

---

# **sc-toolbox**

***Release 0.12.2***

**Lukas Heumos**

**Jun 12, 2023**



## CONTENTS:

<b>1</b>	<b>Installation</b>	<b>3</b>
1.1	Stable release . . . . .	3
1.2	From sources . . . . .	3
<b>2</b>	<b>Usage</b>	<b>5</b>
2.1	Preprocessing . . . . .	5
2.2	Tools . . . . .	6
2.2.1	sc_toolbox.tools.generate_expression_table . . . . .	6
2.2.2	sc_toolbox.tools.relative_frequencies . . . . .	7
2.2.3	sc_toolbox.tools.relative_frequency_per_cluster . . . . .	7
2.2.4	sc_toolbox.tools.correlate_to_signature . . . . .	7
2.2.5	sc_toolbox.tools.remove_outliers . . . . .	8
2.2.6	sc_toolbox.tools.add_percentages . . . . .	8
2.2.7	sc_toolbox.tools.ranksums_between_groups . . . . .	8
2.2.8	sc_toolbox.tools.generate_count_object . . . . .	9
2.2.9	sc_toolbox.tools.tidy_de_table . . . . .	9
2.2.10	sc_toolbox.tools.correlate_means_to_gene . . . . .	10
2.2.11	sc_toolbox.tools.extended_marker_table . . . . .	10
2.2.12	sc_toolbox.tools.generate_pseudobulk . . . . .	10
2.2.13	sc_toolbox.tools.automated_marker_annotation . . . . .	11
2.2.14	sc_toolbox.tools.de_res_to_anndata . . . . .	12
2.3	Plots . . . . .	13
2.3.1	sc_toolbox.plot.Colormaps . . . . .	13
2.3.2	sc_toolbox.plot.custom_plot_size . . . . .	14
2.3.3	sc_toolbox.plot.standard_lineplot . . . . .	14
2.3.4	sc_toolbox.plot.average_expression . . . . .	15
2.3.5	sc_toolbox.plot.average_expression_per_cluster . . . . .	16
2.3.6	sc_toolbox.plot.average_expression_split_cluster . . . . .	17
2.3.7	sc_toolbox.plot.average_expression_per_cell . . . . .	19
2.3.8	sc_toolbox.plot.gene_expression_dpt_ordered . . . . .	19
2.3.9	sc_toolbox.plot.relative_frequencies_boxplots . . . . .	20
2.3.10	sc_toolbox.plot.split_boxplot . . . . .	21
2.3.11	sc_toolbox.plot.marker_dendrogram . . . . .	22
2.3.12	sc_toolbox.plot.volcano_plot . . . . .	23
2.3.13	sc_toolbox.plot.cluster_composition_stacked_barplot . . . . .	24
2.3.14	sc_toolbox.plot.gene_boxplot . . . . .	25
2.3.15	sc_toolbox.plot.colors_overview . . . . .	26
2.3.16	sc_toolbox.plot.relative_frequencies_lineplot . . . . .	27
2.3.17	sc_toolbox.plot.annotated_cell_type_umap . . . . .	28
2.3.18	sc_toolbox.plot.genotype_vs_genotype_umaps . . . . .	29

<b>3</b>	<b>Contributor Guide</b>	<b>31</b>
3.1	How to report a bug . . . . .	31
3.2	How to request a feature . . . . .	31
3.3	Getting the code . . . . .	31
3.4	How to set up your development environment . . . . .	31
3.5	How to test the project . . . . .	32
3.6	How to build and view the documentation . . . . .	32
3.7	How to submit changes . . . . .	33
<b>4</b>	<b>Contributor Covenant Code of Conduct</b>	<b>35</b>
4.1	Our Pledge . . . . .	35
4.2	Our Standards . . . . .	35
4.3	Our Responsibilities . . . . .	35
4.4	Scope . . . . .	36
4.5	Enforcement . . . . .	36
4.6	Attribution . . . . .	36
<b>5</b>	<b>Credits</b>	<b>37</b>
5.1	Development Lead . . . . .	37
5.2	Contributors . . . . .	37
<b>6</b>	<b>Indices and tables</b>	<b>39</b>
	<b>Index</b>	<b>41</b>

Installation New to *sc-toolbox*? Check out the installation guide.

API reference The API reference contains a detailed description of the sc-toolbox API.

Discussion Need help? Reach out on our forum to get your questions answered!

GitHub Find a bug? Interested in improving sc-toolbox? Checkout our GitHub for the latest developments.



## INSTALLATION

### 1.1 Stable release

To install sc-toolbox, run this command in your terminal:

```
$ pip install sc-toolbox
```

This is the preferred method to install sc-toolbox, as it will always install the most recent stable release.

If you don't have [pip](#) installed, this [Python installation guide](#) can guide you through the process.

### 1.2 From sources

The sources for sc-toolbox can be downloaded from the [Github repo](#). Please note that you require [poetry](#) to be installed.

You can either clone the public repository:

```
$ git clone git://github.com/schillerlab/sc-toolbox
```

Or download the [tarball](#):

```
$ curl -OJL https://github.com/schillerlab/sc-toolbox/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ make install
```





Import the sc-toolbox API as follows:

```
import sc_toolbox as sct
```

You can then access the respective modules like:

```
sct.pl.cool_fancy_plot()
```

## 2.1 Preprocessing

## 2.2 Tools

<code>tools.generate_expression_table</code>	Generates a table of cells by genes of expression values as a Pandas DataFrame.
<code>tools.relative_frequencies</code>	Calculates the relative frequencies of conditions grouped by an observation.
<code>tools.relative_frequency_per_cluster</code>	Calculates relative frequencies per cluster
<code>tools.correlate_to_signature</code>	Correlations Score (based on cell type signature (logFC)) - alternative to <code>sc.tl.score</code>
<code>tools.remove_outliers</code>	Remove outlying cells based on UMAP embeddings with DBScan (density based clustering).
<code>tools.add_percentages</code>	Add columns to existing diffxpy table specifying percentage of expressing cells.
<code>tools.ranksums_between_groups</code>	Perform Wilcoxon Rank-sum test between two groups.
<code>tools.generate_count_object</code>	@Meshal what is this really supposed to do?
<code>tools.tidy_de_table</code>	Sorts diffxpy de table and adds percentages of expression per group
<code>tools.correlate_means_to_gene</code>	Calculate gene to gene correlation based on a mean expression table
<code>tools.extended_marker_table</code>	Generates an extended marker table with cell types and percentages of expressed cell types per cluster.
<code>tools.generate_pseudobulk</code>	Generates a pseudobulk for a given key of groups in the AnnData object.
<code>tools.automated_marker_annotation</code>	Calculates a marker gene overlap based on pre-existing annotations.
<code>tools.de_res_to_anndata</code>	Add a tabular differential expression result to AnnData as if it was produced by <code>scanpy.tl.rank_genes_groups</code> .

### 2.2.1 `sc_toolbox.tools.generate_expression_table`

`sc_toolbox.tools.generate_expression_table(adata, cluster='all', subset_by='cell_type', xlabel=None, condition=None, use_raw=None)`

Generates a table of cells by genes of expression values as a Pandas DataFrame.

#### Parameters

- **adata** – Anndata object
- **cluster** (`str`) – Which label of the subsets to generate the table for. Use ‘all’ if for all subsets.
- **subset\_by** (`str`) – Which label to subset the clusters by
- **xlabel** (`Optional[str]`) – Label that will be used for subsequent line plots as x-axis label. Typically a time series such as “days”.
- **condition** (`Optional[str]`) – Column name of the condition to include.
- **use\_raw** (`Optional[bool]`) – Whether to use `adata.raw.X` for the calculations

#### Returns

Gene expression table.

## 2.2.2 `sc_toolbox.tools.relative_frequencies`

`sc_toolbox.tools.relative_frequencies(adata, group_by='cell_type', xlabel='days', condition='batch')`

Calculates the relative frequencies of conditions grouped by an observation.

### Parameters

- **adata** – AnnData Object containing the data
- **group\_by** (`str`) – Column name to group by
- **xlabel** (`str`) – x-axis label
- **condition** (`str`) –

### Returns

Relative frequencies in a Pandas DataFrame

## 2.2.3 `sc_toolbox.tools.relative_frequency_per_cluster`

`sc_toolbox.tools.relative_frequency_per_cluster(adata, group_by='cell_type', xlabel='days', condition=None)`

Calculates relative frequencies per cluster

### Parameters

- **adata** – AnnData object containing the data
- **group\_by** (`str`) – The label to group by for the clusters
- **xlabel** (`str`) – x-axis label
- **condition** – condition to combine by

### Returns

Pandas DataFrame of relative frequencies

## 2.2.4 `sc_toolbox.tools.correlate_to_signature`

`sc_toolbox.tools.correlate_to_signature(adata, marker, log_fc_threshold=0.7, cell_type='AT2 cells', cell_type_label='cell_type', log_fc_label='logfoldchange', gene_label='gene', use_raw=True)`

Correlations Score (based on cell type signature (logFC)) - alternative to `sc.tl.score`

### Parameters

- **adata** – AnnData object containing the data
- **marker** (`DataFrame`) – Pandas DataFrame containing marker genes
- **log\_fc\_threshold** (`float`) – Log fold change label
- **cell\_type** (`str`) – Cell type to calculate the correlation for
- **cell\_type\_label** (`str`) – Label of all cell types in the AnnData object
- **log\_fc\_label** (`str`) – Label of fold change in the AnnData object
- **gene\_label** (`str`) – Label of genes in the AnnData object
- **use\_raw** (`bool`) – Whether to use `adata.raw.X`

**Returns**

List of correlations

## 2.2.5 `sc_toolbox.tools.remove_outliers`

`sc_toolbox.tools.remove_outliers(cords, eps=1, min_samples=2)`

Remove outlying cells based on UMAP embeddings with DBScan (density based clustering).

