Scanpy

Tertiary analysis of large-scale scRNA-seq data

Alex Wolf, Institute of Computational Biology, Helmholtz Munich
March 13, 2017 - Video talk for HCA Red Box Meeting

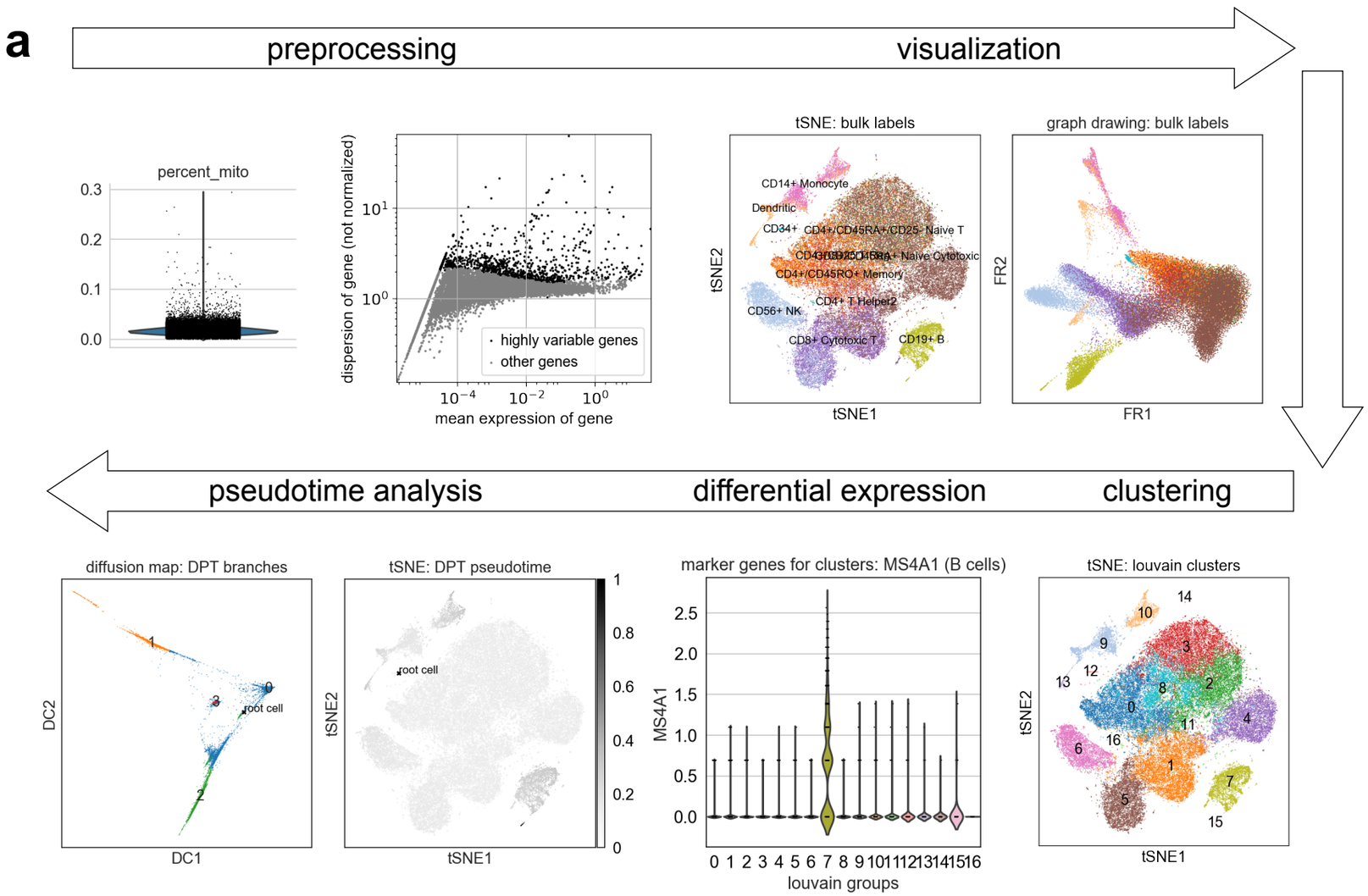
Wolf, Angerer & Theis, Genome Biology (2018)

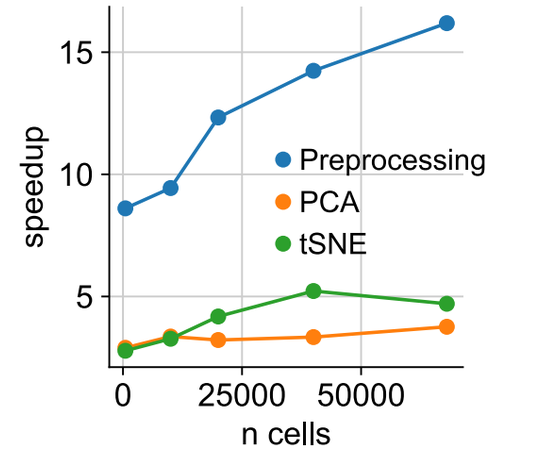
HelmholtzZentrum münchen
Deutsches Forschungszentrum für Gesundheit und Umwelt

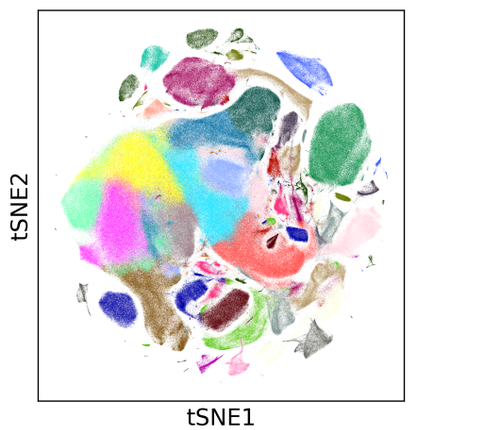
HELMHOLTZ
ASSOCIATION

Scanpy

Scalable Python-based alternative to established R packages for writing clean, comprehensive analysis pipelines.





