Introduction to SpatialOmicsOverlay

Compiled: 2022-04-19

Overview

This vignette demonstrates how to use OME-TIFF files generated by the NanoString GeoMx instrument to enhance the data visualization for a GeoMx experiment. SpatialOmicsOverlay was specifically made to visualize and analyze the free-handed nature of Region of Interest (ROI) selection and the immunofluorescent-guided segmentation process in a GeoMx experiment. The overlay from the instrument is recreated in the R environment allowing for plotting overlays with data like ROI type or gene expression.

NOTE: This package deals with large images causing some functions to take time to run and can be memory intensive. It is recommended to run on a server.

In this vignette, we will be walking through a mouse brain dataset from the Spatial Organ Atlas

Package installation

```
BiocManager::install(version = "3.15")

BiocManager::install("SpatialOmicsOverlay")
BiocManager::install("GeomxTools")
```

Windows install troubleshooting

If you are getting errors on install check your Java version. If you are running x64-bit R you must also have x64-bit Java jdk downloaded. Java download, instructions

Data Ingestion

```
library(GeomxTools)
library(SpatialOmicsOverlay)
```

Files needed:

1. OME-TIFF from GeoMx instrument (other OME-TIFFs should work with custom parsing functions)
2. Lab Worksheet from instrument readout
3. Annotation file(s)
    - Lab Worksheet
    - GeomxSet Object (see GeomxTools and GeoMxWorkflows for more info)

- Annotations from DSPDA

Reading in the SpatialOverlay object can be done with or without the image. We will start without the image as that can be added later.

If `outline = TRUE`, only ROI outline points are saved. This decreases memory needed and figure rendering time downstream. If ANY ROIs are segmented in the study, outline will be FALSE. In this particular example, there are segmented ROIs, so we set `outline = FALSE`.

In this example, we are downloading a tiff image from a Box folder but this variable is simply the file path to a OME-TIFF.

```
tifFile <- downloadMouseBrainImage()

tifFile
```

```
##                                                                          BFC8
## "/home/rstudio/.cache/R/SpatialOmicsOverlay/ed095435d941_mu_brain_004.ome.tiff"
```

```
muBrainLW <- system.file("extdata", "muBrain_LabWorksheet.txt",
                         package = "SpatialOmicsOverlay")

muBrain <- readSpatialOverlay(ometiff = tifFile, annots = muBrainLW,
                              slideName = "4", image = FALSE,
                              saveFile = FALSE, outline = FALSE)
```

```
## [1] "Extracting XML"
## [1] "Parsing XML - scan metadata"
## [1] "Parsing XML - overlay data"
## [1] "Generating Coordinates"
```

The readSpatialOverlay function is a wrapper to walk through all of the necessary steps to store the OME-TIFF file components. This function automates XML extraction & parsing, image extraction, and coordinate generation. These functions can also be run separately if desired (`xmlExtraction`, `parseScanMetadata`, `parseOverlayAttrs`, `addImageOmeTiff`, `createCoordFile`).

SpatialOverlay Accessors

SpatialOverlay objects hold data specific to the image and the ROIs. Here are a couple of functions to access the most important parts.

```
#full object
muBrain
```

```
## SpatialOverlay
## Slide Name: 4
## Overlay Data: 87 samples
##    Overlay Names: DSP-1012996073013-H-A02 ... DSP-1012996073013-H-H09 ( 87 total )
## Scan Metadata
##    Panels: TAP Mouse Whole Transcriptome Atlas
##    Segmentation: Segmented
## Outline: FALSE
```

```
#sample names
head(sampNames(muBrain))
```

```
## [1] "DSP-1012996073013-H-A02" "DSP-1012996073013-H-A03"
## [3] "DSP-1012996073013-H-A04" "DSP-1012996073013-H-A05"
## [5] "DSP-1012996073013-H-A06" "DSP-1012996073013-H-A07"
```

```
#slide name
slideName(muBrain)
```

```
## [1] "4"
```

```
#metadata of ROI overlays
#Height, Width, X, Y values are in pixels
head(meta(overlay(muBrain)))
```

```
##   ROILabel               Sample_ID Height Width     X     Y Segmentation
## 1        1 DSP-1012996073013-H-A02    751   751 10363 15008    Geometric
## 2        2 DSP-1012996073013-H-A03   1239   782 14346 16630    Geometric
## 3        3 DSP-1012996073013-H-A04   1272   902 15664 16603    Geometric
## 4        4 DSP-1012996073013-H-A05   1119  1276 13707 16135    Geometric
## 5        5 DSP-1012996073013-H-A06   1054  1124 15932 16228    Geometric
## 6        6 DSP-1012996073013-H-A07    751   751 10756 14259    Geometric
```

```
#coordinates of each ROI
head(coords(muBrain))
```

```
##                  sampleID ycoor xcoor
## 1 DSP-1012996073013-H-A02 15383 10363
## 2 DSP-1012996073013-H-A02 15356 10364
## 3 DSP-1012996073013-H-A02 15357 10364
## 4 DSP-1012996073013-H-A02 15358 10364
## 5 DSP-1012996073013-H-A02 15359 10364
## 6 DSP-1012996073013-H-A02 15360 10364
```