Call as: `sub.obs[“d_cluster”] = remove_outliers(sub.obsm[“X_umap”], min_samples = 10)`**Parameters**

- **cords** – adata UMAP coordinates, typically `adata.obsm[“X_umap”]`
- **eps** (`int`) – Maximum distance between two clusters to still be considered neighbors
- **min\_samples** (`int`) – Minimum samples of a cluster

**Return type**`Categorical`**Returns**

Pandas Categorical of clusters

## 2.2.6 `sc_toolbox.tools.add_percentages`

`sc_toolbox.tools.add_percentages(adata, table, ids, group_by, threshold=0, gene_label='gene')`

Add columns to existing diffxpy table specifying percentage of expressing cells.

**Parameters**

- **adata** – AnnData object containing the data
- **table** – Table as generated by diffxpy
- **ids** – Identifiers to add percentages for.
- **group\_by** (`str`) – Label to group by
- **threshold** (`int`) – Cell count threshold.
- **gene\_label** (`str`) – Label of the genes

**Returns**

Table containing percentage of expressing cells

## 2.2.7 `sc_toolbox.tools.ranksums_between_groups`

`sc_toolbox.tools.ranksums_between_groups(table, id1='bystander', id2='infected', xlabel='condition',  
cells=None, score='Axin2')`

Perform Wilcoxon Rank-sum test between two groups.

**Parameters**

- **table** –
- **id1** (`str`) –
- **id2** (`str`) –

- **xlabel** (*str*) – x-axis label
- **cells** –
- **score** (*str*) –

**Returns**

Pandas DataFrame containing test statistic and p-value

## 2.2.8 `sc_toolbox.tools.generate_count_object`

```
sc_toolbox.tools.generate_count_object(adata, hue='disease', cell_type_label='cell_type',
                                       cell_type=None, min_samples=2, min_cells=5, ref='healthy',
                                       subset=None, layer='counts', outliers_removal=False)
```

@Meshal what is this really supposed to do?

**Parameters**

- **adata** – AnnData object
- **hue** (*str*) – Value to color by
- **cell\_type\_label** (*str*) – Label containing cell types
- **cell\_type** (*Optional[List[str]]*) – Cells type to generate counts for
- **min\_samples** (*int*) – Minimum samples for outlier removal with DBScan
- **min\_cells** (*int*) – Minimal number of cells
- **ref** (*str*) –
- **subset** (*Optional[List[str]]*) –
- **layer** (*str*) –
- **outliers\_removal** (*bool*) – Whether to remove outliers or not

**Returns**

AnnData object containing counts

Example Call: `subset = ['3d PI-KO', '3d PI-WT']`

```
raw_counts = generate_count_object(adata,
                                   condition = "grouping", cell_type_label = "celltype_refined", cell_type = ["AT2"], ref = "3d PI-WT",
                                   subset = subset)
```

## 2.2.9 `sc_toolbox.tools.tidy_de_table`

```
sc_toolbox.tools.tidy_de_table(de_test, adata, cells, ids=None, qval_thresh=0.9, group_by='treatment',
                              cols=None)
```

Sorts diffxpy de table and adds percentages of expression per group

**Parameters**

- **de\_test** – diffxpy de test
- **adata** – AnnData object
- **cells** –
- **ids** –

- **qval\_thresh** (*float*) –
- **group\_by** (*str*) –
- **cols** –

**Returns**

Pandas Dataframe of diffxpy table with percentages

## 2.2.10 `sc_toolbox.tools.correlate_means_to_gene`

`sc_toolbox.tools.correlate_means_to_gene(means, corr_gene='EOMES')`

Calculate gene to gene correlation based on a mean expression table

**Parameters**

- **means** (*DataFrame*) –
- **corr\_gene** (*str*) –

**Returns**

Pandas DataFrame of correlations

## 2.2.11 `sc_toolbox.tools.extended_marker_table`

`sc_toolbox.tools.extended_marker_table(adata, qval_thresh=0.05, cell_type_label='cell_type',  
gene_ranks_key='rank_genes_groups')`

Generates an extended marker table with cell types and percentages of expressed cell types per cluster.

Run `scanpy.tl.rank_genes_groups` before using this function.

**Parameters**

- **adata** (*AnnData*) – AnnData object containing ranked genes
- **qval\_thresh** (*float*) – Threshold to filter the log fold change for
- **cell\_type\_label** (*str*) – Label containing all cell types
- **gene\_ranks\_key** (*str*) – Key for the ranked gene groups (generated by `sc.tl.rank_genes_groups`)

**Returns**

A Pandas DataFrame

## 2.2.12 `sc_toolbox.tools.generate_pseudobulk`

`sc_toolbox.tools.generate_pseudobulk(adata, group_key='identifier', sep='\t', save=None)`

Generates a pseudobulk for a given key of groups in the AnnData object.

Looks like:

Genes	Group Member 1	Group Member 2
Gene 1	Value 1	Value 2
Gene 2	Value 2	Value 3

**Parameters**

- **adata** (`AnnData`) – AnnData object
- **group\_key** (`str`) – The key to group by. E.g. by mice, by condition, ... (default: 'identifier')
- **sep** – Separator to use when saving the pseudobulk table (default: ' ')
- **save** (`Optional[str]`) – Path to save the pseudobulk table to (default: None)

**Return type**`DataFrame`**Returns**

A Pandas DataFrame containing the pseudobulk table

**2.2.13 sc\_toolbox.tools.automated\_marker\_annotation**

`sc_toolbox.tools.automated_marker_annotation(adata, organism, tissue, marker_file,`  
`key='rank_genes_groups', normalize='reference',`  
`p_value=0.05, log_fold_change=2)`

Calculates a marker gene overlap based on pre-existing annotations.

Currently supported marker files:

Organism	Tissue	Marker File
Mouse	Lung	lung_particle_markers.txt
Human	NA	

**Parameters**

- **adata** (`AnnData`) – AnnData object containing ranked genes
- **organism** (`str`) – Currently supported: 'mouse'
- **tissue** (`str`) – Currently supported: 'lung'
- **marker\_file** (`str`) – Name of the marker file to be used - refer to table
- **key** (`str`) – Key of ranked genes in adata (default: 'rank\_genes\_groups')
- **normalize** (`Optional[Literal['reference', 'data']]`) – Normalization option for the marker gene overlap output (default: 'reference')
- **p\_value** (`float`) – p-value threshold for existing marker genes (default: 0.05)
- **log\_fold\_change** (`float`) – log fold change threshold for existing marker genes (default: 2)

**Returns**

Pandas DataFrame of overlapping genes. Visualize with a Seaborn Heatmap

### 2.2.14 `sc_toolbox.tools.de_res_to_anndata`

```
sc_toolbox.tools.de_res_to_anndata(adata, de_res, *, groupby, gene_id_col='gene_symbol',  
                                   score_col='score', pval_col='pvalue', pval_adj_col=None,  
                                   lfc_col='lfc', key_added='rank_genes_groups')
```

Add a tabular differential expression result to AnnData as if it was produced by `scanpy.tl.rank_genes_groups`.

#### Parameters

- **adata** (`AnnData`) – Annotated data matrix
- **de\_res** (`DataFrame`) – Tabular DE result as Pandas DataFrame
- **groupby** (`str`) – Column in *de\_res* that indicates the group. This column must also exist in *adata.obs*.
- **gene\_id\_col** (`str`) – Column in *de\_res* that holds the gene identifiers
- **score\_col** (`str`) – Column in *de\_res* that holds the score (results will be ordered by score).
- **pval\_col** (`str`) – Column in *de\_res* that holds the unadjusted pvalue
- **pval\_adj\_col** (`Optional[str]`) – Column in *de\_res* that holds the adjusted pvalue. If not specified, the unadjusted p values will be FDR-adjusted.
- **lfc\_col** (`str`) – Column in *de\_res* that holds the log fold change
- **key\_added** (`str`) – Key under which the results will be stored in *adata.uns*

#### Return type

`None`



## 2.3 Plots

<code>plot.Colormaps(value)</code>	Useful Colormaps for e.g.
<code>plot.custom_plot_size(width, height, dpi)</code>	Create a custom axis object of desired sizes.
<code>plot.standard_lineplot(data, order, xlabel, ...)</code>	Draws a standard line plot based on Seaborn's Implot.
<code>plot.average_expression(gene_expression, ...)</code>	Draw a line plot showing the gene expression over time.
<code>plot.average_expression_per_cluster(...[, ...])</code>	Plots gene expression over time split by cluster identity.
<code>plot.average_expression_split_cluster(...[, ...])</code>	Plot average gene expression as line plots for multiple clusters at once.
<code>plot.average_expression_per_cell(...[, ...])</code>	Plots the average gene expression as a line plot per cell.
<code>plot.gene_expression_dpt_ordered(data, ...)</code>	Plot smoothed expression of all cells ordered by pseudo time.
<code>plot.relative_frequencies_boxplots(...[, ...])</code>	Plots the relative frequencies as split boxplots.
<code>plot.split_boxplot(table, order, xlabel, ylabel)</code>	Draws a boxsplit split by hue.
<code>plot.marker_dendrogram(marker_table[, ...])</code>	Plots a dendrogram of used marker genes.
<code>plot.volcano_plot(table[, fdr_thresh, ...])</code>	Scatter plot of differential gene expression results generated by diffxpy
<code>plot.cluster_composition_stacked_barplot(...)</code>	Plot relative frequencies as a stacked barplot.
<code>plot.gene_boxplot(table, palette[, xlabel, ...])</code>	Plot gene values as split boxplots.
<code>plot.colors_overview(colors[, ncols, ...])</code>	Draw an overview plot of all used colors.
<code>plot.relative_frequencies_lineplot(...[, ...])</code>	Plot relative frequencies as a line plot.
<code>plot.annotated_cell_type_umap(adata, ...[, ...])</code>	Plots a UMAP which is colored by the primary_color, but also draws all labels on top of all clusters.
<code>plot.genotype_vs_genotype_umaps(adata, ...)</code>	Plots a two UMAPs of genotypes next to each other displaying only the colors of the second UMAP.

