Spatial Transcriptomics: Workshop Part 1

Dieter Henrik Heiland

2023-05-15

Introduction to the Workshop:

Spatially resolved transcriptomics is a cutting-edge approach used to study gene expression in tissues at a spatial level. It involves mapping RNA transcripts back to their original locations in a tissue, thereby offering insights into the spatial organization of cells within a tissue and the role of their spatial context in the biological function. This field has seen significant advancements with the development of various techniques, including Visium Spatial Gene Expression by 10x Genomics.

Visium Spatial Gene Expression

Visium Spatial Gene Expression is a product by 10x Genomics that allows researchers to understand the spatial context of gene expression in whole transcriptome profiling. Visium captures the gene expression from a whole tissue section and maps the expression data back to the original locations in the tissue.

Method

The Visium Spatial Gene Expression process begins with the placement of a fresh or frozen tissue section onto a Visium Spatial Gene Expression slide. The slide is covered with an array of oligo-dT spots, which capture mRNA molecules from the tissue. The tissue is then permeabilized, allowing the mRNA to hybridize to the oligo-dT spots. The hybridized mRNA is reverse transcribed to cDNA, followed by second-strand synthesis, which creates double-stranded cDNA. This cDNA is then amplified, indexed with barcodes to indicate their spatial origin, and sequenced using next-generation sequencing. The resulting sequencing reads can be mapped back to their original spatial location based on the barcode.

Advantages

- **High-resolution spatial data:** Visium allows for high-resolution mapping of gene expression data, thus providing insights into the spatial context of gene expression within a tissue.
- **Whole transcriptome analysis:** Unlike some spatial transcriptomics methods that focus on a subset of genes, Visium can profile the whole transcriptome, providing a comprehensive view of gene expression.
- **Compatibility with standard workflows:** Visium is designed to be compatible with standard histological workflows, making it easier to incorporate into existing research processes.

Disadvantages

- **Limited resolution:** Although Visium provides high-resolution data compared to traditional methods, it still can't match the cellular resolution provided by single-cell RNA sequencing.

- **Complex data analysis:** The data analysis process for Visium can be complex and requires specialized bioinformatics tools and expertise.
- **Cost:** As with many high-throughput techniques, the cost can be a limiting factor for some research groups.

In summary, Visium Spatial Gene Expression is a powerful tool for studying spatial transcriptomics, providing high-resolution, whole transcriptome data in a spatial context. However, it does come with some challenges, including resolution limitations, complex data analysis, and cost. Despite these challenges, Visium is a significant advancement in spatial transcriptomics and promises to provide valuable insights into many biological questions.

Concept and aims of the Workshop:

In the upcoming sections of this book, we will guide you through the process of handling and analyzing spatially resolved transcriptomic data. Starting from the Space Ranger output files, we will explain how to manage these comprehensive data sets and convert them into suitable formats for downstream analysis. In addition to familiarizing you with the file structure and content, we will introduce the methods for data extraction and conversion to enable compatibility with various data analysis tools. We will then delve into the SPATA (Spatial Transcriptomics Analysis) framework, a powerful toolbox for spatial analysis. Through hands-on examples and detailed explanations, you will learn how to visualize your data, interpret the results, and conduct robust spatial pattern analysis. The SPATA framework will help you transform your scientific questions into meaningful analysis and results. We understand that each research question is unique; therefore, we will also guide you on how to tailor these methods to address your specific questions. Lastly, we will discuss the important aspect of data reporting. We will share best practices for reporting your data and methods, including how to document your analysis process, create compelling figures, and write clear, reproducible methods sections. This knowledge will not only help you communicate your findings effectively but also ensure that your work can be reproduced and built upon by others in the field. By the end of this journey, you should be well-equipped to explore the fascinating world of spatial transcriptomics.

Chapter 1: Import the *spaceranger* output and create a SPATA2 object

Install the software and R environment

Certainly, here's a brief guide on how to install R and associated development tools on various platforms:

macOS

- **Install Xcode:** You can download Xcode from the Mac App Store. After installing Xcode, open Terminal and install the Command Line Tools by typing xcode-select –install. This will prompt you to install the tools, follow the instructions to complete the installation.
- **Install R:** Visit the CRAN (https://cran.r-project.org/) website and select the R package suitable for macOS. Download and follow the instructions in the installer.
- **Install GCC and other development tools:** These should be installed with Xcode and Command Line Tools, but if you need additional tools, consider using Homebrew (https://brew.sh/), a package manager for macOS.

Windows

- **Install R:** Visit the CRAN (https://cran.r-project.org/) website and select the R package suitable for Windows. Download and follow the instructions in the installer.
- **Install Rtools:** This is a collection of tools necessary for building packages on Windows. You can download Rtools from the CRAN website as well. After downloading, follow the instructions in the installer.

Linux (Ubuntu)

- **Install R:** Open Terminal and type the following commands:

```
sudo apt-get update sudo apt-get install r-base Install
```

- **GCC:** If it's not already installed, you can install it with the following command: sudo apt-get install build-essential After completing these steps, you should have a functioning R installation along with the necessary development tools on your operating system. Remember that the specific versions of the software may vary depending on the current date and the specifics of your operating system.

Install RStudio

Installing RStudio is quite straightforward across different operating systems once R has been successfully installed.

macOS and Windows

- Visit the RStudio download page at **https://rstudio.com/products/rstudio/download/**. You'll see different versions of RStudio available for download. Unless you have specific needs, choose the RStudio Desktop Open Source License version, which is free for use.
- Click the download button corresponding to your operating system (Windows or macOS). This will initiate the download of an installer file (.exe for Windows, .dmg for macOS).
- Once the installer file is downloaded, double-click the file to start the installation process. Follow the instructions provided by the installer, accepting the default settings.

Linux

- Visit the RStudio download page at **https://rstudio.com/products/rstudio/download/**. As with macOS and Windows, select the RStudio Desktop Open Source License version.
- Choose the installer file that matches your Linux distribution (.deb for Debian/Ubuntu, .rpm for Fedora/openSUSE). Download the file.
- Open Terminal and navigate to the directory containing the downloaded file. Depending on your Linux distribution, use the appropriate command to install RStudio:
  - For Debian/Ubuntu: `sudo dpkg -i <filename>`
  - For Fedora/openSUSE: `sudo yum install <filename>`

Replace `<filename>` with the name of the downloaded file.

After installation, you can launch RStudio from your applications menu or command line, and it will automatically use the R installation already present on your machine.

Install SPATA2 and required dependencies

To run the analysis, we need the SPATA package which further needs some packages to be installed which are not automated included in the installation of SPATA. For detailed information please follow the introduction of the SPATA package: https://themilolab.github.io/SPATA2/articles/spata-v2-installation.html

```
install.packages("devtools")

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

BiocManager::install(c('BiocGenerics', 'DelayedArray', 'DelayedMatrixStats',
                       'limma', 'S4Vectors', 'SingleCellExperiment',
                       'SummarizedExperiment', 'batchelor', 'Matrix.utils', 'EBImage'))

install.packages("Seurat")

devtools::install_github(repo = "kueckelj/confuns")
devtools::install_github(repo = "theMILOlab/SPATA2")
devtools::install_github(repo = "theMILOlab/SPATAData")

# if you want to use monocle3 related wrappers
devtools::install_github('cole-trapnell-lab/leidenbase')
devtools::install_github('cole-trapnell-lab/monocle3')
```

It's crucial to understand that the SPATA package does not encompass all functions required for spatial transcriptomics analysis. Instead, it serves as a fundamental toolset, which is regularly maintained and stable. However, in the rapidly evolving field of spatial transcriptomics, new analytical needs and techniques arise continuously. To address these advanced requirements, there are numerous other more recent packages that act as add-ons to the SPATA package. These packages provide specialized functions that enhance the capabilities of SPATA, but it's important to note that they are less intensively maintained and can be relatively unstable compared to the core SPATA package. In instances where specific problems necessitate the use of defined sub-functions not available in the core SPATA package, we incorporate these into the SPATAWrappers package. This package acts as a dynamic reservoir of supplemental functions, providing users with up-to-date tools to tackle emerging challenges in spatial transcriptomics analysis.

