExperiment` we can also use `scater` to perform and visualise our data in a lower dimension to look for batch effects in our images. We can see that before normalisation, our UMAP shows a clear batch effect between images.

```
# Usually we specify a subset of the original markers which are informative to separating ou
ct_markers <- c("podoplanin", "CD13", "CD31",
                "panCK", "CD3", "CD4", "CD8a",
                "CD20", "CD68", "CD16", "CD14", "HLADR", "CD66a")

# ct_markers <- c("podoplanin", "CD13", "CD31",
#                 "panCK", "CD3", "CD4", "CD8a",
#                 "CD20", "CD68", "CD14", "CD16",
#                 "CD66a")

set.seed(51773)
# Perform dimension reduction using UMAP.
cells <- scater::runUMAP(
  cells,
  subset_row = ct_markers,
  exprs_values = "counts"
)

# Select a subset of images to plot.
someImages <- unique(cells$imageID)[c(1, 5, 10, 20, 30, 40)]

# UMAP by imageID.
scater::plotReducedDim(
  cells[, cells$imageID %in% someImages],
  dimred = "UMAP",
  colour_by = "imageID"
)
```

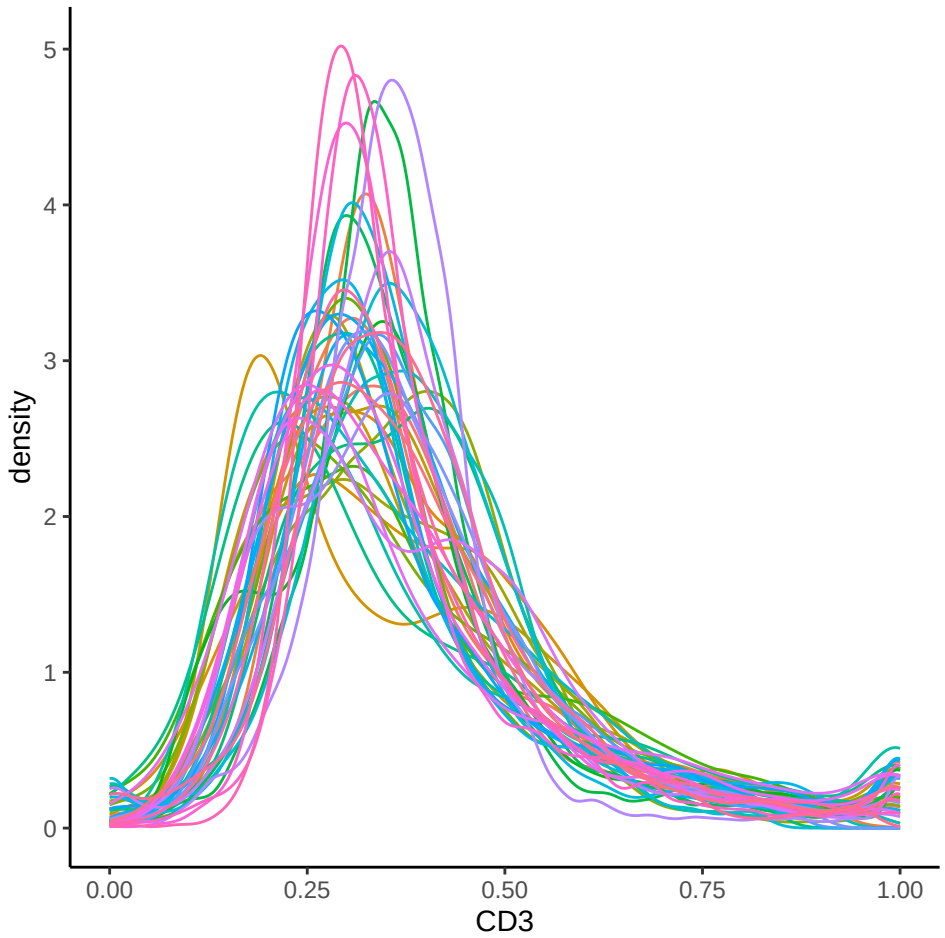



We can transform and normalise our data using the `normalizeCells` function. In the `normalizeCells()` function, we specify the following parameters. `transformation` is an optional argument which specifies the function to be applied to the data. We do not apply an arcsinh transformation here, as we already apply a square root transform in the `simpleSeg()` function. `method = c("trim99", "mean", PC1")` is an optional argument which specifies the normalisation method/s to be performed. Here, we: 1) Trim the 99th percentile 2) Divide by the mean 3) Remove the 1st principal component `assayIn = "counts"` is a required argument which specifies what the assay you'll be taking the intensity data from is named. In our context, this is called `counts`.

This modified data is then stored in the `norm` assay by default. We can see that this normalised data appears more bimodal, not perfectly, but likely to a sufficient degree for clustering, as we can at least observe a clear CD3+ peak at 1.00, and a CD3- peak at around 0.3.

```
# Leave out the nuclei markers from our normalisation process.
useMarkers <- rownames(cells)[!rownames(cells) %in% c("DNA1", "DNA2", "HH3")]

# Transform and normalise the marker expression of each cell type.
cells <- normalizeCells(cells,
                        markers = useMarkers,
                        transformation = NULL,
                        method = c("trim99", "mean", "PC1"),
                        assayIn = "counts",
                        cores = nCores
```

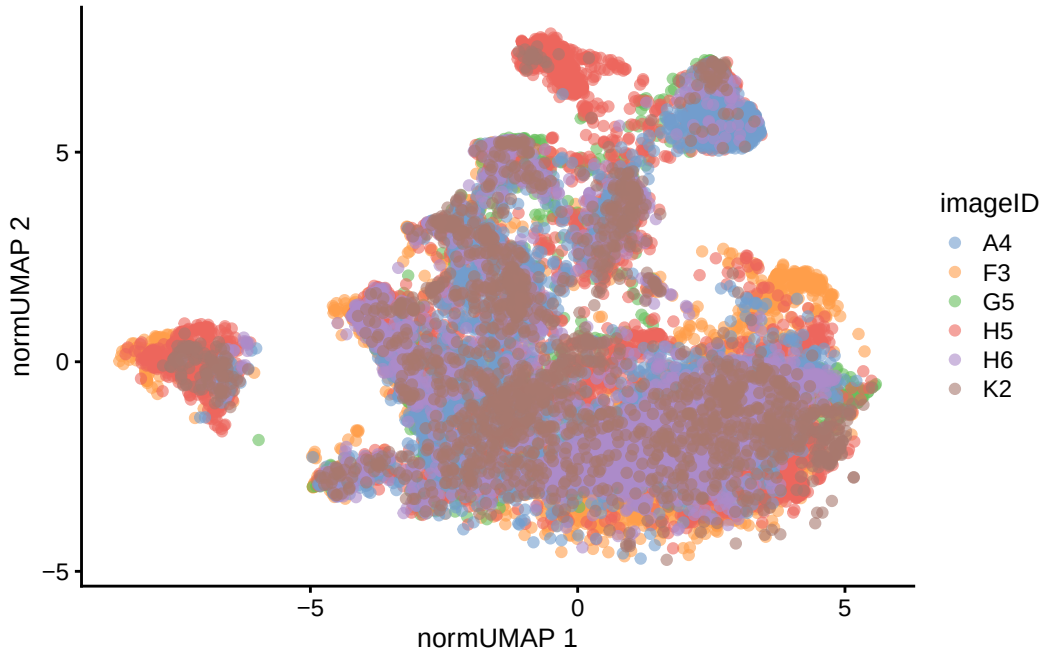
```
)

# Plot densities of CD3 for each image
cells |>
  join_features(features = rownames(cells), shape = "wide", assay = "norm") |>
  ggplot(aes(x = CD3, colour = imageID)) +
  geom_density() +
  theme(legend.position = "none")
```




We can also appreciate through the UMAP a reduction of the batch effect we initially saw.

```
set.seed(51773)
# Perform dimension reduction using UMAP.
cells <- scater::runUMAP(
  cells,
```

```
  subset_row = ct_markers,
  exprs_values = "norm",
  name = "normUMAP"
)

someImages <- unique(cells$imageID)[c(1, 5, 10, 20, 30, 40)]

# UMAP by imageID.
scater::plotReducedDim(
  cells[, cells$imageID %in% someImages],
  dimred = "normUMAP",
  colour_by = "imageID"
)
```




## 8.9 FuseSOM: Cluster cells into cell types

We can appreciate from the UMAP that there is a division of clusters, most likely representing different cell types. We next aim to empirically distinguish each cluster using our FuseSOM package for clustering.

Our FuseSOM R package can be found on bioconductor at https://www.bioconductor.org/packages/release/bioc/html/FuseSOM.html, and provides a pipeline for the clustering of

highly multiplexed in situ imaging cytometry assays. This pipeline uses the Self Organising Map architecture coupled with Multiview hierarchical clustering and provides functions for the estimation of the number of clusters.

Here we cluster using the `runFuseSOM` function. We specify the number of clusters to identify to be `numClusters = 10`. We also specify a set of cell-type specific markers to use, as we want our clusters to be distinct based off cell type markers, rather than markers which might pick up a transitioning cell state.

