scanpy.tl.rank_genes_groups

scanpy.tl.rank_genes_groups(*adata*, *groupby*, *use_raw=None*, *groups='all'*, *reference='rest'*, *n_genes=None*, *rankby_abs=False*, *pts=False*, *key_added=None*, *copy=False*, *method=None*, *corr_method='benjamini-hochberg'*, *tie_correct=False*, *layer=None*, ***kwds*)

Rank genes for characterizing groups.

Expects logarithmized data.

Parameters:

adata : `AnnData`

Annotated data matrix.

groupby : `str`

The key of the observations grouping to consider.

use_raw : `Optional` [`bool`] (default: `None`)

Use `raw` attribute of `adata` if present.

layer : `Optional` [`str`] (default: `None`)

Key from `adata.layers` whose value will be used to perform tests on.

groups : `Union` [`Literal` [`'all'`], `Iterable` [`str`]] (default: `'all'`)

Subset of groups, e.g. [`'g1'` , `'g2'` , `'g3'`], to which comparison shall be restricted, or `'all'` (default), for all groups.

reference : `str` (default: `'rest'`)

If `'rest'`, compare each group to the union of the rest of the group. If a group identifier, compare with respect to this group.

n_genes : `Optional` [`int`] (default: `None`)

The number of genes that appear in the returned tables. Defaults to all genes.

method : `Optional` [`Literal` [`'logreg'` , `'t-test'` , `'wilcoxon'` , `'t-test_overestim_var'`]] (default: `None`)

The default method is `'t-test'`, `'t-test_overestim_var'` overestimates variance of each group, `'wilcoxon'` uses Wilcoxon rank-sum, `'logreg'` uses logistic regression. See [Ntranos18], here and here, for why this is meaningful.

---

corr_method : `Literal` [`'benjamini-hochberg'`, `'bonferroni'`] (default: `'benjamini-hochberg'`)

p-value correction method. Used only for `'t-test'`, `'t-test_overestim_var'`, and `'wilcoxon'`.

---

tie_correct : `bool` (default: `False`)

Use tie correction for `'wilcoxon'` scores. Used only for `'wilcoxon'`.

---

rankby_abs : `bool` (default: `False`)

Rank genes by the absolute value of the score, not by the score. The returned scores are never the absolute values.

---

pts : `bool` (default: `False`)

Compute the fraction of cells expressing the genes.

---

key_added : `Optional` [`str`] (default: `None`)

The key in `adata.uns` information is saved to.

****kwds**

Are passed to test methods. Currently this affects only parameters that are passed to `sklearn.linear_model.LogisticRegression`. For instance, you can pass `penalty='l1'` to try to come up with a minimal set of genes that are good predictors (sparse solution meaning few non-zero fitted coefficients).

Return type: `Optional` [`AnnData`]

Returns:

: **names** : structured `np.ndarray` (`.uns['rank_genes_groups']`)

Structured array to be indexed by group id storing the gene names. Ordered according to scores.

scores : structured `np.ndarray` (`.uns['rank_genes_groups']`)

Structured array to be indexed by group id storing the z-score underlying the computation of a p-value for each gene for each group. Ordered according to scores.

logfoldchanges : structured `np.ndarray` (`.uns['rank_genes_groups']`)

Structured array to be indexed by group id storing the log2 fold change for each gene for each group. Ordered according to scores. Only provided if method is 't-test' like. Note: this is an approximation calculated from mean-log values.

pvals : structured `np.ndarray` (`.uns['rank_genes_groups']`)

p-values.

pvals_adj : structured `np.ndarray` (`.uns['rank_genes_groups']`)

Corrected p-values.

pts : `pandas.DataFrame` (`.uns['rank_genes_groups']`)

Fraction of cells expressing the genes for each group.

pts_rest : `pandas.DataFrame` (`.uns['rank_genes_groups']`)

Only if `reference` is set to `'rest'`. Fraction of cells from the union of the rest of each group expressing the genes.

Notes

There are slight inconsistencies depending on whether sparse or dense data are passed. See here.

Examples

```
>>> import scanpy as sc
>>> adata = sc.datasets.pbmc68k_reduced()
>>> sc.tl.rank_genes_groups(adata, 'bulk_labels', method='wilcoxon')
>>> # to visualize the results
>>> sc.pl.rank_genes_groups(adata)
```