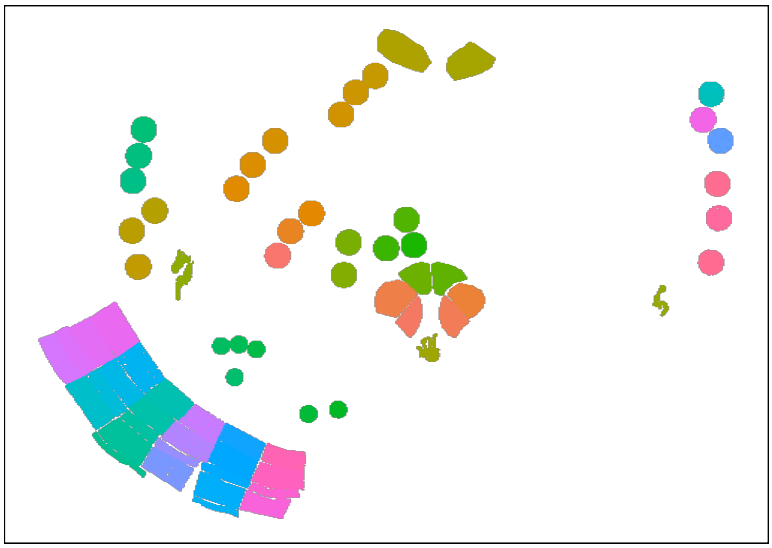
Plotting Without Image

After parsing, ROIs can be plotted without the image in the object. These plots are the highest resolution versions since there is no scaling down to the image size, and might take a little time to render. If the image is attached to the object, coordinates are automatically scaled down to the image size and plotted as if they are on top of the image.

While manipulating the figure, there is a low-resolution option for faster rendering times.

A scale bar is automatically calculated when plotting. This functionality can be turned off using `scaleBar = FALSE`. Scale bars can be fully customized using `corner`, `textDistance`, and variables that start with scaleBar: `scaleBarWidth`, `scaleBarColor`, etc.

```
plotSpatialOverlay(overlay = muBrain, hiRes = FALSE, legend = FALSE)
```



`colorBy`, by default, is Sample_ID but almost any annotation or data can be added instead, including gene expression, tissue morphology annotations, pathway score, etc. These annotations can come from a data.frame, matrix, GeomxSet object, or vector. Below we attach the gene expression for CALM1 from a GeomxSet object and color the segments by that value.

```
muBrainAnnots <- readLabWorksheet(lw = muBrainLW, slideName = "4")
muBrainGeomxSet <- readRDS(system.file("extdata", "muBrain_GxT.RDS",
                                       package = "SpatialOmicsOverlay"))

muBrain <- addPlottingFactor(overlay = muBrain, annots = muBrainAnnots,
                             plottingFactor = "segment")
muBrain <- addPlottingFactor(overlay = muBrain, annots = muBrainGeomxSet,
                             plottingFactor = "Calm1")
muBrain <- addPlottingFactor(overlay = muBrain, annots = 1:length(sampNames(muBrain)),
                             plottingFactor = "ROILabel")

muBrain
```

```
## SpatialOverlay
## Slide Name: 4
## Overlay Data: 87 samples
##    Overlay Names: DSP-1012996073013-H-A02 ... DSP-1012996073013-H-H09 ( 87 total )
## Scan Metadata
##    Panels: TAP Mouse Whole Transcriptome Atlas
##    Segmentation: Segmented
## Plotting Factors:
##    varLabels: segment Calm1 ROILabel
## Outline: FALSE
```

```
head(plotFactors(muBrain))
```

```
##                          segment Calm1 ROILabel
## DSP-1012996073013-H-A02 Full ROI  1105        1
## DSP-1012996073013-H-A03 Full ROI   737        2
## DSP-1012996073013-H-A04 Full ROI   760        3
```

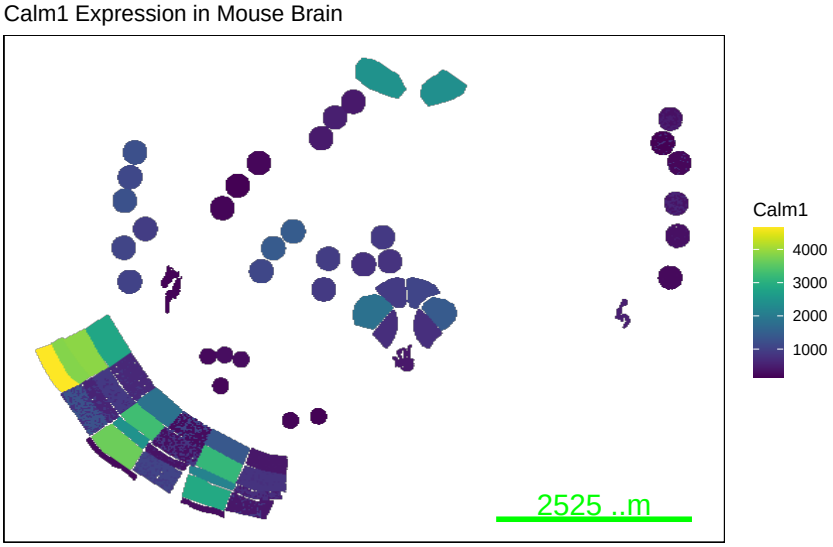
```
## DSP-1012996073013-H-A05 Full ROI  1836        4
## DSP-1012996073013-H-A06 Full ROI  1422        5
## DSP-1012996073013-H-A07 Full ROI  1434        6
```

Customizing the graph

All generated figures are ggplot based so they can be easily customized using functions from that or similar grammar of graphs packages. For example, we can change the color scale to the viridis color palette.