Create a SPATA object from spaceranger outputs

We start the workshop by creating a object. SPATA object can be seen as containers that hold data from various formats such as the gene expression data, the coordinates or the features that are important to describe your spots. In R, "object" are a general format to store data:

Here are a few examples of R objects:

1. **Vectors**: These are the simplest form of R objects and contain elements of the same type, like all numbers or all characters. Think of a vector as a list of items, such as a list of your friends' names or a list of test scores.
2. **Matrices**: A matrix is like a table with rows and columns, where all the elements are of the same type. If you've ever seen a spreadsheet, a matrix is similar, but every cell has to contain the same type of data.

3. **Data Frames**: These are like matrices, but they can contain different types of data in different columns. This makes them more flexible for storing complex data. If you have a table where one column is names (text), another is ages (numbers), and another is employed status (yes/no), you would use a data frame.

4. **Lists**: Lists are another type of object that can hold different types of elements, and each element can be a different size or shape. This could be a mix of numbers, characters, vectors, or even other lists.

5. **Functions**: These are also R objects. A function is like a mini-program that takes certain inputs and gives you an output based on those inputs.

In essence, R objects are just a way to store and organize data in R, so that you can work with it and analyze it.

Detailed information of the structure of the SPATA objects is available here: https://themilolab.github.io/SPATA2/articles/spata-v2-object-initiation-and-manipulation.html

To create a SPATA object we just need the path of the spaceranger output: `directory_10X` and the sample name, which can be chosen by the user: `sample_name`

```
library(SPATA2)
library(tidyverse)

outs_path <-  "path/to/a/10Xvisium-folder" # the directory from which to load the data
object <-initiateSpataObject_10X(
  directory_10X = outs_path,
  sample_name = "gbm-275")
```

SPATA2 is designed to provide a set of baseline analyses for spatial transcriptomics data. This includes scaling and normalization of the data to ensure comparability across different samples or experiments. It also incorporates dimensionality reduction techniques like UMAP (Uniform Manifold Approximation and Projection) and t-SNE (t-Distributed Stochastic Neighbor Embedding) to simplify high-dimensional data into fewer dimensions for visualization and analysis. Furthermore, it applies Shared Nearest Neighbor (SNN) clustering, an algorithm found in the Seurat package, to group similar data points together, thereby identifying distinct clusters or cell types within the data.

This suite of baseline processing methods is instrumental in understanding the overall structure and patterns within your data. However, please note that the intricacies of these methods will not be explored in depth in this tutorial. Our focus will be on how to use these tools rather than their underlying mechanics.

A key consideration when dealing with spatial data is the inherent spatial dependency — the idea that data points (or "spots") in close proximity are likely to be similar. While SPATA2's baseline tools are powerful, they treat each spot as an individual data point, without taking into account its spatial context or relationship to neighboring spots. This is a limitation when analyzing spatial transcriptomics data, as the spatial environment plays a significant role in gene expression patterns. Hence, for a more comprehensive analysis, it's crucial to incorporate methods that explicitly account for spatial relationships, beyond the baseline processing provided by SPATA2.

Explore and visualize with SPATA

QC and basic concepts of surface plots

After constructing a SPATA object, our first objective is to explore the data to understand its quality and heterogeneity. A standard output from this exploration is the total number of UMIs (Unique Molecular Identifiers) per spot. This provides a snapshot of the raw RNA abundance at each spatial location, helping

us gauge the variation in gene expression across the tissue. However, the variance in UMIs per spot can be influenced by several factors, including the number of cells per spot and the efficacy of permeabilization during sample preparation. For a nuanced quality control (QC) evaluation, we require information about the total number of cells per spot, a metric that isn't straightforward to determine. Over the years, we've developed two pipelines to address this challenge. Our most accurate pipeline involves using the Ilastik software to create a mask of cell nuclei, followed by CellProfiler for cell segmentation. This approach provides a detailed mapping of cells per spot but requires the use of external software and more computational resources. Alternatively, we have a less accurate but more accessible method that can be implemented directly in R using the spAnchor package, developed by DHH. While it may not be as precise as the Ilastik-CellProfiler pipeline, spAnchor provides a good balance between accuracy and ease of use, making it a suitable choice for many applications. In this workshop, we'll be using the spAnchor package for cell identification. While it may not provide as detailed information as the Ilastik-CellProfiler pipeline, it offers an efficient and effective method for getting a sense of cell distribution across your tissue sample. This will be instrumental in understanding and interpreting the spatial variation in your transcriptomic data.

Install `spAnchor` and required dependencies:

```
devtools::install_github("https://github.com/heilandd/spAnchor")
install.packages("imager")
remotes::install_github("jbergenstrahle/STUtility")
install.packages("akima")
```

The spAnchor package is using a method implemented in https://github.com/jbergenstrahle/STUtility.

Predict the number of cells per spot:

```
library(spAnchor)
object <- spAnchor::inferCellPositionsfromHE(object, sample.folder = outs_path)
```

```
## Using spotfiles to remove spots outside of tissue
## Loading ~/Desktop/Data/Visium/275_T/outs/filtered_feature_bc_matrix.h5 count matrix from a 'Visium' d
##
## ------------- Filtering (not including images based filtering) --------------
##   Spots removed:  0
##   Genes removed:  13541
## Saving capture area ranges to Staffli object
## After filtering the dimensions of the experiment is: [19997 genes, 3734 spots]
```

The cell positions are saved in the slot `@spatial$`gbm-275`$Cell_coords`:

```
object@spatial$`gbm-275`$Cell_coords %>% head()
```

```
##     Cell       x       y
## 1 Cell_1 107.520 480.000
## 2 Cell_2 187.392 402.432
## 3 Cell_3 215.424 153.600
## 4 Cell_4 290.304 388.224
## 5 Cell_5  86.016 100.608
## 6 Cell_6 185.088 270.336
```

Next we need to estimate the number of cells per spot. For this we need the `Run_extension.R` source file. A implementation of the functions will follow soon.

For know: `source("your_path_to/Run_extensions.R") install.packages("swfscMisc")`

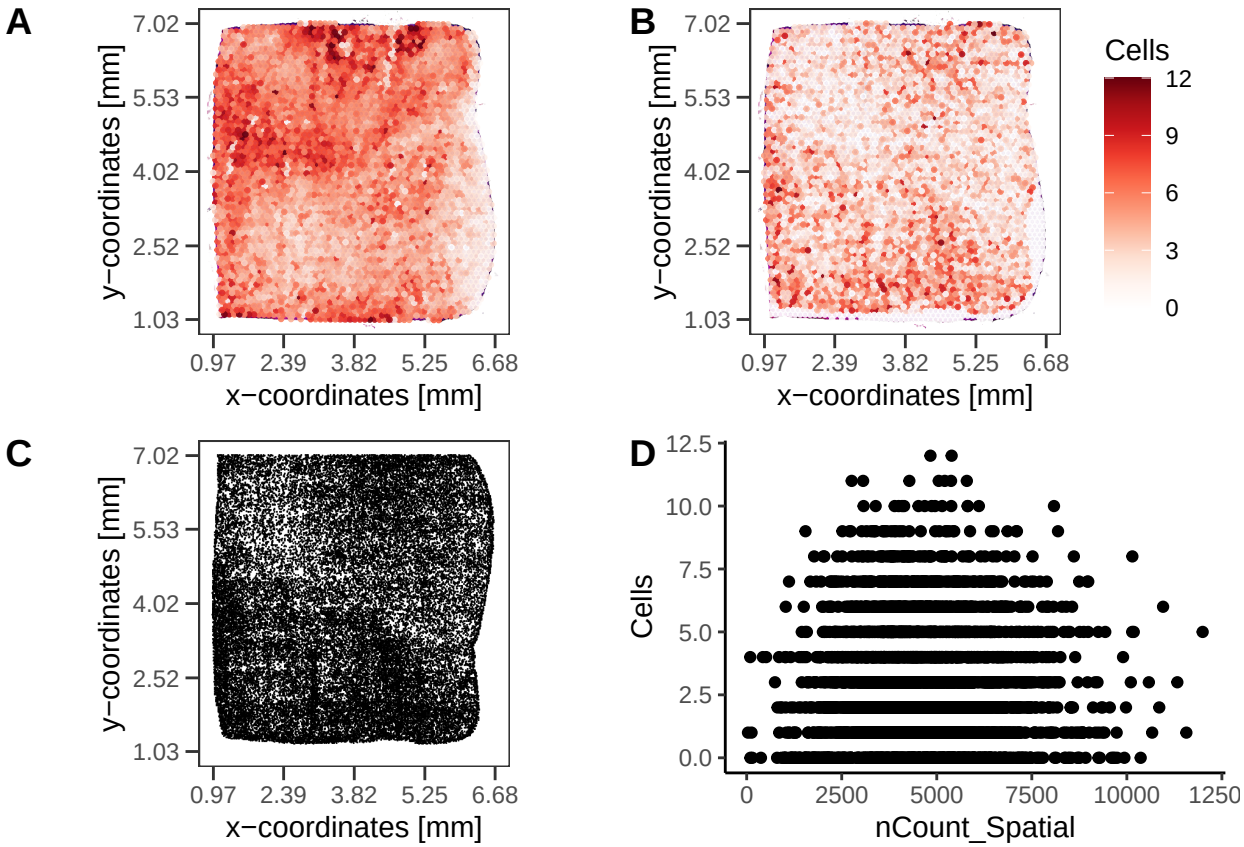
```
return <- runSegmentfromCoords(object, Coord_file = object@spatial$`gbm-275`$Cell_coords)
# With the  addFeatures() function we cann add a feature that characterizes a single spot
object <- addFeatures(object, return$Feature_cells, overwrite = T)
```