### 2.3.1 sc\_toolbox.plot.Colormaps

**class** `sc_toolbox.plot.Colormaps`(*value*)

Useful Colormaps for e.g. UMAPs.

#### Attributes table

<code>grey_red</code>
<code>grey_green</code>
<code>grey_yellow</code>
<code>grey_violet</code>
<code>grey_blue</code>

## Attributes

### grey\_red

`Colormaps.grey_red = <matplotlib.colors.LinearSegmentedColormap object>`

### grey\_green

`Colormaps.grey_green = <matplotlib.colors.LinearSegmentedColormap object>`

### grey\_yellow

`Colormaps.grey_yellow = <matplotlib.colors.LinearSegmentedColormap object>`

### grey\_violet

`Colormaps.grey_violet = <matplotlib.colors.LinearSegmentedColormap object>`

### grey\_blue

`Colormaps.grey_blue = <matplotlib.colors.LinearSegmentedColormap object>`

## 2.3.2 `sc_toolbox.plot.custom_plot_size`

`sc_toolbox.plot.custom_plot_size(width, height, dpi)`

Create a custom axis object of desired sizes.

### Parameters

- **width** (`int`) – Desired plot width
- **height** (`int`) – Desired plot height
- **dpi** (`int`) – Desired plot DPI.

Returns: Axis of desired sizes

## 2.3.3 `sc_toolbox.plot.standard_lineplot`

`sc_toolbox.plot.standard_lineplot(data, order, xlabel, ylabel, hue=None, gene=None, smooth=None, palette=None, title=None, rotation=None, figsize=(15, 5), tick_size=None, label_size=None, order_smooth=3, confidence_interval=None, scatter=None, save=None)`

Draws a standard line plot based on Seaborn's `lmplo`.

### Parameters

- **data** – Data frame containing averaged expression values
- **order** (`List`) – Order of x-axis labels from left to right

- **xlabel** (*str*) – x-axis label
- **ylabel** (*str*) – y-axis label
- **hue** – Subsets of the data which will be drawn on separate facets in the grid. Example: “condition”
- **gene** – Gene of interest
- **smooth** (*Optional[bool]*) – Whether to smoothen (interpolate) the curve
- **palette** – Color palette. For example a list of colors.
- **title** – Title of the plot
- **rotation** (*Optional[int]*) – Rotation of the x-axis labels
- **figsize** (*Tuple[int, int]*) – Size of the figure as specified in matplotlib
- **tick\_size** – Size of the ticks as specified in matplotlib
- **label\_size** – Size of the labels as specified in matplotlib
- **order\_smooth** (*int*) – If greater than 1, numpy.polyfit is used to estimate a polynomial regression
- **confidence\_interval** – Confidence interval
- **scatter** – Set to true in order to add mean expression per sample in form of scatter point
- **save** (*Optional[str]*) – Path to save the plot to

### 2.3.4 sc\_toolbox.plot.average\_expression

```
sc_toolbox.plot.average_expression(gene_expression, genes, order, id_label='identifier', xlabel='days',
                                   cluster='all', hue=None, palette='tab:blue', figsize=(15, 6),
                                   smooth=None, rotation=None, order_smooth=None, conf_int=None,
                                   scatter=None, save=None)
```

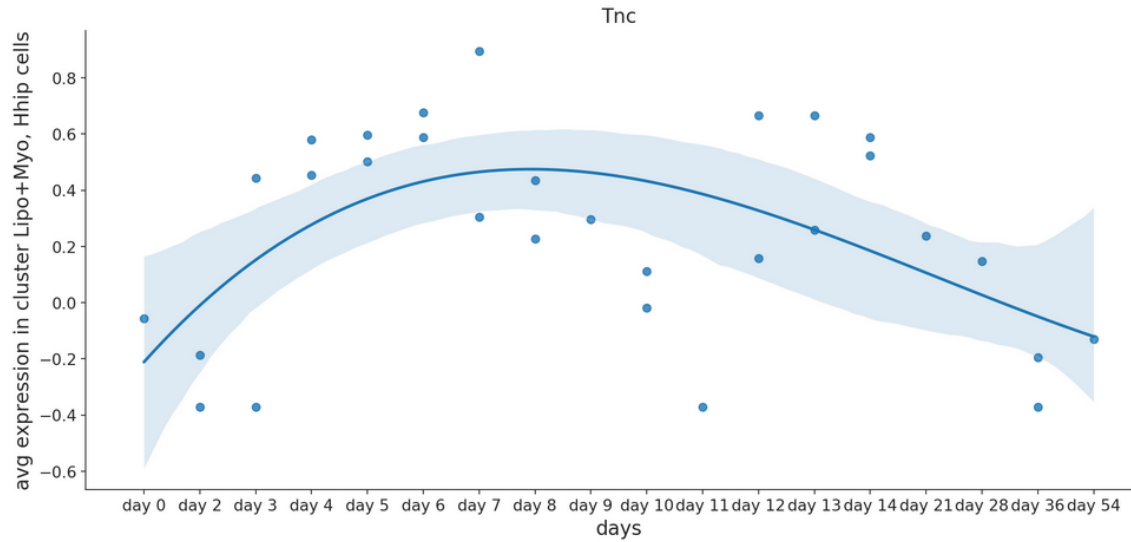
Draw a line plot showing the gene expression over time. Expression values are averaged by individual sample.

#### Parameters

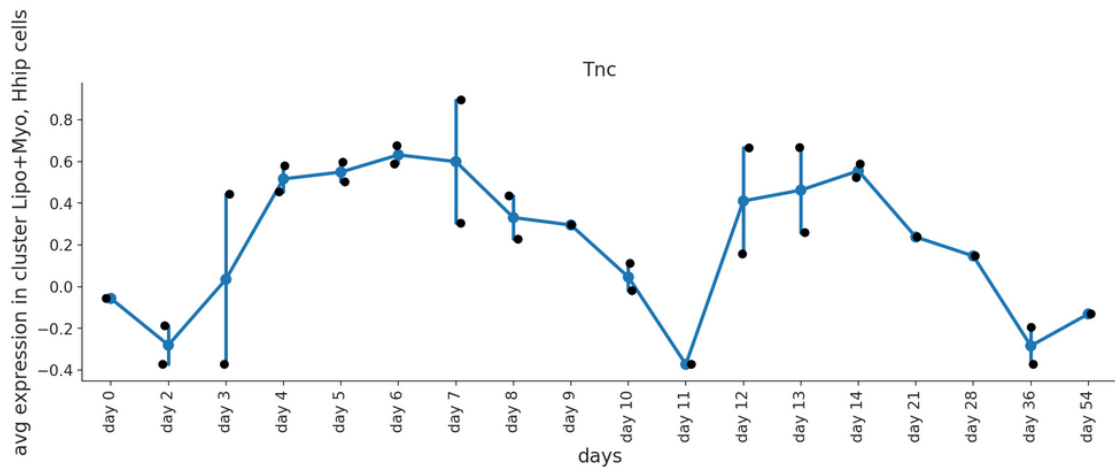
- **gene\_expression** – Data frame containing gene expression values
- **genes** – List of genes for which individual line plots will be generated
- **order** (*List[str]*) – Order of x-axis labels from left to right
- **id\_label** (*str*) – Adata column in which sample id information is stored
- **xlabel** (*str*) – x-axis label
- **cluster** (*str*) – Which clusters to plot. Select ‘all’ if all clusters should be drawn.
- **hue** – Which value to color by
- **figsize** (*Tuple[int, int]*) – Size of the figure as specified in matplotlib
- **smooth** – Set to true for smoothened line plot using polynomial regression
- **rotation** (*Optional[int]*) – set to True to rotate x-axis labels 90 degrees
- **order\_smooth** – If greater than 1, use numpy.polyfit to estimate a polynomial regression
- **conf\_int** – Size of the confidence interval for the regression estimate
- **scatter** – Set to True to add average expression values per sample ID as dots

- **save** (Optional[str]) – Path to save the plot to

#### Example smooth:



#### Example raw:



### 2.3.5 `sc_toolbox.plot.average_expression_per_cluster`

`sc_toolbox.plot.average_expression_per_cluster`(*gene\_expression*, *genes*, *order*, *obs=None*,  
*id\_label='identifier'*, *xlabel='days'*, *cluster='all'*,  
*hue=None*, *figsize=(15, 6)*, *smooth=None*,  
*rotation=None*, *tick\_size=12*, *label\_size=15*,  
*order\_smooth=None*, *conf\_int=None*, *palette=None*,  
*scatter=None*, *save=None*)

Plots gene expression over time split by cluster identity.

One line per cluster.