### 8.9.1 Perform the clustering

```
# Set seed.
set.seed(51773)

# Generate SOM and cluster cells into 10 groups
cells <- runFuseSOM(
  cells,
  markers = ct_markers,
  assay = "norm",
  numClusters = 10
)
```

```
You have provided a dataset of class SpatialExperiment

Everything looks good. Now running the FuseSOM algorithm

Now Generating the Self Organizing Map Grid

Optimal Grid Size is: 8

Now Running the Self Organizing Map Model

Now Clustering the Prototypes

Loading required namespace: fastcluster

Now Mapping Clusters to the Original Data

The Prototypes have been Clustered and Mapped Successfully
```

```
The FuseSOM algorithm has completed successfully
```

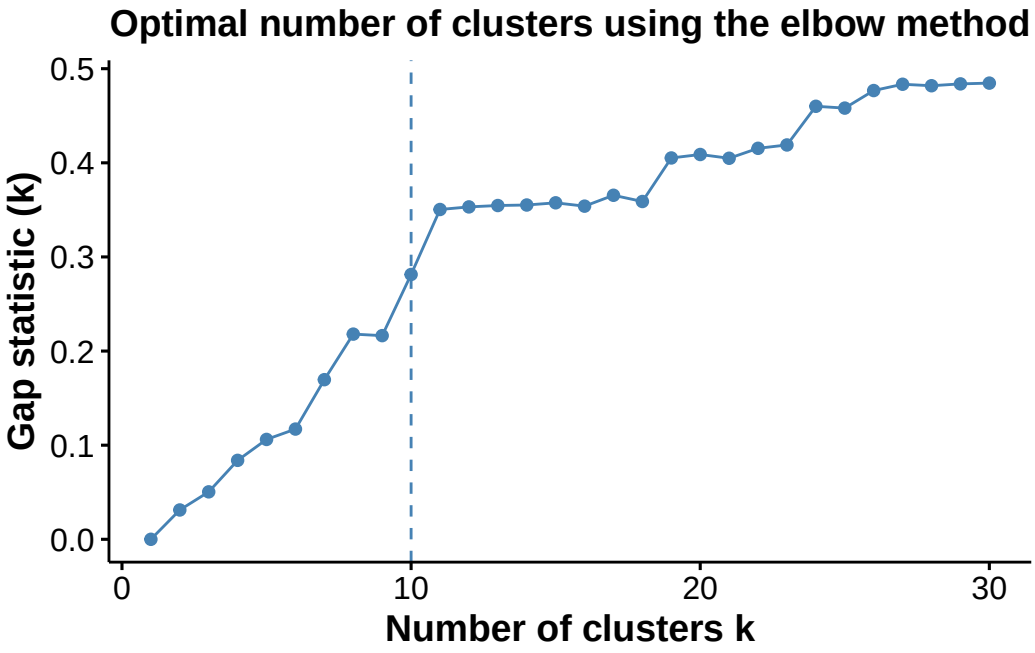
We can also observe how reasonable our choice of `k = 10` was, using the `estimateNumCluster()` and `optiPlot()` functions. Here we examine the Gap method, but others such as Silhouette and Within Cluster Distance are also available. We can see that there are elbow points in the gap statistic at `k = 7`, `k = 10`, and `k = 11`. We've specified `k = 10`, striking a good balance between the number of clusters and the gap statistic.

```
cells <- estimateNumCluster(cells, kSeq = 2:30)
```

```
You have provided a dataset of class: SpatialExperiment

Now Computing the Number of Clusters using Discriminant Analysis

Now Computing The Number Of Clusters Using Distance Analysis
```

```
optiPlot(cells, method = "gap")
```

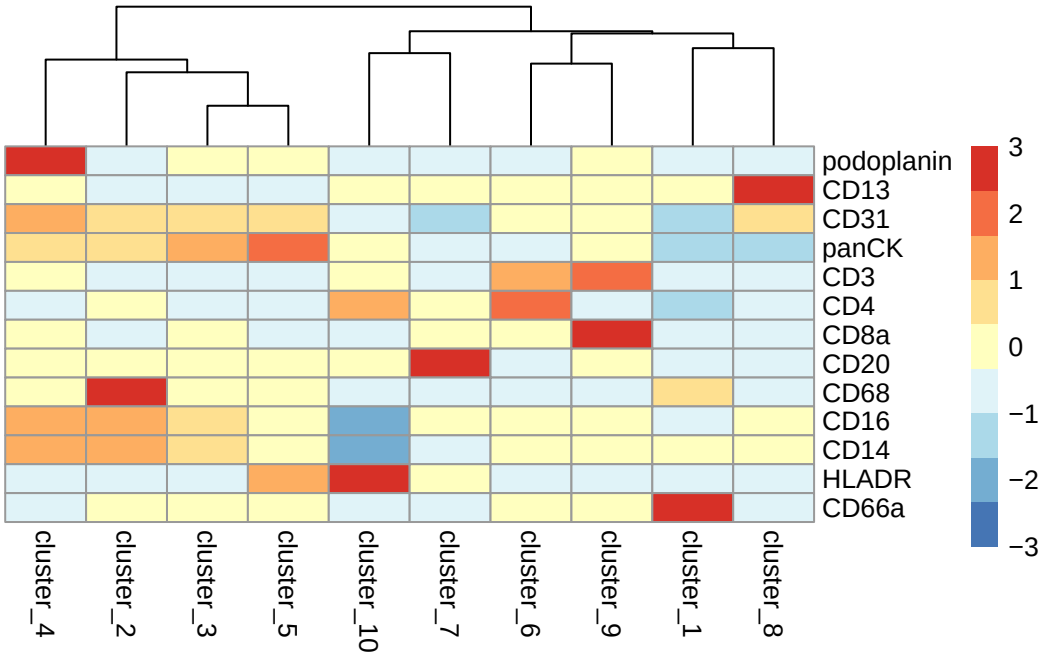
```
You have provided a dataset of class: SpatialExperiment
```



### 8.9.2 Attempt to interpret the phenotype of each cluster

We can begin the process of understanding what each of these cell clusters are by using the `plotGroupedHeatmap` function from `scater`. At the least, here we can see we capture all the major immune populations that we expect to see, including the CD4 and CD8 T cells, the CD20+ B cells, the CD68+ myeloid populations, the CD66+ granulocytes, the podoplanin+ epithelial cells, and the panCK+ tumour cells.

```
# Visualise marker expression in each cluster.
scater::plotGroupedHeatmap(
  cells,
  features = ct_markers,
  group = "clusters",
  exprs_values = "norm",
  center = TRUE,
  scale = TRUE,
  zlim = c(-3, 3),
  cluster_rows = FALSE,
  block = "clusters"
)
```




Given domain-specific knowledge of the tumour-immune landscape, we can go ahead and annotate these clusters as cell types given their expression profiles.

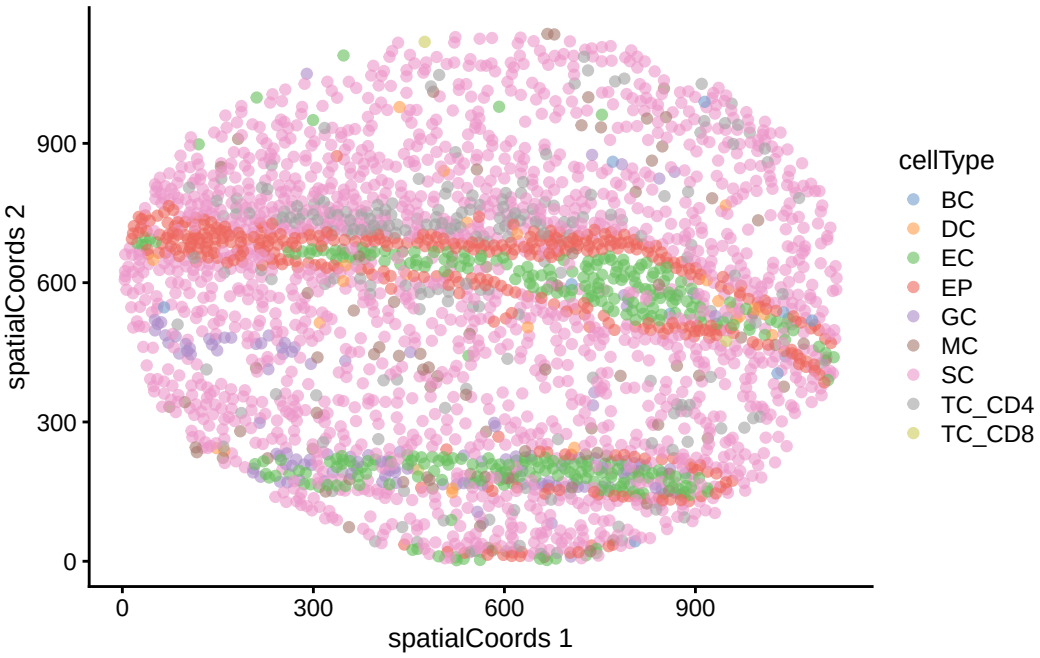
```
cells <- cells |>
  mutate(cellType = case_when(
    clusters == "cluster_1" ~ "GC", # Granulocytes
    clusters == "cluster_2" ~ "MC", # Myeloid cells
    clusters == "cluster_3" ~ "SC", # Squamous cells
    clusters == "cluster_4" ~ "EP", # Epithelial cells
    clusters == "cluster_5" ~ "SC", # Squamous cells
    clusters == "cluster_6" ~ "TC_CD4", # CD4 T cells
    clusters == "cluster_7" ~ "BC", # B cells
    clusters == "cluster_8" ~ "EC", # Endothelial cells
    clusters == "cluster_9" ~ "TC_CD8", # CD8 T cells
    clusters == "cluster_10" ~ "DC" # Dendritic cells
  ))
```

```
New names:
New names:
* `UMAP1` -> `UMAP1...1`
* `UMAP2` -> `UMAP2...2`
* `UMAP1` -> `UMAP1...3`
* `UMAP2` -> `UMAP2...4`
```

We might also be interested in how these cell types are distributed on the images themselves. Here we examine the distribution of clusters on image F3, noting the healthy epithelial and endothelial structures surrounded by tumour cells.

```
reducedDim(cells, "spatialCoords") <- spatialCoords(cells)

cells |>
  filter(imageID == "F3") |>
  plotReducedDim("spatialCoords", colour_by = "cellType")
```

```
New names:
* `UMAP1` -> `UMAP1...1`
* `UMAP2` -> `UMAP2...2`
* `UMAP1` -> `UMAP1...3`
* `UMAP2` -> `UMAP2...4`
```




### 8.9.3 Check cell type frequencies

We find it always useful to check the number of cells in each cluster. Here we can see that cluster 10 contains lots of (most likely tumour - high expression of panCK and non-consistent expression of other markers) cells and cluster 4 contains very few cells.