At this point we learn how to add a feature, something that characterizes a spot (for example a cluster ID or the number of cells in the spot . . .). The return file is a list of data.frames that contain the information of cells per spot. The $Feature_cells is the data.frame that contains the number of cells per spot information.

We can access the features that are stored in the SPATA object by the `getFeatureDf(object)` or the `getFeatureNames(object)` function

Next, we will plot the UMIs per spot, cell counts per spot and the actual cell positions. This can be done by SPATA `plotSurface()` function:

```
#Set the colors of interest
col <-colorRampPalette( c("#FFFFFF", RColorBrewer::brewer.pal(9, "Reds")))

plot_1 <-
  SPATA2::plotSurface(object, color_by = "nCount_Spatial")+
  scale_color_gradientn(colours = col(50), limits=c(500, 10000), oob=scales::squish)+
  SPATA2::ggpLayerAxesSI(object)+
  Seurat::NoLegend()

plot_2 <-
  SPATA2::plotSurface(object, color_by = "Cells")+
  scale_color_gradientn(colours = col(50))+
  SPATA2::ggpLayerAxesSI(object)

plot_3 <-
  ggplot(object@spatial$`gbm-275`$Cell_coords)+
  scattermore::geom_scattermore(mapping=aes(x,y), pointsize = 2)+
  SPATA2::ggpLayerAxesSI(object)+
  coord_fixed()

plot_4 <-
  joinWith(object, features = c("nCount_Spatial", "Cells")) %>%
  ggplot(aes(nCount_Spatial,Cells )) + geom_point()+ theme_classic()

# Illustrate the single plots in a figure like format

library(ggpubr)
ggarrange(plotlist = list(plot_1,plot_2,plot_3, plot_4),
          ncol = 2, nrow = 2,
          labels = c("A", "B", "C", "D"))
```



To break down the code:

- `col <-colorRampPalette( c("#FFFFFF", RColorBrewer::brewer.pal(9, "Reds")))`: This creates a color palette called `col` for use in the plots. `colorRampPalette()` generates a palette of colors that transitions smoothly between the specified colors, in this case from white (`"#FFFFFF"`) to a palette of red shades produced by `RColorBrewer::brewer.pal(9, "Reds")`.
- `SPATA2::plotSurface()`This creates a surface plot of SPATA object `object`, colored by for example the feature variable `nCount_Spatial` , which usually represents the number of spatially-resolved transcript UMIs. The color gradient is specified by the color palette `col`, and the color scale is limited between 500 and 10000, with values outside this range being squished into the limit values. The plot is then decorated with axes using `ggpLayerAxesSI()` and any legend is removed with `NoLegend()`.

The final scatter plot in the given code utilizes the `joinWith()` function to retrieve two key features, "nCount_Spatial" and "Cells", from the SPATA object. The "nCount_Spatial" feature typically represents the number of spatially-resolved transcript counts, while the "Cells" feature might signify the number or type of cells. The purpose of this step is to create a dataset that enables us to compare these two variables directly.

The actual plotting of this data is carried out using `ggplot()`, a function from the `ggplot2` package, which is a cornerstone of data visualization in R. `ggplot2` is based on the Grammar of Graphics, a system for describing and building graphs. With `ggplot2`, you can create complex multi-layered graphics starting from a basic plot, and then adding components gradually.

In the given code, `ggplot(aes(nCount_Spatial,Cells )) + geom_point()+ theme_classic()` creates a scatter plot of "nCount_Spatial" versus "Cells", using points to represent each data pair. The `theme_classic()` function is then used to apply a clean, minimalist aesthetic to the plot.

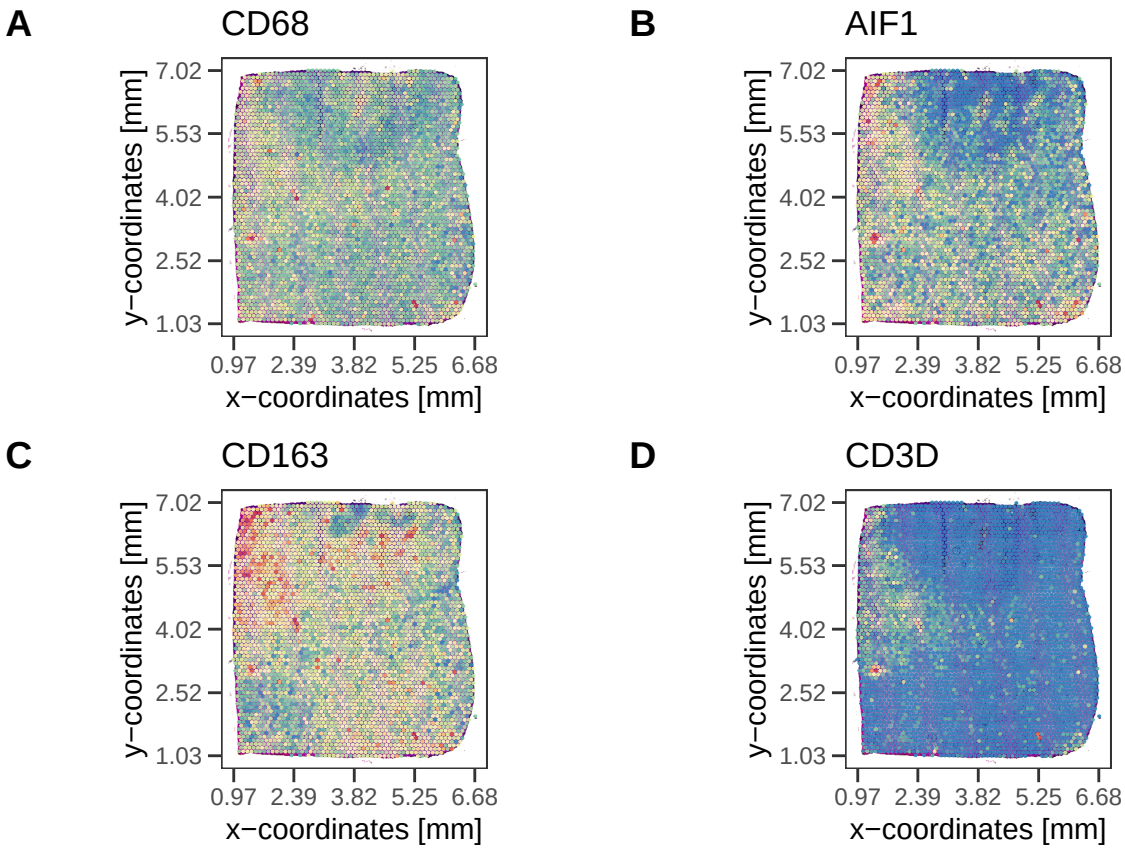
Given the versatility and power of **ggplot2**, it's highly recommended to delve deeper into this package if you're working with data visualization in R. Whether you're creating simple plots or complex multi-faceted graphics, **ggplot2** offers a wide array of options to meet your needs. The more you familiarize yourself with its functionalities, the more effectively you can communicate your data's story.

