# Package 'spatialLIBD'

April 23, 2024

**Title** spatialLIBD: an R/Bioconductor package to visualize spatially-resolved transcriptomics data

**Version** 1.15.4

**Date** 2024-04-19

**Description** Inspect interactively the spatially-resolved transcriptomics data from the 10x Genomics Visium platform as well as data from the Maynard, Collado-Torres et al, Nature Neuroscience, 2021 project analyzed by Lieber Institute for Brain Development (LIBD) researchers and collaborators.

**License** Artistic-2.0

**Encoding** UTF-8

**LazyData** true

**Imports** shiny, golem, ggplot2, cowplot, plotly, viridisLite, shinyWidgets, sessioninfo, grid, grDevices, methods, AnnotationHub, utils, png, scater, DT, ExperimentHub, RColorBrewer, SummarizedExperiment, stats, graphics, S4Vectors, IRanges, fields, benchmarkme, SingleCellExperiment, BiocFileCache, jsonlite, tibble, rtracklayer, Matrix, BiocGenerics, GenomicRanges, magick, paletteer, scuttle, edgeR, limma, statmod, MatrixGenerics, rlang

**RoxygenNote** 7.3.1

**Roxygen** list(markdown = TRUE)

**URL** https://github.com/LieberInstitute/spatialLIBD

**BugReports** https://support.bioconductor.org/tag/spatialLIBD

**Suggests** knitr, RefManageR, rmarkdown, BiocStyle, testthat (>= 2.1.0), covr, here, BiocManager, lobstr, DropletUtils

**VignetteBuilder** knitr

**biocViews** Homo_sapiens_Data, ExperimentHub, SequencingData, SingleCellData, ExpressionData, Tissue, PackageTypeData, SpatialData

**Depends** SpatialExperiment (>= 1.3.3), R (>= 3.6)

**git_url** https://git.bioconductor.org/packages/spatialLIBD

**git_branch** devel

**git_last_commit** 77a5303

**git_last_commit_date** 2024-04-19

**Repository** Bioconductor 3.19

**Date/Publication** 2024-04-23

**Author** Leonardo Collado-Torres [aut, cre]
(<https://orcid.org/0000-0003-2140-308X>),
Kristen R. Maynard [ctb] (<https://orcid.org/0000-0003-0031-8468>),
Andrew E. Jaffe [ctb] (<https://orcid.org/0000-0001-6886-1454>),
Brenda Pardo [ctb] (<https://orcid.org/0000-0001-8103-7136>),
Abby Spangler [ctb] (<https://orcid.org/0000-0002-0028-9348>),
Jesús Vélez Santiago [ctb] (<https://orcid.org/0000-0001-5128-3838>),
Lukas M. Weber [ctb] (<https://orcid.org/0000-0002-3282-1730>),
Louise Huuki-Myers [ctb] (<https://orcid.org/0000-0001-5148-3602>),
Nicholas Eagles [ctb] (<https://orcid.org/0000-0002-9808-5254>)

**Maintainer** Leonardo Collado-Torres <lcolladotor@gmail.com>

# Contents

spatialLIBD-package . . . 3
add10xVisiumAnalysis . . . 4
add_images . . . 5
add_key . . . 6
annotate_registered_clusters . . . 7
check_modeling_results . . . 9
check_sce . . . 10
check_sce_layer . . . 11
check_spe . . . 11
cluster_export . . . 12
cluster_import . . . 13
enough_ram . . . 15
fetch_data . . . 15
frame_limits . . . 17
gene_set_enrichment . . . 19
gene_set_enrichment_plot . . . 20
geom_spatial . . . 23
get_colors . . . 24
img_edit . . . 25
img_update . . . 27
img_update_all . . . 28
layer_boxplot . . . 29
layer_matrix_plot . . . 32
layer_stat_cor . . . 33
layer_stat_cor_plot . . . 35
libd_layer_colors . . . 37

locate_images . . . 38
multi_gene_pca . . . 38
multi_gene_sparsity . . . 39
multi_gene_z_score . . . 40
read10xVisiumAnalysis . . . 40
read10xVisiumWrapper . . . 41
registration_block_cor . . . 43
registration_model . . . 44
registration_pseudobulk . . . 45
registration_stats_anova . . . 46
registration_stats_enrichment . . . 48
registration_stats_pairwise . . . 49
registration_wrapper . . . 51
run_app . . . 53
sce_to_spe . . . 57
sig_genes_extract . . . 58
sig_genes_extract_all . . . 60
sort_clusters . . . 61
tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer . . . 62
vis_clus . . . 62
vis_clus_p . . . 65
vis_gene . . . 67
vis_gene_p . . . 71
vis_grid_clus . . . 73
vis_grid_gene . . . 75

**Index** **78**

---

`spatialLIBD-package` *spatialLIBD: spatialLIBD: an R/Bioconductor package to visualize spatially-resolved transcriptomics data*

---

## Description

Inspect interactively the spatially-resolved transcriptomics data from the 10x Genomics Visium platform as well as data from the Maynard, Collado-Torres et al, Nature Neuroscience, 2021 project analyzed by Lieber Institute for Brain Development (LIBD) researchers and collaborators.

## Author(s)

**Maintainer**: Leonardo Collado-Torres <lcolladotor@gmail.com> (ORCID)

Other contributors:

- Kristen R. Maynard <Kristen.Maynard@libd.org> (ORCID) [contributor]
- Andrew E. Jaffe <andrew.jaffe@libd.org> (ORCID) [contributor]
- Brenda Pardo <bpardo@lcgej.unam.mx> (ORCID) [contributor]
- Abby Spangler <aspangle@gmail.com> (ORCID) [contributor]

- Jesús Vélez Santiago <jvelezmagic@gmail.com> (ORCID) [contributor]
- Lukas M. Weber <lukas.weber.edu@gmail.com> (ORCID) [contributor]
- Louise Huuki-Myers <lahuuki@gmail.com> (ORCID) [contributor]
- Nicholas Eagles <nickeagles77@gmail.com> (ORCID) [contributor]


## See Also

Useful links:

- https://github.com/LieberInstitute/spatialLIBD
- Report bugs at https://support.bioconductor.org/tag/spatialLIBD

---

`add10xVisiumAnalysis` *Add analysis data from a 10x Genomics Visium experiment to a SPE object*

---

## Description

This function adds to a SPE (SpatialExperiment-class) object the output from `read10xVisiumAnalysis()`.

## Usage

```
add10xVisiumAnalysis(spe, visium_analysis)
```

## Arguments

| | |
|---|---|
| `spe` | A SpatialExperiment-class object. |
| `visium_analysis` | The output from `read10xVisiumAnalysis()`. |

## Details

You might want to use `read10xVisiumWrapper()` instead of using this function directly.

## Value

A SpatialExperiment-class object with the clustering results from SpaceRanger added to `colData(spe)` and the dimension reduction results added to `reducedDims(spe)`. Added data starts with the `10x_` prefix to make them easy to differentiate.

## See Also

Other Utility functions for reading data from SpaceRanger output by 10x Genomics: `read10xVisiumAnalysis()`, `read10xVisiumWrapper()`

## Examples

```
## See 'Using spatialLIBD with 10x Genomics public datasets' for
## a full example using this function.
if (interactive()) {
    browseVignettes(package = "spatialLIBD")
}

## Note that ?SpatialExperiment::read10xVisium doesn't include all the files
## we need to illustrate read10xVisiumWrapper().
```

---

# add_images *Add non-standard images with the same dimensions as current ones*

---

## Description

This function re-uses the `SpatialExperiment::scaleFactors()` from current images when adding new images. This is useful if you take for example a multi-channel VisiumIF image and break into several single-channel images that all have the same dimensions. So you could have a set of images such as `channel_01_lowres` and `channel_02_lowres` that have the same dimensions and viewing area as the `lowres` image produced by SpaceRanger, each with only one channel. Similarly, you might have done some image manipulation for a given image and generated one or more images with the same dimensions as existing images.

## Usage

```
add_images(
  spe,
  image_dir,
  image_pattern,
  image_id_current = "lowres",
  image_id = image_pattern,
  image_paths = locate_images(spe, image_dir, image_pattern)
)
```

## Arguments

| | |
|---|---|
| `spe` | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| `image_dir` | A `character(1)` specifying a path to a directory containing image files with the pattern `sampleID_pattern.png`. |
| `image_pattern` | A `character(1)` specifying the pattern for the image files. |
| `image_id_current` | A `character(1)` specifying the name of the current existing image in `spe` that has the same scaling factor that to be used with the additional images. |

| | |
|---|---|
| image_id | A character(1) specifying the name to use in the new images. It cannot be the same as one used for existing images in spe for a given sample. It equals image_pattern by default. |
| image_paths | A named character() vector with the paths to the images. The names have to match the spe$sample_id and cannot be repeated. By default locate_images() is used but you can alternatively specify image_paths and ignore image_dir and image_pattern. |

## Value

A SpatialExperiment-class object with the additional image data in imgData(spe).

## See Also

Other Functions for adding non-standard images: locate_images()

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Add an image
    SpatialExperiment::imgData(add_images(
        spe,
        image_id_current = "lowres",
        image_id = "lowres_aws",
     image_paths = c("151507" = "https://spatial-dlpfc.s3.us-east-2.amazonaws.com/images/151507_tissue_lowres_i
    ))
}
```

---

# add_key *Create a unique spot identifier*

---

## Description

This function adds spe$key to a SpatialExperiment-class object which is unique across all spots.

## Usage

```
add_key(spe, overwrite = TRUE)
```

## Arguments

| | |
|---|---|
| spe | A SpatialExperiment-class object. |
| overwrite | A logical(1) indicating whether to overwrite the spe$key. |

## Value

A SpatialExperiment-class object with `key` added to the `colData(spe)` that is unique across all spots.

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## This object already has a 'key'
    head(spe$key)

    ## We can clean it
    spe$key <- NULL

    ## and then add it back
    head(add_key(spe)$key)

    ## Note that the original 'key' order was 'sample_id'_'barcode' and we'
    ## have since changed it to 'barcode'_'sample_id'.
}
```

---

`annotate_registered_clusters` *Annotated spatially-registered clusters*

---

## Description

Once you have computed the enrichment t-statistics for your sc/snRNA-seq data using `registration_wrapper()` and related functions, you can then use `layer_stat_cor()` and `layer_stat_cor_plot()` to perform the spatial registartion of your sc/snRNA-seq data. This function helps interpret that matrix and assign layer labels to your clusters.

## Usage

```
annotate_registered_clusters(
  cor_stats_layer,
  confidence_threshold = 0.25,
  cutoff_merge_ratio = 0.25
)
```

## Arguments

`cor_stats_layer` The output of `layer_stat_cor()`.

confidence_threshold
: A numeric(1) specifying the minimum correlation that a given cluster must have against any of the layers (by default) to be considered as having a 'good' assignment. Otherwise, the confidence will be 'poor' and the final label will have an asterisk.

cutoff_merge_ratio
: A numeric(1) specifying the threshold for merging or not layer assignments (by default). This is a proportion of the difference between the current correlation and the next highest given the units of the next highest correlation. Defaults to a difference of 25% of the next highest correlation: if the observed difference is lower than this threshold, then we keep merging. Higher values will lead to more layers (by default) being merged.

## Details

If you change the input modeling_results to layer_stat_cor() then the interpretation of this function could change. For example, maybe you have your own spatially-resolved transcriptomics data that doesn't have to be about DLPFC layers.

## Value

A data.frame with 3 columns. Your clusters, the layer_confidence which depends on confidence_threshold, and the layer_label.

## See Also

Other Layer correlation functions: layer_stat_cor(), layer_stat_cor_plot()

## Examples

```
## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}

## Compute the correlations
cor_stats_layer <- layer_stat_cor(
    tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer,
    modeling_results,
    model_type = "enrichment"
)

## Obtain labels
annotate_registered_clusters(cor_stats_layer)

## More relaxed merging threshold
annotate_registered_clusters(cor_stats_layer, cutoff_merge_ratio = 1)
```

check_modeling_results *Check input modeling_results*

## Description

This function checks that the `modeling_results` object has the appropriate structure. For more details please check the vignette documentation.

## Usage

```
check_modeling_results(modeling_results)
```

## Arguments

| | |
|---|---|
| `modeling_results` | Defaults to the output of `fetch_data(type = 'modeling_results')`. This is a list of tables with the columns `f_stat_*` or `t_stat_*` as well as `p_value_*` and `fdr_*` plus `ensembl`. The column name is used to extract the statistic results, the p-values, and the FDR adjusted p-values. Then the `ensembl` column is used for matching in some cases. See `fetch_data()` for more details. |

## Value

The input object if all checks are passed.

## See Also

Other Check input functions: `check_sce()`, `check_sce_layer()`, `check_spe()`

## Examples

```
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}

## Check the object
xx <- check_modeling_results(modeling_results)
```

check_sce *Check input sce*

## Description

This function checks that the sce object has the appropriate structure. This is a legacy function and we highly encourage you to use SpatialExperiment-class objects and check them with check_spe().

## Usage

```
check_sce(
  sce,
  variables = c("GraphBased", "ManualAnnotation", "Maynard", "Martinowich",
   paste0("SNN_k50_k", 4:28), "spatialLIBD", "cell_count", "sum_umi", "sum_gene",
   "expr_chrM", "expr_chrM_ratio", "SpatialDE_PCA", "SpatialDE_pool_PCA", "HVG_PCA",
   "pseudobulk_PCA", "markers_PCA", "SpatialDE_UMAP", "SpatialDE_pool_UMAP", "HVG_UMAP",
    "pseudobulk_UMAP", "markers_UMAP", "SpatialDE_PCA_spatial",
    "SpatialDE_pool_PCA_spatial", "HVG_PCA_spatial", "pseudobulk_PCA_spatial",
   "markers_PCA_spatial", "SpatialDE_UMAP_spatial", "SpatialDE_pool_UMAP_spatial",

    "HVG_UMAP_spatial", "pseudobulk_UMAP_spatial", "markers_UMAP_spatial")
)
```

## Arguments

| | |
|---|---|
| sce | Defaults to the output of fetch_data(type = 'sce'). This is a SingleCellExperiment object with the spot-level Visium data and information required for visualizing the histology. See fetch_data() for more details. |
| variables | A character() vector of variable names expected to be present in colData(sce). |

## Value

The input object if all checks are passed.