Note: hiRes and outline figures use fill, lowRes uses color

```
plotSpatialOverlay(overlay = muBrain, hiRes = FALSE, colorBy = "Calm1",
                   scaleBarWidth = 0.3, scaleBarColor = "green") +
    viridis::scale_color_viridis()+
    labs(title = "Calm1 Expression in Mouse Brain")
```



Adding the Image

Images can be added automatically using `readSpatialOverlay(image = TRUE)` or added after reading in the object.

An OME-TIFF file is a pyramidal file, meaning that many sizes of an image are saved. The largest having the highest resolution and decreasing as the image gets smaller. Images are 1/2 the size as the previous resolution.

The `res` variable says which resolution of the image to extract. 1 = largest image and the higher values get smaller. Each OME-TIFF has a different number of layers, with most having around 8. It is suggested to use the smallest `res` value, and highest resolution, your environment can handle. This is a trial and error process.

Using too big of an image will cause a java memory error. If this error occurs, increase your `res` value. Below is an example of the error you will receive if the resolution is too high for your system.

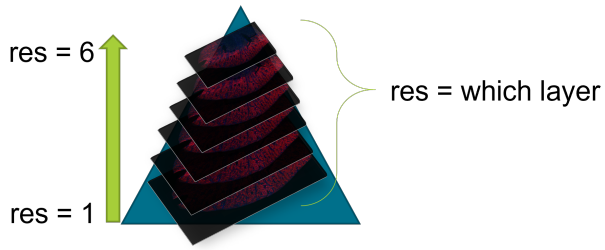


Figure 1: Pyramidal TIFF

```
muBrain <- addImageOmeTiff(overlay = muBrain, ometiff = tifFile, res = 1)
```

```
## Error in .jcall("RBioFormats", "Ljava/lang/Object;", "readPixels", i, : java.lang.NegativeArraySizeE
```

The resolution size will affect speed and image resolution through the rest of the analysis. To check the smallest resolution size available, for the fastest speeds, use `checkValidRes()`. For the rest of this tutorial we will be using res = 6.

```
res <- 6
checkValidRes(ometiff = tifFile)
```

```
## [1] 8
```

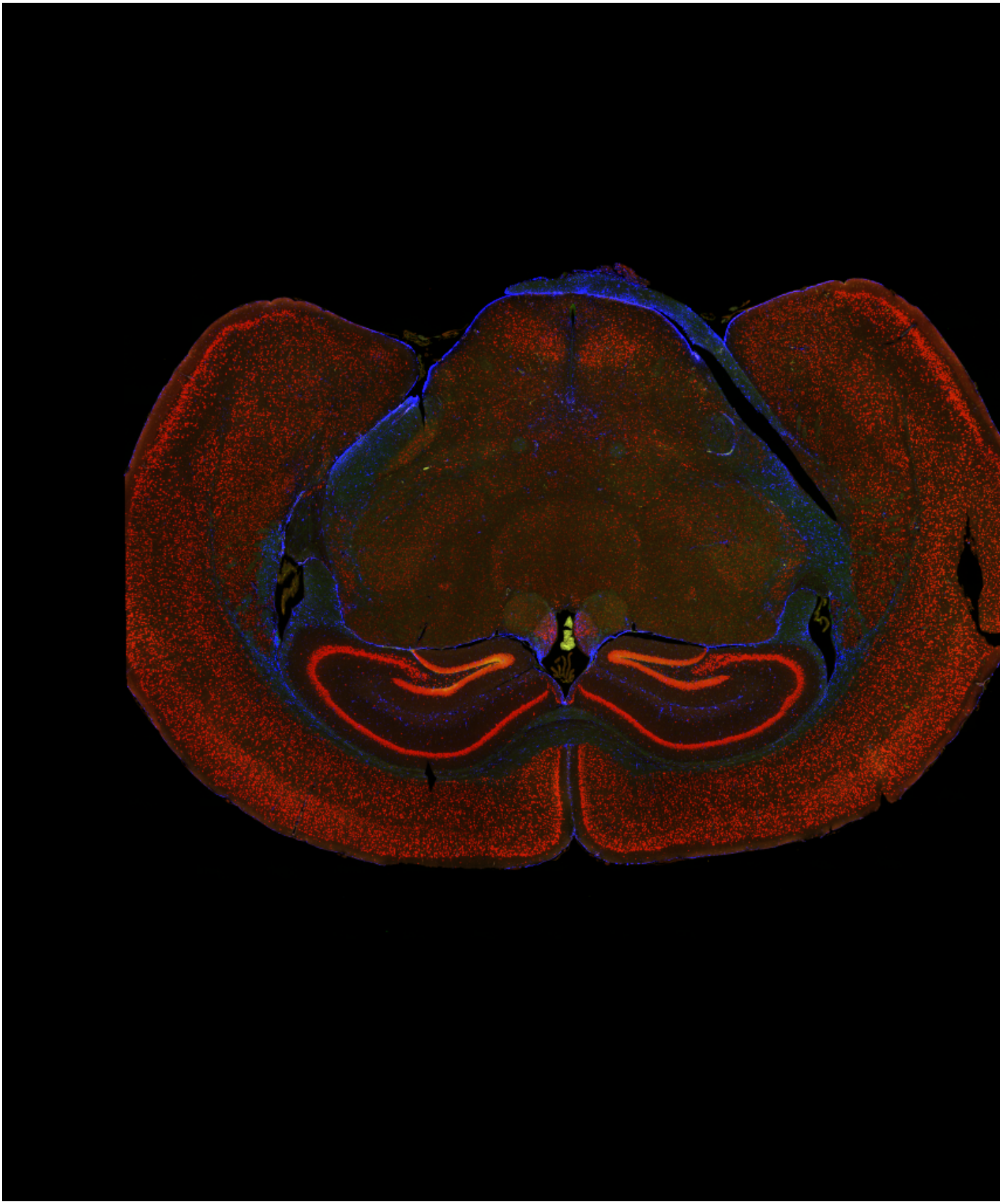
```
muBrain <- addImageOmeTiff(overlay = muBrain, ometiff = tifFile, res = res)
```

```
## [1] "Calculating and scaling coordinates"
```