Exploration of gene expression and gene set enrichment

Much like the spatial visualization of features such as Unique Molecular Identifiers (UMIs) or cells per spot, we can investigate the spatial distribution of gene expression or pathway enrichment. This can be achieved using Gene Set Enrichment Analysis (GSEA), a method for interpreting gene expression data to determine whether defined sets of genes show statistically significant differences between two biological states.In this instance, we're interested in visualizing the expression of marker genes associated with tumor-associated macrophages (TAMs), such as CD68, AIF1, and CD163 or T cells (CD3D). These genes are often upregulated in TAMs, thus providing a molecular signature for these cells in the tissue. To achieve this visualization, we can use the **plotSurface()** function, the same function we've been using for illustrating features like UMIs and cells per spot. However, instead of these features, we'll be focusing on the expression levels of our TAM marker genes. The expression values for these genes can be extracted from the SPATA object using the **joinWith()** function, and then fed into **plotSurface()** to create a spatial expression map. Since all of the TAM marker genes are hypothesized to be co-expressed (as they are all expressed by the same cell type), it's worth conducting a correlation analysis. This involves examining the relationships between the expression patterns of these genes across the spatial transcriptomics data. If they are indeed co-expressed, we should see a positive correlation between them. This type of analysis can help further substantiate the presence of TAMs in the tissue, and possibly reveal insights about their spatial organization and potential functional roles in the tissue microenvironment.

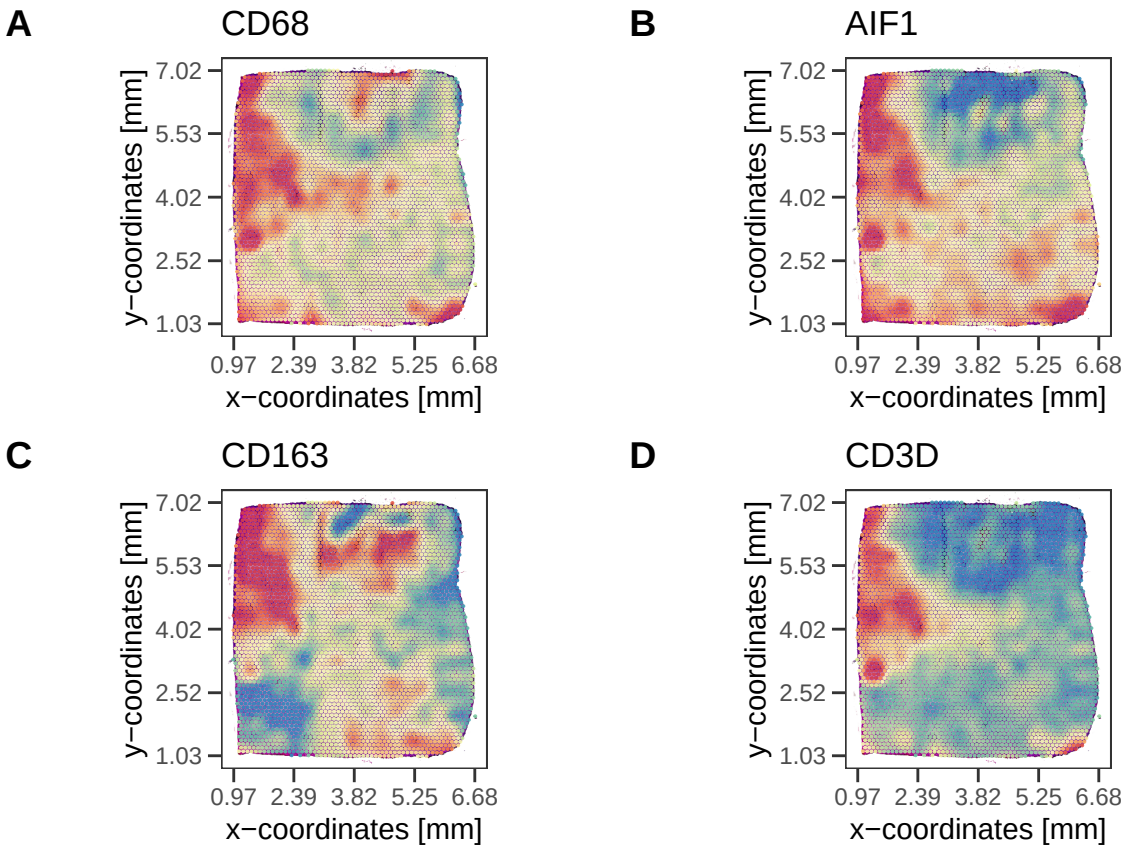
Since we aim to plot multiple genes, it is easier to create a list of plots which can be plotted by the `ggarrange()` function. As input we just want to have a vector containing the genes. A vector uses the function `c()` which means to combine individual genes. Next we create a loop using the `map()` function from the purrr package. This loop will do the same plot for the different inputs (genes). Here the code:

```
#Set the colors of interest
col <-colorRampPalette( rev(RColorBrewer::brewer.pal(9, "Spectral")))
genes <- c("CD68", "AIF1", "CD163", "CD3D")

#We create a loop through the gene vector

all_plots <- map(.x=genes, .f=function(gene){
  plot <-
  SPATA2::plotSurface(object, color_by = gene, pt_size = 1)+
  scale_color_gradientn(colours = col(50), limits=c(0.2, 0.8), oob=scales::squish)+
  SPATA2::ggpLayerAxesSI(object)+
  ggtitle(gene)+
  Seurat::NoLegend()
  return(plot)
})

ggarrange(plotlist = all_plots,
          ncol = 2, nrow = 2,
          labels = c("A", "B", "C", "D"))
```



As you can see, gene expression level of genes which are only expressed by a sub population can be noisy, a better visualization requires some denoising or imputation. There are several options to do this: You can smooth the expression pattern or you perform a auto-encoder denoising. Both options are implemented in SPATA2. We try both options. 1. Smooth the spatial expression pattern:

```
#Set the colors of interest
col <-colorRampPalette( rev(RColorBrewer::brewer.pal(9, "Spectral")))
genes <- c("CD68", "AIF1", "CD163", "CD3D")

#We create a loop through the gene vector

all_plots <- map(.x=genes, .f=function(gene){
  plot <-
  SPATA2::plotSurface(object, color_by = gene, pt_size = 1, smooth = T)+
  scale_color_gradientn(colours = col(50), limits=c(0.2, 0.8), oob=scales::squish)+
  SPATA2::ggpLayerAxesSI(object)+
  ggtitle(gene)+
  Seurat::NoLegend()
  return(plot)
})

ggarrange(plotlist = all_plots,
          ncol = 2, nrow = 2,
          labels = c("A", "B", "C", "D"))
```