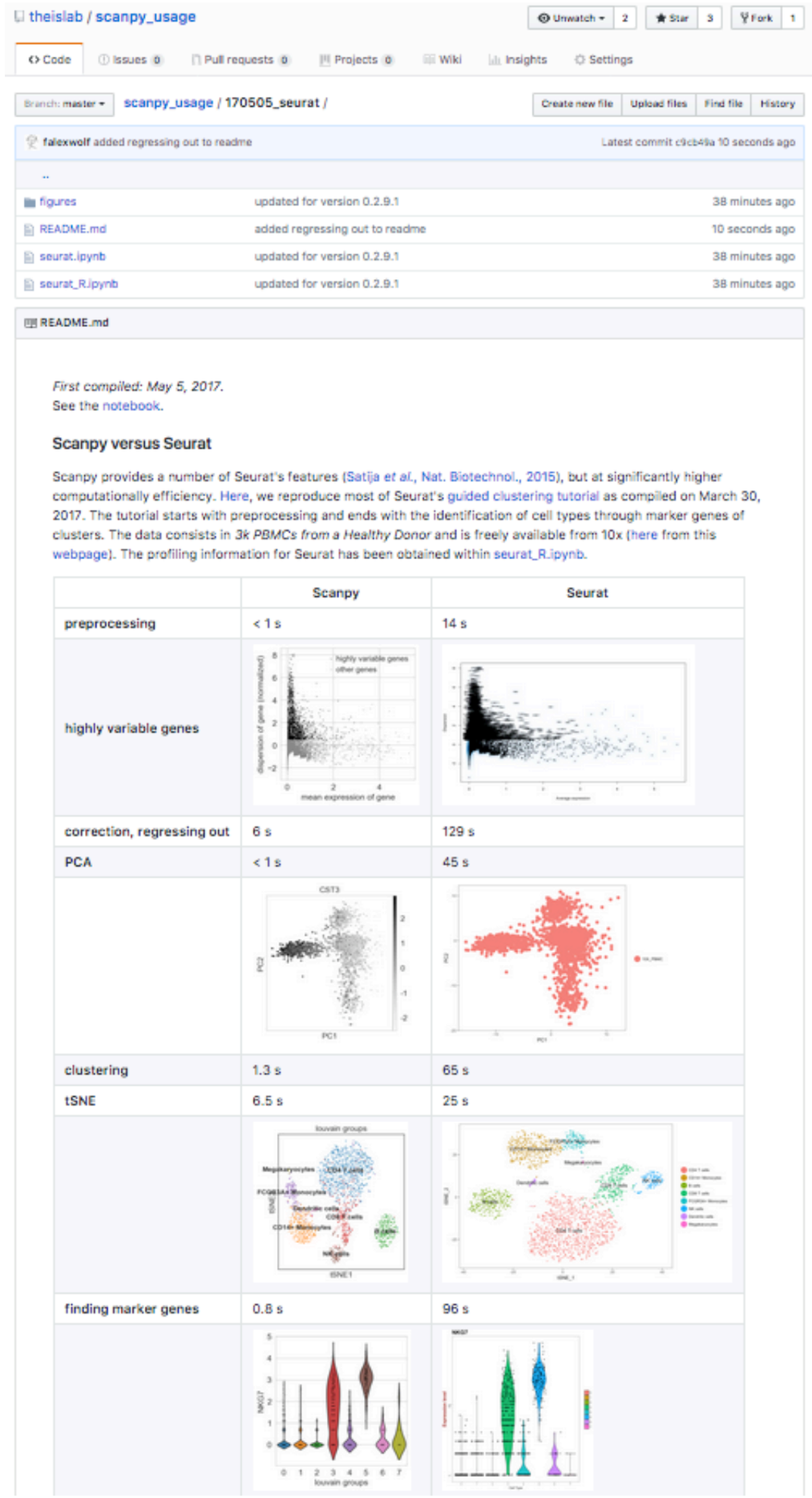


Scanpy vs. Seurat

Satija et al., Nat. Biotechn. (2015)

Benchmarked against Seurat, 2.7K cells.

- preprocessing: <1 s vs. 14 s
- regressing out unwanted sources of variation: 6 s vs. 129 s
- PCA: <1 s vs. 45 s
- clustering: 1.3 s vs. 65 s
- tSNE: 6 s vs. 96 s
- marker genes (approximation): 0.8 s vs. 96 s



theislab / scanpy_usage

Unwatch 2 · Star 3 · Fork 1

Code · Issues 0 · Pull requests 0 · Projects 0 · Wiki · Insights · Settings

Branch: master · scanpy_usage / 170505_seurat / · Create new file · Upload files · Find file · History

falexwolf added regressing out to readme — Latest commit c9cb49a 10 seconds ago

..		
figures	updated for version 0.2.9.1	38 minutes ago
README.md	added regressing out to readme	10 seconds ago
seurat.ipynb	updated for version 0.2.9.1	38 minutes ago
seurat_R.ipynb	updated for version 0.2.9.1	38 minutes ago

README.md

First compiled: May 5, 2017.
See the notebook.

Scanpy versus Seurat

Scanpy provides a number of Seurat's features (Satija *et al.*, Nat. Biotechnol., 2015), but at significantly higher computationally efficiency. Here, we reproduce most of Seurat's guided clustering tutorial as compiled on March 30, 2017. The tutorial starts with preprocessing and ends with the identification of cell types through marker genes of clusters. The data consists in *3k PBMCs from a Healthy Donor* and is freely available from 10x (here from this webpage). The profiling information for Seurat has been obtained within seurat_R.ipynb.

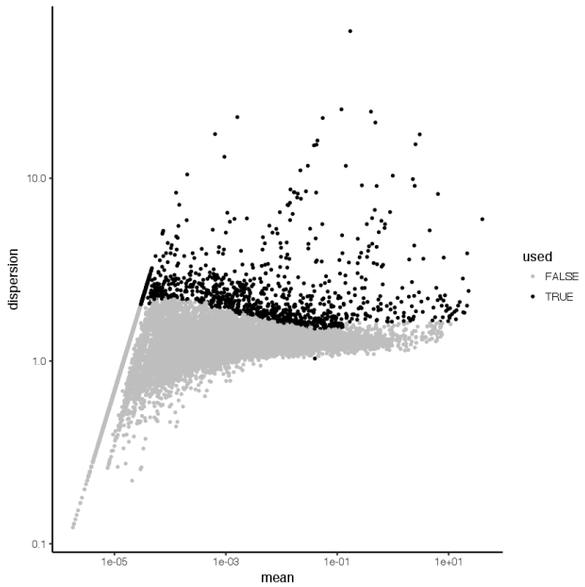
	Scanpy	Seurat
preprocessing	< 1 s	14 s
highly variable genes		
correction, regressing out	6 s	129 s
PCA	< 1 s	45 s
clustering	1.3 s	65 s
tSNE	6.5 s	25 s
finding marker genes	0.8 s	96 s