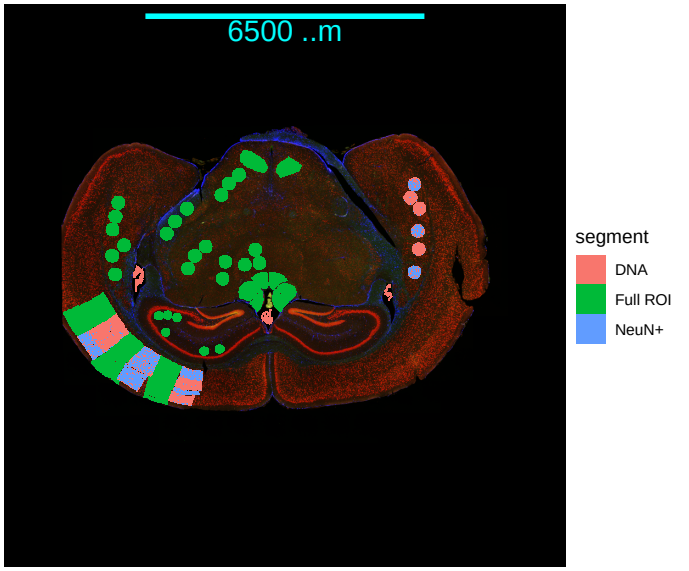
```
muBrain
```

```
## SpatialOverlay
## Slide Name: 4
## Overlay Data: 87 samples
##    Overlay Names: DSP-1012996073013-H-A02 ... DSP-1012996073013-H-H09 ( 87 total )
## Scan Metadata
##    Panels: TAP Mouse Whole Transcriptome Atlas
##    Segmentation: Segmented
## Plotting Factors:
##    varLabels: segment Calm1 ROILabel
## Outline: FALSE
## Image: /home/rstudio/.cache/R/SpatialOmicsOverlay/ed095435d941_mu_brain_004.ome.tiff
```

```
showImage(muBrain)
```



```
plotSpatialOverlay(overlay = muBrain, colorBy = "segment", corner = "topcenter",
                   scaleBarWidth = 0.5, textDistance = 130, scaleBarColor = "cyan")
```

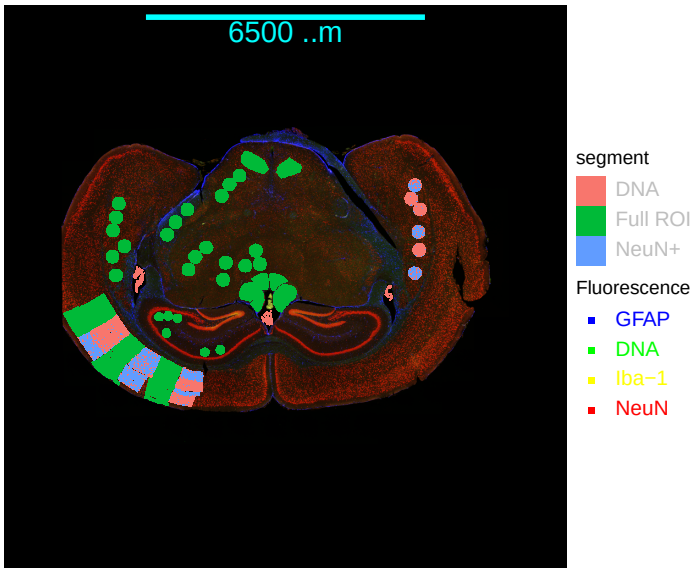


Visualization Marker Legends

There are 2 ways to add a legend to the graph showing the immunofluorescent visualization markers used.

The first is an easy way for data exploration, adding a legend to the ggplot object directly by setting `flourLegend = TRUE`.

```
plotSpatialOverlay(overlay = muBrain, colorBy = "segment", corner = "topcenter",
                   scaleBarWidth = 0.5, textDistance = 130, scaleBarColor = "cyan",
                   fluorLegend = TRUE)
```



The second requires more user manipulation but creates a more publication-ready figure. The `flourLegend` function creates a separate plot that can be added to the graph. The legend shape can be changed with `nrow` and the background can be changed using `boxColor` and `alpha`.