## See Also

Other Check input functions: check_modeling_results(), check_sce_layer(), check_spe()

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("sce_example")) sce_example <- fetch_data("sce_example")

    ## Check the object
    check_sce(sce_example)
}
```

## check_sce_layer *Check input sce_layer*

### Description

This function checks that the `sce_layer` object has the appropriate structure. For more details please check the vignette documentation.

### Usage

```
check_sce_layer(sce_layer, variables = "spatialLIBD")
```

### Arguments

| | |
|---|---|
| `sce_layer` | Defaults to the output of `fetch_data(type = 'sce_layer')`. This is a SingleCellExperiment object with the spot-level Visium data compressed via pseudo-bulking to the layer-level (group-level) resolution. See `fetch_data()` for more details. |
| `variables` | A `character()` vector of variable names expected to be present in `colData(sce_layer)`. |

### Value

The input object if all checks are passed.

### See Also

Other Check input functions: `check_modeling_results()`, `check_sce()`, `check_spe()`

### Examples

```
## Obtain the necessary data
if (!exists("sce_layer")) sce_layer <- fetch_data("sce_layer")

## Check the object
check_sce_layer(sce_layer)
```

## check_spe *Check input spe*

### Description

This function checks that the spe object has the appropriate structure. For more details please check the vignette documentation.

### Usage

```
check_spe(
  spe,
  variables = c("sum_umi", "sum_gene", "expr_chrM", "expr_chrM_ratio")
)
```

### Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| variables | A `character()` vector of variable names expected to be present in `colData(spe)`. |

### Value

The input object if all checks are passed.

### Author(s)

Brenda Pardo, Leonardo Collado-Torres

### See Also

Other Check input functions: `check_modeling_results()`, `check_sce()`, `check_sce_layer()`

### Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Check the object
    check_spe(spe)
}
```

---

## cluster_export *Export a column with cluster results*

---

### Description

This function creates a `clusters.csv` file similar to the ones created by SpaceRanger at `outs/analysis/clustering` but with the `key` column that combines the `barcode` and the `sample_id`, which is needed when the `spe` object contains data from multiple samples given that the barcodes are duplicated.

## Usage

```
cluster_export(
  spe,
  cluster_var,
  cluster_dir = file.path(tempdir(), "exported_clusters"),
  overwrite = TRUE
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| cluster_var | A `character(1)` with the name of the variable you wish to export. |
| cluster_dir | A `character(1)` specifying the output directory, similar to the `outs/analysis/clustering` produced by SpaceRanger. |
| overwrite | A `logical(1)` indicating whether to overwrite the `spe$key`. |

## Value

The path to the exported `clusters.csv` file.

## See Also

Other cluster export/import utility functions: `cluster_import()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Export two cluster variables
    cluster_export(spe, "spatialLIBD")
    cluster_export(spe, "GraphBased")
}
```

---

# cluster_import *Import cluster results*

---

## Description

This function imports previously exported clustering results with `cluster_export()` and adds them to the `colData()` slot of your SpatialExperiment-class object.

## Usage

```
cluster_import(
  spe,
  cluster_dir = file.path(tempdir(), "exported_clusters"),
  prefix = "imported_",
  overwrite = TRUE
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| cluster_dir | A `character(1)` specifying the output directory, similar to the `outs/analysis/clustering` produced by SpaceRanger. |
| prefix | A `character(1)` specifying the prefix to use when naming these new cluster variables. |
| overwrite | A `logical(1)` indicating whether to overwrite the `spe$key`. |

## Value

A SpatialExperiment-class object with the imported clusters appended on the `colData()`.

## See Also

Other cluster export/import utility functions: `cluster_export()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Export two cluster variables
    cluster_export(spe, "spatialLIBD")
    cluster_export(spe, "GraphBased")

    ## Re-import them
    colData(cluster_import(spe))
}
```

enough_ram *Determine if you have enough RAM memory*

## Description

This function determines if you have enough RAM memory on your system.

## Usage

```
enough_ram(how_much = 4e+09)
```

## Arguments

| | |
|---|---|
| how_much | The number of bytes you want to compare against. |

## Details

If `benchmarkme::get_ram()` fails, this function will return `FALSE` as a save bet.

## Value

A `logical(1)` indicating whether your system has enough RAM memory.

## Examples

```
## Do you have ~ 4 GB in your system?
enough_ram(4e9)

## Do you have ~ 100 GB in your system
enough_ram(100e9)
```

fetch_data *Download the Human DLPFC Visium data from LIBD*

## Description

This function downloads from `ExperimentHub` Visium, Visium Spatial Proteogenomics (Visium-SPG), or single nucleus RNA-seq (snRNA-seq) data and results analyzed by LIBD from multiple projects. If `ExperimentHub` is not available, this function will download the files from Dropbox using `BiocFileCache::bfcrpath()` unless the files are present already at `destdir`. Note that `ExperimentHub` and `BiocFileCache` will cache the data and automatically detect if you have previously downloaded it, thus making it the preferred way to interact with the data.

## Usage

```
fetch_data(
  type = c("sce", "sce_layer", "modeling_results", "sce_example", "spe",
    "spatialDLPFC_Visium", "spatialDLPFC_Visium_example_subset",
    "spatialDLPFC_Visium_pseudobulk", "spatialDLPFC_Visium_modeling_results",
    "spatialDLPFC_Visium_SPG", "spatialDLPFC_snRNAseq",
   "Visium_SPG_AD_Visium_wholegenome_spe", "Visium_SPG_AD_Visium_targeted_spe",
    "Visium_SPG_AD_Visium_wholegenome_pseudobulk_spe",
    "Visium_SPG_AD_Visium_wholegenome_modeling_results"),
  destdir = tempdir(),
  eh = ExperimentHub::ExperimentHub(),
  bfc = BiocFileCache::BiocFileCache()
)
```

## Arguments

| | |
|---|---|
| type | A character(1) specifying which file you want to download. It can either be: sce for the SingleCellExperiment object containing the spot-level data that includes the information for visualizing the clusters/genes on top of the Visium histology, sce_layer for the SingleCellExperiment object containing the layer-level data (pseudo-bulked from the spot-level), or modeling_results for the list of tables with the enrichment, pairwise, and anova model results from the layer-level data. It can also be sce_example which is a reduced version of sce just for example purposes. The initial version of spatialLIBD downloaded data only from https://github.com/LieberInstitute/HumanPilot. As of BioC version 3.13 spe downloads a SpatialExperiment-class object. As of version 1.11.6, this function also allows downloading data from the http://research.libd.org/spatialDLPFC/ project. As of version 1.11.12, data from https://github.com/LieberInstitute/Visium_SPG_AD can be downloaded. |
| destdir | The destination directory to where files will be downloaded to in case the ExperimentHub resource is not available. If you already downloaded the files, you can set this to the current path where the files were previously downloaded to avoid re-downloading them. |
| eh | An ExperimentHub object ExperimentHub-class. |
| bfc | A BiocFileCache object BiocFileCache-class. Used when eh is not available. |

## Details

The data was initially prepared by scripts at https://github.com/LieberInstitute/HumanPilot and further refined by https://github.com/LieberInstitute/spatialLIBD/blob/master/inst/scripts/make-data_spatialLIBD.R.

## Value

The requested object: sce, sce_layer, ve or modeling_results that you have to assign to an object. If you didn't you can still avoid re-loading the object by using .Last.value.

## Examples

```
## Download the SingleCellExperiment object
## at the layer-level
if (!exists("sce_layer")) sce_layer <- fetch_data("sce_layer")

## Explore the data
sce_layer

## How to download and load "spatialDLPFC_snRNAseq"
## Not run:
sce_path_zip <- fetch_data("spatialDLPFC_snRNAseq")
sce_path <- unzip(sce_path_zip, exdir = tempdir())
sce <- HDF5Array::loadHDF5SummarizedExperiment(
    file.path(tempdir(), "sce_DLPFC_annotated")
)
sce
#> class: SingleCellExperiment
#> dim: 36601 77604
#> metadata(3): Samples cell_type_colors cell_type_colors_broad
#> assays(2): counts logcounts
#> rownames(36601): MIR1302-2HG FAM138A ... AC007325.4 AC007325.2
#> rowData names(7): source type ... gene_type binomial_deviance
#> colnames(77604): 1_AAACCCAAGTTCTCTT-1 1_AAACCCACAAGGTCTT-1 ... 19_TTTGTTGTCTCATTGT-1 19_TTTGTTGTCTTAAGGC-1
#> colData names(32): Sample Barcode ... cellType_layer layer_annotation
#> reducedDimNames(4): GLMPCA_approx TSNE UMAP HARMONY
#> mainExpName: NULL
#> altExpNames(0):
lobstr::obj_size(sce)
#> 172.28 MB

## End(Not run)
```

---

`frame_limits` *Identify the image limits*

---

## Description

This function is useful for automatically cropping the images. It finds the edge points (min and max on both the X and Y axis) in pixels based on a particular image. This function takes advantage of the known design of Visium slides as documented at https://support.10xgenomics.com/spatial-gene-expression/software/pipelines/latest/output/spatial and https://kb.10xgenomics.com/hc/en-us/articles/360041426992. That is, that for a regular Visium slide, the array row has a range from 0 to 77, the array col from 0 to 127, the capture area has a 6.5 mm edge length, the the fiducial frame area has an edge of 8 mm, and spot centers are about 100 um from each other.

## Usage

```
frame_limits(
```

```
  spe,
  sampleid,
  image_id = "lowres",
  visium_grid = list(row_min = 0, row_max = 77, col_min = 0, col_max = 127,
    fiducial_vs_capture_edge = (8 - 6.5) * 1000/2/100)
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| sampleid | A `character(1)` specifying which sample to plot from `colData(spe)$sample_id` (formerly `colData(spe)$sample_name`). |
| image_id | A `character(1)` with the name of the image ID you want to use in the background. |
| visium_grid | A named list with the parameters known about the Visium grid. This can change for Visium HD vs regular Visium for example. |

## Value

A named list with `y_min`, `y_max`, `x_min`, and `x_max` pixels from the selected image that can be used for cropping the image.

## Author(s)

Louise Huuki-Myers and Leonardo Collado-Torres

## See Also

Other Spatial cluster visualization functions: `vis_clus()`, `vis_clus_p()`, `vis_grid_clus()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Obtain the frame limits for one sample
    frame_limits(spe, sampleid = "151673")
}
```

gene_set_enrichment *Evaluate the enrichment for a list of gene sets*

## Description

Using the layer-level (group-level) data, this function evaluates whether list of gene sets (Ensembl gene IDs) are enriched among the significant genes (FDR < 0.1 by default) genes for a given model type result. Test the alternative hypothesis that OR > 1, i.e. that gene set is over-represented in the set of enriched genes. If you want to check depleted genes, change `reverse` to `TRUE`.

## Usage

```
gene_set_enrichment(
  gene_list,
  fdr_cut = 0.1,
  modeling_results = fetch_data(type = "modeling_results"),
  model_type = names(modeling_results)[1],
  reverse = FALSE
)
```

## Arguments

| | |
|---|---|
| `gene_list` | A named `list` object (could be a `data.frame`) where each element of the list is a character vector of Ensembl gene IDs. |
| `fdr_cut` | A `numeric(1)` specifying the FDR cutoff to use for determining significance among the modeling results genes. |
| `modeling_results` | Defaults to the output of `fetch_data(type = 'modeling_results')`. This is a list of tables with the columns `f_stat_*` or `t_stat_*` as well as `p_value_*` and `fdr_*` plus `ensembl`. The column name is used to extract the statistic results, the p-values, and the FDR adjusted p-values. Then the `ensembl` column is used for matching in some cases. See `fetch_data()` for more details. |
| `model_type` | A named element of the `modeling_results` list. By default that is either `enrichment` for the model that tests one human brain layer against the rest (one group vs the rest), `pairwise` which compares two layers (groups) denoted by `layerA-layerB` such that `layerA` is greater than `layerB`, and `anova` which determines if any layer (group) is different from the rest adjusting for the mean expression level. The statistics for `enrichment` and `pairwise` are t-statistics while the `anova` model ones are F-statistics. |
| `reverse` | A `logical(1)` indicating whether to multiply by `-1` the input statistics and reverse the `layerA-layerB` column names (using the `-`) into `layerB-layerA`. |

## Details

Check https://github.com/LieberInstitute/HumanPilot/blob/master/Analysis/Layer_Guesses/check_clinical_gene_sets.R to see a full script from where this family of functions is derived from.

## Value

A table in long format with the enrichment results using `stats::fisher.test()`.

## Author(s)

Andrew E Jaffe, Leonardo Collado-Torres

## See Also

Other Gene set enrichment functions: `gene_set_enrichment_plot()`

## Examples

```
## Read in the SFARI gene sets included in the package
asd_sfari <- utils::read.csv(
    system.file(
        "extdata",
        "SFARI-Gene_genes_01-03-2020release_02-04-2020export.csv",
        package = "spatialLIBD"
    ),
    as.is = TRUE
)

## Format them appropriately
asd_sfari_geneList <- list(
    Gene_SFARI_all = asd_sfari$ensembl.id,
    Gene_SFARI_high = asd_sfari$ensembl.id[asd_sfari$gene.score < 3],
    Gene_SFARI_syndromic = asd_sfari$ensembl.id[asd_sfari$syndromic == 1]
)

## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}

## Compute the gene set enrichment results
asd_sfari_enrichment <- gene_set_enrichment(
    gene_list = asd_sfari_geneList,
    modeling_results = modeling_results,
    model_type = "enrichment"
)

## Explore the results
asd_sfari_enrichment
```

---

`gene_set_enrichment_plot`

*Plot the gene set enrichment results*

---

## Description

This function takes the output of gene_set_enrichment() and creates a heatmap visualization of the results.

## Usage

```
gene_set_enrichment_plot(
  enrichment,
  xlabs = unique(enrichment$ID),
  PThresh = 12,
  ORcut = 3,
  enrichOnly = FALSE,
  layerHeights = c(0, seq_len(length(unique(enrichment$test)))) * 15,
  mypal = c("white", (grDevices::colorRampPalette(RColorBrewer::brewer.pal(9,
    "YlOrRd")))(50)),
  cex = 1.2
)
```

## Arguments

| | |
|---|---|
| enrichment | The output of gene_set_enrichment(). |
| xlabs | A vector of names in the same order and length as unique(enrichment$ID). Gets passed to layer_matrix_plot(). |
| PThresh | A numeric(1) specifying the P-value threshold for the maximum value in the -log10(p) scale. |
| ORcut | A numeric(1) specifying the P-value threshold for the minimum value in the -log10(p) scale for printing the odds ratio values in the cells of the resulting plot. |
| enrichOnly | A logical(1) indicating whether to show only odds ratio values greater than 1. |
| layerHeights | A numeric() vector of length equal to length(unique(enrichment$test)) + 1 that starts at 0 specifying where to plot the y-axis breaks which can be used for re-creating the length of each brain layer. Gets passed to layer_matrix_plot(). |
| mypal | A vector with the color palette to use. Gets passed to layer_matrix_plot(). |
| cex | Passed to layer_matrix_plot(). |

## Details

Check https://github.com/LieberInstitute/HumanPilot/blob/master/Analysis/Layer_Guesses/check_clinical_gene_sets.R to see a full script from where this family of functions is derived from.

## Value

A plot visualizing the gene set enrichment odds ratio and p-value results.