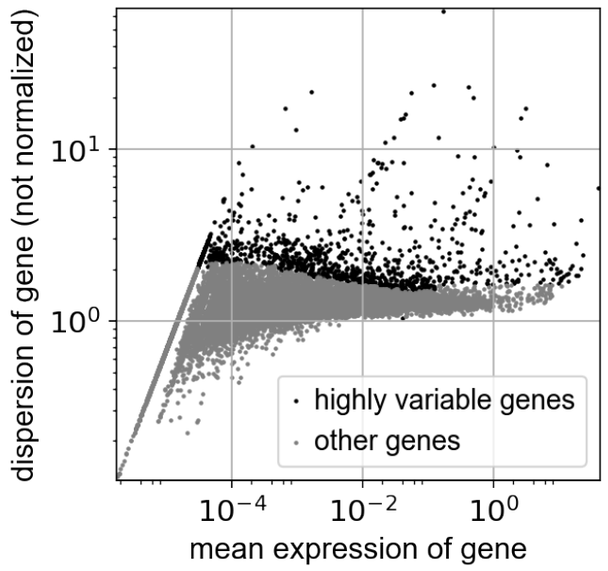
Scanpy vs. Cell Ranger

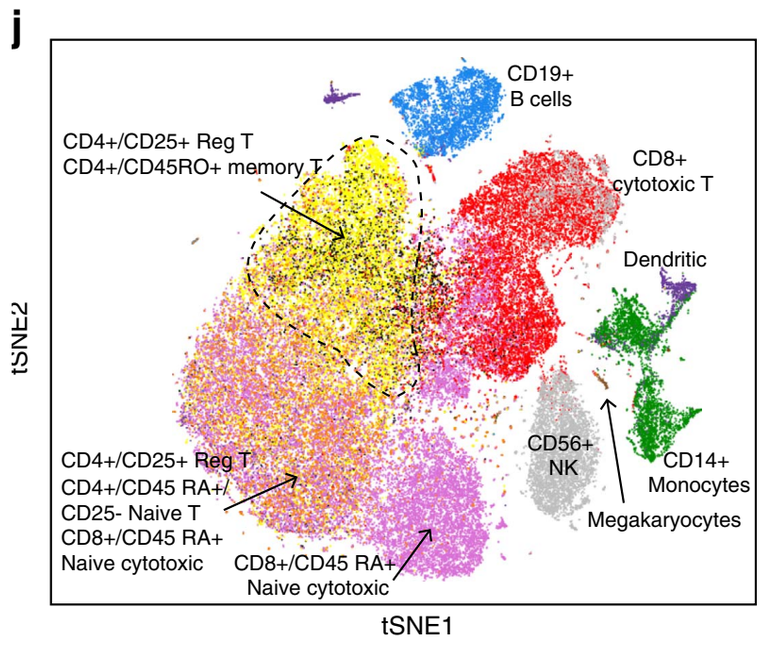
Zheng *et al.*, Nat. Commun. (2017)

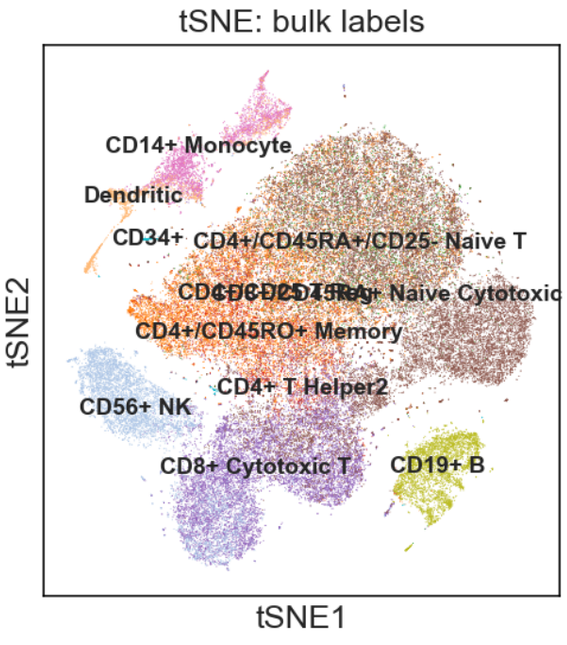
Benchmarked against Cell Ranger R kit, 68K cells.