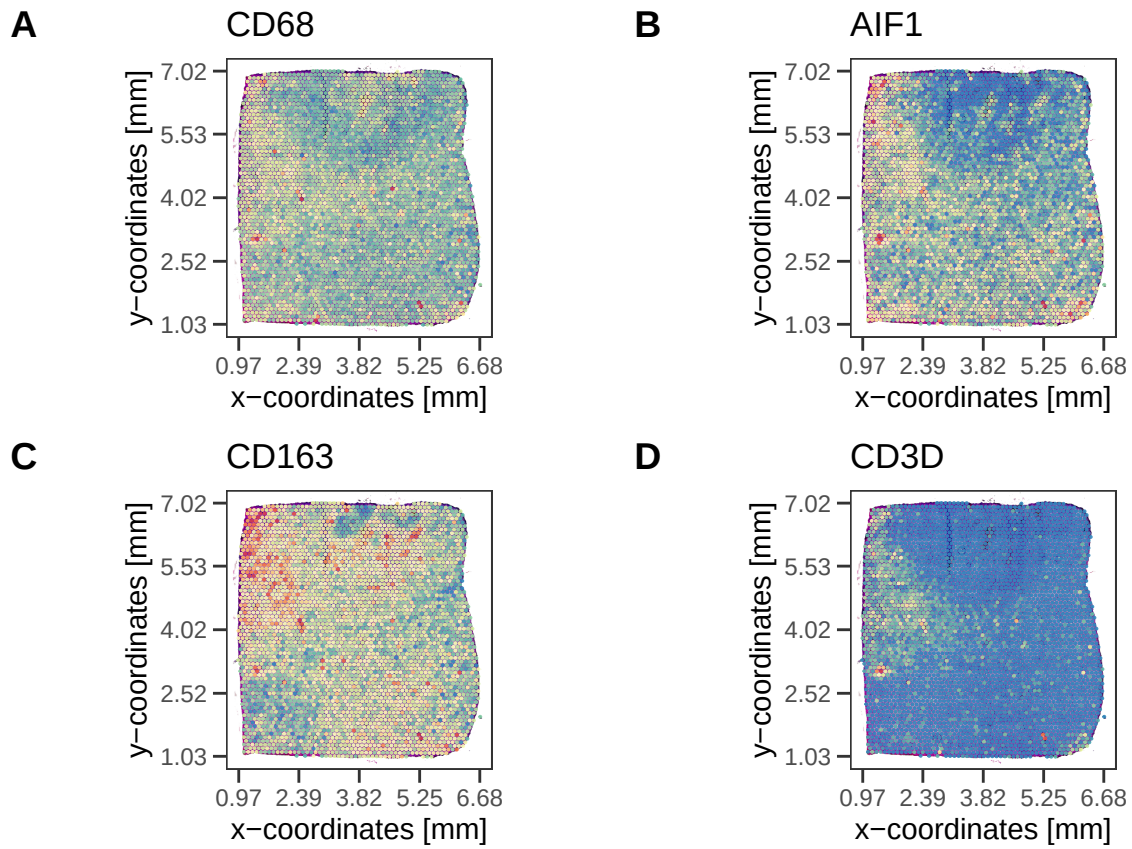


The more powerful option is represented by an autoencoder implemented in SPATA2. An autoencoder is a type of artificial neural network used for learning efficient codings of input data, typically for the purpose of dimensionality reduction or denoising. It has a specific architecture that allows it to compress data into a lower-dimensional form and then reconstruct it back to its original form. Autoencoders consist of two main components: an encoder and a decoder. The encoder compresses the input data and generates a condensed representation, often referred to as the "code" or "latent variables". The decoder then takes this code and reconstructs the original data as closely as it can. The reason autoencoders are useful for denoising spatial transcriptomic data stems from their ability to learn the underlying structure or manifold of the data. When trained on noisy data, the autoencoder learns to map the noisy input to its noise-free version, effectively learning to remove the noise. In the context of spatial transcriptomics, noise can come from various sources like technical variations, batch effects, dropouts, etc. By using an autoencoder, we can reduce this noise and get a clearer picture of the true biological signal. The denoised data is then more suitable for downstream analyses, including clustering, differential expression analysis, or spatial pattern detection, potentially leading to more accurate biological interpretations.

```
#Run the autoencoder
object <- SPATA2::runAutoencoderDenoising(object, activation = "relu", bottleneck=32)
```

```
# Set the denoised expression matrix as default
object@information$active_mtr <- "denoised"

#Set the colors of interest
col <-colorRampPalette( rev(RColorBrewer::brewer.pal(9, "Spectral")))
genes <- c("CD68", "AIF1", "CD163", "CD3D")

#We create a loop through the gene vector
```

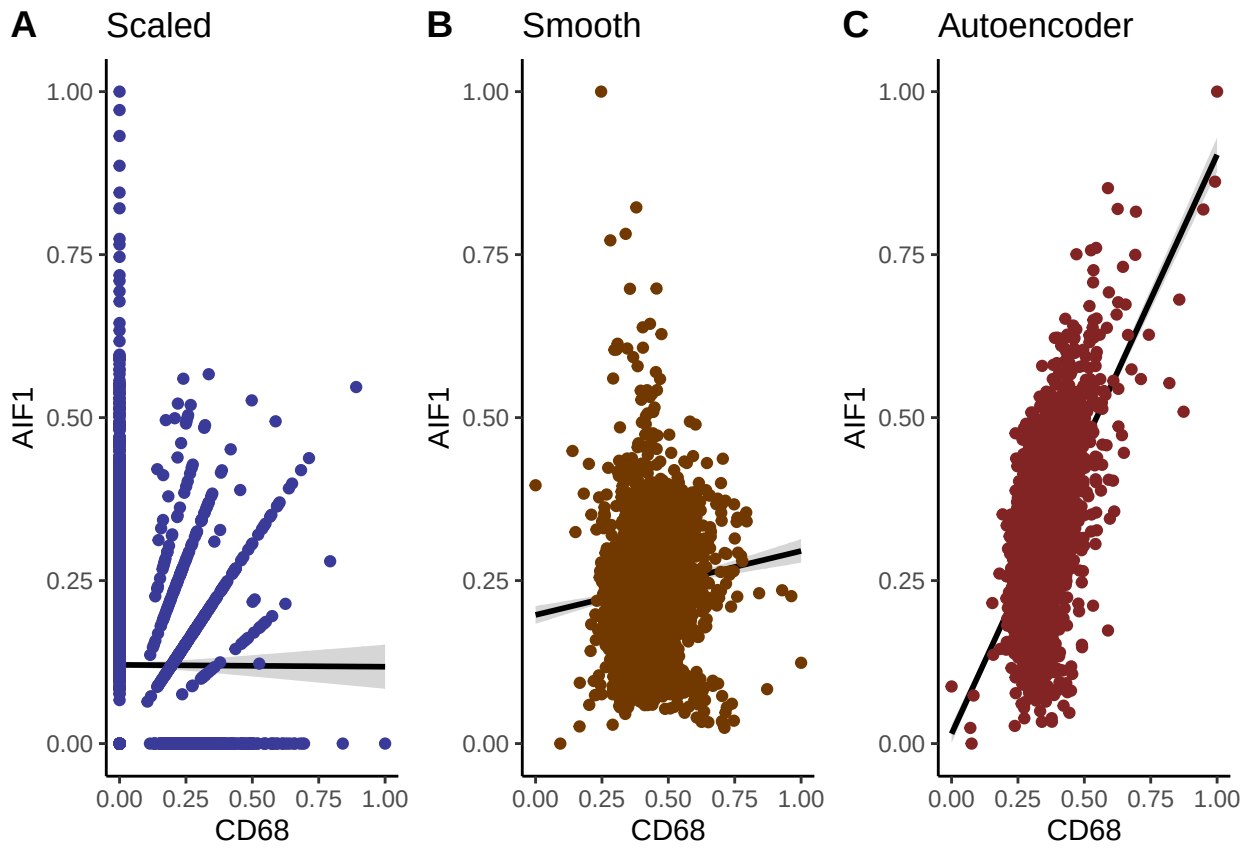
```
all_plots <- map(.x=genes, .f=function(gene){
  plot <-
  SPATA2::plotSurface(object, color_by = gene, pt_size = 1, smooth = F)+
  scale_color_gradientn(colours = col(50), limits=c(0.2, 0.8), oob=scales::squish)+
  SPATA2::ggpLayerAxesSI(object)+
  ggtitle(gene)+
  Seurat::NoLegend()
  return(plot)
})

ggarrange(plotlist = all_plots,
          ncol = 2, nrow = 2,
          labels = c("A", "B", "C", "D"))
```



Our hypothesis was that at least the gene expression of CD68 and AIF1 is correlated. Lets check if this holds true before and after denoising:

```
genes <- c("CD68", "AIF1", "CD163", "CD3D")

object@information$active_mtr <- "denoised"
AE_plot <- joinWith(object, genes = genes) %>%
  ggplot(aes(CD68, AIF1))+
  geom_smooth(method = "lm", color="black")+
  geom_point(color=scales::muted("red"))+
  theme_classic()+
```

```
  ggtitle("Autoencoder")

object@information$active_mtr <- "scaled"
scaled_plot <- joinWith(object, genes = genes) %>%
  ggplot(aes(CD68, AIF1))+
  geom_smooth(method = "lm", color="black")+
  geom_point(color=scales::muted("blue"))+
  theme_classic()+
  ggtitle("Scaled")

smooth_plot <- joinWith(object, genes = genes, smooth = T) %>%
  ggplot(aes(CD68, AIF1))+
  geom_smooth(method = "lm", color="black")+
  geom_point(color=scales::muted("orange"))+
  theme_classic()+
  ggtitle("Smooth")

ggarrange(plotlist = list(scaled_plot,smooth_plot,AE_plot),
          ncol = 3, nrow = 1,
          labels = c("A", "B", "C"))
```



Based on our biological context, the auto-encoder provides the expected correlation of CD68 and AIF1 which can not be found by the "noisy" raw data. With this analysis we adress allready a very important topic: how to correlate features, expression or any variable in spatial data? We will address this problem more in detail in the next section.