See `?draw_plot` for more instructions on how to manipulate the legend position and scale.

```
library(cowplot)
gp <- plotSpatialOverlay(overlay = muBrain, colorBy = "segment",
                         corner = "bottomright")
legend <- fluorLegend(muBrain, nrow = 2, textSize = 4,
                      boxColor = "grey85", alpha = 0.3)

cowplot::ggdraw() +
    cowplot::draw_plot(gp) +
    cowplot::draw_plot(legend, scale = 0.105, x = 0.1, y = -0.25)
```

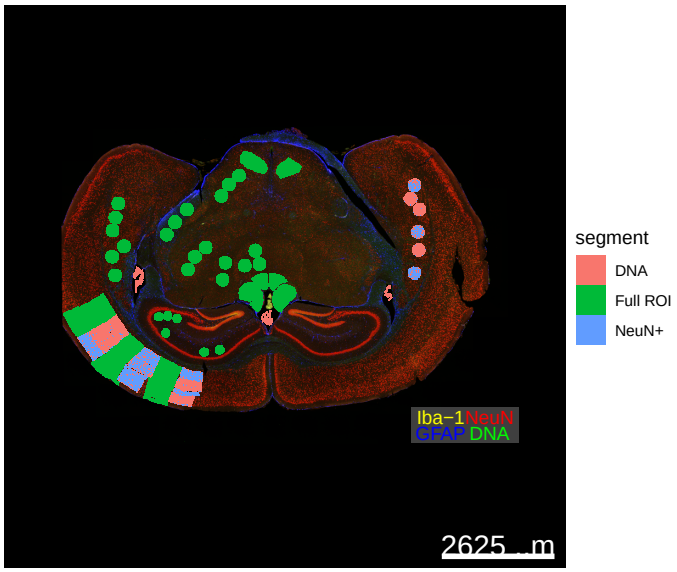
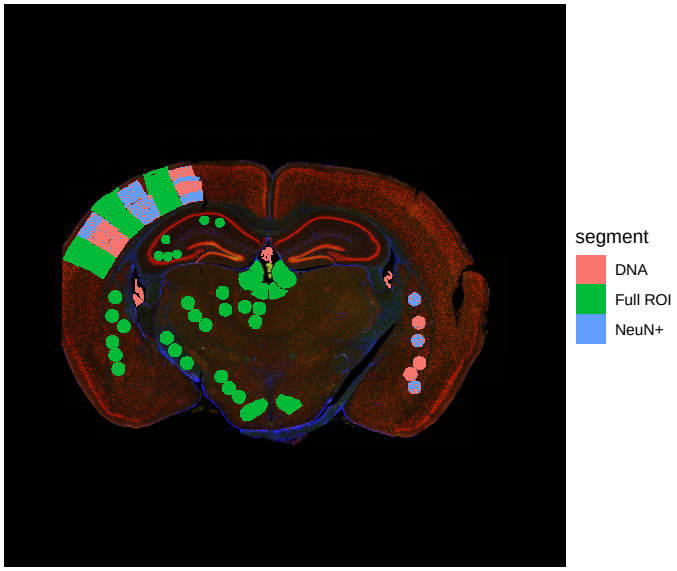


Image Manipulation

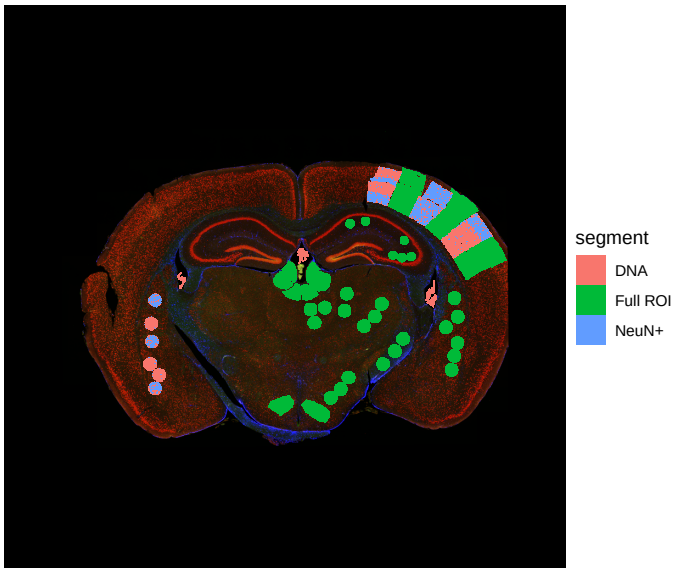
Flipping Axes

Images and overlays can be flipped across either axis to reorient the image. To flip both axes use `flipY(flipX(overlay))`. These functions update the coordinates and image rather than just affecting the figure. In this example, the original image is reversed from the traditional view of the mouse brain, so we shall flip the Y-axis.

```
muBrain <- flipY(muBrain)

plotSpatialOverlay(overlay = muBrain, colorBy = "segment", scaleBar = FALSE)
```




```
plotSpatialOverlay(overlay = flipX(muBrain), colorBy = "segment", scaleBar = FALSE)
```