- preprocessing: 14 s vs. 300 s
- PCA: 17 s vs. 120 s
- tSNE: 5 min vs. 26



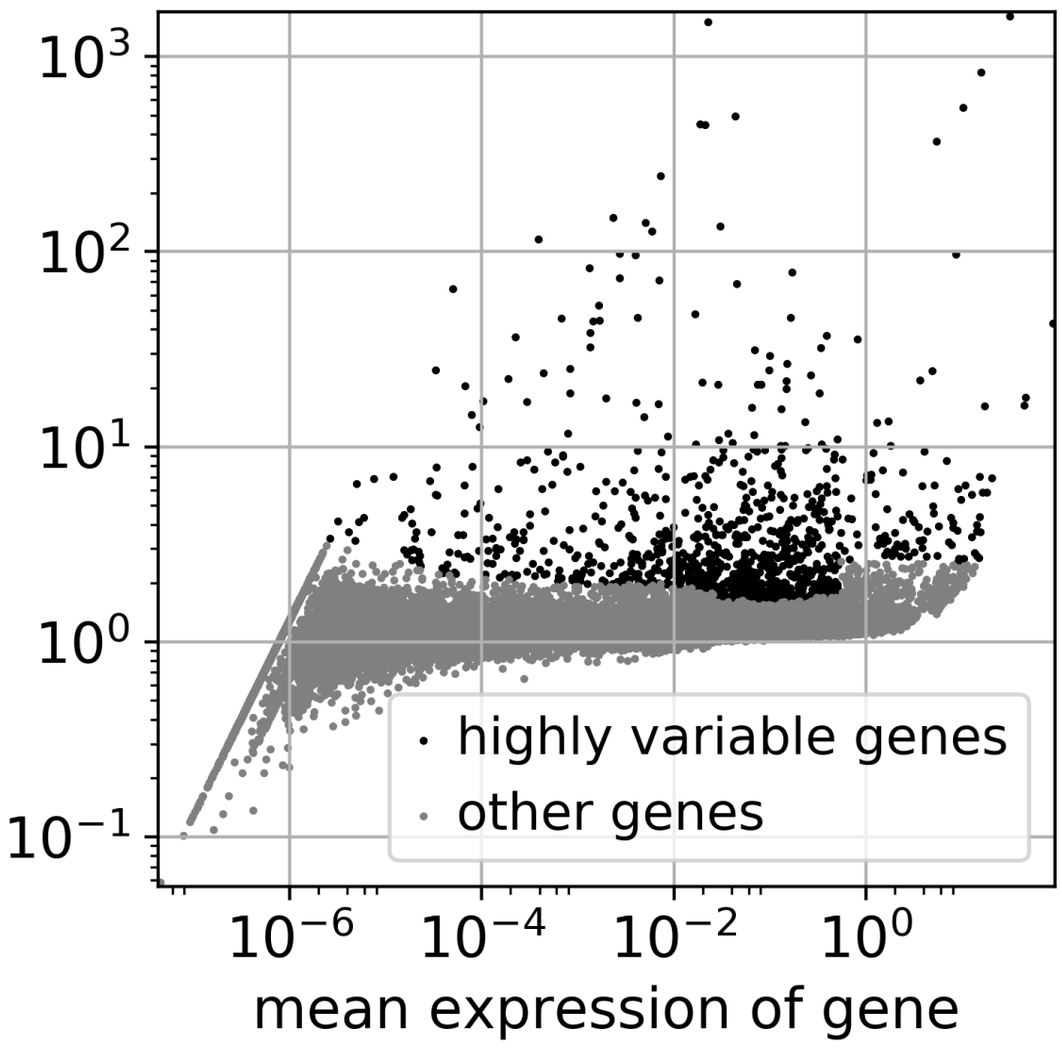


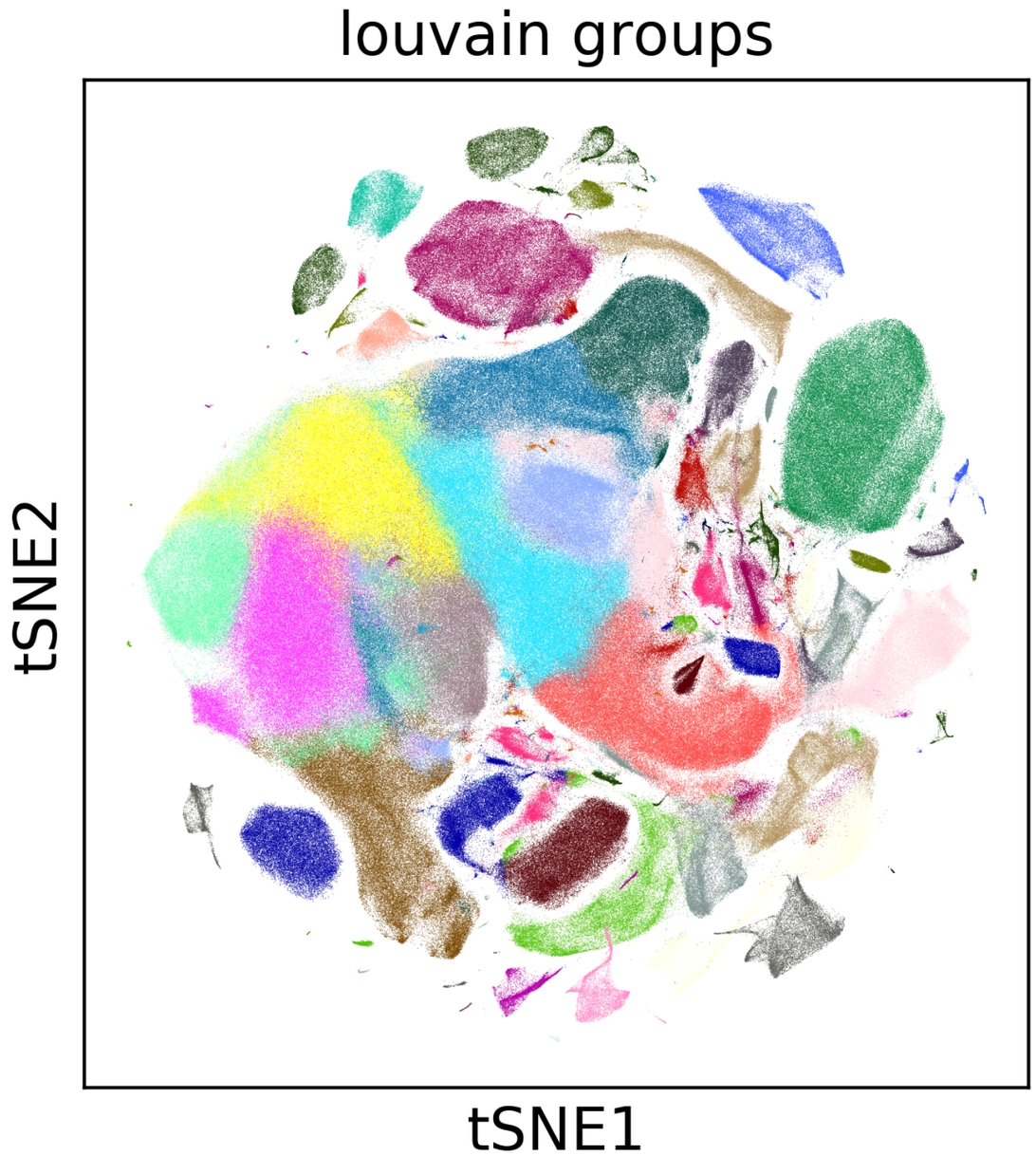




Scanpy scales to >1M cells

1.3M Neurons of 10x Genomics

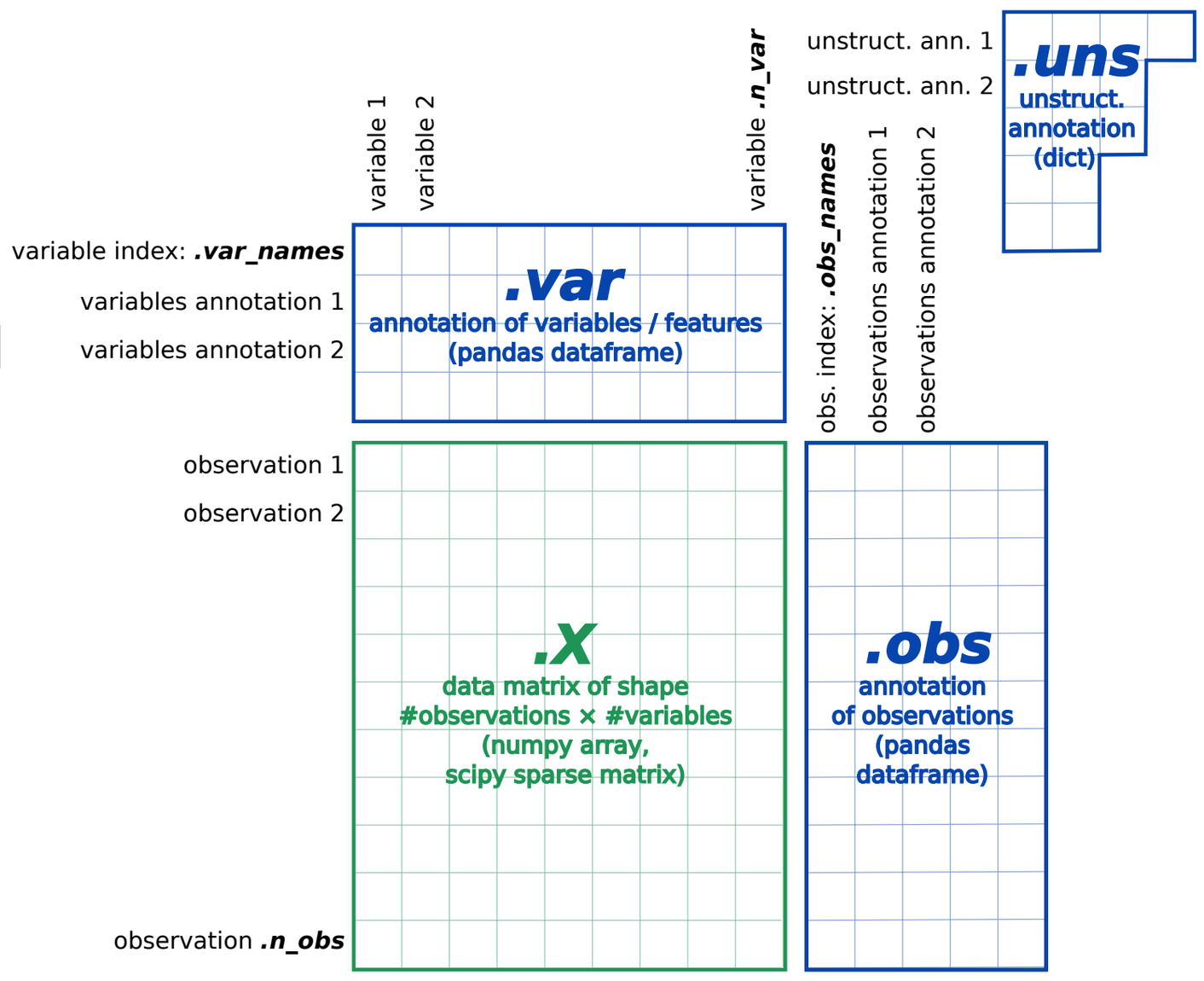




AnnData - Annotated Data

github.com/theislab/anndata

- *backed* and *memory* mode
- *views* on data
- interface to .*loom* files, 10x Genomics HDF5 files
- categorical data on HDF5 level
- sparse data on HDF5 level (*anndata.h5py, h5sparse*)
- arbitrary unstructured annotations
- integration with pandas and conventions of Python ecosystem



anndata.AnnData

class `anndata.AnnData`(*X=None, obs=None, var=None, uns=None, obsm=None, varm=None, raw=None, dtype='float32', single_col=False, filename=None, filemode=None, asview=False, oidx=None, vidx=None*)

github.com/theislab/anndata

Attributes

`X`	Data matrix of shape *n_obs* × *n_vars* (*np.ndarray*, *sp.sparse.spmatrix*).
`filename`	Change to backing mode by setting the filename of a *.h5ad* file.