Spatial autocorrelation and spatial correlation analysis

Spatial autocorrelation and spatial correlation are two important concepts in spatial analysis. Spatial autocorrelation refers to the degree to which a data point's position in space influences its value. In other words, it measures the correlation of a variable with itself through space. Positive spatial autocorrelation occurs when similar values occur near each other, while negative spatial autocorrelation occurs when dissimilar values are nearby. Spatial correlation, on the other hand, refers to the statistical correlation between two or more spatial processes. Any two or more variables that are observed across space may have a spatial correlation. For instance, the amount of rainfall in an area might be spatially correlated with the number of trees in that area. When it comes to spatial data, it's important to understand why standard (Pearson) correlation may not be appropriate. The key reason is the inherent nature of spatial data to violate the assumption of independence, a fundamental premise for Pearson correlation. In spatial data, measurements taken at locations close to each other are often more similar than measurements taken at locations far apart—a phenomenon encapsulated in Tobler's First Law of Geography as "everything is related to everything else, but near things are more related than distant things." This inherent spatial autocorrelation means that the assumption of independence is violated, and thus, the use of Pearson correlation can lead to misleading results. Instead, spatial correlation measures, which take into account the spatial arrangement of data points, should be used when dealing with spatial data. These measures consider the spatial structure of the data and can provide a more accurate understanding of the relationships between variables in a spatial context.

To overcome this problem and how to perform spatial correlation I will show some examples. We will focus on a scientific question: Does TAMs and T cell are spatially correlated? Based on the fact that we have not touched the topic of single cell deconvolution and the power of cell type specific gene expression, we break down the problem by classical marker gene expression of CD68, AIF1, CD168 for TAMs and CD3D for T cells and a mixed score with GFAP, SOX2 and MKI67 for the tumor.

First we need to extract the spatial coordinates on one side and create a matrix with the cell-type scores by gene expression. We will use the `joinwith()` function for this purpose:

```
TAM <- c("CD68", "AIF1", "CD163")
Tcell <- c("CD3D")
Tumor <- c("GFAP", "SOX2", "MKI67")

object@information$active_mtr <- "denoised"

coords <- SPATA2::getCoordsDf(object)

cell_scores <-
  data.frame(TAM=joinWith(object, genes = TAM, average_genes = T) %>% pull(mean_genes),
             Tcell=joinWith(object, genes = Tcell, average_genes = T) %>% pull(mean_genes),
             Tumor=joinWith(object, genes = Tumor, average_genes = T) %>% pull(mean_genes))
head(cell_scores)
```

```
##         TAM      Tcell     Tumor
## 1 0.2997960 0.07019117 0.7231472
## 2 0.2808997 0.12646260 0.6351648
## 3 0.2559615 0.06402640 0.5882665
## 4 0.3763128 0.26038569 0.8237931
## 5 0.3418071 0.24069540 0.6589664
## 6 0.3517055 0.16374694 0.6984851
```

```
# Add feature to the SPATA object
plot_df <- cbind(coords, cell_scores)
object <- addFeatures(object, plot_df %>%
```

```
                          dplyr::select(barcodes, Tumor, TAM, Tcell),
                        overwrite = T)
```

Spatial Autocorrelation

In the upcoming steps, we will employ spatial autocorrelation, a vital tool in spatial analysis, to determine whether the expression of a gene or a gene set is patterned or randomly distributed across space. To be specific, we will use Moran's I, a measure of spatial autocorrelation, to quantify the degree of spatial pattern in our gene expression data. Moran's I ranges from -1 to +1, where a positive value indicates a clustered or patterned distribution, a value near zero suggests a random spatial distribution, and a negative value suggests a dispersed or evenly distributed pattern. To calculate Moran's I, we will perform Monte Carlo simulations. These simulations use random sampling to solve deterministic problems, allowing us to generate a distribution of Moran's I under the null hypothesis of spatial randomness. By comparing our observed Moran's I to this distribution, we can determine whether the spatial pattern of our gene or gene set is significantly different from random.

In essence, spatial autocorrelation analysis using Moran's I will provide a statistically robust way to discern whether there's a spatial pattern in the gene or gene set expression in our data, enriching our spatial transcriptomics analysis.

We will be employing the SPATAWrappers package to generate a plot visualizing the spatial overlap of two distinct features. This package is a useful extension to the principal SPATA package, providing a set of additional tools aimed at enhancing the depth and breadth of spatial transcriptomics analysis.To get started with SPATAWrappers, it can be conveniently installed directly from GitHub using the **devtools** package in R. The installation code would be **devtools::install_github("https://github.com/heilandd/SPATAwrappers")**.

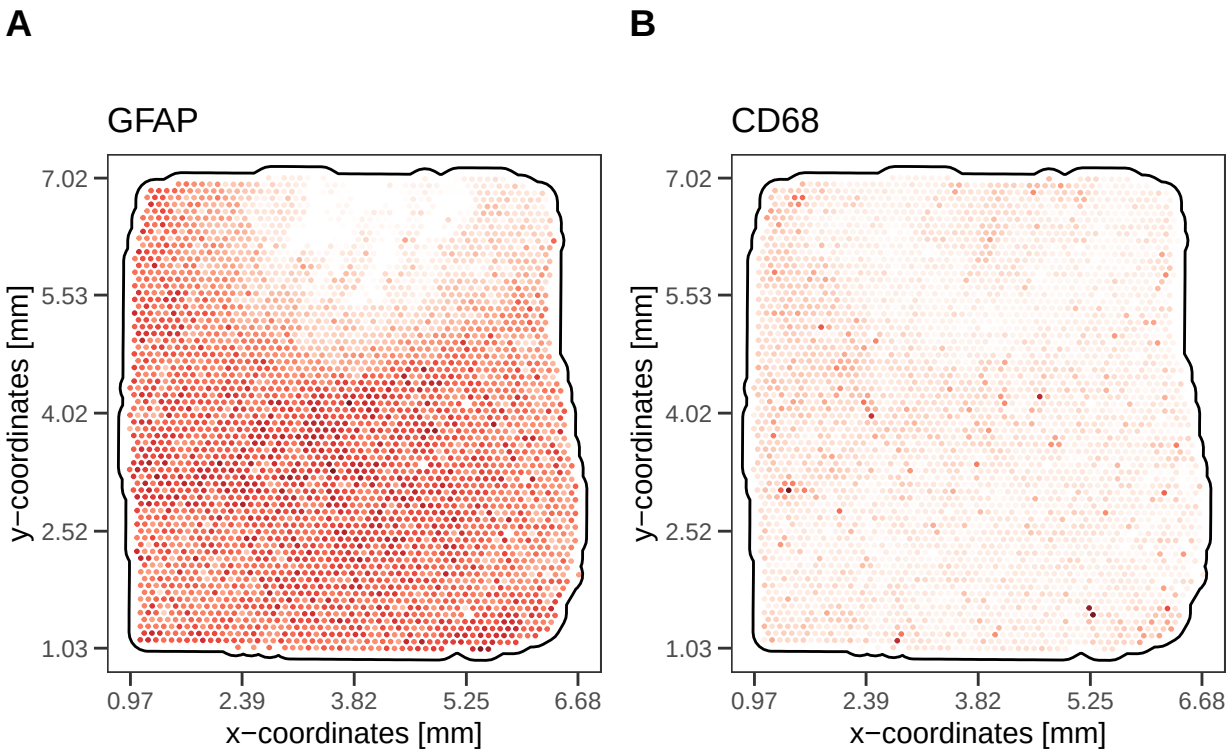
```
library(SPATAwrappers)
object@information$active_mtr <- "denoised"

MoransI <- SPATAwrappers::getMoransI(object, features = c(TAM, Tumor, Tcell))
MoransI
```

```
##           MoranI      p_value
## CD68  0.05340471 0.0009756098
## AIF1  0.12212533 0.0009756098
## CD163 0.10247845 0.0009756098
## GFAP  0.27530980 0.0009756098
## SOX2  0.09966783 0.0009756098
## MKI67 0.16450961 0.0009756098
## CD3D  0.14428140 0.0009756098
```