## Author(s)

Andrew E Jaffe, Leonardo Collado-Torres

## See Also

layer_matrix_plot

Other Gene set enrichment functions: gene_set_enrichment()

## Examples

```
## Read in the SFARI gene sets included in the package
asd_sfari <- utils::read.csv(
    system.file(
        "extdata",
        "SFARI-Gene_genes_01-03-2020release_02-04-2020export.csv",
        package = "spatialLIBD"
    ),
    as.is = TRUE
)

## Format them appropriately
asd_sfari_geneList <- list(
    Gene_SFARI_all = asd_sfari$ensembl.id,
    Gene_SFARI_high = asd_sfari$ensembl.id[asd_sfari$gene.score < 3],
    Gene_SFARI_syndromic = asd_sfari$ensembl.id[asd_sfari$syndromic == 1]
)

## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}

## Compute the gene set enrichment results
asd_sfari_enrichment <- gene_set_enrichment(
    gene_list = asd_sfari_geneList,
    modeling_results = modeling_results,
    model_type = "enrichment"
)

## Visualize the gene set enrichment results
## with a custom color palette
gene_set_enrichment_plot(
    asd_sfari_enrichment,
    xlabs = gsub(".*_", "", unique(asd_sfari_enrichment$ID)),
    mypal = c(
        "white",
        grDevices::colorRampPalette(
            RColorBrewer::brewer.pal(9, "BuGn")
        )(50)
    )
)

## Specify the layer heights so it resembles more the length of each
## layer in the brain
gene_set_enrichment_plot(
    asd_sfari_enrichment,
```

```
    xlabs = gsub(".*_", "", unique(asd_sfari_enrichment$ID)),
    layerHeights = c(0, 40, 55, 75, 85, 110, 120, 135),
)
```

---

`geom_spatial` *A ggplot2 layer for visualizing the Visium histology*

---

## Description

This function defines a `ggplot2::layer()` for visualizing the histology image from Visium. It can be combined with other ggplot2 functions for visualizing the clusters as in `vis_clus_p()` or gene-level information as in `vis_gene_p()`.

## Usage

```
geom_spatial(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `mapping` | Passed to `ggplot2::layer(mapping)` where `grob`, `x` and `y` are required. |
| `data` | Passed to `ggplot2::layer(data)`. |
| `stat` | Passed to `ggplot2::layer(stat)`. |
| `position` | Passed to `ggplot2::layer(position)`. |
| `na.rm` | Passed to `ggplot2::layer(params = list(na.rm))`. |
| `show.legend` | Passed to `ggplot2::layer(show.legend)`. |
| `inherit.aes` | Passed to `ggplot2::layer(inherit.aes)`. |
| `...` | Other arguments passed to `ggplot2::layer(params = list(...))`. |

## Value

A `ggplot2::layer()` for the histology information.

## Author(s)

10x Genomics

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Select the first sample and extract the data
    sample_id <- unique(spe$sample_id)[1]
    spe_sub <- spe[, spe$sample_id == sample_id]
    sample_df <- as.data.frame(colData(spe_sub), optional = TRUE)

    ## Obtain the histology image
    img <- SpatialExperiment::imgRaster(spe_sub)

    ## Transform to a rasterGrob object
   grob <- grid::rasterGrob(img, width = grid::unit(1, "npc"), height = grid::unit(1, "npc"))

    ## Make a plot using geom_spatial
    p <- ggplot2::ggplot(
        sample_df,
        ggplot2::aes(
            x = pxl_col_in_fullres * SpatialExperiment::scaleFactors(spe_sub),
            y = pxl_row_in_fullres * SpatialExperiment::scaleFactors(spe_sub),
        )
    ) +
        geom_spatial(
            data = tibble::tibble(grob = list(grob)),
            ggplot2::aes(grob = grob),
            x = 0.5,
            y = 0.5
        )

    ## Show the plot
    print(p)

    ## Clean up
    rm(spe_sub)
}
```

---

`get_colors` *Obtain the colors for a set of cluster names*

---

## Description

This function returns a vector of colors based on a vector of cluster names. It can be used to automatically assign colors.

## Usage

```
get_colors(colors = NULL, clusters)
```

### Arguments

| | |
|---|---|
| colors | A vector of colors. If NULL then a set of default colors will be used when clusters has less than 12 unique values, otherwise Polychrome::palette36 will be used which can generate up to 36 unique colors. If the number of unique clusters is beyond 36 then this function will fail. |
| clusters | A vector of cluster names. |

### Value

A named vector where the values are the colors to use for displaying them different clusters. For some use cases, you might have to either change the names or use unname().

### Examples

```
## Obtain the necessary data
if (!exists("sce_layer")) sce_layer <- fetch_data("sce_layer")

## Example layer colors with the corresponding names
get_colors(libd_layer_colors, sce_layer$layer_guess)
get_colors(libd_layer_colors, sce_layer$layer_guess_reordered_short)

## Example where colors are assigned automatically
## based on a pre-defined set of colors
get_colors(clusters = sce_layer$kmeans_k7)

## Example where Polychrome::palette36.colors() gets used
get_colors(clusters = letters[seq_len(13)])
```

---

## img_edit *Edit a background image*

---

### Description

This function uses the magick package to edit the color and perform other image manipulations on a background image. It can be useful if you want to highlight certain features of these images.

### Usage

```
img_edit(
  spe,
  sampleid,
  image_id = "lowres",
  channel = NA,
  brightness = 100,
  saturation = 100,
  hue = 100,
  enhance = FALSE,
  contrast_sharpen = NA,
```

```
  quantize_max = NA,
  quantize_dither = TRUE,
  equalize = FALSE,
  normalize = FALSE,
  transparent_color = NA,
  transparent_fuzz = 0,
  background_color = NA,
  median_radius = NA,
  negate = FALSE
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| sampleid | A `character(1)` specifying which sample to plot from `colData(spe)$sample_id` (formerly `colData(spe)$sample_name`). |
| image_id | A `character(1)` with the name of the image ID you want to use in the background. |
| channel | A `character(1)` passed to magick::image_channel. If `NA` this step is skipped. |
| brightness | A `numeric(1)` passed to magick::image_modulate. |
| saturation | A `numeric(1)` passed to magick::image_modulate. |
| hue | A `numeric(1)` passed to magick::image_modulate. |
| enhance | A `logical(1)` controlling whether to use magick::enhance. |
| contrast_sharpen | A `numeric(1)` passed to magick::image_contrast. If `NA` this step is skipped. |
| quantize_max | A `numeric(1)` passed to magick::image_quantize. If `NA` this step is skipped. |
| quantize_dither | A `logical(1)` passed to magick::image_quantize. |
| equalize | A `logical(1)` controlling whether to use magick::equalize. |
| normalize | A `logical(1)` controlling whether to use magick::normalize. |
| transparent_color | A `character(1)` passed to magick::image_transparent. If `NA` this step is skipped. |
| transparent_fuzz | A `numeric(1)` passed to magick::image_transparent. |
| background_color | A `character(1)` passed to magick::image_background. If `NA` this step is skipped. |
| median_radius | A `numeric(1)` passed to magick::image_median. If `NA` this step is skipped. |
| negate | A `logical(1)` controlling whether to use magick::negate. |

## Details

The `magick` functions are used in the sequence represented by the arguments to this function. You can alternatively use this function sequentially. Or directly use the `magick` package.

## Value

A magick image object such as the one returned by magick::image_read.

## See Also

Other Image editing functions: img_update(), img_update_all()

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Reduce brightness to 25%
    x <- img_edit(spe, sampleid = "151507", brightness = 25)
    plot(x)
}
```

---

# img_update *Update the image for one sample*

---

## Description

Edit the image with img_edit() then update the imgData().

## Usage

```
img_update(
  spe,
  sampleid,
  image_id = "lowres",
  new_image_id = paste0("edited_", image_id),
  overwrite = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of fetch_data(type = 'spe'). This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See fetch_data() for more details. |
| sampleid | A character(1) specifying which sample to plot from colData(spe)$sample_id (formerly colData(spe)$sample_name). |
| image_id | A character(1) with the name of the image ID you want to use in the background. |
| new_image_id | A character(1) specifying the new image_id to use. |
| overwrite | A logical(1) specifying whether to overwrite the image_id if it already exists. |
| ... | Parameters passed to img_edit(). |

## Value

A SpatialExperiment-class object with an updated `imgData()` slot.

## See Also

Other Image editing functions: `img_edit()`, `img_update_all()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Reduce brightness to 25% and update the imgData()
    imgData(img_update(spe, sampleid = "151507", brightness = 25))
}
```

---

# `img_update_all` *Update the images for all samples*

---

## Description

This function uses `img_update()` for all samples. That is, it loops through every sample and edits the image with `img_edit()` and then updates the `imgData()`.

## Usage

```
img_update_all(
  spe,
  image_id = "lowres",
  new_image_id = paste0("edited_", image_id),
  overwrite = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `spe` | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| `image_id` | A `character(1)` with the name of the image ID you want to use in the background. |
| `new_image_id` | A `character(1)` specifying the new `image_id` to use. |
| `overwrite` | A `logical(1)` specifying whether to overwrite the `image_id` if it already exists. |
| `...` | Parameters passed to `img_edit()`. |

## Value

A SpatialExperiment-class object with an updated `imgData()` slot.

## See Also

Other Image editing functions: `img_edit()`, `img_update()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Reduce brightness to 25% for the 'lowres' image for all samples and
    ## update the imgData()
    imgData(img_update_all(spe, brightness = 25))
}
```

---

`layer_boxplot` *Layer-level (group-level) boxplots*

---

## Description

This function uses the output of `sig_genes_extract_all()` as well as the logcounts from the layer-level (group-level) data to visualize the expression of a given gene and display the modeling results for the given gene.

## Usage

```
layer_boxplot(
  i = 1,
  sig_genes = sig_genes_extract(),
  short_title = TRUE,
  sce_layer = fetch_data(type = "sce_layer"),
  col_bkg_box = "grey80",
  col_bkg_point = "grey40",
  col_low_box = "violet",
  col_low_point = "darkviolet",
  col_high_box = "skyblue",
  col_high_point = "dodgerblue4",
  cex = 2,
  group_var = "layer_guess_reordered_short",
  assayname = "logcounts"
)
```

## Arguments

| | |
|---|---|
| i | A integer(1) indicating which row of sig_genes do you want to plot. |
| sig_genes | The output of sig_genes_extract_all(). |
| short_title | A logical(1) indicating whether to print a short title or not. |
| sce_layer | Defaults to the output of fetch_data(type = 'sce_layer'). This is a SingleCellExperiment object with the spot-level Visium data compressed via pseudo-bulking to the layer-level (group-level) resolution. See fetch_data() for more details. |
| col_bkg_box | Box background color for layers not used when visualizing the pairwise model results. |
| col_bkg_point | Similar to col_bkg_box but for the points. |
| col_low_box | Box background color for layer(s) with the expected lower expression based on the actual test for row i of sig_genes. |
| col_low_point | Similar to col_low_box but for the points. |
| col_high_box | Similar to col_low_box but for the expected layer(s) with higher expression. |
| col_high_point | Similar to col_high_box but for the points. |
| cex | Controls the size of the text, points and axis legends. |
| group_var | A character(1) specifying a colData(sce_layer) column name to use for the x-axis. |
| assayname | A character(1) specifying the default assay to use from assays(sce_layer). |

## Value

This function creates a boxplot of the layer-level data (group-level) separated by layer and colored based on the model type from row i of sig_genes.

## References

Adapted from https://github.com/LieberInstitute/HumanPilot/blob/master/Analysis/Layer_Guesses/layer_specificity.R

## See Also

Other Layer modeling functions: sig_genes_extract(), sig_genes_extract_all()

## Examples

```
## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}
if (!exists("sce_layer")) sce_layer <- fetch_data(type = "sce_layer")

## Top 2 genes from the enrichment model
sig_genes <- sig_genes_extract_all(
    n = 2,
    modeling_results = modeling_results,
```

```
    sce_layer = sce_layer
)

## Example default boxplot
set.seed(20200206)
layer_boxplot(sig_genes = sig_genes, sce_layer = sce_layer)

## Now show the long title version
set.seed(20200206)
layer_boxplot(
    sig_genes = sig_genes,
    short_title = FALSE,
    sce_layer = sce_layer
)

set.seed(20200206)
layer_boxplot(
    i = which(sig_genes$model_type == "anova")[1],
    sig_genes = sig_genes,
    sce_layer = sce_layer
)

set.seed(20200206)
layer_boxplot(
    i = which(sig_genes$model_type == "pairwise")[1],
    sig_genes = sig_genes,
    sce_layer = sce_layer
)

## Viridis colors displayed in the shiny app
library("viridisLite")
set.seed(20200206)
layer_boxplot(
    sig_genes = sig_genes,
    sce_layer = sce_layer,
    col_low_box = viridis(4)[2],
    col_low_point = viridis(4)[1],
    col_high_box = viridis(4)[3],
    col_high_point = viridis(4)[4]
)

## Paper colors displayed in the shiny app
set.seed(20200206)
layer_boxplot(
    sig_genes = sig_genes,
    sce_layer = sce_layer,
    col_low_box = "palegreen3",
    col_low_point = "springgreen2",
    col_high_box = "darkorange2",
    col_high_point = "orange1"
)

## Blue/red colors displayed in the shiny app
```

```
set.seed(20200206)
layer_boxplot(
    i = which(sig_genes$model_type == "pairwise")[1],
    sig_genes = sig_genes,
    sce_layer = sce_layer,
    col_bkg_box = "grey90",
    col_bkg_point = "grey60",
    col_low_box = "skyblue2",
    col_low_point = "royalblue3",
    col_high_box = "tomato2",
    col_high_point = "firebrick4",
    cex = 3
)
```

---

`layer_matrix_plot` *Visualize a matrix of values across human brain layers*

---

## Description

This function visualizes a numerical matrix where the Y-axis represents the human brain layers and can be adjusted to represent the length of each brain layer. Cells can optionally have text values. This function is used by `gene_set_enrichment_plot()` and `layer_stat_cor_plot()`.

## Usage

```
layer_matrix_plot(
  matrix_values,
  matrix_labels = NULL,
  xlabs = NULL,
  layerHeights = NULL,
  mypal = c("white", (grDevices::colorRampPalette(RColorBrewer::brewer.pal(9,
    "YlOrRd")))(50)),
  breaks = NULL,
  axis.args = NULL,
  srt = 45,
  mar = c(8, 4 + (max(nchar(rownames(matrix_values)))%/%3) * 0.5, 4, 2) + 0.1,
  cex = 1.2
)
```

## Arguments

| | |
|---|---|
| `matrix_values` | A `matrix()` with one column per set of interest and one row per layer (group) with numeric values. |
| `matrix_labels` | Optionally a character `matrix()` with the same dimensions and `dimnames()` as `matrix_values` with text labels for the cells. |
| `xlabs` | A vector of names in the same order and length as `colnames(matrix_values)`. |

| | |
|---|---|
| layerHeights | A numeric() vector of length equal to nrow(matrix_values) + 1 that starts at 0 specifying where to plot the y-axis breaks which can be used for re-creating the length of each brain layer. |
| mypal | A vector with the color palette to use. |
| breaks | Passed to fields::image.plot(). Used by layer_stat_cor_plot(). |
| axis.args | Passed to fields::image.plot(). Used by layer_stat_cor_plot(). |
| srt | The angle for the x-axis labels. Used by layer_stat_cor_plot(). |
| mar | Passed to graphics::par(). |
| cex | Used for the x-axis labels and the text inside the cells. |

## Value

A base R plot visualizing the input matrix_values with optional text labels for matrix_labels.