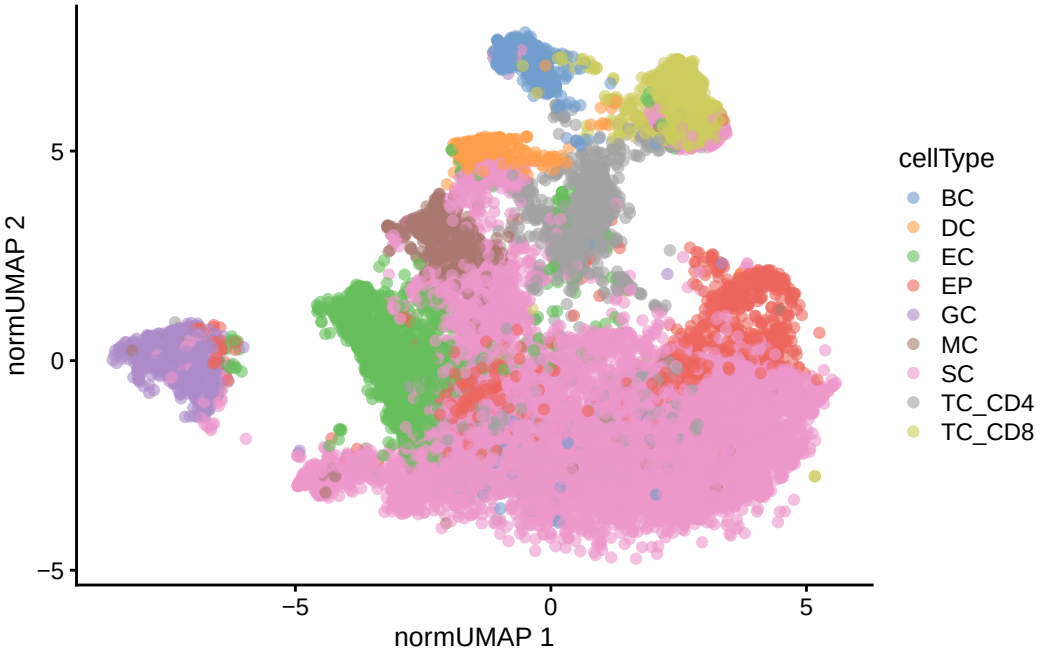
```
# Check cell type frequencies.
cells$cellType |>
  table() |>
  sort()
```

```
   DC     BC     MC     GC TC_CD8 TC_CD4     EC     EP     SC
 2411   4322   5283   7993   8534  11753  14159  14170  87288
```

We can also use the UMAP we computed earlier to visualise our data in a lower dimension and see how well our annotated cell types cluster out.

```
# UMAP by cell type
scater::plotReducedDim(
  cells[, cells$imageID %in% someImages],
  dimred = "normUMAP",
```

```
  colour_by = "cellType"
)
```




### 8.9.4 Testing for association between the proportion of each cell type and progressor status

We recommend using a package such as `diffcyt` for testing for changes in abundance of cell types. However, the `colTest` function allows us to quickly test for associations between the proportions of the cell types and progression status using either Wilcoxon rank sum tests or t-tests. Here we see a p-value less than 0.05, but this does not equate to a small FDR.

```
# Perform simple student's t-tests on the columns of the proportion matrix.
testProp <- colTest(cells,
                    condition = "group",
                    feature = "cellType",
                    type = "ttest")

head(testProp)
```

```
       mean in group NP mean in group P tval.t  pval adjPval cluster
TC_CD8            0.056           0.033   2.70 0.011   0.099  TC_CD8
MC                0.033           0.040  -1.70 0.099   0.400      MC
```

```
EC                0.098           0.085   1.50 0.160   0.400      EC
BC                0.026           0.016   1.30 0.190   0.400      BC
SC                0.560           0.590  -1.30 0.220   0.400      SC
TC_CD4            0.074           0.084  -0.94 0.360   0.540  TC_CD4
```

Let's examine one of these clusters using our `getProp()` function from `spicyR`, which conveniently transforms our proportions into a feature matrix of images by cell type, enabling convenient downstream classification or analysis.

Next, let's visualise how different the proportions are

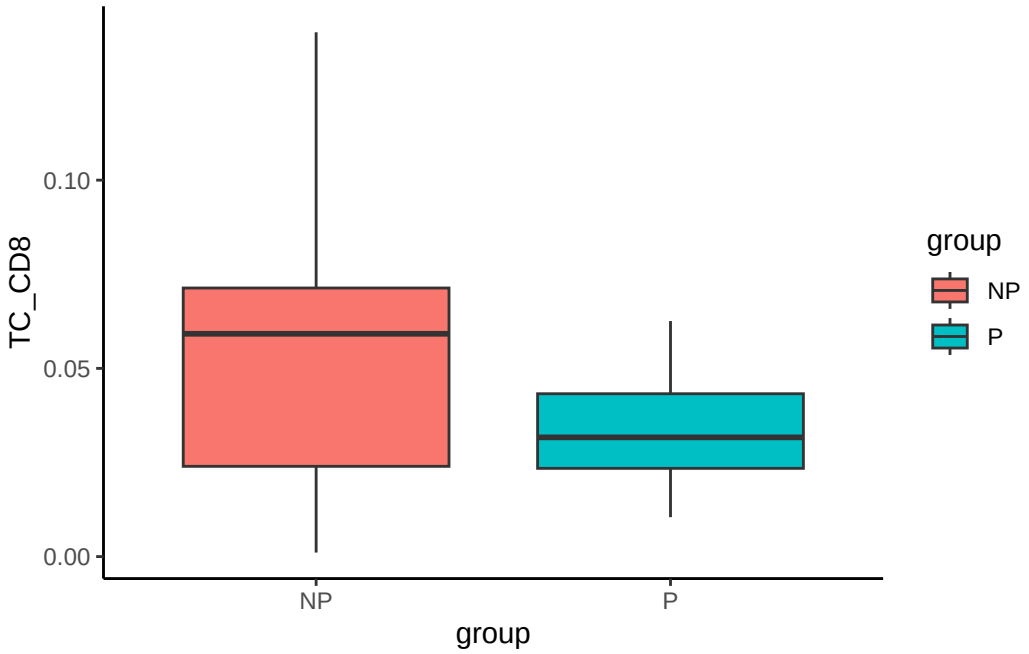
boxplot.

```
prop <- getProp(cells, feature = "cellType")
prop[1:5, 1:5]
```

```
           BC         DC         EC         EP          GC
A2 0.01501502 0.00000000 0.09129129 0.04684685 0.031831832
A3 0.01854193 0.00000000 0.09334176 0.01769912 0.001474926
A4 0.01196581 0.00322887 0.07901235 0.02792023 0.003988604
A5 0.02135513 0.03190087 0.07882942 0.15344055 0.140522014
A6 0.01783492 0.01824969 0.08710079 0.14765657 0.125259229
```

It appears that the CD8 T cells are the most differentially abundant cell type across our progressors and non-progressors. A boxplot visualisation of CD8 T cell proportion clearly shows that progressors have a lower proportion of CD8 T cells in the tumour core.

```
clusterToUse <- rownames(testProp)[1]

prop |>
  select(all_of(clusterToUse)) |>
  tibble::rownames_to_column("imageID") |>
  left_join(clinical, by = "imageID") |>
  ggplot(aes(x = group, y = .data[[clusterToUse]], fill = group)) +
  geom_boxplot()
```




**NB**: If you have already clustered and annotated your cells, you may only be interested in our downstream analysis capabilities, looking into identifying localisation (spicyR), cell regions (lisaClust), and cell-cell interactions (SpatioMark & Kontextual). Therefore, for the sake of convenience, we've provided capability to directly load in the SpatialExperiment (SPE) object that we've generated up to this point, complete with clusters and normalised intensities.

```
load(system.file("extdata/computed_cells.rda", package = "spicyWorkflow"))
```

## 8.10 spicyR: Test spatial relationships

Our spicyR package is available on bioconductor on https://www.bioconductor.org/packages/devel/bioc/html/spicyR.html and provides a series of functions to aid in the analysis of both immunofluorescence and imaging mass cytometry data as well as other assays that can deeply phenotype individual cells and their spatial location. Here we use the `spicy()` function to test for changes in the spatial relationships between pair-wise combinations of cells.

Put simply, spicyR uses the L-function to determine localisation or dispersion between cell types. The L-function is an arbitrary measure of "closeness" between points, with greater values suggesting increased localisation, and lower values suggesting dispersion.