`isbacked`	*True* if object is backed on disk, *False* otherwise.
`isview`	*True* if object is view of another AnnData object, *False* otherwise.
`n_obs`	Number of observations.
`n_vars`	Number of variables/features.
`shape`	Shape of data matrix: (n_obs, n_vars).
`obs`	One-dimensional annotation of observations (*pd.DataFrame*).
`obsm`	Multi-dimensional annotation of observations (mutable structure
`obs_names`	Names of observations (alias for *.obs.index*).
`raw`	Store raw version of *.X* and *.var* as *.raw.X* and *.raw.X*.
`var`	One-dimensional annotation of variables/ features (*pd.DataFrame*
`varm`	Multi-dimensional annotation of variables/ features (mutable stru
`var_names`	Names of variables (alias for *.var.index*).

Methods

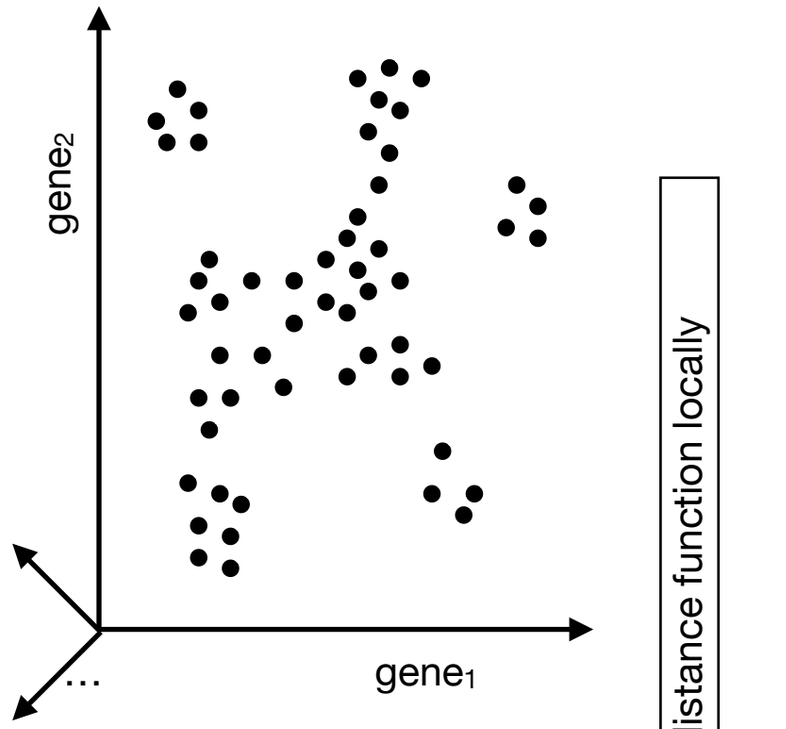
`concatenate`	(adatas[, ...])
`copy`	([filename])
`transpose`	()
`obs_names_make_unique`	([join])
`var_names_make_unique`	([join])
`write`	([filename, compression, compression_opts])
`write_csvs`	(dirname[, ...])
`write_loom`	(filename)