#### Parameters

- **gene\_expression** – Data frame containing gene expression values

- **genes** – List of genes for which individual line plots will be generated
- **order** – Order of x-axis labels from left to right
- **obs** – Data frame containing meta data information
- **xlabel** (**str**) – x-axis label
- **cluster** (**str**) – Which clusters to plot. Select ‘all’ if all clusters should be drawn.
- **id\_label** (**str**) – Meta data column in which sample id information is stored
- **hue** – Split expression values by this grouping, one line per category will be drawn
- **figsize** (**Tuple[int, int]**) – Size of the figure as specified in matplotlib
- **smooth** – Set to True for smoothened line plot using polynomial regression
- **rotation** – Set to True to rotate x-axis labels 90 degrees
- **tick\_size** (**int**) – Size of the ticks as specified in matplotlib
- **label\_size** (**int**) – Size of the labels as specified in matplotlib
- **order\_smooth** – If greater than 1, use numpy.polyfit to estimate a polynomial regression
- **conf\_int** – Size of the confidence interval for the regression estimate
- **palette** – Color palette that gets passed to Seaborn’s lineplot. For example a list of colors.
- **scatter** – Set to True to add average expression values per sample ID as dots
- **save** (**Optional[str]**) – Path to save the plot to

### 2.3.6 sc\_toolbox.plot.average\_expression\_split\_cluster

```
sc_toolbox.plot.average_expression_split_cluster(gene_expression, genes, order, id_label='identifier',
                                                xlabel='days', hue='genotype', cluster=None,
                                                figsize=(15, 6), smooth=None, rotation=None,
                                                cols=None, tick_size=12, label_size=15,
                                                order_smooth=None, conf_int=None, scatter=None,
                                                save=None)
```

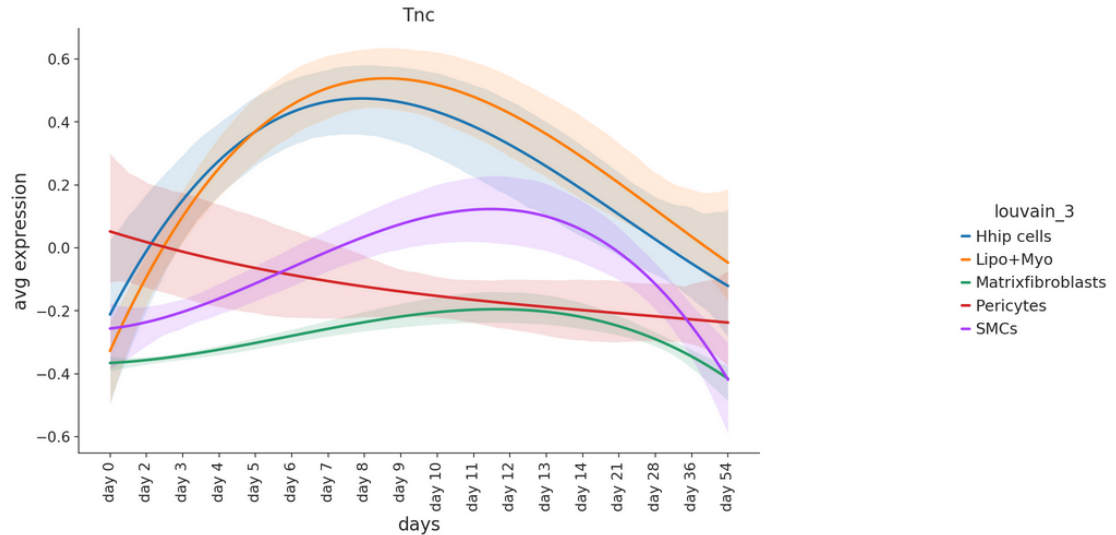
Plot average gene expression as line plots for multiple clusters at once.

#### Parameters

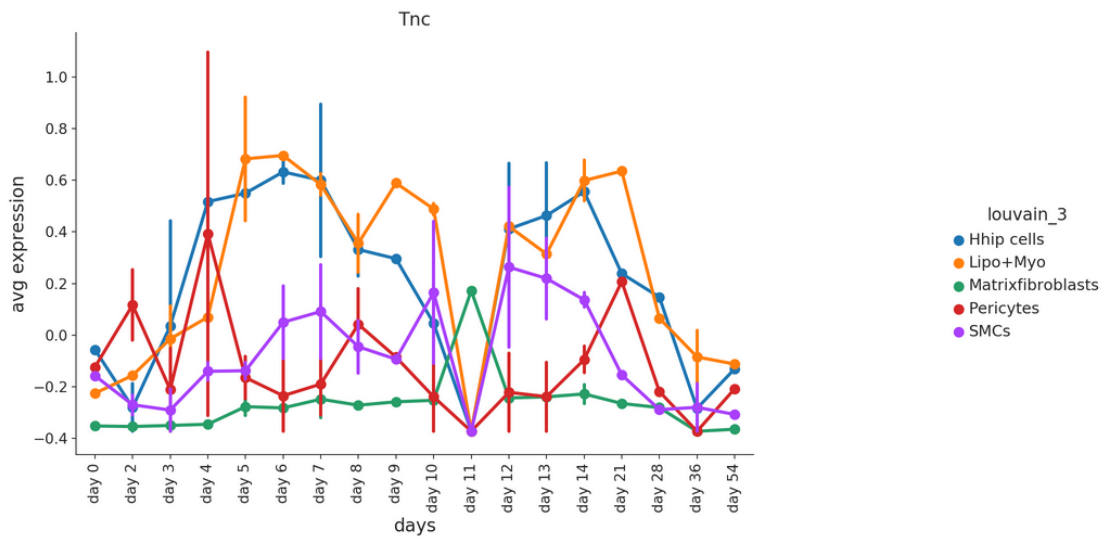
- **gene\_expression** – Data frame containing gene expression values
- **genes** – List of genes for which individual line plots will be generated
- **order** – Order of x-axis labels from left to right
- **id\_label** – Meta data column in which sample id information is stored
- **xlabel** – x-axis label
- **hue** – Split expression values by this grouping, one line per category, will be drawn
- **cluster** – Which clusters to plot. Select ‘all’ if all clusters should be drawn.
- **figsize** – Size of the figure as specified in matplotlib
- **smooth** – Set to True for smoothened line plot using polynomial regression
- **rotation** – x-axis label rotation
- **cols** – List of colors to use for line plot

- **tick\_size** – Size of the ticks as specified in matplotlib
- **label\_size** – Size of the labels as specified in matplotlib
- **order\_smooth** – If greater than 1, numpy.polyfit is used to estimate a polynomial regression
- **conf\_int** – Size of the confidence interval for the regression estimate
- **scatter** – Set to True to add average expression values per sample ID as dots
- **save** – Path to save the plot to

#### Example smooth:



#### Example raw:



### 2.3.7 `sc_toolbox.plot.average_expression_per_cell`

```
sc_toolbox.plot.average_expression_per_cell(gene_expression, genes, order, xlabel='days', cluster='all',
                                           hue=None, figsize=(15, 6), smooth=None, rotation=None,
                                           tick_size=12, label_size=15, order_smooth=None,
                                           conf_int=None, scatter=None, cols=None, save=None)
```

Plots the average gene expression as a line plot per cell. Ideally used when the scatter point should not be sample wise, but cell wise. :type gene\_expression: :param gene\_expression: Data frame containing gene expression values :type genes: :param genes: List of genes for which individual line plots will be generated :type order: :param order: Order of x-axis labels from left to right :type xlabel: `str` :param xlabel: x-axis label :type cluster: `str` :param cluster: Which clusters to plot. Select 'all' if all clusters should be drawn. :type hue: :param hue: Split expression values by this grouping, one line per category, will be drawn :type figsize: `Tuple[int, int]` :param figsize: Size of the figure as specified in matplotlib :type smooth: :param smooth: Set to true for smoothened line plot using polynomial regression :type rotation: :param rotation: Set to True to rotate x-axis labels 90 degrees :type tick\_size: :param tick\_size: Size of the ticks as specified in matplotlib :type label\_size: :param label\_size: Size of the labels as specified in matplotlib :type order\_smooth: :param order\_smooth: If greater than 1, use numpy.polyfit to estimate a polynomial regression :type conf\_int: :param conf\_int: Size of the confidence interval for the regression estimate :type scatter: :param scatter: Set to True to add average expression values per sample ID as dots :type cols: :param cols: List of colors to use for line plot :type save: `Optional[str]` :param save: Path to save the plot to