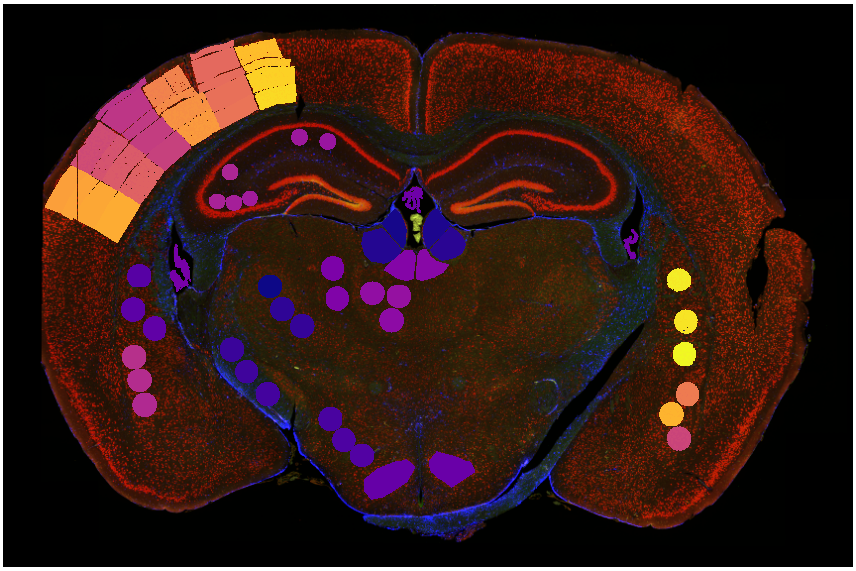


Cropping

Images can be cropped 2 ways. The amount of area added to the cropped area in both methods can be defined by `buffer`. This adds a percentage of the final image size to each edge.

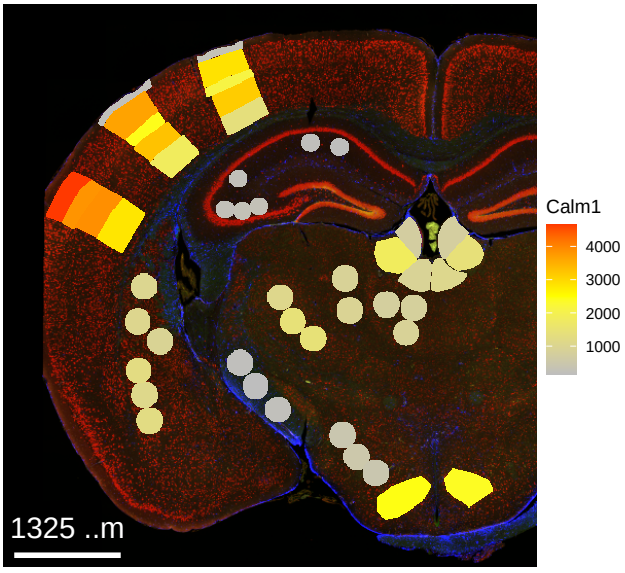
1. `cropTissue` automatically detects where the tissue is and removes non-tissue area from around the tissue.

```
muBrain <- cropTissue(overlay = muBrain, buffer = 0.05)

plotSpatialOverlay(overlay = muBrain, colorBy = "ROILabel", legend = FALSE, scaleBar = FALSE)+
    viridis::scale_fill_viridis(option = "C")
```



2. `cropSamples` automatically crops the image around the ROIs given. Other ROIs in the cropped image can be kept in or ignored. Below we will crop to only ROIs that are unsegmented, hiding ROIs profiled which are segmented. Setting `sampsOnly = TRUE` hides the segmented ROIs which are within the plotted region.

```
samps <- muBrainAnnots$Sample_ID[muBrainAnnots$segment == "Full ROI" & muBrainAnnots$slide.name == slid

muBrainCrop <- cropSamples(overlay = muBrain, sampleIDs = samps, sampsOnly = TRUE)

plotSpatialOverlay(overlay = muBrainCrop, colorBy = "Calm1", scaleBar = TRUE, corner = "bottomleft", te
    scale_fill_gradient2(low = "grey", high = "red", mid = "yellow", midpoint = 2500)
```




```
muBrainCrop <- cropSamples(overlay = muBrain, sampleIDs = samps, sampsOnly = FALSE)

plotSpatialOverlay(overlay = muBrainCrop, colorBy = "segment", scaleBar = TRUE, corner = "bottomleft",
```

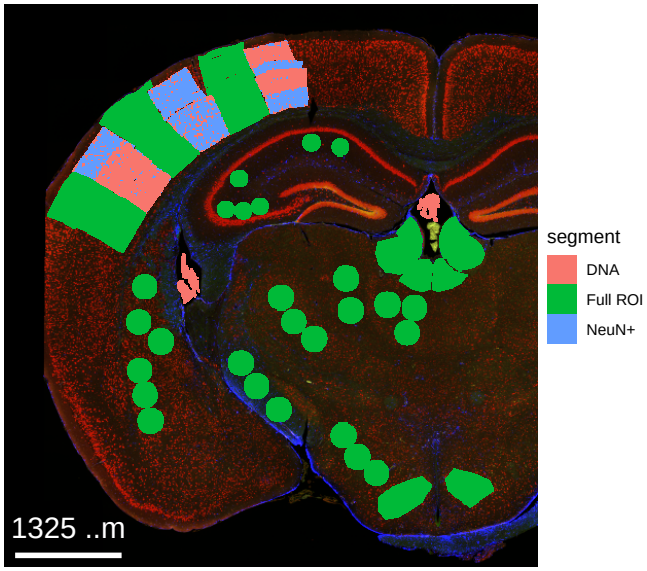


Image Coloring

Image colors are typically determined before downloading the OME-TIFF from the instrument but can be recolored here.