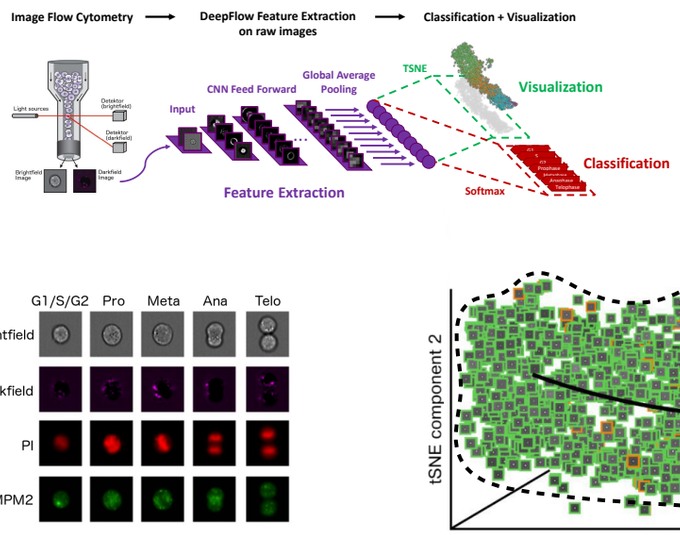
Graph representation Islam *et al.*, Genome Research (2011)

- High-dimensional data → make guess for distance metric $d(\boldsymbol{x}, \boldsymbol{y})$ → evaluate d locally → generate neighborhood graph of single cells

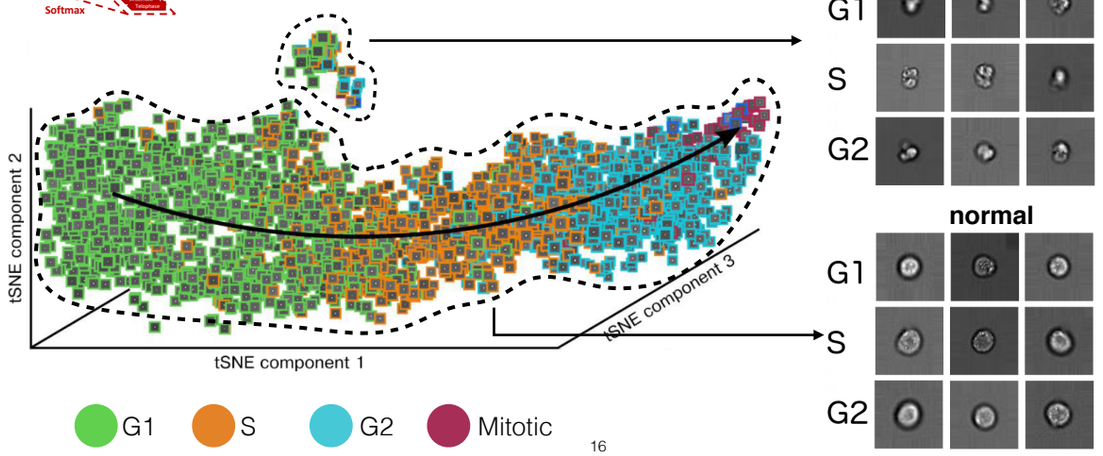
- Typically, obtain d from preprocessing and something like euclidean distance.

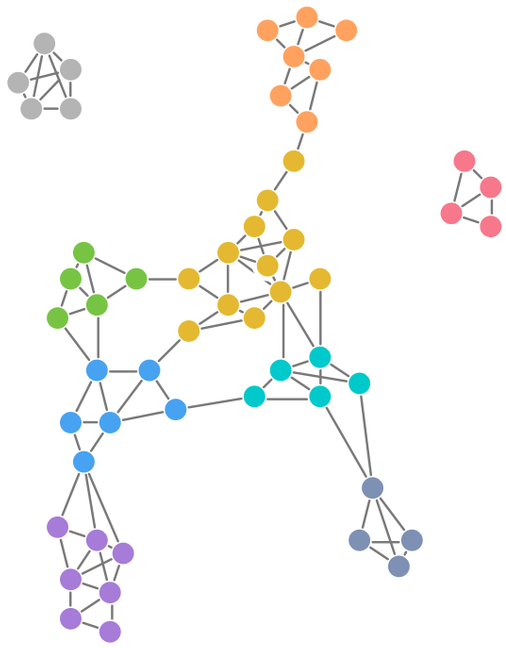
- Alternatively, *learn* the distance d:





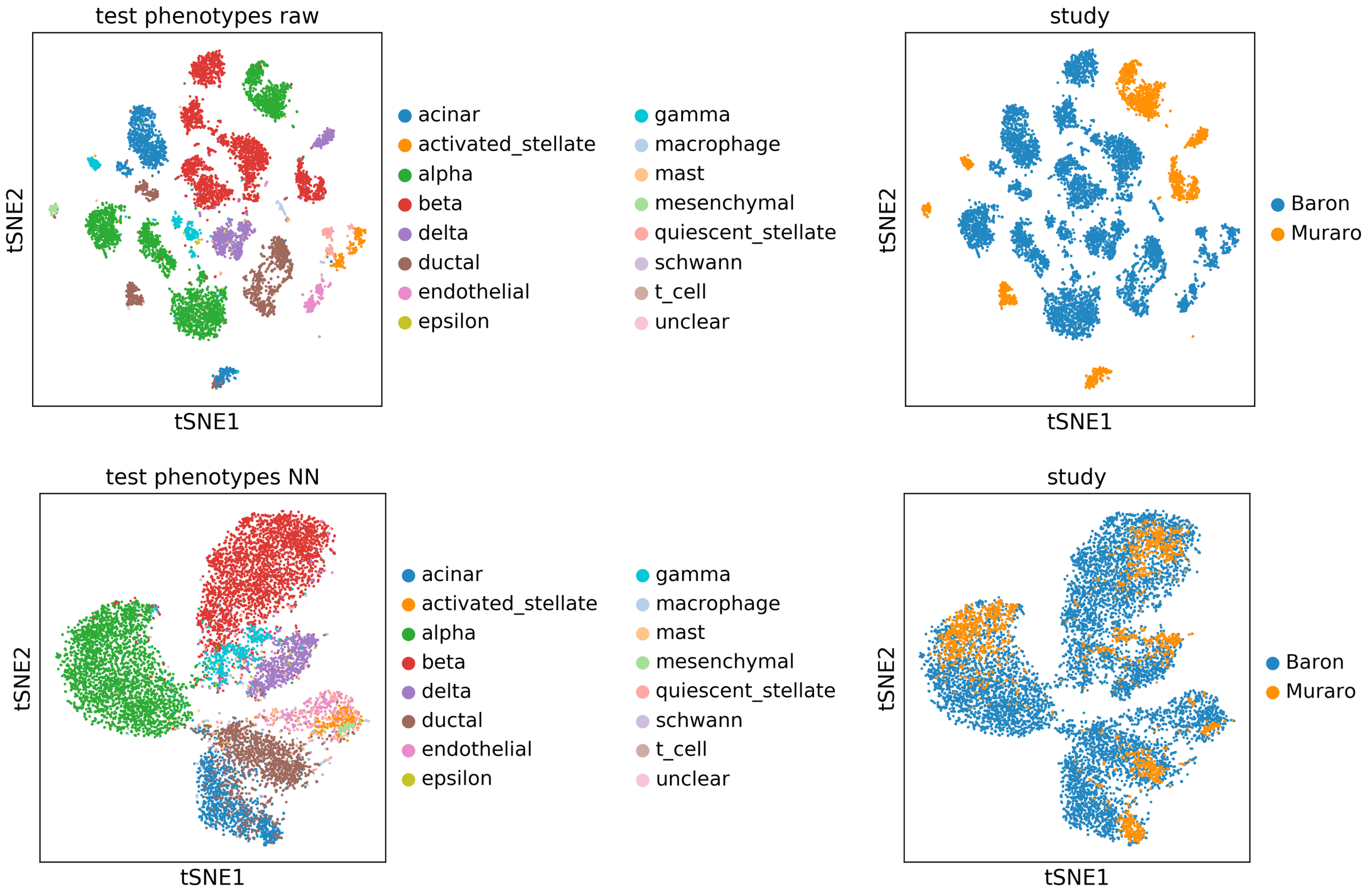
Eulenberg, Köhler et al., Nat. Commun. (2017)





Learned distances resolve batch effects

Scanpy 1.0 "Universal preprocessing". No manual alignment necessary.



Neighbors *Scanpy 1.0*

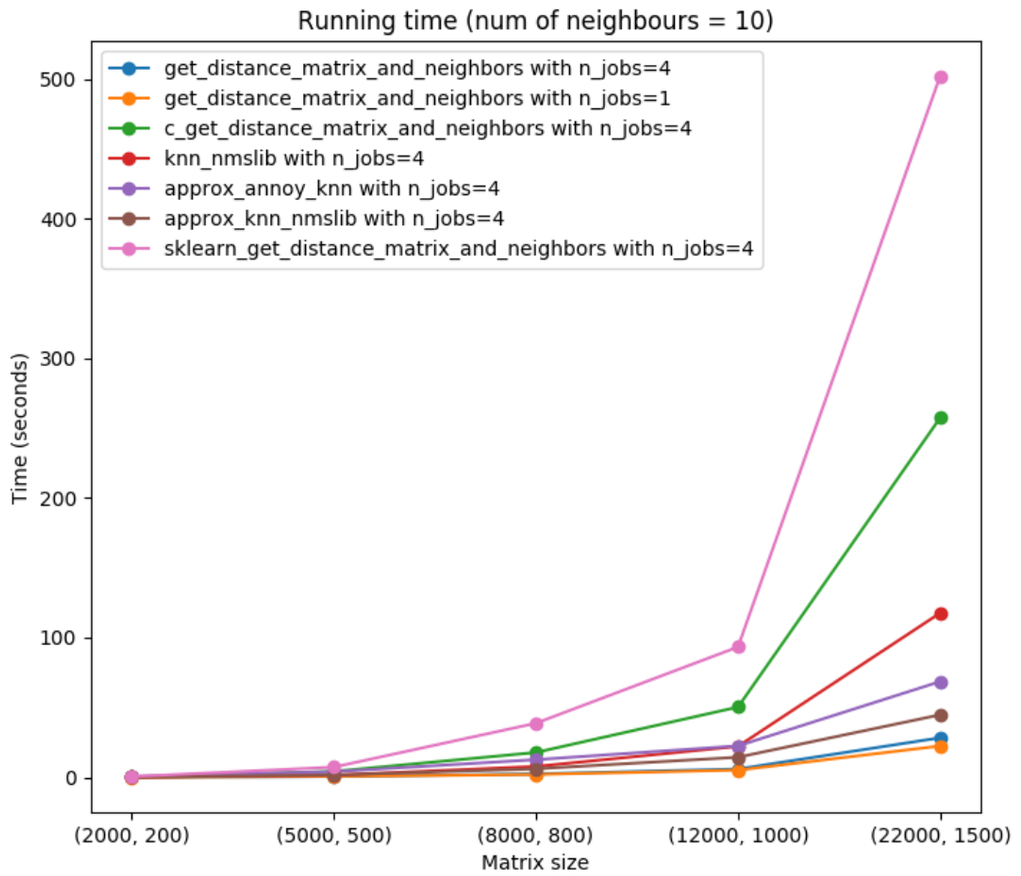
Class for representing data as a graph of neighborhood relations between data points.