Here, we quantify spatial relationships using a combination of 10 radii from 10 to 100 by specifying `Rs = 1:10*10` and mildly account for some global tissue structure using `sigma = 50`. Further information on how to optimise these parameters can be found in the vignette and the spicyR paper.

```
spicyTest <- spicy(cells,
                   condition = "group",
                   cellTypeCol = "cellType",
                   imageIDCol = "imageID",
                   Rs = 1:10*10,
                   sigma = 50,
                   BPPARAM = BPPARAM)
```

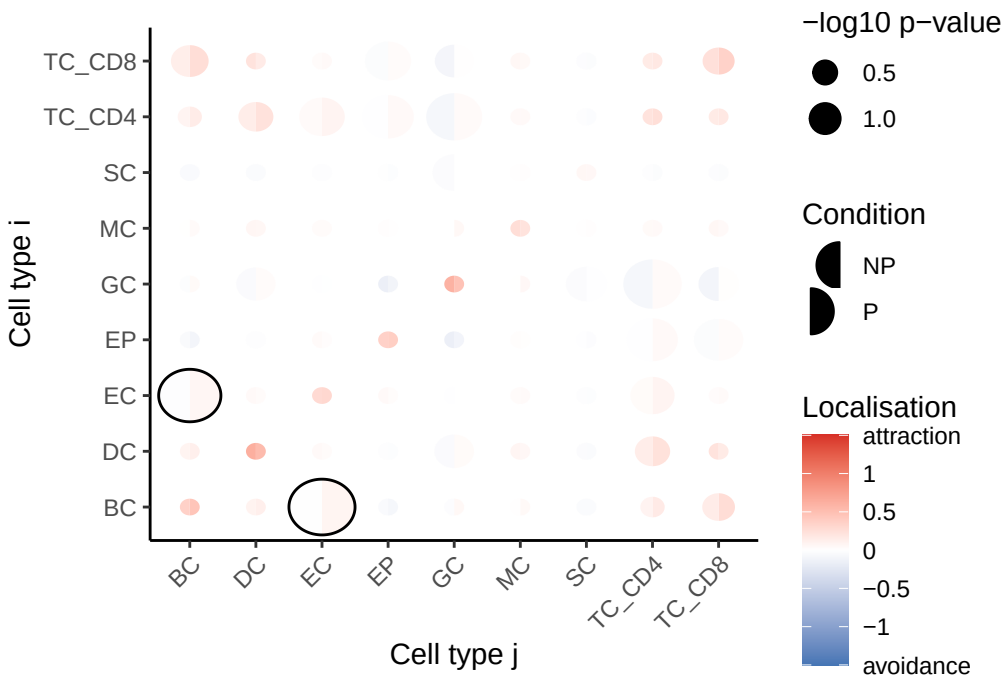
```
Coercing condition into factor. Using group = NP as base comparison group. If
this is not the desired base group, please convert cells$group into a factor
and change the order of levels(cells$group) so that the base group is at index
1.
```

```
topPairs(spicyTest, n = 10)
```

```
            intercept coefficient    p.value adj.pvalue   from     to
BC__EC     -1.1346644    8.304020 0.03391529  0.6931721     BC     EC
EC__BC     -1.5909700    7.919971 0.04134661  0.6931721     EC     BC
GC__TC_CD4 -7.6316673   11.005639 0.05047096  0.6931721     GC TC_CD4
TC_CD4__GC -7.5406782   11.077602 0.05709461  0.6931721 TC_CD4     GC
TC_CD4__EP -0.7117393    4.853317 0.07261120  0.6931721 TC_CD4     EP
EP__TC_CD4 -0.7070918    4.820661 0.07510118  0.6931721     EP TC_CD4
EP__TC_CD8 -3.1445464    5.779536 0.08047994  0.6931721     EP TC_CD8
TC_CD8__EP -2.9313474    5.688420 0.09522058  0.6931721 TC_CD8     EP
TC_CD4__EC  3.1984391    4.366122 0.09716499  0.6931721 TC_CD4     EC
EC__TC_CD4  3.1052061    4.263971 0.10461424  0.6931721     EC TC_CD4
```

We can visualise these tests using `signifPlot` where we observe that cell type pairs appear to become less attractive (or avoid more) in the progression sample.

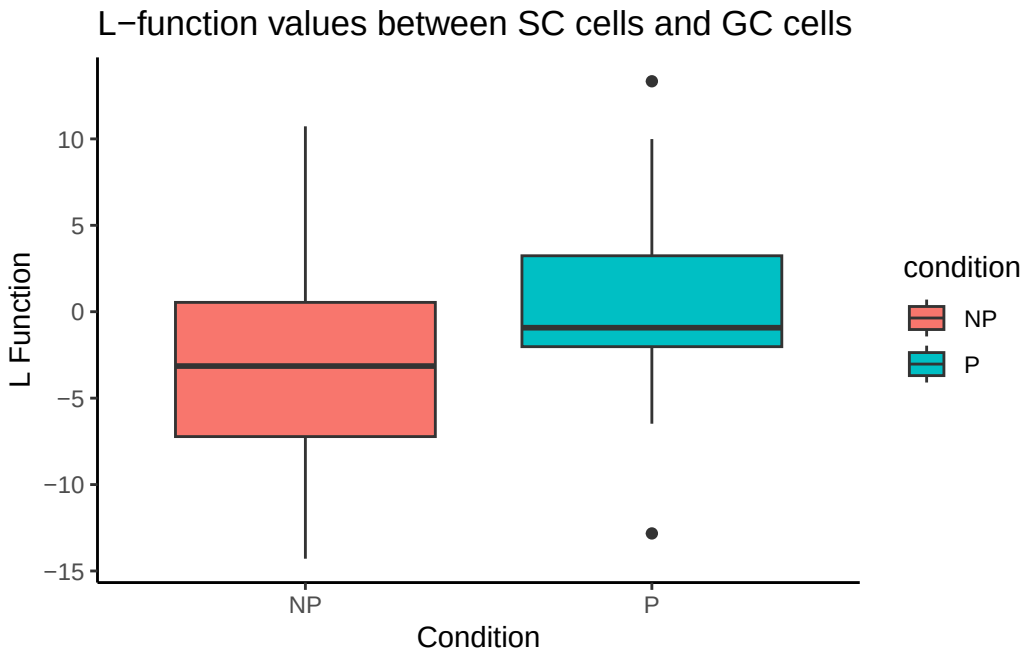
```
# Visualise which relationships are changing the most.
signifPlot(
  spicyTest,
  breaks = c(-1.5, 1.5, 0.5)
)
```




`spicyR` also has functionality for plotting out individual pairwise relationships. We can first try look into whether the `SC` tumour cell type localises with the `GC` granular cell type, and whether this localisation affects progression vs non-progression of the tumour.

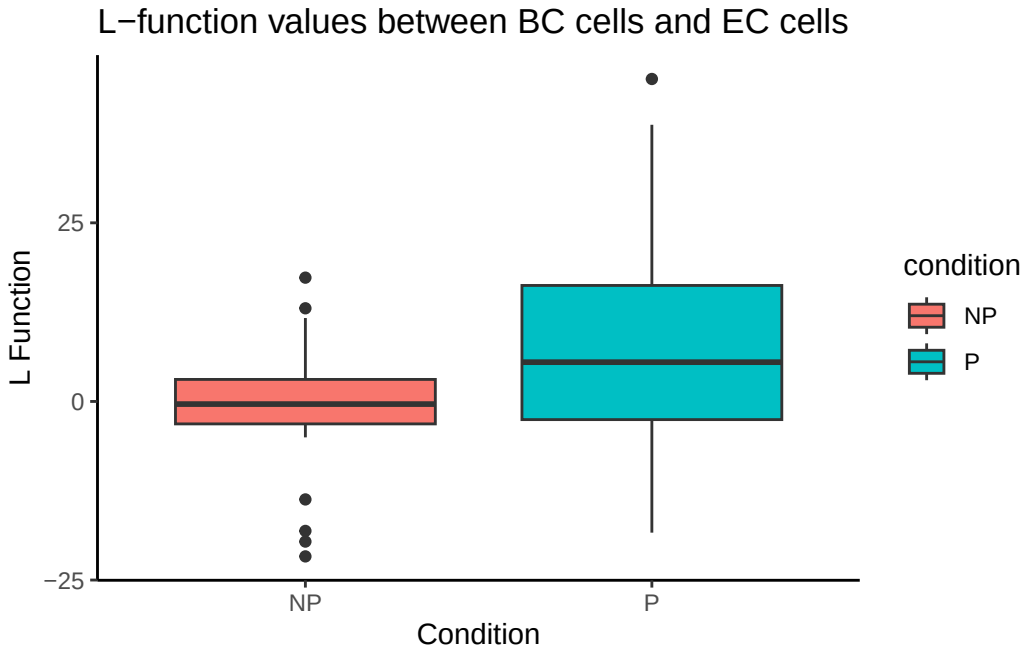
```
spicyBoxPlot(spicyTest,
             from = "SC",
             to = "GC")
```

```
Warning: Removed 1 row containing non-finite outside the scale range
(`stat_boxplot()`).
```




Alternatively, we can look at the most differentially localised relationship between progressors and non-progressors by specifying `rank = 1`.

```
spicyBoxPlot(spicyTest,
             rank = 1)
```



## 8.11 lisaClust: Find cellular neighbourhoods

Our lisaClust package (https://www.bioconductor.org/packages/devel/bioc/html/lisaClust.html)[https://www. provides a series of functions to identify and visualise regions of tissue where spatial associations between cell-types is similar. This package can be used to provide a high-level summary of cell-type co-localisation in multiplexed imaging data that has been segmented at a single-cell resolution. Here we use the `lisaClust` function to clusters cells into 5 regions with distinct spatial ordering.

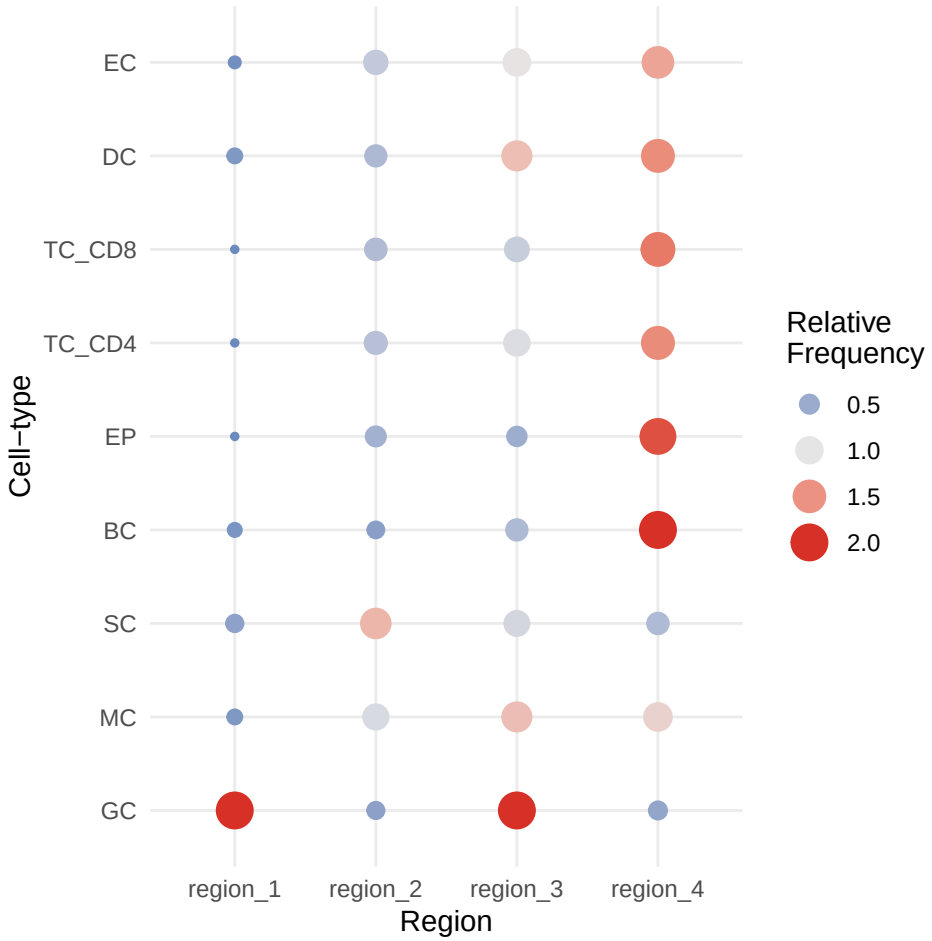
```
set.seed(51773)

# Cluster cells into spatial regions with similar composition.
cells <- lisaClust(
  cells,
  k = 4,
  sigma = 50,
  cellType = "cellType",
  BPPARAM = BPPARAM
)
```

```
Generating local L-curves.
```