## Author(s)

Andrew E Jaffe, Leonardo Collado-Torres

## Examples

```
## Create some random data
set.seed(20200224)
mat <- matrix(runif(7 * 8, min = -1), nrow = 7)
rownames(mat) <- c("WM", paste0("L", rev(seq_len(6))))
colnames(mat) <- paste0("Var", seq_len(8))

## Create some text labels
mat_text <- matrix("", nrow = 7, ncol = 8, dimnames = dimnames(mat))
diag(mat_text) <- as.character(round(diag(mat), 2))

## Make the plot
layer_matrix_plot(mat, mat_text)

## Try to re-create the anatomical proportions of the human brain layers
layer_matrix_plot(
    mat,
    mat_text,
    layerHeights = c(0, 40, 55, 75, 85, 110, 120, 135),
    cex = 2
)
```

---

# layer_stat_cor *Layer modeling correlation of statistics*

---

## Description

Layer modeling correlation of statistics

## Usage

```
layer_stat_cor(
  stats,
  modeling_results = fetch_data(type = "modeling_results"),
  model_type = names(modeling_results)[1],
  reverse = FALSE,
  top_n = NULL
)
```

## Arguments

| | |
|---|---|
| stats | A data.frame where the row names are Ensembl gene IDs, the column names are labels for clusters of cells or cell types, and where each cell contains the given statistic for that gene and cell type. These statistics should be computed similarly to the modeling results from the data we provide. For example, like the `enrichment` t-statistics that are derived from comparing one layer against the rest. The `stats` will be matched and then correlated with our statistics. |
| modeling_results | Defaults to the output of `fetch_data(type = 'modeling_results')`. This is a list of tables with the columns `f_stat_*` or `t_stat_*` as well as `p_value_*` and `fdr_*` plus `ensembl`. The column name is used to extract the statistic results, the p-values, and the FDR adjusted p-values. Then the `ensembl` column is used for matching in some cases. See `fetch_data()` for more details. |
| model_type | A named element of the `modeling_results` list. By default that is either `enrichment` for the model that tests one human brain layer against the rest (one group vs the rest), `pairwise` which compares two layers (groups) denoted by `layerA-layerB` such that `layerA` is greater than `layerB`, and `anova` which determines if any layer (group) is different from the rest adjusting for the mean expression level. The statistics for `enrichment` and `pairwise` are t-statistics while the `anova` model ones are F-statistics. |
| reverse | A `logical(1)` indicating whether to multiply by `-1` the input statistics and reverse the `layerA-layerB` column names (using the `-`) into `layerB-layerA`. |
| top_n | An `integer(1)` specifying whether to filter to the top n marker genes. The default is `NULL` in which case no filtering is done. |

## Details

Check https://github.com/LieberInstitute/HumanPilot/blob/master/Analysis/Layer_Guesses/dlpfc_snRNAseq_annotation.R for a full analysis from which this family of functions is derived from.

## Value

A correlation matrix between `stats` and our statistics using only the Ensembl gene IDs present in both tables. The columns are sorted using a hierarchical cluster.

## Author(s)

Andrew E Jaffe, Leonardo Collado-Torres

## See Also

Other Layer correlation functions: annotate_registered_clusters(), layer_stat_cor_plot()

## Examples

```
## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}

## Compute the correlations
cor_stats_layer <- layer_stat_cor(
    tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer,
    modeling_results,
    model_type = "enrichment"
)

## Explore the correlation matrix
head(cor_stats_layer[, seq_len(3)])
summary(cor_stats_layer)

## Repeat with top_n set to 10
summary(layer_stat_cor(
    tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer,
    modeling_results,
    model_type = "enrichment",
    top_n = 10
))
```

---

# layer_stat_cor_plot *Visualize the layer modeling correlation of statistics*

---

## Description

This function makes a heatmap from the layer_stat_cor() correlation matrix between a given set of cell cluster/type statistics derived from scRNA-seq or snRNA-seq data (among other types) and the layer statistics from the Human DLPFC Visium data (when using the default arguments).

## Usage

```
layer_stat_cor_plot(
  cor_stats_layer,
  max = 0.81,
  min = -max,
  layerHeights = NULL,
  cex = 1.2
)
```

## Arguments

| | |
|---|---|
| cor_stats_layer | The output of layer_stat_cor(). |
| max | A numeric(1) specifying the highest correlation value for the color scale (should be between 0 and 1). |
| min | A numeric(1) specifying the lowest correlation value for the color scale (should be between 0 and -1). |
| layerHeights | A numeric() vector of length equal to ncol(cor_stats_layer) + 1 that starts at 0 specifying where to plot the y-axis breaks which can be used for re-creating the length of each brain layer. Gets passed to layer_matrix_plot(). |
| cex | Passed to layer_matrix_plot(). |

## Details

Check https://github.com/LieberInstitute/HumanPilot/blob/master/Analysis/Layer_Guesses/dlpfc_snRNAseq_annotation.R for a full analysis from which this family of functions is derived from.

## Value

A heatmap for the correlation matrix between statistics.

## Author(s)

Andrew E Jaffe, Leonardo Collado-Torres

## See Also

layer_matrix_plot annotate_registered_clusters

Other Layer correlation functions: annotate_registered_clusters(), layer_stat_cor()

## Examples

```
## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}

## Compute the correlations
cor_stats_layer <- layer_stat_cor(
    tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer,
    modeling_results,
    model_type = "enrichment"
)

## Visualize the correlation matrix
layer_stat_cor_plot(cor_stats_layer, max = max(cor_stats_layer))

## Annotate then re-plot
rownames(cor_stats_layer) <- paste0(
```

```
    rownames(cor_stats_layer),
    " - ",
    annotate_registered_clusters(cor_stats_layer)$layer_label
)
layer_stat_cor_plot(cor_stats_layer, max = max(cor_stats_layer))

## Restrict the range of colors further
layer_stat_cor_plot(cor_stats_layer, max = 0.25)

## Repeat with just the top 10 layer marker genes
layer_stat_cor_plot(layer_stat_cor(
    tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer,
    modeling_results,
    model_type = "enrichment",
    top_n = 10
), max = 0.25)

## Now with the "pairwise" modeling results and also top_n = 10
layer_stat_cor_plot(layer_stat_cor(
    tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer,
    modeling_results,
    model_type = "pairwise",
    top_n = 10
), max = 0.25)
```

---

`libd_layer_colors` *Vector of LIBD layer colors*

---

## Description

A named vector of colors to use for the LIBD layers designed by Lukas M. Weber with feedback from the spatialLIBD collaborators.

## Usage

```
libd_layer_colors
```

## Format

A vector of length 9 with colors for Layers 1 through 9, WM, NA and a special WM2 that is present in some of the unsupervised clustering results.

## locate_images *Locate image files*

### Description

Creates a named `character()` vector that can be helpful for locating image files and used with `add_images()`. This function is not necessary if the image files don't use the `spe$sample_id`.

### Usage

```
locate_images(spe, image_dir, image_pattern)
```

### Arguments

| | |
|---|---|
| `spe` | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| `image_dir` | A `character(1)` specifying a path to a directory containing image files with the pattern `sampleID_pattern.png`. |
| `image_pattern` | A `character(1)` specifying the pattern for the image files. |

### Value

A named `character()` vector with the path to images.

### See Also

Other Functions for adding non-standard images: `add_images()`

### Examples

```
## Not run:
locate_images(spe, tempdir(), "testImage")

## End(Not run)
```

## multi_gene_pca *Combine multiple continuous variables through PCA*

### Description

PCA is performed on `cont_mat`, the matrix of multiple continuous features. The first PC is returned, representing the dominant spatial signature of the feature set. Its direction is negated if necessary so that the majority of coefficients across features are positive (when the features are highly correlated, this encourages spots with higher values to represent areas of higher expression of the features).

### Usage

```
multi_gene_pca(cont_mat)
```

### Arguments

| | |
|---|---|
| cont_mat | A matrix() with spots as rows and 2 or more continuous variables as columns. |

### Value

A numeric() vector with one element per spot, summarizing the multiple continuous variables.

### Author(s)

Nicholas J. Eagles

### See Also

Other functions for summarizing expression of multiple continuous variables simultaneously: multi_gene_sparsity(), multi_gene_z_score()

---

## multi_gene_sparsity *Combine multiple continuous variables by proportion of positive values*

---

### Description

To summarize multiple features, the proportion of features with positive values for each spot is computed.

### Usage

```
multi_gene_sparsity(cont_mat)
```

### Arguments

| | |
|---|---|
| cont_mat | A matrix() with spots as rows and 2 or more continuous variables as columns. |

### Value

A numeric() vector with one element per spot, summarizing the multiple continuous variables.

### Author(s)

Nicholas J. Eagles

### See Also

Other functions for summarizing expression of multiple continuous variables simultaneously: multi_gene_pca(), multi_gene_z_score()

## multi_gene_z_score *Combine multiple continuous variables by averaging Z scores*

### Description

To summarize multiple features, each is normalized to represent a Z-score. Scores are averaged to return a single vector.

### Usage

```
multi_gene_z_score(cont_mat)
```

### Arguments

| | |
|---|---|
| cont_mat | A `matrix()` with spots as rows and 2 or more continuous variables as columns. |

### Value

A `numeric()` vector with one element per spot, summarizing the multiple continuous variables.

### Author(s)

Nicholas J. Eagles

### See Also

Other functions for summarizing expression of multiple continuous variables simultaneously: `multi_gene_pca()`, `multi_gene_sparsity()`

## read10xVisiumAnalysis *Load analysis data from a 10x Genomics Visium experiment*

### Description

This function expands `SpatialExperiment::read10xVisium()` by reading analysis outputs from SpaceRanger by 10x Genomics.

### Usage

```
read10xVisiumAnalysis(
  samples = "",
  sample_id = paste0("sample", sprintf("%02d", seq_along(samples)))
)
```

### Arguments

| | |
|---|---|
| samples | Passed to SpatialExperiment::read10xVisium(). |
| sample_id | Passed to SpatialExperiment::read10xVisium(). |

### Details

You might want to use read10xVisiumWrapper() instead of using this function directly.

### Value

A named list() with the information about the clustering and the dimension reduction (projections) from the SpaceRanger output by 10x Genomics.

### See Also

Other Utility functions for reading data from SpaceRanger output by 10x Genomics: add10xVisiumAnalysis(), read10xVisiumWrapper()

### Examples

```
## See 'Using spatialLIBD with 10x Genomics public datasets' for
## a full example using this function.
if (interactive()) {
    browseVignettes(package = "spatialLIBD")
}

## Note that ?SpatialExperiment::read10xVisium doesn't include all the files
## we need to illustrate read10xVisiumWrapper().
```

---

## read10xVisiumWrapper *Load data from a 10x Genomics Visium experiment and make it spatialLIBD-ready*

---

### Description

This function expands SpatialExperiment::read10xVisium() to include analysis results from SpaceRanger by 10x Genomics as well as add information needed by run_app() to visualize the data with the spatialLIBD shiny web application.

### Usage

```
read10xVisiumWrapper(
  samples = "",
  sample_id = paste0("sample", sprintf("%02d", seq_along(samples))),
  type = c("HDF5", "sparse"),
  data = c("filtered", "raw"),
  images = c("lowres", "hires", "detected", "aligned"),
  load = TRUE,
```

```
  reference_gtf = NULL,
  chrM = "chrM",
 gtf_cols = c("source", "type", "gene_id", "gene_version", "gene_name", "gene_type"),
  verbose = TRUE
)
```

## Arguments

| | |
|---|---|
| samples | Passed to SpatialExperiment::read10xVisium(). |
| sample_id | Passed to SpatialExperiment::read10xVisium(). |
| type | Passed to SpatialExperiment::read10xVisium(). |
| data | Passed to SpatialExperiment::read10xVisium(). |
| images | Passed to SpatialExperiment::read10xVisium(). |
| load | Passed to SpatialExperiment::read10xVisium(). |
| reference_gtf | A character(1) specifying the path to the reference genes.gtf file. If not specified, it will be automatically inferred from the web_summary.html file for the first samples. |
| chrM | A character(1) specifying the chromosome name of the mitochondrial chromosome. Defaults to chrM. |
| gtf_cols | A character() specifying which columns to keep from the GTF file. "gene_name" and "gene_id" have to be included in gtf_cols. |
| verbose | A logical(1) specifying whether to show progress updates. |

## Value

A SpatialExperiment object with the clustering and dimension reduction (projection) results from SpaceRanger by 10x Genomics as well as other information used by run_app() for visualzing the gene expression data.

## See Also

Other Utility functions for reading data from SpaceRanger output by 10x Genomics: add10xVisiumAnalysis(), read10xVisiumAnalysis()

## Examples

```
## See 'Using spatialLIBD with 10x Genomics public datasets' for
## a full example using this function.
if (interactive()) {
    browseVignettes(package = "spatialLIBD")
}

## Note that ?SpatialExperiment::read10xVisium doesn't include all the files
## we need to illustrate read10xVisiumWrapper().
```

`registration_block_cor` *Spatial registration: block correlation*

## Description

This function computes the block correlation using the sample ID as the blocking factor. This takes into account that cells in scRNA-seq data or spots in spatially-resolved transcriptomics data from Visium (or similar) have a sample ID batch effect.

## Usage

```
registration_block_cor(
  sce_pseudo,
  registration_model,
  var_sample_id = "registration_sample_id"
)
```

## Arguments

| | |
|---|---|
| `sce_pseudo` | The output of `registration_pseudobulk()`. |
| `registration_model` | The output from `registration_model()`. |
| `var_sample_id` | A `character(1)` specifying the `colData(sce_pseudo)` variable with the sample ID. |

## Value

A `numeric(1)` with the block correlation at the sample ID level.

## See Also

Other spatial registration and statistical modeling functions: `registration_model()`, `registration_pseudobulk()`, `registration_stats_anova()`, `registration_stats_enrichment()`, `registration_stats_pairwise()`, `registration_wrapper()`

## Examples

```
example("registration_model", package = "spatialLIBD")
block_cor <- registration_block_cor(sce_pseudo, registration_mod)
```

registration_model *Spatial registration: model*

## Description

This function defines the statistical model that will be used for computing the block correlation as well as pairwise statistics. It is useful to check it in case your sample-level covariates need to be casted. For example, an `integer()` variable might have to be casted into a `factor()` if you wish to model it as a categorical variable and not a continuous one.

## Usage

```
registration_model(
  sce_pseudo,
  covars = NULL,
  var_registration = "registration_variable"
)
```

## Arguments

| | |
|---|---|
| `sce_pseudo` | The output of `registration_pseudobulk()`. |
| `covars` | A `character()` with names of sample-level covariates. |
| `var_registration` | A `character(1)` specifying the `colData(sce_pseudo)` variable of interest against which will be used for computing the relevant statistics. |

## Value

The output of `model.matrix()` which you can inspect to verify that your sample-level covariates are being properly modeled.