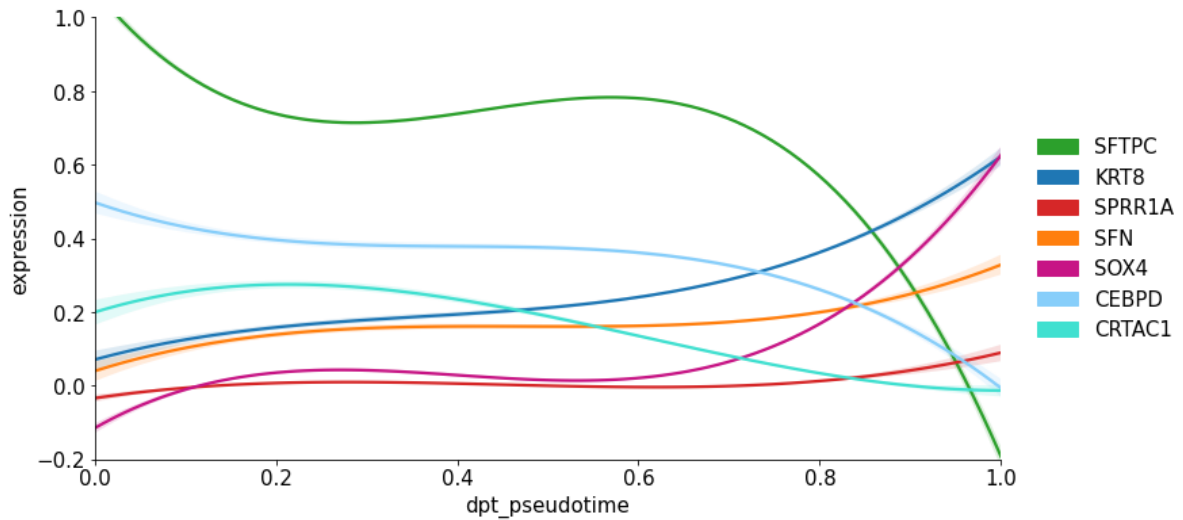
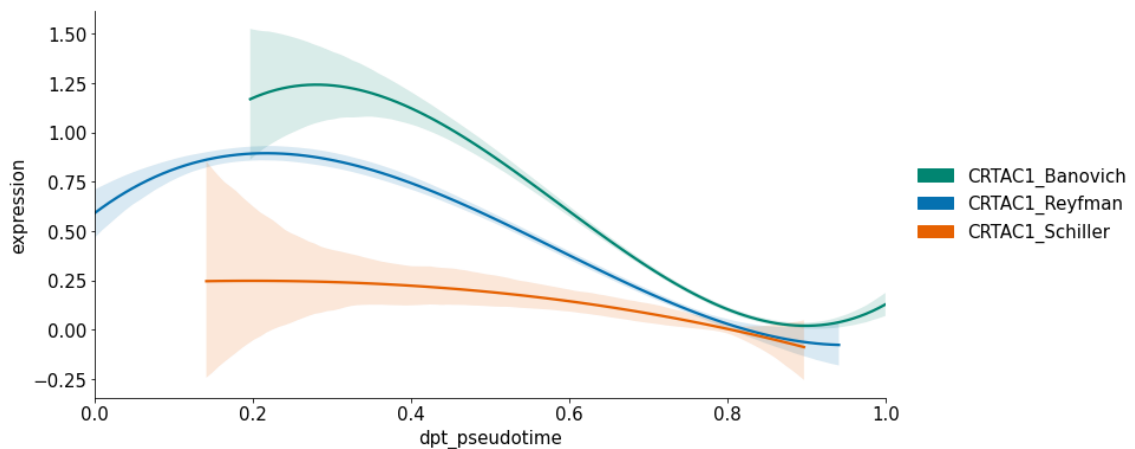
### 2.3.8 `sc_toolbox.plot.gene_expression_dpt_ordered`

```
sc_toolbox.plot.gene_expression_dpt_ordered(data, genes, xlabel, order=3, conf_int=95, figsize=(12, 6),
                                           condition=None, label_size=15, cols=None, scale=None,
                                           ylim=None, save=None)
```

Plot smoothed expression of all cells ordered by pseudo time.

#### Parameters

- **data** – AnnData object
- **genes** – List of genes for which individual line plots will be generated
- **xlabel** – x-axis label
- **order** – Order of x-axis labels from left to right
- **conf\_int** – Size of the confidence interval for the regression estimate
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **condition** – Split expression values by this grouping, one line per category will be drawn
- **label\_size** (`int`) – Size of the labels as specified in matplotlib
- **cols** – List of colors to use for line plot
- **scale** – Set to True to scale expression value to a range between 0 and 1
- **ylim** – Upper limit on the y-axis if desired
- **save** (`Optional[str]`) – Path to save the plot to

**Example****Example with columns:****2.3.9 sc\_toolbox.plot.relative\_frequencies\_boxplots**

`sc_toolbox.plot.relative_frequencies_boxplots(relative_frequencies, cluster, cols, order, xlabel='days', hue='batch', figsize=(15, 6), width=0.5, jitter=None, save=None)`

Plots the relative frequencies as split boxplots.

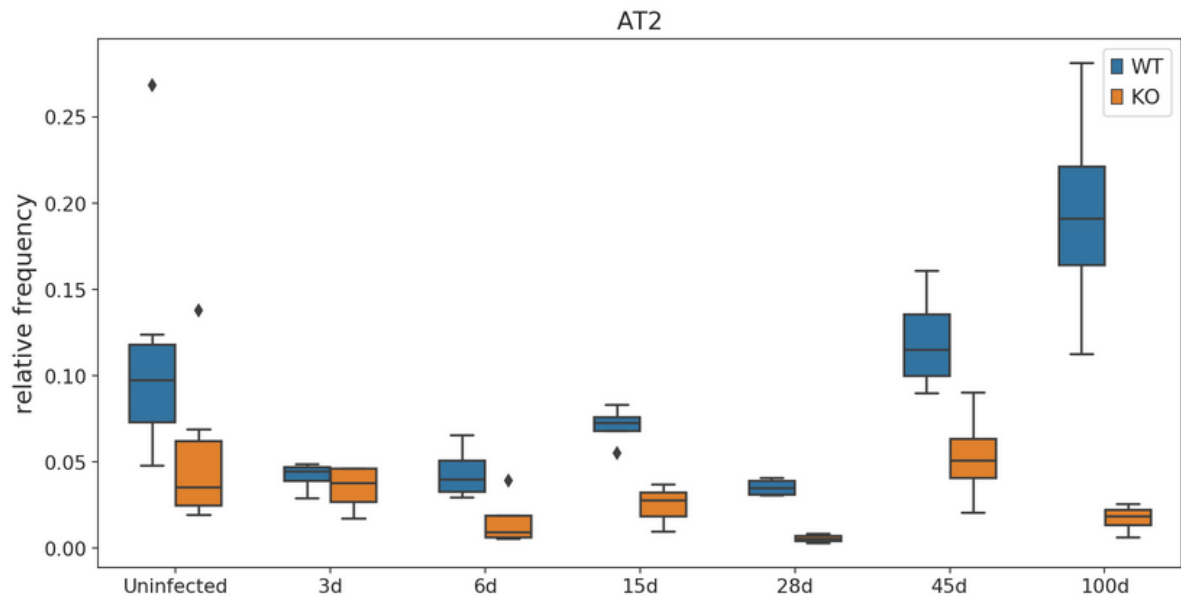
Use `calc_relative_frequencies` to get the required input format.

**Parameters**

- **relative\_frequencies** (`DataFrame`) – Calculated by `calc_relative_frequencies` as Pandas `DataFrame`
- **cluster** – Cluster to be plotted
- **cols** – List of colors to use for boxes
- **order** – Order of x-axis labels from left to right
- **xlabel** (`str`) – x-axis label



- **hue** (`str`) – Value to color by
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **width** (`float`) – Width of the plot as specified in matplotlib
- **jitter** – Set to True for individual dots per sample
- **save** – Path to save the plot to

**Return type**`None`**Example****2.3.10 sc\_toolbox.plot.split\_boxplot**

`sc_toolbox.plot.split_boxplot`(*table, order, xlabel, ylabel, column=None, hue=None, cols=None, width=1, title=None, figsize=(15, 6), jitter=None, save=None*)

Draws a boxsplit split by hue.

**Parameters**

- **table** – Table containing the data to draw the boxplots for
- **order** – Order of the boxplot labels
- **xlabel** (`str`) – x-axis label
- **ylabel** (`str`) – y-axis label
- **column** –
- **hue** – Value to split relative frequencies by
- **cols** – List of colors to use for boxes
- **width** (`float`) – Width of the desired plot

- **title** – Title of the plot
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **jitter** – Set to True for individual dots per sample
- **save** (`Optional[str]`) – Path to save the plot to

**Return type**

`None`

### 2.3.11 `sc_toolbox.plot.marker_dendrogram`

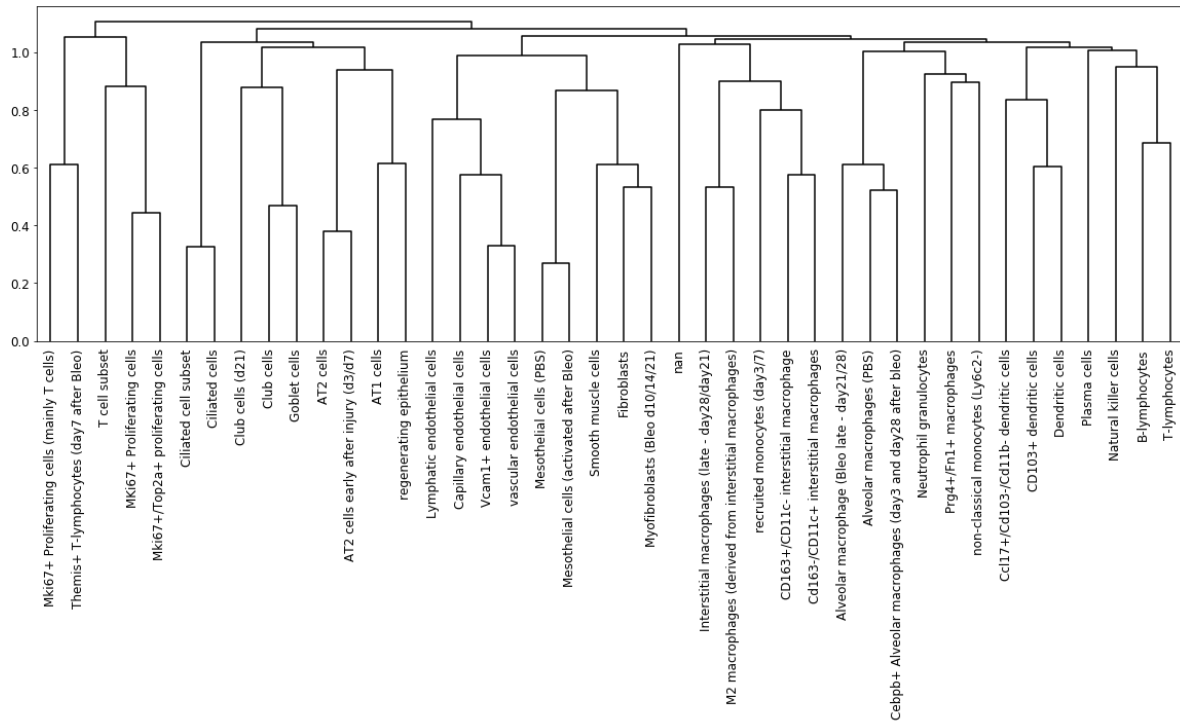
`sc_toolbox.plot.marker_dendrogram(marker_table, threshold=0.7, column='cluster', log_fc_key='log_FC', label_size=10, orientation='top', figsize=(10, 4), save=None)`

Plots a dendrogram of used marker genes.