### 8.11.1 Region - cell type enrichment heatmap

We can try to interpret which spatial orderings the regions are quantifying using the `regionMap` function. This plots the frequency of each cell type in a region relative to what you would expect by chance. We can see here that our regions have neatly separated according to biological milieu, with region 1 and 2 representing our immune cell regions, region 3 representing our tumour cells, and region 4 representing our healthy epithelial and endothelial cells.

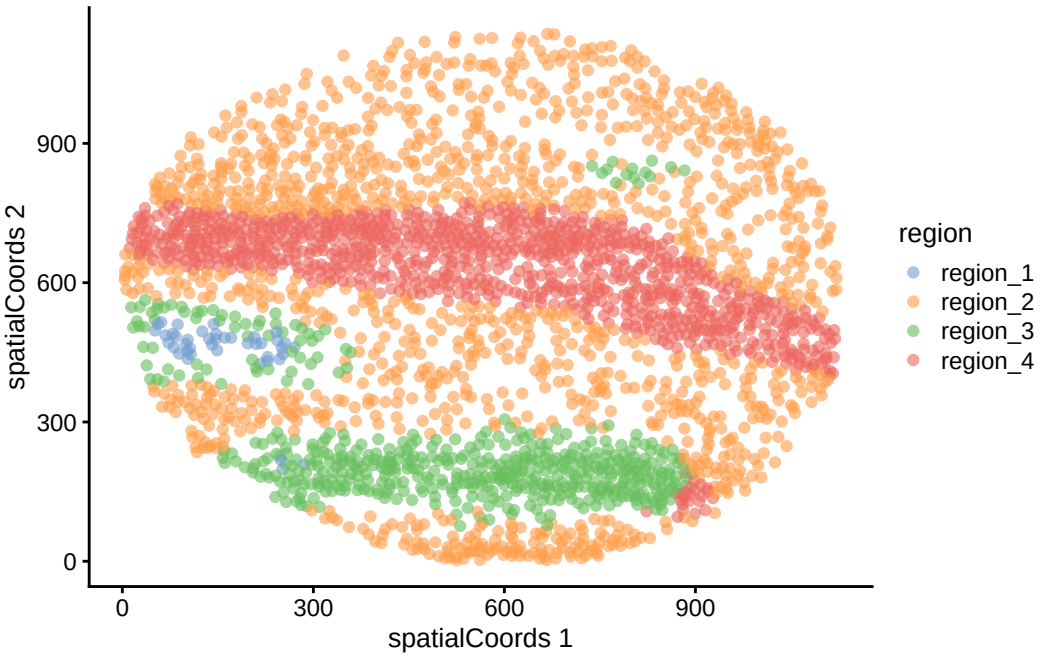
```
# Visualise the enrichment of each cell type in each region
regionMap(cells, cellType = "cellType", limit = c(0.2, 2))
```




### 8.11.2 Visualise regions

By default, these identified regions are stored in the `regions` column in the `colData` of our object. We can quickly examine the spatial arrangement of these regions using `ggplot` on image F3, where we can see the same division of immune, healthy, and tumour tissue that we identified in our `regionMap`.

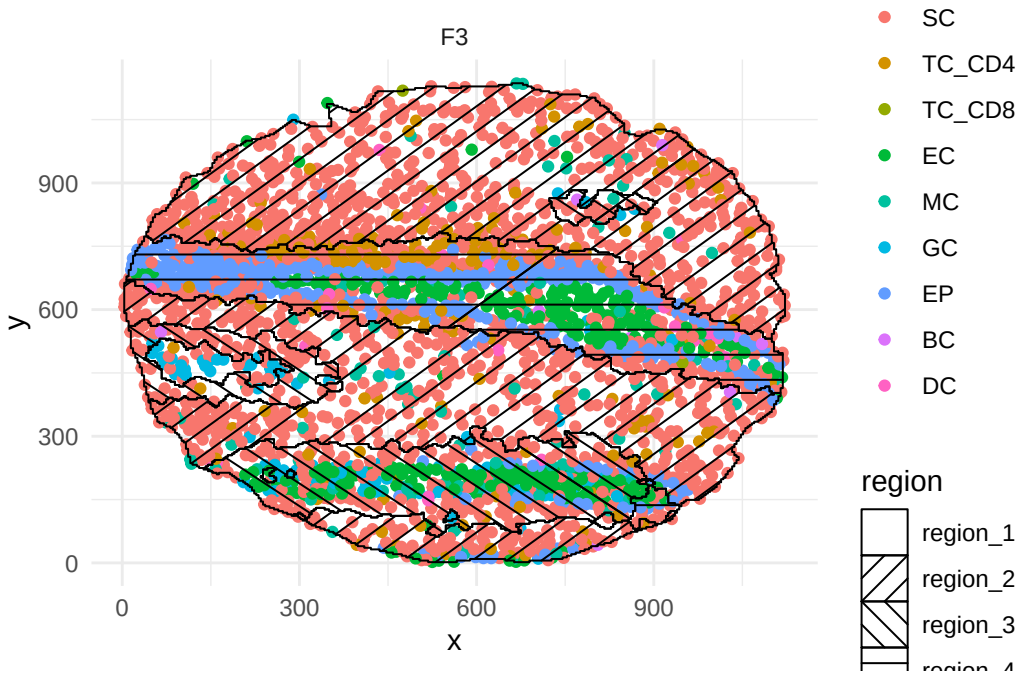
```
cells |>
  filter(imageID == "F3") |>
  plotReducedDim("spatialCoords", colour_by = "region")
```




While much slower, we have also implemented a function for overlaying the region information as a hatching pattern so that the information can be viewed simultaneously with the cell type calls.

```
# Use hatching to visualise regions and cell types.
hatchingPlot(
  cells,
  useImages = "F3",
  cellType = "cellType",
  nbp = 300
)
```

```
Concave windows are temperamental. Try choosing values of window.length > and < 1 if you have
```




### 8.11.3 Test for association with progression

Similar to cell type proportions, we can quickly use the `colTest` function to test for associations between the proportions of cells in each region and progression status by specifying `feature = "region"`.

```
# Test if the proportion of each region is associated
# with progression status.
testRegion <- colTest(
  cells,
  feature = "region",
  condition = "group",
  type = "ttest"
)

testRegion
```

```
         mean in group NP mean in group P tval.t  pval adjPval  cluster
region_1            0.022          0.0079   1.90 0.066    0.20 region_1
region_3            0.092          0.1300  -1.70 0.100    0.20 region_3
region_4            0.340          0.3300   0.63 0.530    0.71 region_4
region_2            0.540          0.5400   0.20 0.840    0.84 region_2
```

## 8.12 Statial: Identify changes in cell state.

Our Statial package (https://www.bioconductor.org/packages/release/bioc/html/Statial.html) provides a suite of functions (Kontextual) for robust quantification of cell type localisation which are invariant to changes in tissue structure. In addition, we provide a suite of functions (SpatioMark) for uncovering continuous changes in marker expression associated with varying levels of localisation.

### 8.12.1 SpatioMark: Continuous changes in marker expression associated with varying levels of localisation.

The first step in analysing these changes is to calculate the spatial proximity (`getDistances`) of each cell to every cell type. These values will then be stored in the `reducedDims` slot of the `SingleCellExperiment` object under the names `distances`. SpatioMark also provides functionality to look into proximal cell abundance using the `getAbundance()` function, which is further explored within the `Statial` package vignette.