From the results, we can interpretative that GFAP followed a strict pattern (Moran's I=0.27) compared to SOX2 (Moran's I=0.09). We can visualize the expression to confirm the analysis:

```
#Set the colors of interest
col <-colorRampPalette(c("#FFFFFF", RColorBrewer::brewer.pal(9, "Reds")))
genes <- c("GFAP", "CD68")

#We create a loop through the gene vector
all_plots <- map(.x=genes, .f=function(gene){
  plot <-
  SPATA2::plotSurface(object, color_by = gene, pt_size = 1, smooth = F, display_image = F)+
  scale_color_gradientn(colours = col(50), limits=c(0.2, 1), oob=scales::squish)+
```

```
  SPATA2::ggpLayerAxesSI(object)+
  ggforce::geom_mark_hull(getCoordsDf(object), mapping = ggplot2::aes(x = x,
                                                                      y = y,
                                                                      group = sample),
                          expand = ggplot2::unit(0.15, "cm"))+
  ggtitle(gene)+
  Seurat::NoLegend()
  return(plot)
})

ggarrange(plotlist = all_plots,
          ncol = 2, nrow = 1,
          labels = c("A", "B"))
```



Spatial Correlation of features

In the next step of our analysis, we will employ two distinct methods of correlation calculation: classical Pearson correlation and spatial correlation. While the Pearson correlation provides a measure of the linear correlation between two variables, it does not take into account the spatial relationship between data points, which can be a significant factor in spatial transcriptomic data. Therefore, to gain a more accurate and comprehensive understanding of the relationships in our data, we will also perform spatial correlation.

For this purpose, we will be using the "MERINGUE" package in R. This package offers a set of tools specifically designed for spatial analysis and is particularly well-suited for handling spatial transcriptomic data.

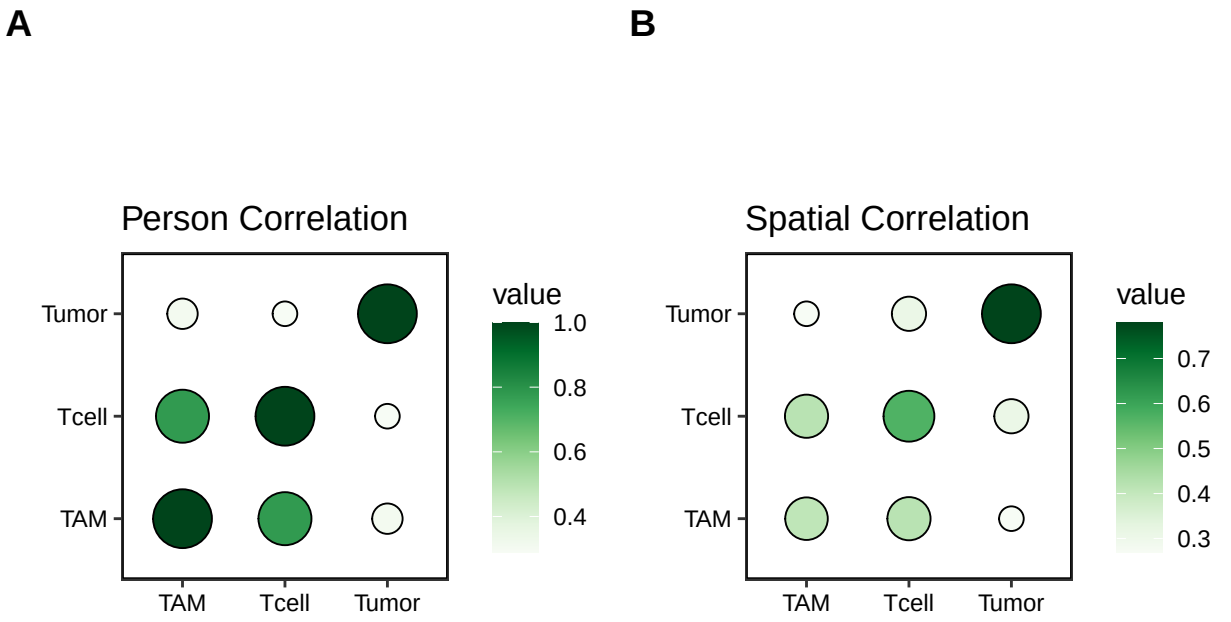
The key advantage of performing both types of correlation is that it allows us to compare and contrast the results, helping us to better understand the implications of spatial structure in our data. While the Pearson correlation may give us an initial overview of the relationships between variables, the spatial correlation, by accounting for spatial relationships, will provide more nuanced insights, capturing patterns that the Pearson correlation might miss. Please note that the application of these methods should be accompanied by a thorough interpretation of the results, considering the spatial nature of the data, to draw meaningful conclusions.

```
# Pearson correlation:
pearson <- cor(cell_scores, method="pearson")

library(MERINGUE)
barcodes <- coords$barcodes
pos <- data.frame(x=coords$x, y=coords$y)
rownames(pos) <- barcodes
weight <- MERINGUE::getSpatialNeighbors(pos)

feature_mat <- cell_scores %>% as.matrix()
rownames(feature_mat) <- barcodes
scc <- MERINGUE::spatialCrossCorMatrix(t(feature_mat), weight)
```

We can illustrate the correlation by dot-plots:

```
col <-colorRampPalette((RColorBrewer::brewer.pal(9, "Greens")))

library(ggcorrplot)
plot_1 <-
  ggcorrplot(t(pearson), method = "circle", outline.color = "black")+
  scale_fill_gradientn(colours = col(50))+
  ggtitle("Person Correlation")+
  xlab("")+ylab("")+
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_rect(colour = "black", size=1),
          axis.text.x = element_text(colour="black"),
          axis.text.y = element_text(colour="black"))+
  coord_fixed()

plot_2 <-
  ggcorrplot(t(scc), method = "circle", outline.color = "black")+
  scale_fill_gradientn(colours = col(50))+
  ggtitle("Spatial Correlation")+
  xlab("")+ylab("")+
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_rect(colour = "black", size=1),
          axis.text.x = element_text(colour="black"),
          axis.text.y = element_text(colour="black"))+
  coord_fixed()


ggarrange(plotlist = list(plot_1,plot_2),
```

```
          ncol = 2, nrow = 1,
          labels = c("A", "B"))
```



The analyses have yielded comparable outcomes, highlighting a stronger correlation between T cell and tumor-associated macrophage (TAM) populations as compared to the tumor itself. This suggests that the spatial arrangement of T cells and TAMs in the tissue is more closely related than their arrangement with respect to the tumor cells.

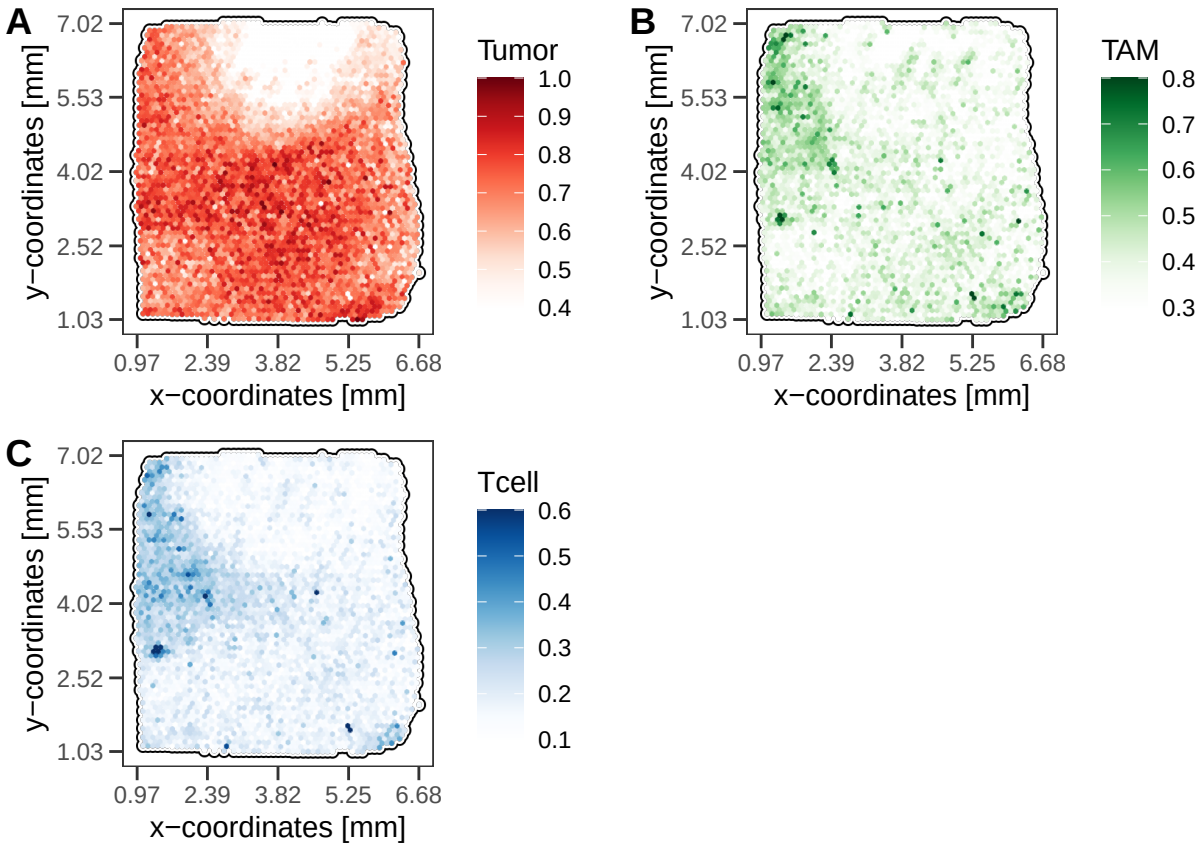
To further illustrate this relationship, we can create surface plots using the `ggplot2` package. These plots will offer a spatial visualization of the expression profiles of T cells, TAMs, and tumor cells. To enhance the clarity of these plots, we will mark the outline of the tissue specimens with a black line.