This recoloring must be done on the 4 channel image before converting to RGB. The color code and min/max intensities determine the coloring of the RGB image. To view the current color definition use the `fluor` function.

The color can be a hex color or a valid R color name. The dye can either come from the Dye or DisplayName columns from fluor(overlay). To change a color use the `changeImageColoring` function.

```
chan4 <- add4ChannelImage(overlay = muBrain)
```

```
## [1] "Calculating and scaling coordinates"
```

```
fluor(chan4)
```

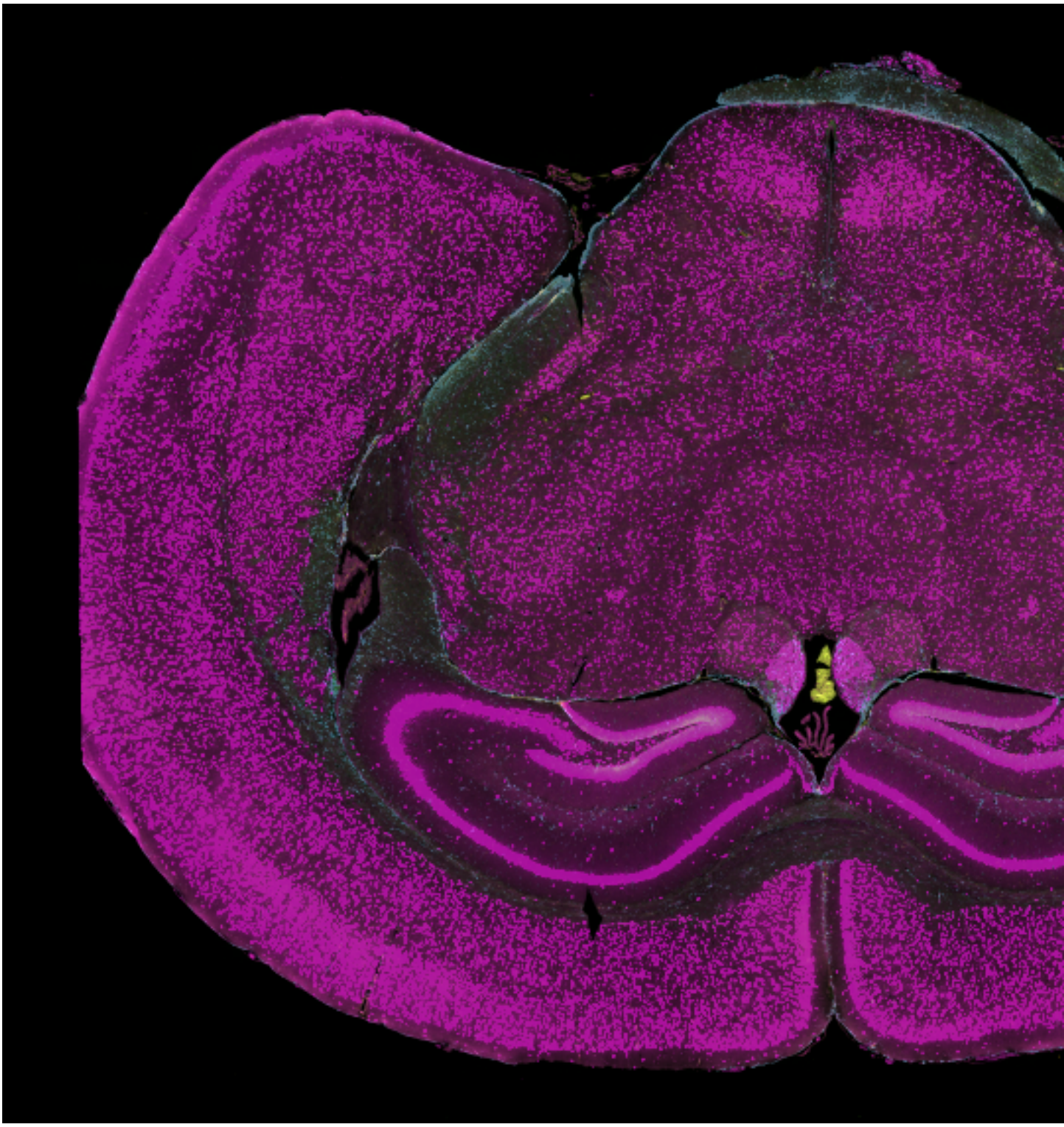
```
##         Dye DisplayName  Color WaveLength Target ExposureTime MinIntensity
## 1 Alexa 488        FITC   Blue      525nm   GFAP     200.0 µs           50
## 2   SYTO 83         Cy3  Green      568nm    DNA      50.0 µs            4
## 3 Alexa 594   Texas Red Yellow      615nm  Iba-1     300.0 µs           33
## 4 Alexa 647         Cy5    Red      666nm   NeuN     100.0 µs           88
##   MaxIntensity ColorCode
## 1        18509 #0000feff
## 2          805 #00fe00ff
## 3         3174 #fefe00ff
## 4        30000 #fe0000ff
```

```
chan4 <- changeImageColoring(overlay = chan4, color = "#32a8a4", dye = "FITC")
chan4 <- changeImageColoring(overlay = chan4, color = "magenta", dye = "Alexa 647")

chan4 <- changeColoringIntensity(overlay = chan4, minInten = 500,
                                 maxInten = 10000, dye = "Cy5")
```

```
fluor(chan4)
```

```
##         Dye DisplayName         Color WaveLength Target ExposureTime
## 1 Alexa 488        FITC Lightseagreen      525nm   GFAP     200.0 µs
## 2   SYTO 83         Cy3         Green      568nm    DNA      50.0 µs
## 3 Alexa 594   Texas Red        Yellow      615nm  Iba-1     300.0 µs
## 4 Alexa 647         Cy5       Magenta      666nm   NeuN     100.0 µs
##   MinIntensity MaxIntensity ColorCode
## 1           50        18509   #32a8a4
## 2            4          805 #00fe00ff
## 3           33         3174 #fefe00ff
## 4          500        10000   magenta
```

```
# change 4 channel TIFF to RGB
chan4 <- recolor(chan4)

showImage(chan4)
```