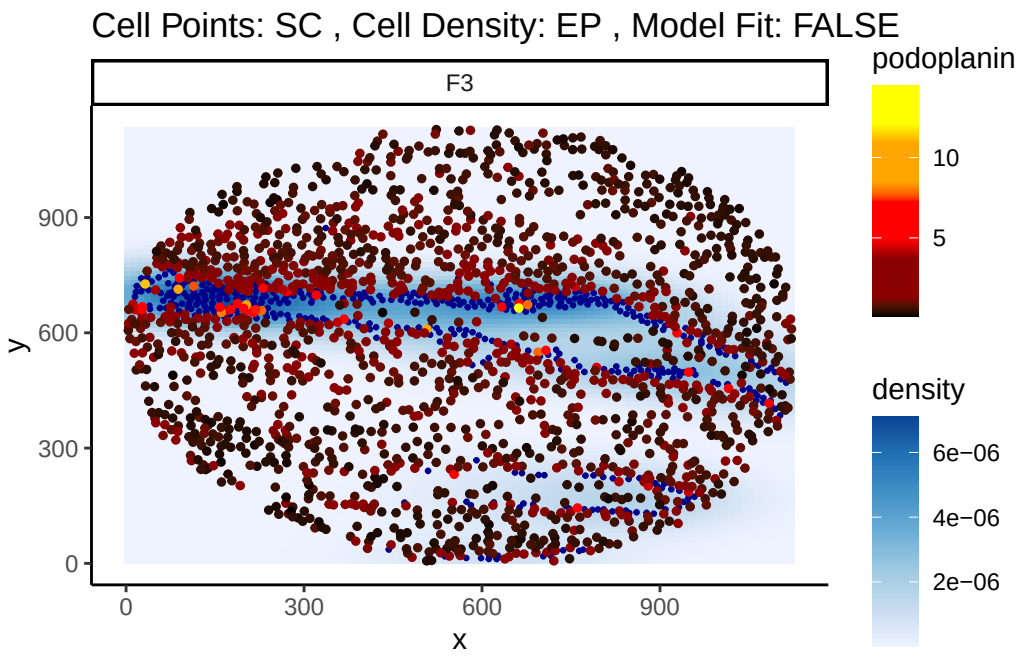
```
cells$m.cx <- spatialCoords(cells)[,"x"]
cells$m.cy <- spatialCoords(cells)[,"y"]

cells <- getDistances(cells,
  maxDist = 200,
  nCores = nCores,
  cellType = "cellType",
  spatialCoords = c("m.cx", "m.cy")
)
```

We can then visualise an example image, specified with `image = "F3"` and a particular marker interaction with cell type localisation. To visualise these changes, we specify two cell types with the `from` and `to` parameters, and a marker with the `marker` parameter (cell-cell-marker interactions). Here, we specify the changes in the marker podoplanin in `SC` tumour cells as its localisation to `EP` epithelial cells increases or decreases, where we can observe that podoplanin decreases in tumour cells as its distance to the central cluster of epithelial cells increases.

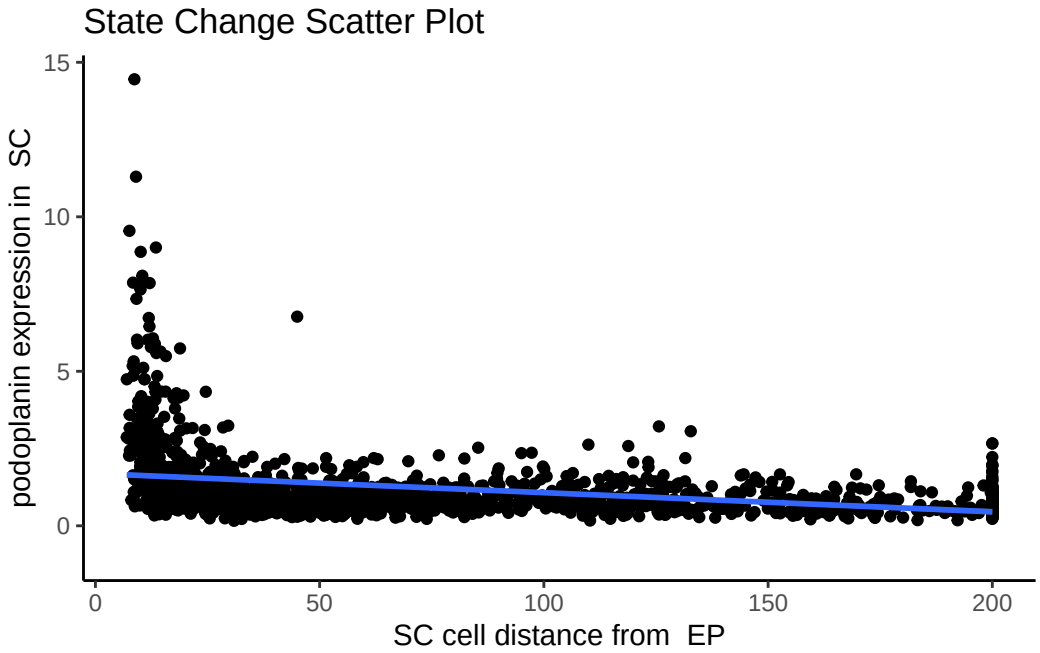
```
p <- plotStateChanges(
  cells = cells,
  cellType = "cellType",
  spatialCoords = c("m.cx", "m.cy"),
  type = "distances",
  image = "F3",
  from = "SC",
```

```
  to = "EP",
  marker = "podoplanin",
  size = 1,
  shape = 19,
  interactive = FALSE,
  plotModelFit = FALSE,
  method = "lm"
)

p
```

```
$image
```




```
$scatter

`geom_smooth()` using formula = 'y ~ x'
```




SpatioMark aims to holistically uncover all such significant relationships by looking at all interactions across all images. The `calcStateChanges` function provided by Statial can be expanded for this exact purpose - by not specifying cell types, a marker, or an image, `calcStateChanges` will examine the most significant correlations between distance and marker expression across the entire dataset.

```
state_dist <- calcStateChanges(
  cells = cells,
  cellType = "cellType",
  type = "distances",
  assay = 2,
  nCores = nCores,
  minCells = 100
)

head(state_dist[state_dist$imageID == "F3",], n = 10)
```

```
      imageID primaryCellType otherCellType     marker          coef      tval
51990      F3              SC            EP podoplanin -0.0006911103 -16.90036
23703      F3              EP            GC      CXCR3  0.0012103305  15.04988
23704      F3              EP            GC     pSTAT3  0.0012299816  14.79933
51978      F3              SC            GC       PDL2  0.0008802428  13.12601
23669      F3              EP            MC       CCR7  0.0011476888  13.32900
23595      F3              EP        TC_CD8      CXCR3  0.0016741880  12.95156
```

```
51981      F3              SC            GC       ICOS  0.0005186181  11.75830
23696      F3              EP            GC      CADM1  0.0011398625  12.18028
11426      F3              EC            GC       CD31  0.0010070252  12.40866
23650      F3              EP            EC       TIM3  0.0033539928  12.05028
              pval          fdr
51990 4.203881e-59 9.004712e-57
23703 8.041089e-41 8.191646e-39
23704 8.916789e-40 8.712976e-38
51978 1.839014e-37 1.637728e-35
23669 9.360248e-34 6.863845e-32
23595 3.025912e-32 2.085497e-30
51981 1.113579e-30 7.070268e-29
23696 3.239532e-29 1.903862e-27
11426 4.839241e-29 2.825645e-27
23650 1.030900e-28 5.930114e-27
```

The results from our SpatioMark outputs can be converted from a `data.frame` to a `matrix`, using the `prepMatrix()` function. Note, the choice of extracting either the t-statistic or the coefficient of the linear regression can be specified using the `column = "tval"` parameter, with the coefficient being the default extracted parameter. We can see that with SpatioMark, we get some features which are significant after adjusting for FDR.

```
# Preparing outcome vector
outcome <- cells$group[!duplicated(cells$imageID)]
names(outcome) <- cells$imageID[!duplicated(cells$imageID)]

# Preparing features for Statial
distMat <- prepMatrix(state_dist)

distMat <- distMat[names(outcome), ]

# Remove some very small values
distMat <- distMat[, colMeans(abs(distMat) > 0.0001) > .8]

survivalResults <- colTest(distMat, outcome, type = "ttest")

head(survivalResults)
```

```
                 mean in group NP mean in group P tval.t    pval adjPval
EC__EC__CD13             -0.00240        -9.6e-04   -3.6 0.00086    0.27
SC__SC__VISTA            -0.00470        -1.7e-03   -3.3 0.00190    0.27
SC__TC_CD4__CXCR3        -0.00015         8.8e-04   -3.4 0.00230    0.27
```

```
SC__EP__Ki67               -0.00037        -7.3e-05   -3.2 0.00260    0.27
EC__SC__CD68                0.00029        -3.9e-04    3.2 0.00300    0.27
SC__BC__DNA1                0.02600        -1.4e-01    3.0 0.00480    0.27
                             cluster
EC__EC__CD13           EC__EC__CD13
SC__SC__VISTA         SC__SC__VISTA
SC__TC_CD4__CXCR3 SC__TC_CD4__CXCR3
SC__EP__Ki67           SC__EP__Ki67
EC__SC__CD68           EC__SC__CD68
SC__BC__DNA1           SC__BC__DNA1
```

### 8.12.2 Kontextual: Robust quantification of cell type localisation which is invariant to changes in tissue structure

`Kontextual` is a method to evaluate the localisation relationship between two cell types in an image. `Kontextual` builds on the L-function by contextualising the relationship between two cell types in reference to the typical spatial behaviour of a $3^{rd}$ cell type/population. By taking this approach, `Kontextual` is invariant to changes in the window of the image as well as tissue structures which may be present.

The definitions of cell types and cell states are somewhat ambiguous, cell types imply well defined groups of cells that serve different roles from one another, on the other hand cell states imply that cells are a dynamic entity which cannot be discretised, and thus exist in a continuum. For the purposes of using `Kontextual` we treat cell states as identified clusters of cells, where larger clusters represent a "parent" cell population, and finer sub-clusters representing a "child" cell population. For example a CD4 T cell may be considered a child to a larger parent population of Immune cells. `Kontextual` thus aims to see how a child population of cells deviate from the spatial behaviour of their parent population, and how that influences the localisation between the child cell state and another cell state.

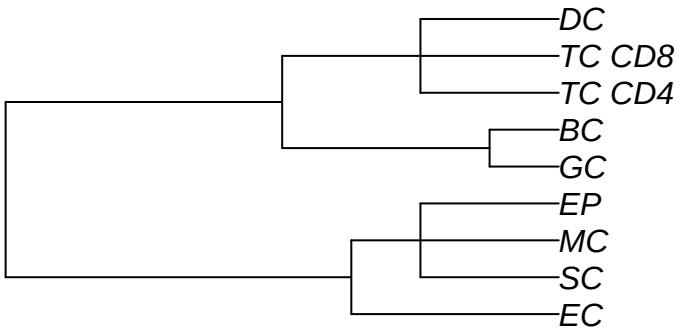
#### 8.12.2.1 Cell type hierarchy

A key input for Kontextual is an annotation of cell type hierarchies. We will need these to organise all the cells present into cell state populations or clusters, e.g. all the different B cell types are put in a vector called bcells.