**Parameters**

- **marker\_table** (`DataFrame`) – A marker table as generated by `sct.calc.extended_marker_table`
- **threshold** (`float`) – Threshold for the log fold change
- **column** (`str`) – Column to create pivot by; usually just the clusters
- **log\_fc\_key** (`str`) – Key for the stored log fold changes in the marker table
- **label\_size** (`int`) – Font size of the labels
- **orientation** (`str`) – Orientation of the figure; Currently just 'top' or no orientation
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **save** (`Optional[str]`) – Path to save the plot to

## Example



## 2.3.12 sc\_toolbox.plot.volcano\_plot

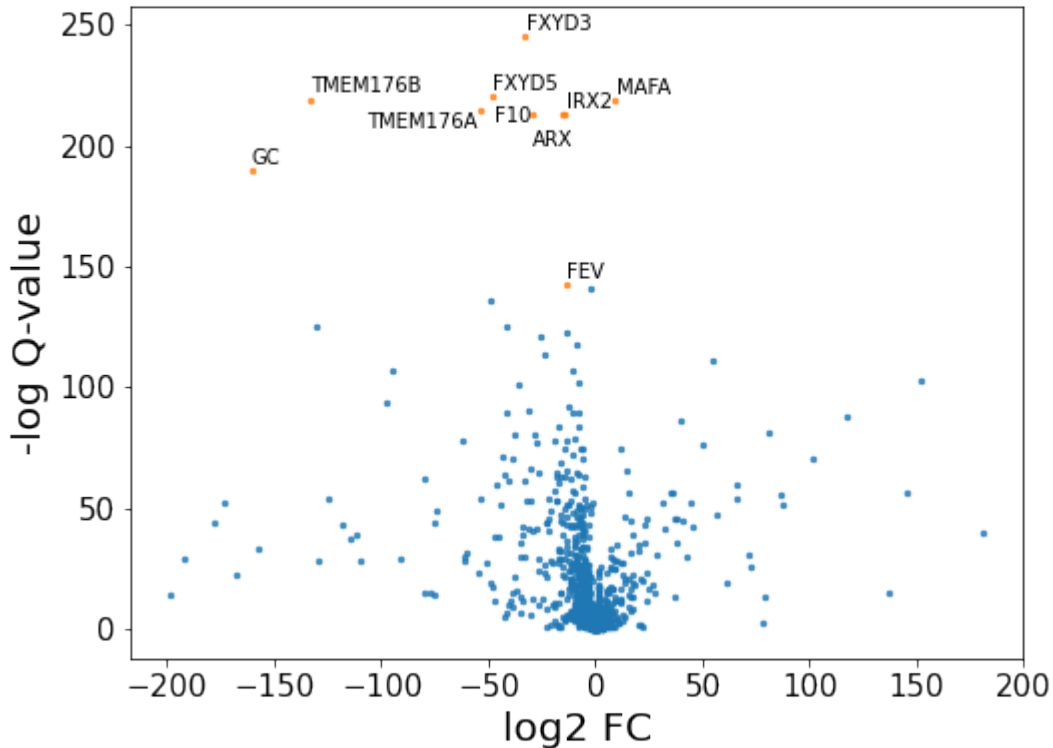
```
sc_toolbox.plot.volcano_plot(table, fdr_thresh=None, log_fc_thresh=0, adj_p_val='adj_p_val',
                             log_fc='avg_logFC', gene='gene', sig_col='tab:orange', col='tab:blue',
                             figsize=(8, 6), save=None)
```

Scatter plot of differential gene expression results generated by diffxpy

### Parameters

- **table** – diffxpy generated table of results
- **fdr\_thresh** (Optional[float]) –  $-\log(\text{FDR})$  threshold for labeling genes. If set to None, we will consider the 99th percentile of  $-\log(\text{FDR})$  values the threshold.
- **log\_fc\_thresh** (float) – absolute(log\_fc) threshold for labeling genes.
- **adj\_p\_val** (str) – Label of the adjusted p value, these are considered FDRs
- **log\_fc** (str) – Label of the log fold change
- **gene** (str) – Label of column with gene names
- **col** (str) – Color of dots
- **sig\_col** (str) – Colour of dots surpassing defined FDR threshold
- **figsize** (Tuple[int, int]) – Size of the figure as specified in matplotlib
- **save** – Path to save the plot to

### Example



### 2.3.13 `sc_toolbox.plot.cluster_composition_stacked_barplot`

`sc_toolbox.plot.cluster_composition_stacked_barplot`(*relative\_frequencies*, *xlabel*='name', *figsize*=(6, 10), *width*=0.8, *order*=None, *error\_bar*=None, *label\_size*=15, *tick\_size*=13, *capsize*=None, *margins*=(0.02, 0.04), *colors*=None, *save*=None)

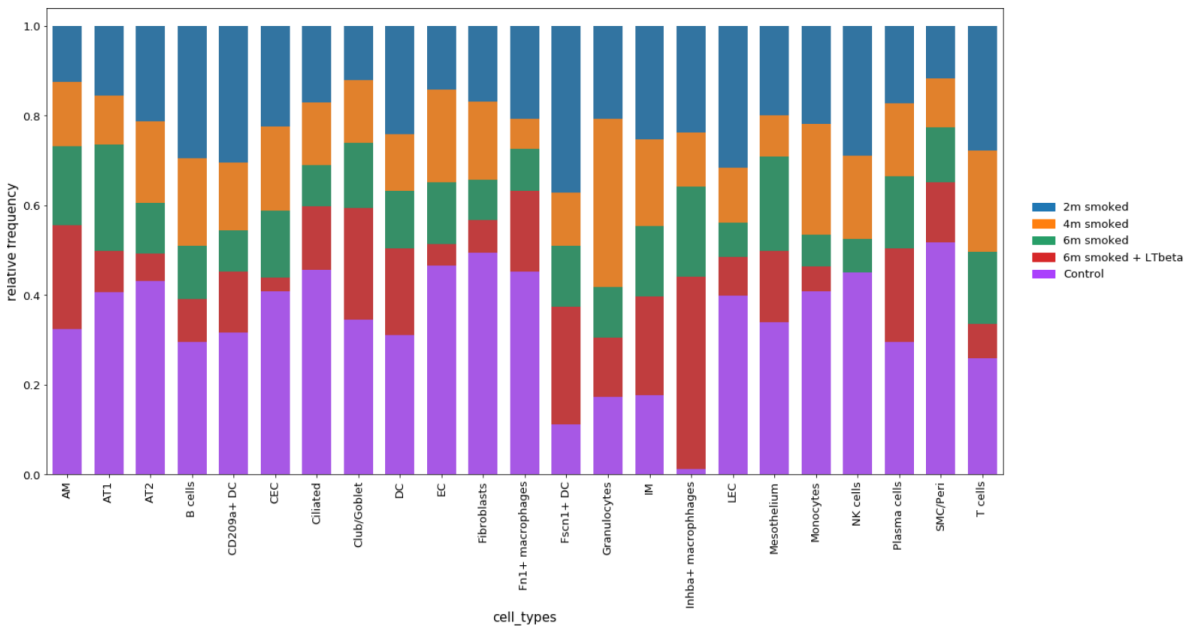
Plot relative frequencies as a stacked barplot.

#### Parameters

- **relative\_frequencies** (`DataFrame`) – Data frame containing relative Frequencies as calculated by `calc_relFreq()`
- **xlabel** (`str`) – x-axis label
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **width** (`float`) – Width of the bars
- **order** – Order of x-axis labels from left to right
- **error\_bar** – Set to True to add error bars (only possible when grouping the frequencies)
- **tick\_size** (`int`) – Size of the ticks as specified in matplotlib
- **label\_size** (`int`) – Size of the labels as specified in matplotlib

- **capsize** (`Optional[int]`) – Size of the horizontal lines of the error bar
- **margins** (`Tuple[float, float]`) – Change margins of the plot if desired
- **colors** – List of colors to use for the bands
- **save** (`Optional[str]`) – Path to save the plot to

### Example



### 2.3.14 `sc_toolbox.plot.gene_boxplot`

`sc_toolbox.plot.gene_boxplot(table, palette, xlabel='cell_types', hue=None, figsize=(10, 5), legend=True, score='Axi2', scatter=None, rotate=False, width=0.7, save=None)`

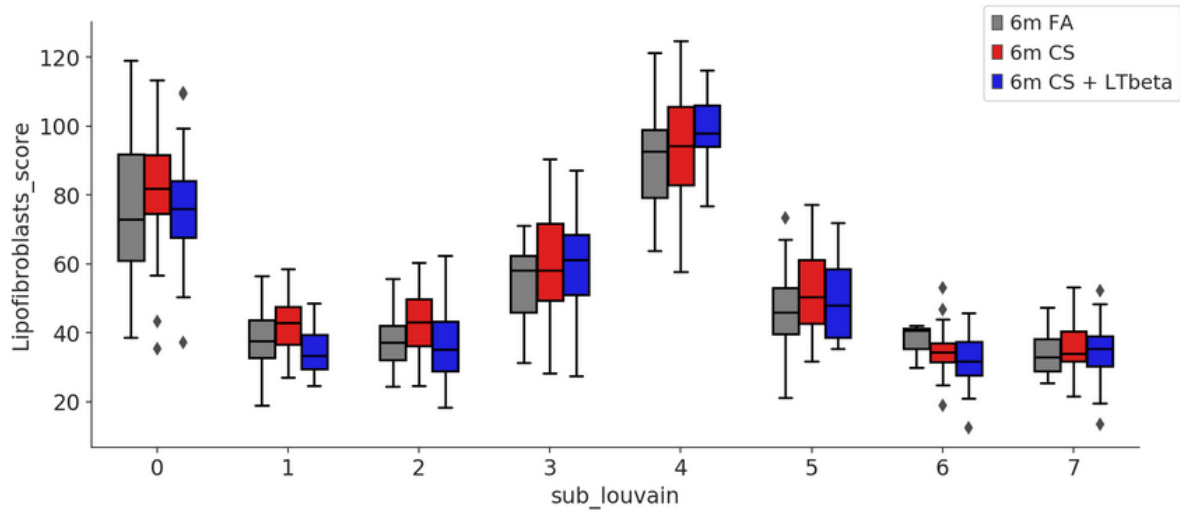
Plot gene values as split boxplots.