## See Also

Other spatial registration and statistical modeling functions: `registration_block_cor()`, `registration_pseudobulk()`, `registration_stats_anova()`, `registration_stats_enrichment()`, `registration_stats_pairwise()`, `registration_wrapper()`

## Examples

```
example("registration_pseudobulk", package = "spatialLIBD")
registration_mod <- registration_model(sce_pseudo, "age")
head(registration_mod)
```

registration_pseudobulk *Spatial registration: pseudobulk*

## Description

Pseudo-bulk the gene expression, filter lowly-expressed genes, and normalize. This is the first step for spatial registration and for statistical modeling.

## Usage

```
registration_pseudobulk(
  sce,
  var_registration,
  var_sample_id,
  covars = NULL,
  min_ncells = 10,
  pseudobulk_rds_file = NULL
)
```

## Arguments

| | |
|---|---|
| sce | A SingleCellExperiment-class object or one that inherits its properties. |
| var_registration | A `character(1)` specifying the `colData(sce)` variable of interest against which will be used for computing the relevant statistics. |
| var_sample_id | A `character(1)` specifying the `colData(sce)` variable with the sample ID. |
| covars | A `character()` with names of sample-level covariates. |
| min_ncells | An `integer(1)` greater than 0 specifying the minimum number of cells (for scRNA-seq) or spots (for spatial) that are combined when pseudo-bulking. Pseudo-bulked samples with less than `min_ncells` on `sce_pseudo$ncells` will be dropped. |
| pseudobulk_rds_file | A `character(1)` specifying the path for saving an RDS file with the pseudo-bulked object. It's useful to specify this since pseudo-bulking can take hours to run on large datasets. |

## Value

A pseudo-bulked SingleCellExperiment-class object.

## See Also

Other spatial registration and statistical modeling functions: `registration_block_cor()`, `registration_model()`, `registration_stats_anova()`, `registration_stats_enrichment()`, `registration_stats_pairwise()`, `registration_wrapper()`

## Examples

```
## Ensure reproducibility of example data
set.seed(20220907)

## Generate example data
sce <- scuttle::mockSCE()

## Add some sample IDs
sce$sample_id <- sample(LETTERS[1:5], ncol(sce), replace = TRUE)

## Add a sample-level covariate: age
ages <- rnorm(5, mean = 20, sd = 4)
names(ages) <- LETTERS[1:5]
sce$age <- ages[sce$sample_id]

## Add gene-level information
rowData(sce)$ensembl <- paste0("ENSG", seq_len(nrow(sce)))
rowData(sce)$gene_name <- paste0("gene", seq_len(nrow(sce)))

## Pseudo-bulk
sce_pseudo <- registration_pseudobulk(sce, "Cell_Cycle", "sample_id", c("age"), min_ncells = NULL)
colData(sce_pseudo)
```

---

# registration_stats_anova

*Spatial registration: compute ANOVA statistics*

---

## Description

This function computes the gene ANOVA F-statistics (at least one group is different from the rest). These F-statistics can be used for spatial registration with `layer_stat_cor()` and related functions. Although, they are more typically used for identifying ANOVA-marker genes.

## Usage

```
registration_stats_anova(
  sce_pseudo,
  block_cor,
  covars = NULL,
  var_registration = "registration_variable",
  var_sample_id = "registration_sample_id",
  gene_ensembl = NULL,
  gene_name = NULL,
  suffix = ""
)
```

## Arguments

| | |
|---|---|
| sce_pseudo | The output of registration_pseudobulk(). |
| block_cor | A numeric(1) computed with registration_block_cor(). |
| covars | A character() with names of sample-level covariates. |
| var_registration | A character(1) specifying the colData(sce_pseudo) variable of interest against which will be used for computing the relevant statistics. |
| var_sample_id | A character(1) specifying the colData(sce_pseudo) variable with the sample ID. |
| gene_ensembl | A character(1) specifying the rowData(sce_pseudo) column with the ENSEMBL gene IDs. This will be used by layer_stat_cor(). |
| gene_name | A character(1) specifying the rowData(sce_pseudo) column with the gene names (symbols). |
| suffix | A character(1) specifying the suffix to use for the F-statistics column. This is particularly useful if you will run this function more than once and want to be able to merge the results. |

## Value

A data.frame() with the ANOVA statistical results. This is similar to fetch_data("modeling_results")$anova.

## See Also

Other spatial registration and statistical modeling functions: registration_block_cor(), registration_model(), registration_pseudobulk(), registration_stats_enrichment(), registration_stats_pairwise(), registration_wrapper()

## Examples

```
example("registration_block_cor", package = "spatialLIBD")
results_anova <- registration_stats_anova(sce_pseudo,
    block_cor, "age",
    gene_ensembl = "ensembl", gene_name = "gene_name", suffix = "example"
)
head(results_anova)

## Specifying `block_cor = NaN` then ignores the correlation structure
results_anova_nan <- registration_stats_anova(sce_pseudo,
    block_cor = NaN, "age",
    gene_ensembl = "ensembl", gene_name = "gene_name", suffix = "example"
)
head(results_anova_nan)

## Note that you can merge multiple of these data.frames if you run this
## function for different sets. For example, maybe you drop one group
## before pseudo-bulking if you know that there are many differences between
## that group and others. For example, we have dropped the white matter (WM)
## prior to computing ANOVA F-statistics.
```

```
## no covariates
results_anova_nocovar <- registration_stats_anova(sce_pseudo,
    block_cor,
    covars = NULL,
    gene_ensembl = "ensembl", gene_name = "gene_name", suffix = "nocovar"
)
head(results_anova_nocovar)

## Merge both results into a single data.frame, thanks to having different
## 'suffix' values.
results_anova_merged <- merge(results_anova, results_anova_nocovar)
head(results_anova_merged)
```

---

`registration_stats_enrichment` *Spatial registration: compute enrichment statistics*

---

## Description

This function computes the gene enrichment t-statistics (one group > the rest). These t-statistics are the ones typically used for spatial registration with `layer_stat_cor()` and related functions.

## Usage

```
registration_stats_enrichment(
  sce_pseudo,
  block_cor,
  covars = NULL,
  var_registration = "registration_variable",
  var_sample_id = "registration_sample_id",
  gene_ensembl = NULL,
  gene_name = NULL
)
```

## Arguments

| | |
|---|---|
| `sce_pseudo` | The output of `registration_pseudobulk()`. |
| `block_cor` | A `numeric(1)` computed with `registration_block_cor()`. |
| `covars` | A `character()` with names of sample-level covariates. |
| `var_registration` | A `character(1)` specifying the `colData(sce_pseudo)` variable of interest against which will be used for computing the relevant statistics. |
| `var_sample_id` | A `character(1)` specifying the `colData(sce_pseudo)` variable with the sample ID. |
| `gene_ensembl` | A `character(1)` specifying the `rowData(sce_pseudo)` column with the ENSEMBL gene IDs. This will be used by `layer_stat_cor()`. |
| `gene_name` | A `character(1)` specifying the `rowData(sce_pseudo)` column with the gene names (symbols). |

## Value

A `data.frame()` with the enrichment statistical results. This is similar to `fetch_data("modeling_results")$enrichment`

## See Also

Other spatial registration and statistical modeling functions: `registration_block_cor()`, `registration_model()`, `registration_pseudobulk()`, `registration_stats_anova()`, `registration_stats_pairwise()`, `registration_wrapper()`

## Examples

```
example("registration_block_cor", package = "spatialLIBD")
results_enrichment <- registration_stats_enrichment(sce_pseudo,
    block_cor, "age",
    gene_ensembl = "ensembl", gene_name = "gene_name"
)
head(results_enrichment)

## Specifying `block_cor = NaN` then ignores the correlation structure
results_enrichment_nan <- registration_stats_enrichment(sce_pseudo,
    block_cor = NaN, "age",
    gene_ensembl = "ensembl", gene_name = "gene_name"
)
head(results_enrichment_nan)
```

---

# registration_stats_pairwise

*Spatial registration: compute pairwise statistics*

---

## Description

This function computes the gene pairwise t-statistics (one group > another, for all combinations). These t-statistics can be used for spatial registration with `layer_stat_cor()` and related functions. Although, they are more typically used for identifying pairwise-marker genes.

## Usage

```
registration_stats_pairwise(
  sce_pseudo,
  registration_model,
  block_cor,
  var_registration = "registration_variable",
  var_sample_id = "registration_sample_id",
  gene_ensembl = NULL,
  gene_name = NULL
)
```

## Arguments

| | |
|---|---|
| sce_pseudo | The output of registration_pseudobulk(). |
| registration_model | The output from registration_model(). |
| block_cor | A numeric(1) computed with registration_block_cor(). |
| var_registration | A character(1) specifying the colData(sce_pseudo) variable of interest against which will be used for computing the relevant statistics. |
| var_sample_id | A character(1) specifying the colData(sce_pseudo) variable with the sample ID. |
| gene_ensembl | A character(1) specifying the rowData(sce_pseudo) column with the ENSEMBL gene IDs. This will be used by layer_stat_cor(). |
| gene_name | A character(1) specifying the rowData(sce_pseudo) column with the gene names (symbols). |

## Value

A data.frame() with the pairwise statistical results. This is similar to fetch_data("modeling_results")$pairwise.

## See Also

Other spatial registration and statistical modeling functions: registration_block_cor(), registration_model(), registration_pseudobulk(), registration_stats_anova(), registration_stats_enrichment(), registration_wrapper()

## Examples

```
example("registration_block_cor", package = "spatialLIBD")
results_pairwise <- registration_stats_pairwise(sce_pseudo,
    registration_mod, block_cor,
    gene_ensembl = "ensembl", gene_name = "gene_name"
)
head(results_pairwise)

## Specifying `block_cor = NaN` then ignores the correlation structure
results_pairwise_nan <- registration_stats_pairwise(sce_pseudo,
    registration_mod,
    block_cor = NaN,
    gene_ensembl = "ensembl", gene_name = "gene_name"
)
head(results_pairwise_nan)
```

`registration_wrapper` *Spatial registration: wrapper function*

## Description

This function is provided for convenience. It runs all the functions required for computing the `modeling_results`. This can be useful for finding marker genes on a new spatially-resolved transcriptomics dataset and thus using it for `run_app()`. The results from this function can also be used for performing spatial registration through `layer_stat_cor()` and related functions of sc/snRNA-seq datasets.

## Usage

```
registration_wrapper(
  sce,
  var_registration,
  var_sample_id,
  covars = NULL,
  gene_ensembl = NULL,
  gene_name = NULL,
  suffix = "",
  min_ncells = 10,
  pseudobulk_rds_file = NULL
)
```

## Arguments

| | |
|---|---|
| `sce` | A SingleCellExperiment-class object or one that inherits its properties. |
| `var_registration` | A `character(1)` specifying the `colData(sce)` variable of interest against which will be used for computing the relevant statistics. |
| `var_sample_id` | A `character(1)` specifying the `colData(sce)` variable with the sample ID. |
| `covars` | A `character()` with names of sample-level covariates. |
| `gene_ensembl` | A `character(1)` specifying the `rowData(sce_pseudo)` column with the ENSEMBL gene IDs. This will be used by `layer_stat_cor()`. |
| `gene_name` | A `character(1)` specifying the `rowData(sce_pseudo)` column with the gene names (symbols). |
| `suffix` | A `character(1)` specifying the suffix to use for the F-statistics column. This is particularly useful if you will run this function more than once and want to be able to merge the results. |
| `min_ncells` | An `integer(1)` greater than 0 specifying the minimum number of cells (for scRNA-seq) or spots (for spatial) that are combined when pseudo-bulking. Pseudo-bulked samples with less than `min_ncells` on `sce_pseudo$ncells` will be dropped. |

pseudobulk_rds_file
: A character(1) specifying the path for saving an RDS file with the pseudo-bulked object. It's useful to specify this since pseudo-bulking can take hours to run on large datasets.

## Details

We chose a default of min_ncells = 10 based on OSCA from section 4.3 at http://bioconductor.org/books/3.15/OSCA.multisample/multi-sample-comparisons.html. They cite https://doi.org/10.1038/s41467-020-19894-4 as the paper where they came up with the definition of "very low" being 10. You might want to use registration_pseudobulk() and manually explore sce_pseudo$ncells to choose the best cutoff.

## Value

A list() of data.frame() with the statistical results. This is similar to fetch_data("modeling_results").

## See Also

Other spatial registration and statistical modeling functions: registration_block_cor(), registration_model(), registration_pseudobulk(), registration_stats_anova(), registration_stats_enrichment(), registration_stats_pairwise()

## Examples

```
## Ensure reproducibility of example data
set.seed(20220907)

## Generate example data
sce <- scuttle::mockSCE()

## Add some sample IDs
sce$sample_id <- sample(LETTERS[1:5], ncol(sce), replace = TRUE)

## Add a sample-level covariate: age
ages <- rnorm(5, mean = 20, sd = 4)
names(ages) <- LETTERS[1:5]
sce$age <- ages[sce$sample_id]

## Add gene-level information
rowData(sce)$ensembl <- paste0("ENSG", seq_len(nrow(sce)))
rowData(sce)$gene_name <- paste0("gene", seq_len(nrow(sce)))

## Compute all modeling results
example_modeling_results <- registration_wrapper(
    sce,
    "Cell_Cycle", "sample_id", c("age"), "ensembl", "gene_name", "wrapper"
)
```

run_app *Run the spatialLIBD Shiny Application*

## Description

This function runs the shiny application that allows users to interact with the Visium spatial transcriptomics data from LIBD (by default) or any other data that you have shaped according to our object structure.