Here, we use the `treeKor` bioconductor package treekoR to define these hierarchies in a data driven way.

```
fergusonTree <- treekoR::getClusterTree(t(assay(cells, "norm")),
                                        cells$cellType,
                                        hierarchy_method="hopach")
```

```
parent1 <- c("TC_CD8", "TC_CD4", "DC")
parent2 <- c("BC", "GC")
parent3 <- c(parent1, parent2)

parent4 <- c("MC", "EP", "SC")
parent5 <- c(parent4, "EC")

all = c(parent1, parent2, parent3, parent4, parent5)

treeDf = Statial::parentCombinations(all, parent1, parent2, parent3, parent4, parent5)

fergusonTree$clust_tree |> plot()
```




`Kontextual` accepts a `SingleCellExperiment` object, a single image, or list of images from a `SingleCellExperiment` object, which gets passed into the `cells` argument. Here, we've specified Kontextual to perform calculations on all pairwise combinations for every cluster using the `parentCombinations()` function to create the `treeDf` dataframe which we've specified in the `parentDf` parameter. The argument `r` will specify the radius which the cell relationship will be evaluated on. `Kontextual` supports parallel processing, the number of cores can be specified using the `cores` argument. `Kontextual` can take a single value or multiple values for each argument and will test all combinations of the arguments specified.

We can calculate all pairwise relationships across all images for a single radius.

```
kontext <- Kontextual(
  cells = cells,
  cellType = "cellType",
  spatialCoords = c("m.cx", "m.cy"),
  parentDf = treeDf,
  r = 50,
  cores = nCores
)
```

Again, we can use the same `colTest()` to quickly test for associations between the Kontextual values and progression status using either Wilcoxon rank sum tests or t-tests. Similar to SpatioMark, we can specify using either the original L-function by specifying `column = "original"` in our `prepMatrix()` function.

```
# Converting Kontextual result into data matrix
kontextMat <- prepMatrix(kontext)

# Replace NAs with 0
kontextMat[is.na(kontextMat)] <- 0

survivalResults <- spicyR::colTest(kontextMat, outcome, type = "ttest")

head(survivalResults)
```

```
                    mean in group NP mean in group P tval.t    pval adjPval
SC__BC__parent2                  4.9           -3.80    3.9 0.00039   0.053
SC__GC__parent3                 -3.7            2.60   -3.4 0.00140   0.095
SC__TC_CD4__parent3              2.9            0.24    2.7 0.00920   0.420
TC_CD4__EC__parent5              4.0           10.00   -2.5 0.01700   0.520
BC__TC_CD8__parent3             -5.5            3.10   -2.2 0.03100   0.520
SC__BC__parent3                  1.1           -3.80    2.2 0.03300   0.520
                               cluster
SC__BC__parent2          SC__BC__parent2
SC__GC__parent3          SC__GC__parent3
SC__TC_CD4__parent3 SC__TC_CD4__parent3
TC_CD4__EC__parent5 TC_CD4__EC__parent5
BC__TC_CD8__parent3 BC__TC_CD8__parent3
SC__BC__parent3          SC__BC__parent3
```

## 8.13 ClassifyR: Classification

Our ClassifyR package, https://github.com/SydneyBioX/ClassifyR, formalises a convenient framework for evaluating classification in R. We provide functionality to easily include four key modelling stages; Data transformation, feature selection, classifier training and prediction; into a cross-validation loop. Here we use the `crossValidate` function to perform 100 repeats of 5-fold cross-validation to evaluate the performance of a random forest applied to five quantifications of our IMC data; 1) Cell type proportions 2) Cell type localisation from `spicyR` 3) Region proportions from `lisaClust` 4) Cell type localisation in reference to a parent cell type from `Kontextual` 5) Cell changes in response to proximal changes from `SpatioMark`

```r
# Create list to store data.frames
data <- list()

# Add proportions of each cell type in each image
data[["Proportions"]] <- getProp(cells, "cellType")

# Add pair-wise associations
spicyMat <- bind(spicyTest)
spicyMat[is.na(spicyMat)] <- 0
spicyMat <- spicyMat |>
  select(!condition) |>
  tibble::column_to_rownames("imageID")

data[["SpicyR"]] <- spicyMat

# Add proportions of each region in each image
# to the list of dataframes.
data[["LisaClust"]] <- getProp(cells, "region")


# Add SpatioMark features
data[["SpatioMark"]] <- distMat

# Add Kontextual features
data[["Kontextual"]] <- kontextMat
```

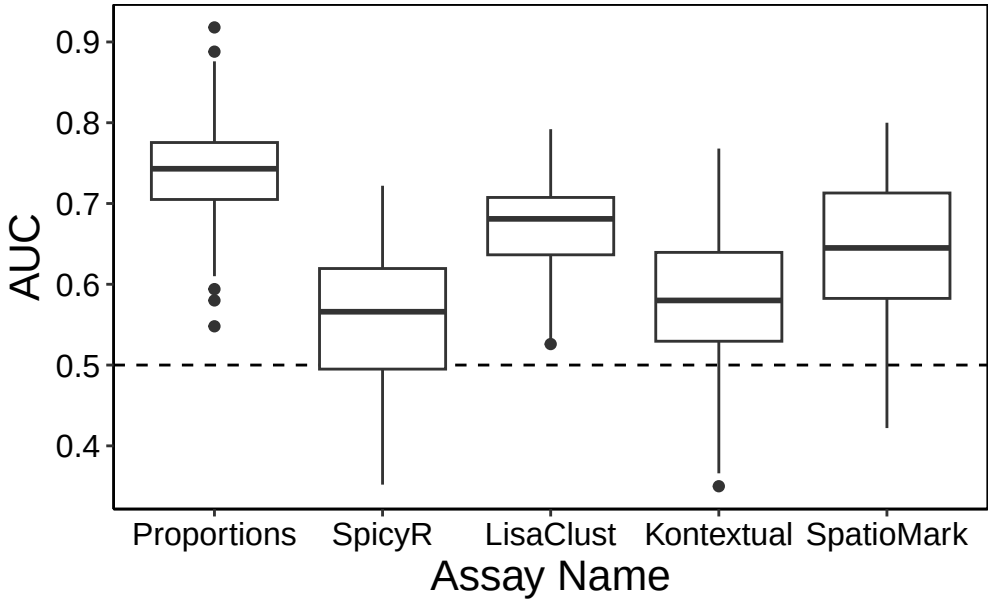
```r
# Set seed
set.seed(51773)

# Perform cross-validation of a random forest model
# with 100 repeats of 5-fold cross-validation.
cv <- crossValidate(
  measurements = data,
  outcome = outcome,
  classifier = "randomForest",
  nFolds = 5,
  nRepeats = 50,
  nCores = nCores
)
```

### 8.13.1 Visualise cross-validated prediction performance

Here we use the `performancePlot` function to assess the AUC from each repeat of the 5-fold cross-validation. We see that the lisaClust regions appear to capture information which is predictive of progression status of the patients.

```
# Calculate AUC for each cross-validation repeat and plot.
performancePlot(
  cv,
  metric = "AUC",
  characteristicsList = list(x = "Assay Name"),
  orderingList = list("Assay Name" = c("Proportions", "SpicyR", "LisaClust", "Kontextual", "S
)
```

```
Warning in .local(results, ...): AUC not found in all elements of results.
Calculating it now.
```

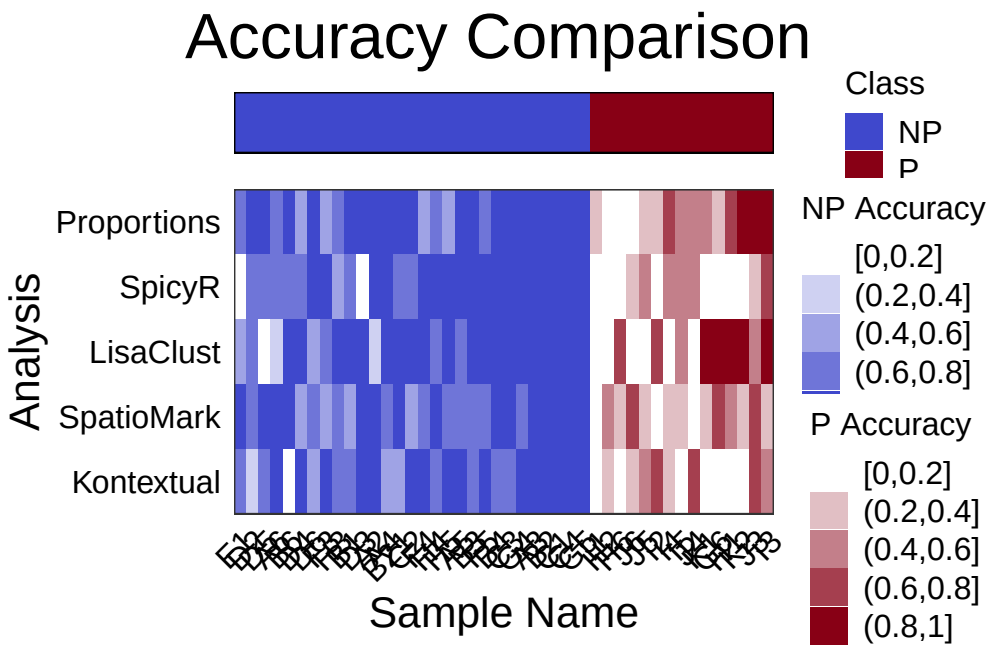



We can also visualise which features were good at classifying which patients using the `sampleMetricMap()` function from `ClassifyR`.

```
samplesMetricMap(cv)
```

```
Warning in .local(results, ...): Sample Accuracy not found in all elements of
results. Calculating it now.
```

```
Warning: Removed 2 rows containing missing values or values outside the scale range
(`geom_tile()`).
```




```
TableGrob (2 x 1) "arrange": 2 grobs
  z     cells    name                grob
1 1 (2-2,1-1) arrange       gtable[layout]
2 2 (1-1,1-1) arrange text[GRID.text.1936]
```

## 8.14 Summary

Here we have used a pipeline of our spatial analysis R packages to demonstrate an easy way to segment, cluster, normalise, quantify and classify high dimensional in situ cytometry data all within R.