#### Parameters

- **table** – Pandas DataFrame
- **palette** (`List[str]`) –
- **xlabel** (`str`) – x-axis label
- **hue** (`Optional[str]`) –
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **legend** – Whether to draw a legend or not
- **score** –
- **scatter** –
- **rotate** –
- **width** – Width of the desired plot

- **save** – Path to save the plot to

### Example



### 2.3.15 sc\_toolbox.plot.colors\_overview


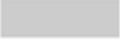





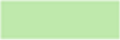














`sc_toolbox.plot.colors_overview(colors, ncols=2, figsize=(8, 5), save=None)`

Draw an overview plot of all used colors.

#### Parameters

- **colors** (`Dict`) – Dictionary of color name and color
- **ncols** (`int`) – How many columns for the plot
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **save** (`Optional[str]`) – Path to save the plot to

**Example**

	Mesothelium #7f7f7f		Monocytes #cccccc
	Fscn1+ DC #8c564b		EC #fc9983
	SMC/Peri #ff7f0e		B cells #dfb678
	Fibroblasts #f1c40f		Club/Goblet #bfe9ac
	LEC #49de16		T cells #138310
	Inhba+ macrophages #4be7b0		CEC #bdecf0
	Granulocytes #17becf		AT1 #17202a
	DC #1e37d4		Ciliated #b36bf6
	AM #65209b		Fn1+ macrophages #c5b0d5
	IM #ef6bf1		Plasma cells #f7b6d2
	NK cells #bb7784		AT2 #d62728

**2.3.16 sc\_toolbox.plot.relative\_frequencies\_lineplot**

`sc_toolbox.plot.relative_frequencies_lineplot`(*relative\_frequencies*, *order*, *cluster*, *xlabel*='days', *ylabel*='relative frequency', *hue*=None, *smooth*=None, *cols*=None, *title*=None, *rotation*=None, *figsize*=(15, 5), *tick\_size*=None, *label\_size*=None, *order\_smooth*=3, *conf\_int*=None, *scatter*=None, *save*=None)

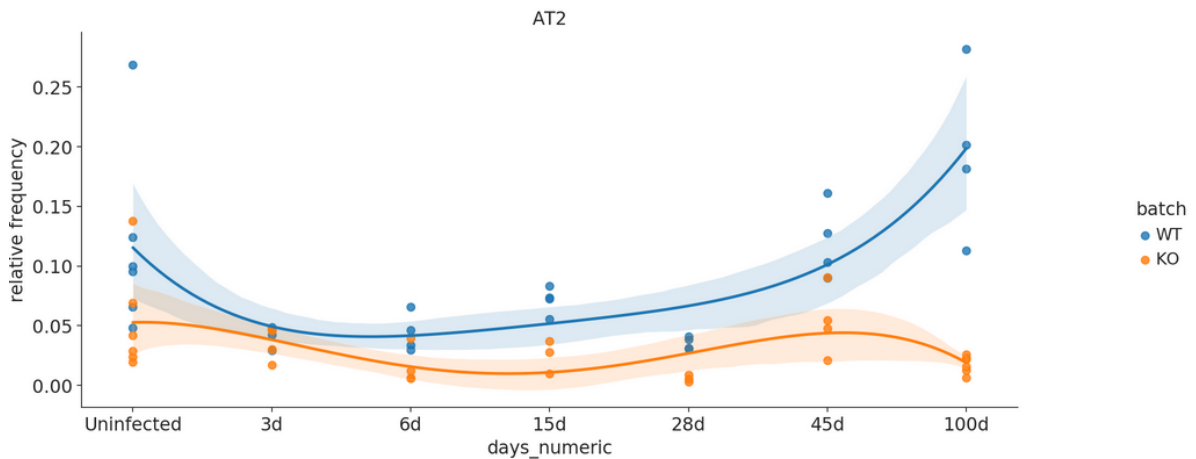
Plot relative frequencies as a line plot.

**Parameters**

- **relative\_frequencies** (`DataFrame`) – Data frame containing relative Frequencies as calculated by `calc_relFreq()`
- **order** – Order of x-axis labels from left to right
- **cluster** – Which cluster to plot
- **xlabel** (`str`) – x-axis label
- **ylabel** (`str`) – y-axis label
- **hue** (`Optional[str]`) – Value to color by
- **smooth** (`Optional[bool]`) – Whether to smoothen the plot
- **cols** – List of colors to use for line plot
- **title** (`Optional[str]`) – Title of the plot
- **rotation** (`Optional[int]`) – Rotation of the x-axis labels
- **figsize** (`Tuple[int, int]`) – Size of the figure as specified in matplotlib
- **tick\_size** (`Optional[int]`) – Size of the ticks as specified in matplotlib

- **label\_size** (`Optional[int]`) – Size of the labels as specified in matplotlib
- **order\_smooth** (`int`) – If greater than 1, `numpy.polyfit` is used to estimate a polynomial regression
- **conf\_int** – Size of the confidence interval for the regression estimate
- **scatter** – Set to `True` to add average expression values per sample ID as dots
- **save** (`Optional[str]`) – Path to save the plot to

### Example



## 2.3.17 `sc_toolbox.plot.annotated_cell_type_umap`

`sc_toolbox.plot.annotated_cell_type_umap(adata, primary_color, cell_type_color, legend_loc='on data', legend_fontsize=8, title='Plot title', palette=None, cmap=None, figsize=(8, 6), save=None)`

Plots a UMAP which is colored by the `primary_color`, but also draws all labels on top of all clusters.

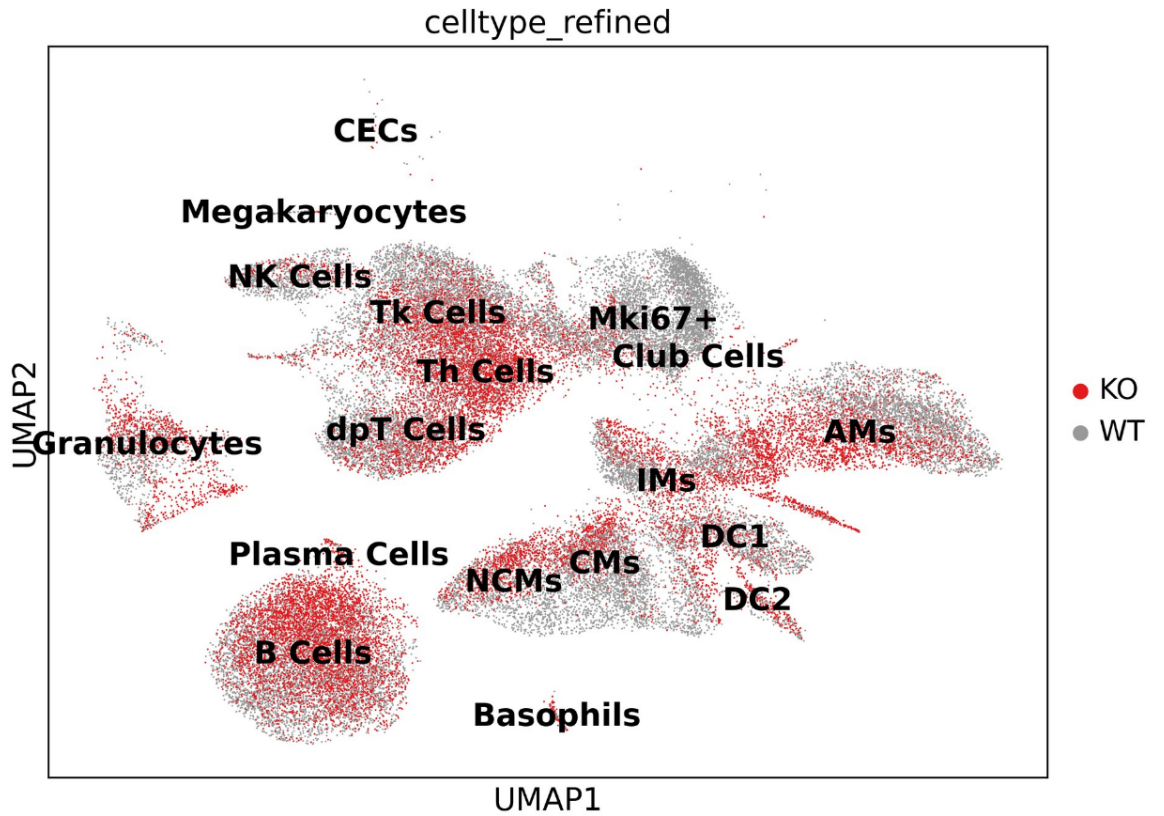
### Parameters

- **adata** – `Anndata` object
- **primary\_color** (`Union[str, Sequence[str]]`) – Primary color to color all cells by, e.g. 'genotype'
- **cell\_type\_color** (`str`) – Key containing all cell types, e.g. 'cell\_type'
- **legend\_loc** (`str`) – Location of the legend (default: 'on data')
- **legend\_fontsize** (`int`) – Font size of the legend (default: 8)
- **title** (`str`) – Title of the plot
- **palette** – Color
- **cmap** – Color map of the UMAP
- **figsize** – Size of the figure
- **save** – Path to save the plot to