## Usage

```
run_app(
  spe = fetch_data(type = "spe"),
  sce_layer = fetch_data(type = "sce_layer"),
  modeling_results = fetch_data(type = "modeling_results"),
  sig_genes = sig_genes_extract_all(n = nrow(sce_layer), modeling_results =
    modeling_results, sce_layer = sce_layer),
  docs_path = system.file("app", "www", package = "spatialLIBD"),
  title = "spatialLIBD",
 spe_discrete_vars = c("spatialLIBD", "GraphBased", "ManualAnnotation", "Maynard",
  "Martinowich", paste0("SNN_k50_k", 4:28), "SpatialDE_PCA", "SpatialDE_pool_PCA",
  "HVG_PCA", "pseudobulk_PCA", "markers_PCA", "SpatialDE_UMAP", "SpatialDE_pool_UMAP",
    "HVG_UMAP", "pseudobulk_UMAP", "markers_UMAP", "SpatialDE_PCA_spatial",
    "SpatialDE_pool_PCA_spatial", "HVG_PCA_spatial", "pseudobulk_PCA_spatial",
  "markers_PCA_spatial", "SpatialDE_UMAP_spatial", "SpatialDE_pool_UMAP_spatial",
    "HVG_UMAP_spatial", "pseudobulk_UMAP_spatial",
     "markers_UMAP_spatial"),
  spe_continuous_vars = c("cell_count", "sum_umi", "sum_gene", "expr_chrM",
    "expr_chrM_ratio"),
  default_cluster = "spatialLIBD",
  auto_crop_default = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| sce_layer | Defaults to the output of `fetch_data(type = 'sce_layer')`. This is a SingleCellExperiment object with the spot-level Visium data compressed via pseudobulking to the layer-level (group-level) resolution. See `fetch_data()` for more details. |
| modeling_results | Defaults to the output of `fetch_data(type = 'modeling_results')`. This is a list of tables with the columns `f_stat_*` or `t_stat_*` as well as `p_value_*` and `fdr_*` plus `ensembl`. The column name is used to extract the statistic results, |

the p-values, and the FDR adjusted p-values. Then the ensembl column is used for matching in some cases. See fetch_data() for more details.

sig_genes — The output of sig_genes_extract_all() which is a table in long format with the modeling results. You can subset this if the object requires too much memory.

docs_path — A character(1) specifying the path to the directory containing the website documentation files. The directory has to contain the files: documentation_sce_layer.md, documentation_spe.md, favicon.ico, footer.html and README.md.

title — A character(1) specifying the title for the app.

spe_discrete_vars — A character() vector of discrete variables that will be available to visualize in the app. Basically, the set of variables with spot-level groups. They will have to be present in colData(spe).

spe_continuous_vars — A character() vector of continuous variables that will be available to visualize in the app using the same scale as genes. They will have to be present in colData(sce).

default_cluster — A character(1) with the name of the main cluster (discrete) variable to use. It will have to be present in both colData(spe) and colData(sce_layer).

auto_crop_default — A logical(1) specifying the default value for automatically cropping the images. Set this to FALSE if your images do not follow the Visium grid size expectations, which are key for enabling auto-cropping.

... — Other arguments passed to the list of golem options for running the application.

## Details

If you don't have the pseudo-bulked analysis results like we computed them in our project https://doi.org/10.1038/s41593-020-00787-0 you can set sce_layer, modeling_results and sig_genes to NULL. Doing so will disable the pseudo-bulked portion of the web application. See the examples for one such case as well as the vignette that describes how you can use spatialLIBD with public data sets provided by 10x Genomics. That vignette is available at http://research.libd.org/spatialLIBD/articles/TenX_data_download.html.

## Value

A shiny.appobj that contains the input data.

## Examples

```
## Not run:
## The default arguments will download the data from the web
## using fetch_data(). If this is the first time you have run this,
## the files will need to be cached by ExperimentHub. Otherwise it
## will re-use the files you have previously downloaded.
if (enough_ram(4e9)) {
    ## Obtain the necessary data
```

```
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Create the interactive website
    run_app(spe)

    ## You can also run a custom version without the pseudo-bulked
    ## layer information. This is useful if you are only interested
    ## in the spatial transcriptomics features.
    run_app(spe,
        sce_layer = NULL, modeling_results = NULL, sig_genes = NULL,
        title = "spatialLIBD without layer info"
    )

    ## When using shinyapps.io aim for less than 3 GB of RAM with your
    ## objects. Check each input object with:
    ## lobstr::obj_size(x)
    ## Do not create the large input objects on the app.R script before
    ## subsetting them. Do this outside app.R since the app.R script is
    ## run at shinyapps.io, so subsetting on that script to reduce the
    ## memory load is pointless. You have to do it outside of app.R.
}

## How to run locally the spatialDLPFC Sp09 spatialLIBD app. That is,
## from http://research.libd.org/spatialDLPFC/#interactive-websites
## how to run https://libd.shinyapps.io/spatialDLPFC_Visium_Sp09 locally.
if (enough_ram(9e9)) {
    ## Download the 3 main objects needed
    spe <- fetch_data("spatialDLPFC_Visium")
    sce_pseudo <- fetch_data("spatialDLPFC_Visium_pseudobulk")
    modeling_results <- fetch_data("spatialDLPFC_Visium_modeling_results")

    ## These are optional commands to further reduce the memory required.
    #
    ## Keep only the "lowres" images. Reduces the object from 6.97 GB to 4.59 GB
    # imgData(spe) <- imgData(spe)[imgData(spe)$image_id == "lowres", ]
    ## Drop the regular counts (keep only the logcounts). Reduces the object
    ## from 4.59 GB to 2.45 GB.
    # counts(spe) <- NULL

    ## For sig_genes_extract_all() to work
    sce_pseudo$spatialLIBD <- sce_pseudo$BayesSpace
    ## Compute the significant genes
    sig_genes <- sig_genes_extract_all(
        n = nrow(sce_pseudo),
        modeling_results = modeling_results,
        sce_layer = sce_pseudo
    )
    ## Reduce the memory from 423.73 MB to 78.88 MB
    lobstr::obj_size(sig_genes)
    sig_genes$in_rows <- NULL
    sig_genes$in_rows_top20 <- NULL
    lobstr::obj_size(sig_genes)
```

```
## Specify the default variable
spe$BayesSpace <- spe$BayesSpace_harmony_09
## Get all variables
vars <- colnames(colData(spe))

## Set default cluster colors
colors_BayesSpace <- Polychrome::palette36.colors(28)
names(colors_BayesSpace) <- c(1:28)
m <- match(as.character(spe$BayesSpace_harmony_09), names(colors_BayesSpace))
stopifnot(all(!is.na(m)))
spe$BayesSpace_colors <- spe$BayesSpace_harmony_09_colors <- colors_BayesSpace[m]

## Download documentation files we use
temp_www <- file.path(tempdir(), "www")
dir.create(temp_www)
download.file(
    "https://raw.githubusercontent.com/LieberInstitute/spatialDLPFC/main/README.md",
    file.path(temp_www, "README.md")
)
download.file(
 "https://raw.githubusercontent.com/LieberInstitute/spatialDLPFC/main/code/deploy_app_k09/www/documentatio
    file.path(temp_www, "documentation_sce_layer.md")
)
download.file(
 "https://raw.githubusercontent.com/LieberInstitute/spatialDLPFC/main/code/deploy_app_k09/www/documentatio
    file.path(temp_www, "documentation_spe.md")
)
download.file(
 "https://raw.githubusercontent.com/LieberInstitute/spatialDLPFC/main/img/favicon.ico",
    file.path(temp_www, "favicon.ico")
)
download.file(
 "https://raw.githubusercontent.com/LieberInstitute/spatialDLPFC/main/code/deploy_app_k09/www/footer.html"
    file.path(temp_www, "footer.html")
)
list.files(temp_www)

## Run the app locally
run_app(
    spe,
    sce_layer = sce_pseudo,
    modeling_results = modeling_results,
    sig_genes = sig_genes,
    title = "spatialDLPFC, Visium, Sp09",
  spe_discrete_vars = c( # this is the variables for the spe object not the sce_pseudo object
        "BayesSpace",
        "ManualAnnotation",
        vars[grep("^SpaceRanger_|^scran_", vars)],
        vars[grep("^BayesSpace_harmony", vars)],
        vars[grep("^BayesSpace_pca", vars)],
        "graph_based_PCA_within",
        "PCA_SNN_k10_k7",
        "Harmony_SNN_k10_k7",
```

```
            "manual_layer_label",
            "wrinkle_type",
            "BayesSpace_colors"
        ),
        spe_continuous_vars = c(
            "sum_umi",
            "sum_gene",
            "expr_chrM",
            "expr_chrM_ratio",
            vars[grep("^VistoSeg_", vars)],
            vars[grep("^layer_", vars)],
            vars[grep("^broad_", vars)]
        ),
        default_cluster = "BayesSpace",
        docs_path = temp_www
    )
}
## See also:
## * https://github.com/LieberInstitute/spatialDLPFC/tree/main/code/deploy_app_k09
## * https://github.com/LieberInstitute/spatialDLPFC/tree/main/code/deploy_app_k09_position
## * https://github.com/LieberInstitute/spatialDLPFC/tree/main/code/deploy_app_k09_position_noWM
## * https://github.com/LieberInstitute/spatialDLPFC/tree/main/code/deploy_app_k16
## * https://github.com/LieberInstitute/spatialDLPFC/tree/main/code/analysis_IF/03_spatialLIBD_app

## End(Not run)
```

---

`sce_to_spe` *Convert a SCE object to a SPE one*

---

## Description

This function converts a spot-level SingleCellExperiment-class (SCE) object as generated by `fetch_data()` to a SpatialExperiment-class (SPE) object.

## Usage

```
sce_to_spe(sce = fetch_data("sce"), imageData = NULL)
```

## Arguments

| | |
|---|---|
| `sce` | Defaults to the output of `fetch_data(type = 'sce')`. This is a SingleCellExperiment object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| `imageData` | A `DataFrame()` with image data. Will be used with SpatialExperiment::imgData. If `NULL`, then this will be constructed for you assuming that you are working with the original data from `spatialLIBD::fetch_data("sce")`. |

## Details

Note that the resulting object is a bit more complex than a regular SPE because it contains the data from the spatialLIBD project which you might otherwise have to generate for your own data.

## Value

A a SpatialExperiment-class object.

## Author(s)

Brenda Pardo, Leonardo Collado-Torres

## Examples

```
if (enough_ram()) {
    ## Download the sce data
    sce <- fetch_data("sce")
    ## Transform it to a SpatialExperiment object
    spe <- sce_to_spe(sce)
}
```

---

# sig_genes_extract *Extract significant genes*

---

## Description

From the layer-level modeling results, this function extracts the top `n` significant genes. This is the workhorse function used by `sig_genes_extract_all()` through which we obtain the information that can then be used by functions such as `layer_boxplot()` for constructing informative titles.

## Usage

```
sig_genes_extract(
  n = 10,
  modeling_results = fetch_data(type = "modeling_results"),
  model_type = names(modeling_results)[1],
  reverse = FALSE,
  sce_layer = fetch_data(type = "sce_layer")
)
```

## Arguments

| | |
|---|---|
| `n` | The number of the top ranked genes to extract. |
| `modeling_results` | Defaults to the output of `fetch_data(type = 'modeling_results')`. This is a list of tables with the columns `f_stat_*` or `t_stat_*` as well as `p_value_*` and `fdr_*` plus `ensembl`. The column name is used to extract the statistic results, the p-values, and the FDR adjusted p-values. Then the `ensembl` column is used for matching in some cases. See `fetch_data()` for more details. |

model_type
: A named element of the `modeling_results` list. By default that is either `enrichment` for the model that tests one human brain layer against the rest (one group vs the rest), `pairwise` which compares two layers (groups) denoted by `layerA-layerB` such that `layerA` is greater than `layerB`, and `anova` which determines if any layer (group) is different from the rest adjusting for the mean expression level. The statistics for `enrichment` and `pairwise` are t-statistics while the `anova` model ones are F-statistics.

reverse
: A `logical(1)` indicating whether to multiply by `-1` the input statistics and reverse the `layerA-layerB` column names (using the `-`) into `layerB-layerA`.

sce_layer
: Defaults to the output of `fetch_data(type = 'sce_layer')`. This is a SingleCellExperiment object with the spot-level Visium data compressed via pseudobulking to the layer-level (group-level) resolution. See `fetch_data()` for more details.

## Value

A `data.frame()` with the top `n` significant genes (as ordered by their statistics in decreasing order) in long format. The specific columns are described further in the vignette.

## References

Adapted from https://github.com/LieberInstitute/HumanPilot/blob/master/Analysis/Layer_Guesses/layer_specificity_functio

## See Also

Other Layer modeling functions: `layer_boxplot()`, `sig_genes_extract_all()`

## Examples

```
## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}
if (!exists("sce_layer")) sce_layer <- fetch_data(type = "sce_layer")

## anova top 10 genes
sig_genes_extract(
    modeling_results = modeling_results,
    sce_layer = sce_layer
)

## Extract all genes
sig_genes_extract(
    modeling_results = modeling_results,
    sce_layer = sce_layer,
    n = nrow(sce_layer)
)
```

`sig_genes_extract_all` *Extract significant genes for all modeling results*

## Description

This function combines the output of `sig_genes_extract()` from all the layer-level (group-level) modeling results and builds the data required for functions such as `layer_boxplot()`.

## Usage

```
sig_genes_extract_all(
  n = 10,
  modeling_results = fetch_data(type = "modeling_results"),
  sce_layer = fetch_data(type = "sce_layer")
)
```

## Arguments

| | |
|---|---|
| `n` | The number of the top ranked genes to extract. |
| `modeling_results` | Defaults to the output of `fetch_data(type = 'modeling_results')`. This is a list of tables with the columns `f_stat_*` or `t_stat_*` as well as `p_value_*` and `fdr_*` plus `ensembl`. The column name is used to extract the statistic results, the p-values, and the FDR adjusted p-values. Then the `ensembl` column is used for matching in some cases. See `fetch_data()` for more details. |
| `sce_layer` | Defaults to the output of `fetch_data(type = 'sce_layer')`. This is a SingleCellExperiment object with the spot-level Visium data compressed via pseudo-bulking to the layer-level (group-level) resolution. See `fetch_data()` for more details. |

## Value

A DataFrame-class with the extracted statistics in long format. The specific columns are described further in the vignette.

## See Also

Other Layer modeling functions: `layer_boxplot()`, `sig_genes_extract()`

## Examples

```
## Obtain the necessary data
if (!exists("modeling_results")) {
    modeling_results <- fetch_data(type = "modeling_results")
}
if (!exists("sce_layer")) sce_layer <- fetch_data(type = "sce_layer")

## top 10 genes for all models
```

```
sig_genes_extract_all(
    modeling_results = modeling_results,
    sce_layer = sce_layer
)
```

---

`sort_clusters` *Sort clusters by frequency*

---

## Description

This function takes a vector with cluster labels and sorts it by frequency such that the most frequent cluster is the first one and so on.

## Usage

```
sort_clusters(clusters, map_subset = NULL)
```

## Arguments

| | |
|---|---|
| `clusters` | A vector with cluster labels. |
| `map_subset` | A logical vector of length equal to `clusters` specifying which elements of `clusters` to use to determine the ranking of the clusters. |

## Value

A factor of length equal to `clusters` where the levels are the new ordered clusters and the names of the factor are the original values from `clusters`.

## Examples

```
## Build an initial set of cluster labels
clus <- letters[unlist(lapply(4:1, function(x) rep(x, x)))]

## In this case, it's a character vector
class(clus)

## Sort them and obtain a factor
sort_clusters(clus)
```

## tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer *Cell cluster t-statistics from Tran et al*

### Description

Using the DLPFC snRNA-seq data from Matthew N Tran et al we computed enrichment t-statistics for the cell clusters. This is a subset of them used in examples such as in `layer_stat_cor_plot()`.

### Usage

```
tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer
```

### Format

A matrix with 692 rows and 31 variables where each column is a given cell cluster from Tran et al and each row is one gene. The row names are Ensembl gene IDs which are used by `layer_stat_cor()` to match to our modeling results.

### Source

https://github.com/LieberInstitute/HumanPilot/blob/master/Analysis/Layer_Guesses/dlpfc_snRNAseq_annotation.R and https://github.com/LieberInstitute/spatialLIBD/blob/master/dev/02_dev.R#L107-L194.

## vis_clus *Sample spatial cluster visualization*

### Description

This function visualizes the clusters for one given sample at the spot-level using (by default) the histology information on the background. To visualize gene-level (or any continuous variable) use `vis_gene()`.

### Usage

```
vis_clus(
  spe,
  sampleid = unique(spe$sample_id)[1],
  clustervar,
 colors = c("#b2df8a", "#e41a1c", "#377eb8", "#4daf4a", "#ff7f00", "gold", "#a65628",
    "#999999", "black", "grey", "white", "purple"),
  spatial = TRUE,
  image_id = "lowres",
  alpha = NA,
```

```
  point_size = 2,
  auto_crop = TRUE,
  na_color = "#CCCCCC40",
  ...
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| sampleid | A `character(1)` specifying which sample to plot from `colData(spe)$sample_id` (formerly `colData(spe)$sample_name`). |
| clustervar | A `character(1)` with the name of the `colData(spe)` column that has the cluster values. |
| colors | A vector of colors to use for visualizing the clusters from `clustervar`. If the vector has names, then those should match the values of `clustervar`. |
| spatial | A `logical(1)` indicating whether to include the histology layer from `geom_spatial()`. If you plan to use ggplotly() then it's best to set this to `FALSE`. |
| image_id | A `character(1)` with the name of the image ID you want to use in the background. |
| alpha | A `numeric(1)` in the `[0, 1]` range that specifies the transparency level of the data on the spots. |
| point_size | A `numeric(1)` specifying the size of the points. Defaults to `1.25`. Some colors look better if you use `2` for instance. |
| auto_crop | A `logical(1)` indicating whether to automatically crop the image / plotting area, which is useful if the Visium capture area is not centered on the image and if the image is not a square. |
| na_color | A `character(1)` specifying a color for the NA values. If you set `alpha = NA` then it's best to set `na_color` to a color that has alpha blending already, which will make non-NA values pop up more and the NA values will show with a lighter color. This behavior is lost when `alpha` is set to a non-`NA` value. |
| ... | Passed to paste0() for making the title of the plot following the `sampleid`. |

## Details

This function subsets `spe` to the given sample and prepares the data and title for `vis_clus_p()`.