## 8.15 sessionInfo

```
sessionInfo()
```

```
R version 4.4.1 (2024-06-14)
Platform: x86_64-pc-linux-gnu
Running under: Debian GNU/Linux 12 (bookworm)

Matrix products: default
BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/libopenblasp-r0.3.21.so;  LAPACK version 3

locale:
 [1] LC_CTYPE=C.UTF-8       LC_NUMERIC=C           LC_TIME=C.UTF-8
 [4] LC_COLLATE=C.UTF-8     LC_MONETARY=C.UTF-8    LC_MESSAGES=C.UTF-8
 [7] LC_PAPER=C.UTF-8       LC_NAME=C              LC_ADDRESS=C
[10] LC_TELEPHONE=C         LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C

time zone: Australia/Sydney
tzcode source: system (glibc)

attached base packages:
[1] stats4    stats     graphics  grDevices utils     datasets  methods
[8] base

other attached packages:
 [1] SpatialDatasets_1.4.0           ExperimentHub_2.14.0
 [3] AnnotationHub_3.14.0            BiocFileCache_2.14.0
 [5] dbplyr_2.5.0                    SpatialExperiment_1.16.0
 [7] ttservice_0.4.1                 tidyr_1.3.1
 [9] tidySingleCellExperiment_1.16.0 Statial_1.8.0
[11] lisaClust_1.14.4                ClassifyR_3.10.0
[13] survival_3.7-0                  BiocParallel_1.40.0
[15] MultiAssayExperiment_1.32.0     generics_0.1.3
[17] spicyR_1.18.0                   scater_1.34.0
[19] scuttle_1.16.0                  ggpubr_0.6.0
[21] FuseSOM_1.8.0                   simpleSeg_1.8.0
[23] ggplot2_3.5.1                   dplyr_1.1.4
[25] cytomapper_1.18.0               SingleCellExperiment_1.28.1
[27] SummarizedExperiment_1.36.0     Biobase_2.66.0
[29] GenomicRanges_1.58.0            GenomeInfoDb_1.42.0
[31] IRanges_2.40.0                  S4Vectors_0.44.0
[33] BiocGenerics_0.52.0             MatrixGenerics_1.18.0
[35] matrixStats_1.4.1               EBImage_4.48.0

loaded via a namespace (and not attached):
  [1] tiff_0.1-12                 FCPS_1.3.4
```

```
 [3] nnet_7.3-19                 goftest_1.2-3
 [5] Biostrings_2.74.0           HDF5Array_1.34.0
 [7] TH.data_1.1-2               vctrs_0.6.5
 [9] spatstat.random_3.3-2       digest_0.6.37
[11] png_0.1-8                   shape_1.4.6.1
[13] proxy_0.4-27                ggrepel_0.9.6
[15] deldir_2.0-4                permute_0.9-7
[17] magick_2.8.5                MASS_7.3-61
[19] reshape2_1.4.4              httpuv_1.6.15
[21] foreach_1.5.2               withr_3.0.2
[23] ggfun_0.1.7                 psych_2.4.6.26
[25] xfun_0.49                   ellipsis_0.3.2
[27] memoise_2.0.1               ggbeeswarm_0.7.2
[29] RProtoBufLib_2.18.0         diptest_0.77-1
[31] princurve_2.1.6             systemfonts_1.1.0
[33] tidytree_0.4.6              zoo_1.8-12
[35] GlobalOptions_0.1.2         V8_6.0.0
[37] DEoptimR_1.1-3              Formula_1.2-5
[39] prabclus_2.3-4              KEGGREST_1.46.0
[41] promises_1.3.0              httr_1.4.7
[43] rstatix_0.7.2               rhdf5filters_1.18.0
[45] fpc_2.2-13                  rhdf5_2.50.0
[47] rstudioapi_0.17.1           UCSC.utils_1.2.0
[49] concaveman_1.1.0            curl_6.0.1
[51] zlibbioc_1.52.0             ScaledMatrix_1.14.0
[53] analogue_0.17-7             polyclip_1.10-7
[55] GenomeInfoDbData_1.2.13     SparseArray_1.6.0
[57] fftwtools_0.9-11            xtable_1.8-4
[59] stringr_1.5.1               doParallel_1.0.17
[61] evaluate_1.0.1              S4Arrays_1.6.0
[63] irlba_2.3.5.1               colorspace_2.1-1
[65] filelock_1.0.3              spatstat.data_3.1-2
[67] flexmix_2.3-19              magrittr_2.0.3
[69] ggtree_3.14.0               later_1.3.2
[71] viridis_0.6.5               modeltools_0.2-23
[73] lattice_0.22-6              genefilter_1.88.0
[75] spatstat.geom_3.3-3         robustbase_0.99-4-1
[77] XML_3.99-0.17               cowplot_1.1.3
[79] RcppAnnoy_0.0.22            ggupset_0.4.0
[81] class_7.3-22                svgPanZoom_0.3.4
[83] pillar_1.9.0                nlme_3.1-166
[85] iterators_1.0.14            compiler_4.4.1
[87] beachmat_2.22.0             stringi_1.8.4
```

```
 [89] tensor_1.5                minqa_1.2.8
 [91] plyr_1.8.9                treekoR_1.14.0
 [93] crayon_1.5.3              abind_1.4-8
 [95] gridGraphics_0.5-1        locfit_1.5-9.10
 [97] sp_2.1-4                  bit_4.5.0
 [99] terra_1.7-83              sandwich_3.1-1
[101] multcomp_1.4-26           fastcluster_1.2.6
[103] codetools_0.2-20          BiocSingular_1.22.0
[105] coop_0.6-3                GetoptLong_1.0.5
[107] plotly_4.10.4             mime_0.12
[109] splines_4.4.1             circlize_0.4.16
[111] Rcpp_1.0.13-1             profileModel_0.6.1
[113] knitr_1.49                blob_1.2.4
[115] utf8_1.2.4                clue_0.3-66
[117] BiocVersion_3.20.0        lme4_1.1-35.5
[119] fs_1.6.5                  nnls_1.6
[121] ggsignif_0.6.4            ggplotify_0.1.2
[123] tibble_3.2.1              Matrix_1.7-1
[125] scam_1.2-17               statmod_1.5.0
[127] svglite_2.1.3             tweenr_2.0.3
[129] pkgconfig_2.0.3           pheatmap_1.0.12
[131] tools_4.4.1               cachem_1.1.0
[133] RSQLite_2.3.7             viridisLite_0.4.2
[135] DBI_1.2.3                 numDeriv_2016.8-1.1
[137] fastmap_1.2.0             rmarkdown_2.29
[139] scales_1.3.0              grid_4.4.1
[141] shinydashboard_0.7.2      broom_1.0.7
[143] patchwork_1.3.0           brglm_0.7.2
[145] BiocManager_1.30.25       carData_3.0-5
[147] farver_2.1.2              mgcv_1.9-1
[149] yaml_2.3.10               ggthemes_5.1.0
[151] cli_3.6.3                 purrr_1.0.2
[153] hopach_2.66.0             lifecycle_1.0.4
[155] uwot_0.2.2                mvtnorm_1.3-2
[157] kernlab_0.9-33            backports_1.5.0
[159] annotate_1.84.0           cytolib_2.18.0
[161] gtable_0.3.6              rjson_0.2.23
[163] parallel_4.4.1            ape_5.8
[165] limma_3.62.1              edgeR_4.4.0
[167] jsonlite_1.8.9            bitops_1.0-9
[169] bit64_4.5.2               Rtsne_0.17
[171] FlowSOM_2.14.0            yulab.utils_0.1.8
[173] vegan_2.6-8               spatstat.utils_3.1-1
```

```
[175] BiocNeighbors_2.0.0         ranger_0.17.0
[177] flowCore_2.18.0             bdsmatrix_1.3-7
[179] spatstat.univar_3.1-1       lazyeval_0.2.2
[181] ConsensusClusterPlus_1.70.0 shiny_1.9.1
[183] htmltools_0.5.8.1           diffcyt_1.26.0
[185] rappdirs_0.3.3              tinytex_0.54
[187] glue_1.8.0                  XVector_0.46.0
[189] RCurl_1.98-1.16             treeio_1.30.0
[191] mclust_6.1.1                mnormt_2.1.1
[193] coxme_2.2-22                jpeg_0.1-10
[195] gridExtra_2.3               boot_1.3-31
[197] igraph_2.1.1                R6_2.5.1
[199] ggiraph_0.8.10              labeling_0.4.3
[201] ggh4x_0.2.8                 cluster_2.1.6
[203] Rhdf5lib_1.28.0             aplot_0.2.3
[205] nloptr_2.1.1                DelayedArray_0.32.0
[207] tidyselect_1.2.1            vipor_0.4.7
[209] ggforce_0.4.2               raster_3.6-30
[211] car_3.1-3                   AnnotationDbi_1.68.0
[213] rsvd_1.0.5                  munsell_0.5.1
[215] DataVisualizations_1.3.2    data.table_1.16.2
[217] htmlwidgets_1.6.4           ComplexHeatmap_2.22.0
[219] RColorBrewer_1.1-3          rlang_1.1.4
[221] spatstat.sparse_3.1-0       spatstat.explore_3.3-3
[223] colorRamps_2.3.4            lmerTest_3.1-3
[225] uuid_1.2-1                  ggnewscale_0.5.0
[227] fansi_1.0.6                 beeswarm_0.4.0
```