Here's a step-by-step outline of the upcoming process:

1. Create a surface plot for the T cell population, marking the boundaries of the tissue specimens.
2. Repeat this process for the TAM population, and again for the tumor cells.
3. Finally, generate an "overlap" plot. This special plot can be used to display the gene expression proximity, providing a visual representation of the spatial correlation between different cell populations in the tissue.

These plots not only offer a visual representation of the distribution of different cell populations in the tissue, but they also provide an opportunity to visualize the spatial correlation between T cells and TAMs, and their relationship with the tumor.

```r
# combine coords and cell scores
plot_df <- cbind(coords, cell_scores)

# To increase the speed of plotting, we will use raster plots
library(scattermore)
col <-colorRampPalette(c("#FFFFFF", RColorBrewer::brewer.pal(9, "Reds")))
plot_tumor <-
  ggplot(plot_df)+
  geom_scattermore(mapping=aes(x,y), color="black", pointsize = 12)+
  geom_scattermore(mapping=aes(x,y), color="white", pointsize = 9)+
  geom_scattermore(mapping=aes(x,y, color=Tumor), pointsize = 5)+
  scale_color_gradientn(colours = col(50),limits=c(0.4, 1), oob=scales::squish)+
  SPATA2::ggpLayerAxesSI(object)+
  coord_fixed()


col <-colorRampPalette(c("#FFFFFF", RColorBrewer::brewer.pal(9, "Greens")))
plot_TAM <-
  ggplot(plot_df)+
  geom_scattermore(mapping=aes(x,y), color="black", pointsize = 12)+
  geom_scattermore(mapping=aes(x,y), color="white", pointsize = 9)+
  geom_scattermore(mapping=aes(x,y, color=TAM), pointsize = 5)+
  scale_color_gradientn(colours = col(50),limits=c(0.3, 0.8), oob=scales::squish)+
  SPATA2::ggpLayerAxesSI(object)+
  coord_fixed()

col <-colorRampPalette(c("#FFFFFF", RColorBrewer::brewer.pal(9, "Blues")))
plot_TC <-
  ggplot(plot_df)+
  geom_scattermore(mapping=aes(x,y), color="black", pointsize = 12)+
  geom_scattermore(mapping=aes(x,y), color="white", pointsize = 9)+
  geom_scattermore(mapping=aes(x,y, color=Tcell), pointsize = 5)+
  scale_color_gradientn(colours = col(50),limits=c(0.1, 0.6), oob=scales::squish)+
  SPATA2::ggpLayerAxesSI(object)+
  coord_fixed()

ggarrange(plotlist = list(plot_tumor,plot_TAM,plot_TC),
          ncol = 2, nrow = 2,
          labels = c("A", "B", "C"))
```

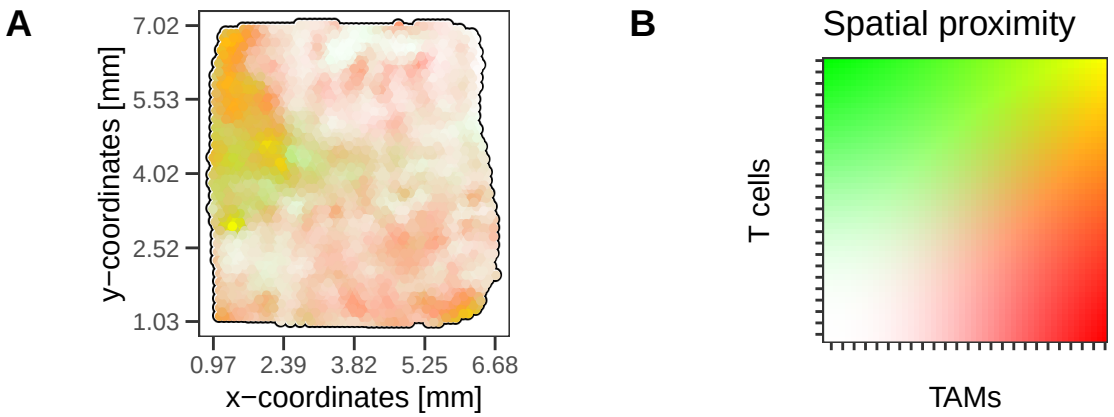


Once we have the package ready to use, we will leverage its function that enables plotting the expression proximity of two features. This function, however, requires that the features of interest are already stored in the SPATA object. Luckily, we have previously covered the **`addFeatures()`** function, a tool that allows us to add and store any specific features in the SPATA object.

By using **`addFeatures()`**, we can ensure our features of interest are accessible within the SPATA object, thereby meeting the prerequisite for using the SPATAWrappers' function for plotting expression proximity. This process underscores the importance of understanding how to manipulate and manage the data within the SPATA framework, as it enables us to take full advantage of the analytical capabilities provided by both the core package and its extensions like SPATAWrappers.

```
library(SPATAwrappers)

## Create the plot with outlines

# 1. Overlap plot
p_o <- plotColorOverlap(object,
                  feature1="TAM",
                  feature2="Tcell",
                  two.colors = c("red", "green"),
                  negative.color = "white",
                  col.threshold=0.4,
                  pt_size = 1)
# 2. Add outlines

p <-
  ggplot(data=p_o$data, aes(x,y))+
```

```
  geom_scattermore(mapping=aes(x,y), color="black", pointsize = 12)+
  geom_scattermore(mapping=aes(x,y), color="white", pointsize = 9)+
  p_o$layers[[1]]+
  coord_fixed()

# 3. Add Scales
p <- p+SPATA2::ggpLayerAxesSI(object)

# 4. Create 2D colorbar
map <- plotColorOverlap(object,
                        get.map = T ,
                        feature1="TAM",
                        feature2="Tcell",
                        two.colors = c("red", "green"),
                        negative.color = "white",
                        col.threshold=0.4,
                        pt_size = 2)+
  xlab("TAMs")+ylab("T cells")+
  ggtitle("Spatial proximity")+
  coord_equal()+
  theme_bw() +
    theme(panel.grid.major = element_blank(),
          panel.grid.minor = element_blank(),
          panel.background = element_rect(colour = "black", size=1),
          axis.text.x = element_text(colour="white"),
          axis.text.y = element_text(colour="white"))


# Create a plot
ggarrange(plotlist = list(p,map),
          ncol = 2, nrow = 2,
          labels = c("A", "B"))
```



Summary

In the first part of the workshop, you should be able to import data to create SPATA objects and visualize gene expression data. Another option to get a fast overview of the spatial data is to open the Shiny gui, which allows interactive visualization of features, genes and gene Sets. `plotSurfaceInteractive()`

In the second part, we have discussed some QC criteria and spatial correlation analysis. Also we showed how to used autoencoder to denoise the data.