## Value

A ggplot2 object.

## See Also

Other Spatial cluster visualization functions: `frame_limits()`, `vis_clus_p()`, `vis_grid_clus()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Check the colors defined by Lukas M Weber
    libd_layer_colors

    ## Use the manual color palette by Lukas M Weber
    p1 <- vis_clus(
        spe = spe,
        clustervar = "layer_guess_reordered",
        sampleid = "151673",
        colors = libd_layer_colors,
        ... = " LIBD Layers"
    )
    print(p1)

    ## Without auto-cropping the image
    p2 <- vis_clus(
        spe = spe,
        clustervar = "layer_guess_reordered",
        sampleid = "151673",
        colors = libd_layer_colors,
        auto_crop = FALSE,
        ... = " LIBD Layers"
    )
    print(p2)

    ## Without histology
    p3 <- vis_clus(
        spe = spe,
        clustervar = "layer_guess_reordered",
        sampleid = "151673",
        colors = libd_layer_colors,
        ... = " LIBD Layers",
        spatial = FALSE
    )
    print(p3)

    ## With some NA values
    spe$tmp <- spe$layer_guess_reordered
    spe$tmp[spe$sample_id == "151673"][seq_len(500)] <- NA
    p4 <- vis_clus(
        spe = spe,
        clustervar = "tmp",
        sampleid = "151673",
        colors = libd_layer_colors,
        na_color = "white",
        ... = " LIBD Layers"
    )
    print(p4)
```

```
}
```

---

`vis_clus_p` *Sample spatial cluster visualization workhorse function*

---

## Description

This function visualizes the clusters for one given sample at the spot-level using (by default) the histology information on the background. This is the function that does all the plotting behind `vis_clus()`. To visualize gene-level (or any continuous variable) use `vis_gene_p()`.

## Usage

```
vis_clus_p(
  spe,
  d,
  clustervar,
  sampleid = unique(spe$sample_id)[1],
  colors,
  spatial,
  title,
  image_id = "lowres",
  alpha = NA,
  point_size = 2,
  auto_crop = TRUE,
  na_color = "#CCCCCC40"
)
```

## Arguments

| | |
|---|---|
| `spe` | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| `d` | A data.frame with the sample-level information. This is typically obtained using `cbind(colData(spe), spatialCoords(spe))`. |
| `clustervar` | A `character(1)` with the name of the `colData(spe)` column that has the cluster values. |
| `sampleid` | A `character(1)` specifying which sample to plot from `colData(spe)$sample_id` (formerly `colData(spe)$sample_name`). |
| `colors` | A vector of colors to use for visualizing the clusters from `clustervar`. If the vector has names, then those should match the values of `clustervar`. |
| `spatial` | A `logical(1)` indicating whether to include the histology layer from `geom_spatial()`. If you plan to use ggplotly() then it's best to set this to `FALSE`. |
| `title` | The title for the plot. |

| | |
|---|---|
| image_id | A character(1) with the name of the image ID you want to use in the background. |
| alpha | A numeric(1) in the [0, 1] range that specifies the transparency level of the data on the spots. |
| point_size | A numeric(1) specifying the size of the points. Defaults to 1.25. Some colors look better if you use 2 for instance. |
| auto_crop | A logical(1) indicating whether to automatically crop the image / plotting area, which is useful if the Visium capture area is not centered on the image and if the image is not a square. |
| na_color | A character(1) specifying a color for the NA values. If you set alpha = NA then it's best to set na_color to a color that has alpha blending already, which will make non-NA values pop up more and the NA values will show with a lighter color. This behavior is lost when alpha is set to a non-NA value. |

## Value

A ggplot2 object.

## See Also

Other Spatial cluster visualization functions: frame_limits(), vis_clus(), vis_grid_clus()

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")
    spe_sub <- spe[, spe$sample_id == "151673"]

    ## Use the manual color palette by Lukas M Weber
    ## Don't plot the histology information
    p <- vis_clus_p(
        spe = spe_sub,
     d = as.data.frame(cbind(colData(spe_sub), SpatialExperiment::spatialCoords(spe_sub)), optional = TRUE),
        clustervar = "layer_guess_reordered",
        sampleid = "151673",
        colors = libd_layer_colors,
        title = "151673 LIBD Layers",
        spatial = FALSE
    )
    print(p)

    ## Clean up
    rm(spe_sub)
}
```

vis_gene *Sample spatial gene visualization*

## Description

This function visualizes the gene expression stored in `assays(spe)` or any continuous variable stored in `colData(spe)` for one given sample at the spot-level using (by default) the histology information on the background. To visualize clusters (or any discrete variable) use `vis_clus()`.

## Usage

```
vis_gene(
  spe,
  sampleid = unique(spe$sample_id)[1],
  geneid = rowData(spe)$gene_search[1],
  spatial = TRUE,
  assayname = "logcounts",
  minCount = 0,
  viridis = TRUE,
  image_id = "lowres",
  alpha = NA,
  cont_colors = if (viridis) viridisLite::viridis(21) else c("aquamarine4",
    "springgreen", "goldenrod", "red"),
  point_size = 2,
  auto_crop = TRUE,
  na_color = "#CCCCCC40",
  multi_gene_method = c("z_score", "pca", "sparsity"),
  ...
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| sampleid | A `character(1)` specifying which sample to plot from `colData(spe)$sample_id` (formerly `colData(spe)$sample_name`). |
| geneid | A `character()` specifying the gene ID(s) stored in `rowData(spe)$gene_search` or a continuous variable(s) stored in `colData(spe)` to visualize. For each ID, if `rowData(spe)$gene_search` is missing, then `rownames(spe)` is used to search for the gene ID. When a vector of length > 1 is supplied, the continuous variables are combined according to `multi_gene_method`, producing a single value for each spot. |
| spatial | A `logical(1)` indicating whether to include the histology layer from `geom_spatial()`. If you plan to use ggplotly() then it's best to set this to `FALSE`. |

assayname — The name of the assays(spe) to use for extracting the gene expression data. Defaults to logcounts.

minCount — A numeric(1) specifying the minimum gene expression (or value in the continuous variable) to visualize. Values at or below this threshold will be set to NA. Defaults to 0.

viridis — A logical(1) whether to use the color-blind friendly palette from viridis or the color palette used in the paper that was chosen for contrast when visualizing the data on top of the histology image. One issue is being able to differentiate low values from NA ones due to the purple-ish histology information that is dependent on cell density.

image_id — A character(1) with the name of the image ID you want to use in the background.

alpha — A numeric(1) in the [0, 1] range that specifies the transparency level of the data on the spots.

cont_colors — A character() vector of colors that supersedes the viridis argument.

point_size — A numeric(1) specifying the size of the points. Defaults to 1.25. Some colors look better if you use 2 for instance.

auto_crop — A logical(1) indicating whether to automatically crop the image / plotting area, which is useful if the Visium capture area is not centered on the image and if the image is not a square.

na_color — A character(1) specifying a color for the NA values. If you set alpha = NA then it's best to set na_color to a color that has alpha blending already, which will make non-NA values pop up more and the NA values will show with a lighter color. This behavior is lost when alpha is set to a non-NA value.

multi_gene_method — A character(1): either "pca", "sparsity", or "z_score". This parameter controls how multiple continuous variables are combined for visualization, and only applies when geneid has length great than 1. z_score: to summarize multiple continuous variables, each is normalized to represent a Z-score. The multiple scores are then averaged. pca: PCA dimension reduction is conducted on the matrix formed by the continuous variables, and the first PC is then used and multiplied by -1 if needed to have the majority of the values for PC1 to be positive. sparsity: the proportion of continuous variables with positive values for each spot is computed. For more details, check the multi gene vignette at https://research.libd.org/spatialLIBD/articles/multi_gene_plots.html.

... — Passed to paste0() for making the title of the plot following the sampleid.

## Details

This function subsets spe to the given sample and prepares the data and title for vis_gene_p(). It also adds a caption to the plot.

## Value

A ggplot2 object.

## See Also

Other Spatial gene visualization functions: vis_gene_p(), vis_grid_gene()

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Valid `geneid` values are those in
    head(rowData(spe)$gene_search)
    ## or continuous variables stored in colData(spe)
    ## or rownames(spe)

    ## Visualize a default gene on the non-viridis scale
    p1 <- vis_gene(
        spe = spe,
        sampleid = "151507",
        viridis = FALSE
    )
    print(p1)

    ## Use a custom set of colors in the reverse order than usual
    p2 <- vis_gene(
        spe = spe,
        sampleid = "151507",
        cont_colors = rev(viridisLite::viridis(21, option = "magma"))
    )
    print(p2)

    ## Turn the alpha to 1, which makes the NA values have a full alpha
    p2b <- vis_gene(
        spe = spe,
        sampleid = "151507",
        cont_colors = rev(viridisLite::viridis(21, option = "magma")),
        alpha = 1
    )
    print(p2b)

    ## Turn the alpha to NA, and use an alpha-blended "forestgreen" for
    ## the NA values
   # https://gist.githubusercontent.com/mages/5339689/raw/2aaa482dfbbecbfcb726525a3d81661f9d802a8e/add.alpha.R
    # add.alpha("forestgreen", 0.5)
    p2c <- vis_gene(
        spe = spe,
        sampleid = "151507",
        cont_colors = rev(viridisLite::viridis(21, option = "magma")),
        alpha = NA,
        na_color = "#228B2280"
    )
    print(p2c)
```

```
## Visualize a continuous variable, in this case, the ratio of chrM
## gene expression compared to the total expression at the spot-level
p3 <- vis_gene(
    spe = spe,
    sampleid = "151507",
    geneid = "expr_chrM_ratio"
)
print(p3)

## Visualize a gene using the rownames(spe)
p4 <- vis_gene(
    spe = spe,
    sampleid = "151507",
    geneid = rownames(spe)[which(rowData(spe)$gene_name == "MOBP")]
)
print(p4)

## Repeat without auto-cropping the image
p5 <- vis_gene(
    spe = spe,
    sampleid = "151507",
    geneid = rownames(spe)[which(rowData(spe)$gene_name == "MOBP")],
    auto_crop = FALSE
)
print(p5)

#    Define several markers for white matter
white_matter_genes <- c(
    "ENSG00000197971", "ENSG00000131095", "ENSG00000123560",
    "ENSG00000171885"
)

## Plot all white matter markers at once using the Z-score combination
## method
p6 <- vis_gene(
    spe = spe,
    sampleid = "151507",
    geneid = white_matter_genes,
    multi_gene_method = "z_score"
)
print(p6)

## Plot all white matter markers at once using the sparsity combination
## method
p7 <- vis_gene(
    spe = spe,
    sampleid = "151507",
    geneid = white_matter_genes,
    multi_gene_method = "sparsity"
)
print(p7)

## Plot all white matter markers at once using the PCA combination
```

```
    ## method
    p8 <- vis_gene(
        spe = spe,
        sampleid = "151507",
        geneid = white_matter_genes,
        multi_gene_method = "pca"
    )
    print(p8)
}
```

---

vis_gene_p *Sample spatial gene visualization workhorse function*

---

## Description

This function visualizes the gene expression stored in `assays(spe)` or any continuous variable stored in `colData(spe)` for one given sample at the spot-level using (by default) the histology information on the background. This is the function that does all the plotting behind `vis_gene()`. To visualize clusters (or any discrete variable) use `vis_clus_p()`.

## Usage

```
vis_gene_p(
  spe,
  d,
  sampleid = unique(spe$sample_id)[1],
  spatial,
  title,
  viridis = TRUE,
  image_id = "lowres",
  alpha = NA,
  cont_colors = if (viridis) {
     viridisLite::viridis(21)
 } else {

    c("aquamarine4", "springgreen", "goldenrod", "red")
 },
  point_size = 2,
  auto_crop = TRUE,
  na_color = "#CCCCCC40",
  legend_title = ""
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |

| | |
|---|---|
| d | A data.frame with the sample-level information. This is typically obtained using `cbind(colData(spe), spatialCoords(spe))`. The data.frame has to contain a column with the continuous variable data to plot stored under `d$COUNT`. |
| sampleid | A `character(1)` specifying which sample to plot from `colData(spe)$sample_id` (formerly `colData(spe)$sample_name`). |
| spatial | A `logical(1)` indicating whether to include the histology layer from `geom_spatial()`. If you plan to use ggplotly() then it's best to set this to `FALSE`. |
| title | The title for the plot. |
| viridis | A `logical(1)` whether to use the color-blind friendly palette from viridis or the color palette used in the paper that was chosen for contrast when visualizing the data on top of the histology image. One issue is being able to differentiate low values from NA ones due to the purple-ish histology information that is dependent on cell density. |
| image_id | A `character(1)` with the name of the image ID you want to use in the background. |
| alpha | A `numeric(1)` in the `[0, 1]` range that specifies the transparency level of the data on the spots. |
| cont_colors | A `character()` vector of colors that supersedes the `viridis` argument. |
| point_size | A `numeric(1)` specifying the size of the points. Defaults to `1.25`. Some colors look better if you use `2` for instance. |
| auto_crop | A `logical(1)` indicating whether to automatically crop the image / plotting area, which is useful if the Visium capture area is not centered on the image and if the image is not a square. |
| na_color | A `character(1)` specifying a color for the NA values. If you set `alpha = NA` then it's best to set `na_color` to a color that has alpha blending already, which will make non-NA values pop up more and the NA values will show with a lighter color. This behavior is lost when `alpha` is set to a non-`NA` value. |
| legend_title | A `character(1)` specifying the legend title. |

## Value

A ggplot2 object.

## See Also

Other Spatial gene visualization functions: `vis_gene()`, `vis_grid_gene()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Prepare the data for the plotting function
    spe_sub <- spe[, spe$sample_id == "151673"]
  df <- as.data.frame(cbind(colData(spe_sub), SpatialExperiment::spatialCoords(spe_sub)), optional = TRUE)
```

```
    df$COUNT <- df$expr_chrM_ratio

    ## Use the manual color palette by Lukas M Weber
    ## Don't plot the histology information
    p <- vis_gene_p(
        spe = spe_sub,
        d = df,
        sampleid = "151673",
        title = "151673 chrM expr ratio",
        spatial = FALSE
    )
    print(p)

    ## Clean up
    rm(spe_sub)
}
```

---

`vis_grid_clus` *Sample spatial cluster visualization grid*

---

## Description

This function visualizes the clusters for a set of samples at the spot-level using (by default) the histology information on the background. To visualize gene-level (or any continuous variable) use `vis_grid_gene()`.