**Returns**

fig and axs Matplotlib objects

**Example****2.3.18 sc\_toolbox.plot.genotype\_vs\_genotype\_umaps**

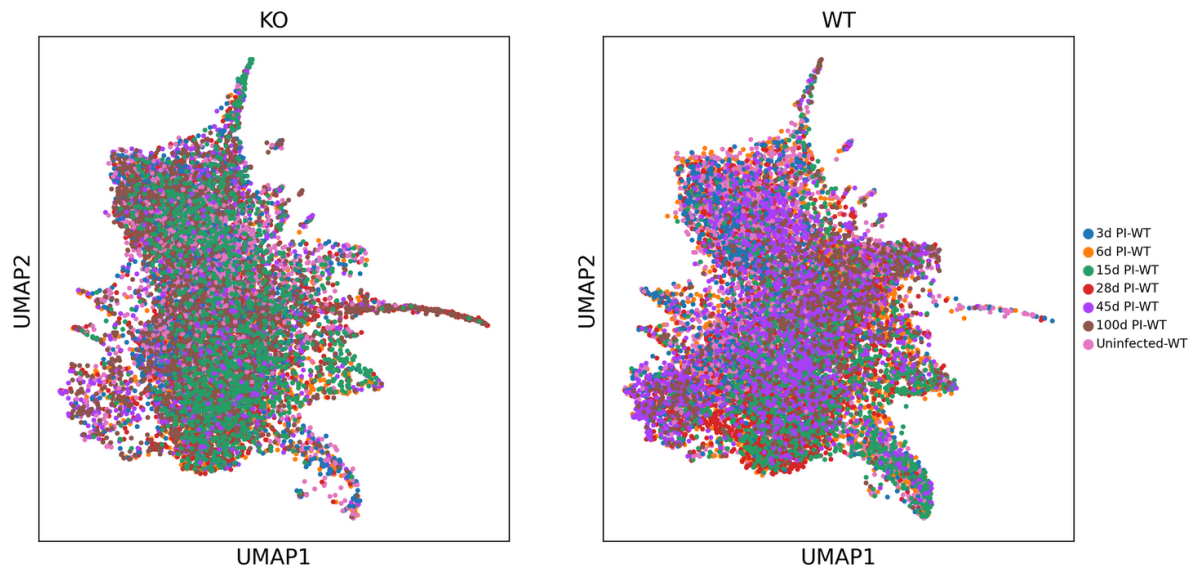
```
sc_toolbox.plot.genotype_vs_genotype_umaps(adata, genotype_key, genotype_label_1, genotype_label_2,
                                             color, hide_one_legend=True, figsize=(12, 6))
```

Plots a two UMAPs of genotypes next to each other displaying only the colors of the second UMAP.

**Parameters**

- **adata** – AnnData object
- **genotype\_key** (str) – Key of the genotypes
- **genotype\_label\_1** (str) – Name of the first genotype; Must be contained in the genotypes
- **genotype\_label\_2** (str) – Name of the second genotype; Must be contained in the genotypes
- **color** (str) – Key to color by
- **hide\_one\_legend** (bool) – Whether to hide the legend of the genotype\_label\_1
- **figsize** (Tuple[int, int]) – Size of the figure

## Example



## CONTRIBUTOR GUIDE

Thank you for your interest in improving this project. This project is open-source under the [Apache2.0 license](#) and highly welcomes contributions in the form of bug reports, feature requests, and pull requests.

Here is a list of important resources for contributors:

- [Source Code](#)
- [Documentation](#)
- [Issue Tracker](#)
- [Code of Conduct](#)

### 3.1 How to report a bug

Report bugs on the [Issue Tracker](#).

### 3.2 How to request a feature

Request features on the [Issue Tracker](#).

### 3.3 Getting the code

```
$ git clone --recurse-submodules https://github.com/schillerlab/sc-toolbox
```

### 3.4 How to set up your development environment

You need Python 3.8+ and the following tools:

- [Poetry](#)
- [Nox](#)
- [nox-poetry](#)

You can install them with:

```
$ pip install poetry nox nox-poetry
```

Install the package with development requirements:

```
$ make install
```

You can now run an interactive Python session, or the command-line interface:

```
$ poetry run python
$ poetry run sc_toolbox
```

## 3.5 How to test the project

Run the full test suite:

```
$ nox
```

List the available Nox sessions:

```
$ nox --list-sessions
```

You can also run a specific Nox session. For example, invoke the unit test suite like this:

```
$ nox --session=tests
```

Unit tests are located in the `tests` directory, and are written using the [pytest](#) testing framework.

## 3.6 How to build and view the documentation

This project uses [Sphinx] together with several extensions to build the documentation. It further requires [Pandoc] to translate various formats.

To install all required dependencies for the documentation run:

```
$ pip install -r docs/requirements.txt
```

Please note that `ehrapy` itself must also be installed. To build the documentation run:

```
$ make html
```

from inside the docs folder. The generated static HTML files can be found in the `_build/html` folder. Simply open them with your favorite browser.

## 3.7 How to submit changes

Open a [pull request](#) to submit changes to this project against the `development` branch.

Your pull request needs to meet the following guidelines for acceptance:

- The Nox test suite must pass without errors and warnings.
- Include unit tests. This project maintains a high code coverage.
- If your changes add functionality, update the documentation accordingly.

To run linting and code formatting checks before committing your change, you can install pre-commit as a Git hook by running the following command:

```
$ nox --session=pre-commit -- install
```

It is recommended to open an issue before starting work on anything. This will allow a chance to talk it over with the owners and validate your approach.



## CONTRIBUTOR COVENANT CODE OF CONDUCT

### 4.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, nationality, personal appearance, race, religion, or sexual identity and orientation.

### 4.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

### 4.3 Our Responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

## 4.4 Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

## 4.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by opening an issue. The project team will review and investigate all complaints, and will respond in a way that it deems appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

## 4.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at <https://www.contributor-covenant.org/version/1/4/code-of-conduct.html>



## CREDITS

### 5.1 Development Lead

- Lukas Heumos <<mailto:lukas.heumos@helmholtz-munich.de>>

### 5.2 Contributors

- Meshal Ansari <<mailto:meshal.ansari@helmholtz-munich.de>>



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## A

add\_percentages() (in module *sc\_toolbox.tools*), 8  
 annotated\_cell\_type\_umap() (in module *sc\_toolbox.plot*), 28  
 automated\_marker\_annotation() (in module *sc\_toolbox.tools*), 11  
 average\_expression() (in module *sc\_toolbox.plot*), 15  
 average\_expression\_per\_cell() (in module *sc\_toolbox.plot*), 19  
 average\_expression\_per\_cluster() (in module *sc\_toolbox.plot*), 16  
 average\_expression\_split\_cluster() (in module *sc\_toolbox.plot*), 17

## C

cluster\_composition\_stacked\_barplot() (in module *sc\_toolbox.plot*), 24  
 Colormaps (class in *sc\_toolbox.plot*), 13  
 colors\_overview() (in module *sc\_toolbox.plot*), 26  
 correlate\_means\_to\_gene() (in module *sc\_toolbox.tools*), 10  
 correlate\_to\_signature() (in module *sc\_toolbox.tools*), 7  
 custom\_plot\_size() (in module *sc\_toolbox.plot*), 14

## D

de\_res\_to\_anndata() (in module *sc\_toolbox.tools*), 12

## E

extended\_marker\_table() (in module *sc\_toolbox.tools*), 10

## G

gene\_boxplot() (in module *sc\_toolbox.plot*), 25  
 gene\_expression\_dpt\_ordered() (in module *sc\_toolbox.plot*), 19  
 generate\_count\_object() (in module *sc\_toolbox.tools*), 9  
 generate\_expression\_table() (in module *sc\_toolbox.tools*), 6  
 generate\_pseudobulk() (in module *sc\_toolbox.tools*), 10

genotype\_vs\_genotype\_umaps() (in module *sc\_toolbox.plot*), 29  
 grey\_blue (*sc\_toolbox.plot.Colormaps* attribute), 14  
 grey\_green (*sc\_toolbox.plot.Colormaps* attribute), 14  
 grey\_red (*sc\_toolbox.plot.Colormaps* attribute), 14  
 grey\_violet (*sc\_toolbox.plot.Colormaps* attribute), 14  
 grey\_yellow (*sc\_toolbox.plot.Colormaps* attribute), 14

## M

marker\_dendrogram() (in module *sc\_toolbox.plot*), 22

## R

ranksums\_between\_groups() (in module *sc\_toolbox.tools*), 8  
 relative\_frequencies() (in module *sc\_toolbox.tools*), 7  
 relative\_frequencies\_boxplots() (in module *sc\_toolbox.plot*), 20  
 relative\_frequencies\_lineplot() (in module *sc\_toolbox.plot*), 27  
 relative\_frequency\_per\_cluster() (in module *sc\_toolbox.tools*), 7  
 remove\_outliers() (in module *sc\_toolbox.tools*), 8

## S

split\_boxplot() (in module *sc\_toolbox.plot*), 21  
 standard\_lineplot() (in module *sc\_toolbox.plot*), 14

## T

tidy\_de\_table() (in module *sc\_toolbox.tools*), 9

## V

volcano\_plot() (in module *sc\_toolbox.plot*), 23