Future Directions

In future releases of SpatialOmicsOverlay, we will be

1. Ensuring capability with NanoString's CosMx Spatial Molecular Imager outputs
2. Adding ability to add graphing features on top of the image
3. Adding image analysis capabilities
4. Adding extraction of image data to use in ML/AI applications

```
sessionInfo()
```

```
## R version 4.1.3 (2022-03-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.4 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8          LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8           LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8       LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8          LC_NAME=en_US.UTF-8
##  [9] LC_ADDRESS=en_US.UTF-8        LC_TELEPHONE=en_US.UTF-8
## [11] LC_MEASUREMENT=en_US.UTF-8    LC_IDENTIFICATION=en_US.UTF-8
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] cowplot_1.1.1              SpatialOmicsOverlay_0.99.0
## [3] GeomxTools_2.1.7           NanoStringNCTools_1.3.1
## [5] ggplot2_3.3.5              S4Vectors_0.32.4
## [7] Biobase_2.54.0             BiocGenerics_0.40.0
##
## loaded via a namespace (and not attached):
##   [1] ggbeeswarm_0.6.0       minqa_1.2.4            colorspace_2.0-3
##   [4] rjson_0.2.21           ellipsis_0.3.2         EnvStats_2.7.0
##   [7] markdown_1.1           XVector_0.34.0         base64enc_0.1-3
##  [10] fftwtools_0.9-11       gridtext_0.1.4         ggtext_0.1.1
##  [13] rstudioapi_0.13        farver_2.1.0           bit64_4.0.5
##  [16] fansi_1.0.3            xml2_1.3.3             splines_4.1.3
##  [19] cachem_1.0.6           knitr_1.38             nloptr_2.0.0
##  [22] rJava_1.0-6            dbplyr_2.1.1           png_0.1-7
##  [25] pheatmap_1.0.12        compiler_4.1.3         httr_1.4.2
##  [28] assertthat_0.2.1       SeuratObject_4.0.4     Matrix_1.4-1
##  [31] fastmap_1.1.0          cli_3.2.0              htmltools_0.5.2
##  [34] tools_4.1.3            lmerTest_3.1-3         gtable_0.3.0
##  [37] glue_1.6.2             GenomeInfoDbData_1.2.7 reshape2_1.4.4
##  [40] dplyr_1.0.8            ggthemes_4.2.4         rappdirs_0.3.3
##  [43] Rcpp_1.0.8.3           scattermore_0.8        EBImage_4.36.0
##  [46] cellranger_1.1.0       vctrs_0.4.0            Biostrings_2.62.0
##  [49] nlme_3.1-157           xfun_0.30              stringr_1.4.0
##  [52] lme4_1.1-28            lifecycle_1.0.1        XML_3.99-0.9
##  [55] zlibbioc_1.40.0        MASS_7.3-55            scales_1.1.1
##  [58] parallel_4.1.3         RColorBrewer_1.1-3     yaml_2.3.5
```

```
##  [61] curl_4.3.2             gridExtra_2.3          memoise_2.0.1
##  [64] pbapply_1.5-0          reshape_0.8.8          stringi_1.7.6
##  [67] RSQLite_2.2.12         highr_0.9              plotrix_3.8-2
##  [70] tiff_0.1-11            filelock_1.0.2         boot_1.3-28
##  [73] GenomeInfoDb_1.30.1    rlang_1.0.2            pkgconfig_2.0.3
##  [76] systemfonts_1.0.4      bitops_1.0-7           ggiraph_0.8.2
##  [79] evaluate_0.15          lattice_0.20-45        purrr_0.3.4
##  [82] htmlwidgets_1.5.4      labeling_0.4.2         bit_4.0.4
##  [85] tidyselect_1.1.2       GGally_2.1.2           plyr_1.8.7
##  [88] magrittr_2.0.3         R6_2.5.1               IRanges_2.28.0
##  [91] magick_2.7.3           generics_0.1.2         DBI_1.1.2
##  [94] pillar_1.7.0           withr_2.5.0            abind_1.4-5
##  [97] RCurl_1.98-1.6         tibble_3.1.6           crayon_1.5.1
## [100] uuid_1.0-4             utf8_1.2.2             BiocFileCache_2.2.1
## [103] rmarkdown_2.13         viridis_0.6.2          jpeg_0.1-9
## [106] locfit_1.5-9.5         grid_4.1.3             readxl_1.4.0
## [109] data.table_1.14.2      blob_1.2.2             digest_0.6.29
## [112] numDeriv_2016.8-1.1    munsell_0.5.0          viridisLite_0.4.0
## [115] beeswarm_0.4.0         vipor_0.4.5
```