## Usage

```
vis_grid_clus(
  spe,
  clustervar,
  pdf_file,
  sort_clust = TRUE,
  colors = NULL,
  return_plots = FALSE,
  spatial = TRUE,
  height = 24,
  width = 36,
  image_id = "lowres",
  alpha = NA,
  sample_order = unique(spe$sample_id),
  point_size = 2,
  auto_crop = TRUE,
  na_color = "#CCCCCC40",
  ...
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| clustervar | A `character(1)` with the name of the `colData(spe)` column that has the cluster values. |
| pdf_file | A `character(1)` specifying the path for the resulting PDF. |
| sort_clust | A `logical(1)` indicating whether you want to sort the clusters by frequency using `sort_clusters()`. |
| colors | A vector of colors to use for visualizing the clusters from `clustervar`. If the vector has names, then those should match the values of `clustervar`. |
| return_plots | A `logical(1)` indicating whether to print the plots to a PDF or to return the list of plots that you can then print using plot_grid. |
| spatial | A `logical(1)` indicating whether to include the histology layer from `geom_spatial()`. If you plan to use ggplotly() then it's best to set this to `FALSE`. |
| height | A `numeric(1)` passed to pdf. |
| width | A `numeric(1)` passed to pdf. |
| image_id | A `character(1)` with the name of the image ID you want to use in the background. |
| alpha | A `numeric(1)` in the `[0, 1]` range that specifies the transparency level of the data on the spots. |
| sample_order | A `character()` with the names of the samples to use and their order. |
| point_size | A `numeric(1)` specifying the size of the points. Defaults to `1.25`. Some colors look better if you use `2` for instance. |
| auto_crop | A `logical(1)` indicating whether to automatically crop the image / plotting area, which is useful if the Visium capture area is not centered on the image and if the image is not a square. |
| na_color | A `character(1)` specifying a color for the NA values. If you set `alpha = NA` then it's best to set `na_color` to a color that has alpha blending already, which will make non-NA values pop up more and the NA values will show with a lighter color. This behavior is lost when `alpha` is set to a non-`NA` value. |
| ... | Passed to paste0() for making the title of the plot following the `sampleid`. |

## Details

This function prepares the data and then loops through `vis_clus()` for computing the list of ggplot2 objects.

## Value

A list of ggplot2 objects.

## See Also

Other Spatial cluster visualization functions: `frame_limits()`, `vis_clus()`, `vis_clus_p()`

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Subset to two samples of interest and obtain the plot list
    p_list <-
        vis_grid_clus(
            spe[, spe$sample_id %in% c("151673", "151674")],
            "layer_guess_reordered",
            spatial = FALSE,
            return_plots = TRUE,
            sort_clust = FALSE,
            colors = libd_layer_colors
        )

    ## Visualize the spatial adjacent replicates for position = 0 micro meters
    ## for subject 3
    cowplot::plot_grid(plotlist = p_list, ncol = 2)
}
```

---

# vis_grid_gene *Sample spatial gene visualization grid*

---

## Description

This function visualizes the gene expression stored in `assays(spe)` or any continuous variable stored in `colData(spe)` for a set of samples at the spot-level using (by default) the histology information on the background. To visualize clusters (or any discrete variable) use `vis_grid_clus()`.

## Usage

```
vis_grid_gene(
  spe,
  geneid = rowData(spe)$gene_search[1],
  pdf_file,
  assayname = "logcounts",
  minCount = 0,
  return_plots = FALSE,
  spatial = TRUE,
  viridis = TRUE,
  height = 24,
  width = 36,
  image_id = "lowres",
  alpha = NA,
  cont_colors = if (viridis) viridisLite::viridis(21) else c("aquamarine4",
    "springgreen", "goldenrod", "red"),
  sample_order = unique(spe$sample_id),
```

```
  point_size = 2,
  auto_crop = TRUE,
  na_color = "#CCCCCC40",
  ...
)
```

## Arguments

| | |
|---|---|
| spe | Defaults to the output of `fetch_data(type = 'spe')`. This is a SpatialExperiment-class object with the spot-level Visium data and information required for visualizing the histology. See `fetch_data()` for more details. |
| geneid | A `character()` specifying the gene ID(s) stored in `rowData(spe)$gene_search` or a continuous variable(s) stored in `colData(spe)` to visualize. For each ID, if `rowData(spe)$gene_search` is missing, then `rownames(spe)` is used to search for the gene ID. When a vector of length > 1 is supplied, the continuous variables are combined according to `multi_gene_method`, producing a single value for each spot. |
| pdf_file | A `character(1)` specifying the path for the resulting PDF. |
| assayname | The name of the `assays(spe)` to use for extracting the gene expression data. Defaults to `logcounts`. |
| minCount | A `numeric(1)` specifying the minimum gene expression (or value in the continuous variable) to visualize. Values at or below this threshold will be set to `NA`. Defaults to `0`. |
| return_plots | A `logical(1)` indicating whether to print the plots to a PDF or to return the list of plots that you can then print using plot_grid. |
| spatial | A `logical(1)` indicating whether to include the histology layer from `geom_spatial()`. If you plan to use ggplotly() then it's best to set this to `FALSE`. |
| viridis | A `logical(1)` whether to use the color-blind friendly palette from viridis or the color palette used in the paper that was chosen for contrast when visualizing the data on top of the histology image. One issue is being able to differentiate low values from NA ones due to the purple-ish histology information that is dependent on cell density. |
| height | A `numeric(1)` passed to pdf. |
| width | A `numeric(1)` passed to pdf. |
| image_id | A `character(1)` with the name of the image ID you want to use in the background. |
| alpha | A `numeric(1)` in the `[0, 1]` range that specifies the transparency level of the data on the spots. |
| cont_colors | A `character()` vector of colors that supersedes the `viridis` argument. |
| sample_order | A `character()` with the names of the samples to use and their order. |
| point_size | A `numeric(1)` specifying the size of the points. Defaults to `1.25`. Some colors look better if you use `2` for instance. |
| auto_crop | A `logical(1)` indicating whether to automatically crop the image / plotting area, which is useful if the Visium capture area is not centered on the image and if the image is not a square. |

| | |
|---|---|
| na_color | A character(1) specifying a color for the NA values. If you set alpha = NA then it's best to set na_color to a color that has alpha blending already, which will make non-NA values pop up more and the NA values will show with a lighter color. This behavior is lost when alpha is set to a non-NA value. |
| ... | Passed to paste0() for making the title of the plot following the sampleid. |

## Details

This function prepares the data and then loops through vis_gene() for computing the list of ggplot2 objects.

## Value

A list of ggplot2 objects.

## See Also

Other Spatial gene visualization functions: vis_gene(), vis_gene_p()

## Examples

```
if (enough_ram()) {
    ## Obtain the necessary data
    if (!exists("spe")) spe <- fetch_data("spe")

    ## Subset to two samples of interest and obtain the plot list
    p_list <-
        vis_grid_gene(
            spe[, spe$sample_id %in% c("151673", "151674")],
            spatial = FALSE,
            return_plots = TRUE
        )

    ## Visualize the spatial adjacent replicates for position = 0 micro meters
    ## for subject 3
    cowplot::plot_grid(plotlist = p_list, ncol = 2)
}
```

# Index


∗ **Check input functions**
check_modeling_results, 9
check_sce, 10
check_sce_layer, 11
check_spe, 11
∗ **Functions for adding non-standard images**
add_images, 5
locate_images, 38
∗ **Gene set enrichment functions**
gene_set_enrichment, 19
gene_set_enrichment_plot, 20
∗ **Genomics**
add10xVisiumAnalysis, 4
read10xVisiumAnalysis, 40
read10xVisiumWrapper, 41
∗ **Image editing functions**
img_edit, 25
img_update, 27
img_update_all, 28
∗ **Layer correlation functions**
annotate_registered_clusters, 7
layer_stat_cor, 33
layer_stat_cor_plot, 35
∗ **Layer modeling functions**
layer_boxplot, 29
sig_genes_extract, 58
sig_genes_extract_all, 60
∗ **Spatial cluster visualization functions**
frame_limits, 17
vis_clus, 62
vis_clus_p, 65
vis_grid_clus, 73
∗ **Spatial gene visualization functions**
vis_gene, 67
vis_gene_p, 71
vis_grid_gene, 75
∗ **SpatialExperiment-related functions**
sce_to_spe, 57
∗ **Utility functions for reading data from SpaceRanger output by 10x**
add10xVisiumAnalysis, 4
read10xVisiumAnalysis, 40
read10xVisiumWrapper, 41
∗ **cluster export/import utility functions**
cluster_export, 12
cluster_import, 13
∗ **datasets**
libd_layer_colors, 37
tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer, 62
∗ **functions for summarizing expression of multiple continuous variables simultaneously**
multi_gene_pca, 38
multi_gene_sparsity, 39
multi_gene_z_score, 40
∗ **internal**
multi_gene_pca, 38
multi_gene_sparsity, 39
multi_gene_z_score, 40
spatialLIBD-package, 3
∗ **spatial registration and statistical modeling functions**
registration_block_cor, 43
registration_model, 44
registration_pseudobulk, 45
registration_stats_anova, 46
registration_stats_enrichment, 48
registration_stats_pairwise, 49
registration_wrapper, 51

add10xVisiumAnalysis, 4, *41, 42*
add_images, 5, *38*
add_key, 6
annotate_registered_clusters, 7, *35, 36*

BiocFileCache-class, *16*
BiocFileCache::bfcrpath(), *15*

check_modeling_results, 9, *10–12*
check_sce, *9*, 10, *11, 12*
check_sce_layer, *9, 10*, 11, *12*
check_spe, *9–11*, 11
cluster_export, 12, *14*
cluster_import, *13*, 13

DataFrame-class, *60*

enough_ram, 15
ExperimentHub-class, *16*

fetch_data, 15
fetch_data(), *5*, *9–14*, *18, 19*, *26–28*, *30*, *34*, *38*, *53, 54*, *57–60*, *63*, *65*, *67*, *71*, *74*, *76*
fields::image.plot(), *33*
frame_limits, 17, *63*, *66*, *74*

gene_set_enrichment, 19, *22*
gene_set_enrichment(), *21*
gene_set_enrichment_plot, *20*, 20
gene_set_enrichment_plot(), *32*
geom_spatial, 23
geom_spatial(), *63*, *65*, *67*, *72*, *74*, *76*
get_colors, 24
ggplot2, *63*, *66*, *68*, *72*, *74*, *77*
ggplot2::layer(), *23*
ggplotly(), *63*, *65*, *67*, *72*, *74*, *76*
graphics::par(), *33*

img_edit, 25, *28, 29*
img_update, *27*, 27, *29*
img_update_all, *27, 28*, 28

layer_boxplot, 29, *59, 60*
layer_boxplot(), *58*, *60*
layer_matrix_plot, 32
layer_matrix_plot(), *21*, *36*
layer_stat_cor, *8*, 33, *36*
layer_stat_cor(), *7*, *35, 36*, *62*
layer_stat_cor_plot, *8*, *35*, 35
layer_stat_cor_plot(), *32, 33*, *62*
libd_layer_colors, 37
locate_images, *6*, 38

magick::enhance, *26*
magick::equalize, *26*
magick::image_background, *26*
magick::image_channel, *26*
magick::image_contrast, *26*
magick::image_median, *26*
magick::image_modulate, *26*
magick::image_quantize, *26*
magick::image_read, *27*
magick::image_transparent, *26*
magick::negate, *26*
magick::normalize, *26*
multi_gene_pca, 38, *39, 40*
multi_gene_sparsity, *39*, 39, *40*
multi_gene_z_score, *39*, 40

paste0(), *63*, *68*, *74*, *77*
pdf, *74*, *76*
plot_grid, *74*, *76*

read10xVisiumAnalysis, *4*, 40, *42*
read10xVisiumWrapper, *4*, *41*, 41
registration_block_cor, 43, *44, 45*, *47*, *49, 50*, *52*
registration_model, *43*, 44, *45*, *47*, *49, 50*, *52*
registration_pseudobulk, *43, 44*, 45, *47*, *49, 50*, *52*
registration_stats_anova, *43–45*, 46, *49, 50*, *52*
registration_stats_enrichment, *43–45*, *47*, 48, *50*, *52*
registration_stats_pairwise, *43–45*, *47*, *49*, 49, *52*
registration_wrapper, *43–45*, *47*, *49, 50*, 51
run_app, 53

sce_to_spe, 57
shiny.appobj, *54*
sig_genes_extract, *30*, 58, *60*
sig_genes_extract(), *60*
sig_genes_extract_all, *30*, *59*, 60
sig_genes_extract_all(), *29, 30*, *54*, *58*
SingleCellExperiment, *10, 11*, *16*, *30*, *53*, *57*, *59, 60*
SingleCellExperiment-class, *45*, *51*, *57*
sort_clusters, 61
sort_clusters(), *74*
SpatialExperiment, *42*
SpatialExperiment-class, *4–7*, *10*, *12–14*, *16*, *18*, *26–29*, *38*, *53*, *57, 58*, *63*, *65*, *67*, *71*, *74*, *76*

SpatialExperiment::imgData, *57*
SpatialExperiment::read10xVisium(), *40–42*
spatialLIBD *(*spatialLIBD-package*)*, 3
spatialLIBD-package, 3
stats::fisher.test(), *20*

tstats_Human_DLPFC_snRNAseq_Nguyen_topLayer, 62

unname(), *25*

viridis, *68*, *72*, *76*
vis_clus, *18*, 62, *66*, *74*
vis_clus(), *65*, *67*, *74*
vis_clus_p, *18*, *63*, 65, *74*
vis_clus_p(), *23*, *63*, *71*
vis_gene, 67, *72*, *77*
vis_gene(), *62*, *71*, *77*
vis_gene_p, *69*, 71, *77*
vis_gene_p(), *23*, *65*, *68*
vis_grid_clus, *18*, *63*, *66*, 73
vis_grid_clus(), *75*
vis_grid_gene, *69*, *72*, 75
vis_grid_gene(), *73*