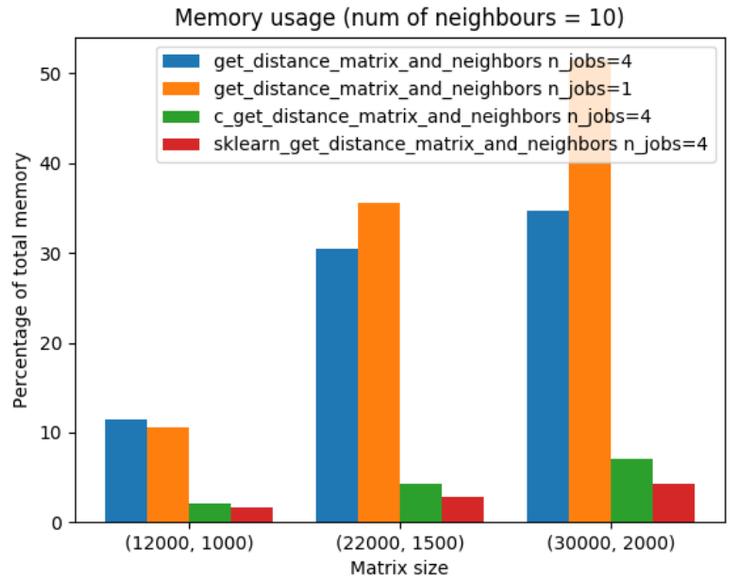
Simplest case: knn graph.

- much faster than *sklearn.neighbors*
- much faster than C, annoy, nmslib for < 100k cells

Also, *Neighbors* offers functions related to stochastic processes on graphs, absent in *igraph*, *networks*, *graph-tools*.

Scanpy 1.0 use umap implementation. Easily installed and very fast.





scanpy.api.Neighbors *Scanpy 1.0*

class `scanpy.api.Neighbors`(*adata*, *n_jobs=None*)

Data represented as graph of nearest neighbors.

Represent a data matrix as a graph of nearest neighbor relations (edges) among data points (nodes).

Attributes

`distances`	Distances between data points.
`similarities`	Similarities between data points, closely related to a transition matrix.
`eigen_values`	Eigen values of similarity matrix.
`eigen_basis`	Eigen basis of similarity matrix.
`laplacian`	Graph laplacian.

Methods

`compute_distances` ([n_neighbors, knn, n_pcs])	Compute distances.
`compute_similarities` ([alpha])	Compute similarities.
`compute_eigen` ([n_comps, sym, sort, matrix])	Compute eigen decomposition of similarity matr
`compute_laplacian` ()	Graph Laplacian for K.
`to_igraph` ()	Generate igraph object.

Many Scanpy tools use *Neighbors*

A single representation of the data for all common analysis tasks.

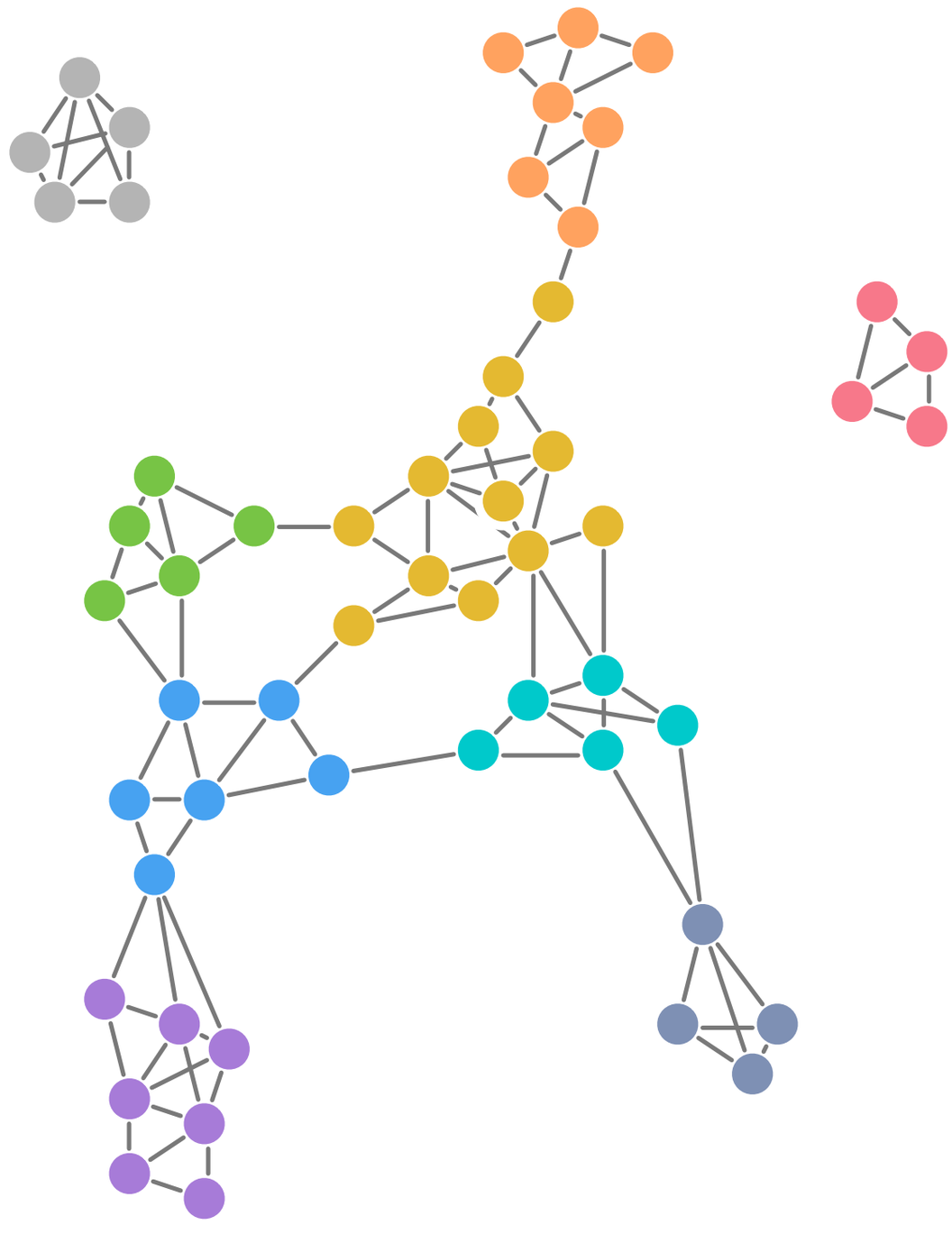
- clustering

  Levine et al., Cell (2015), Xu et al. Bioinf (2015) …
- pseudotime inference

  Trapnell et al., Bendall et al. (2014), Haghverdi et al. (2016), …
- graph drawing

  Islam *et al.*, Genome Research (2011), …
- manifold learning (tSNE, diffmap, …)

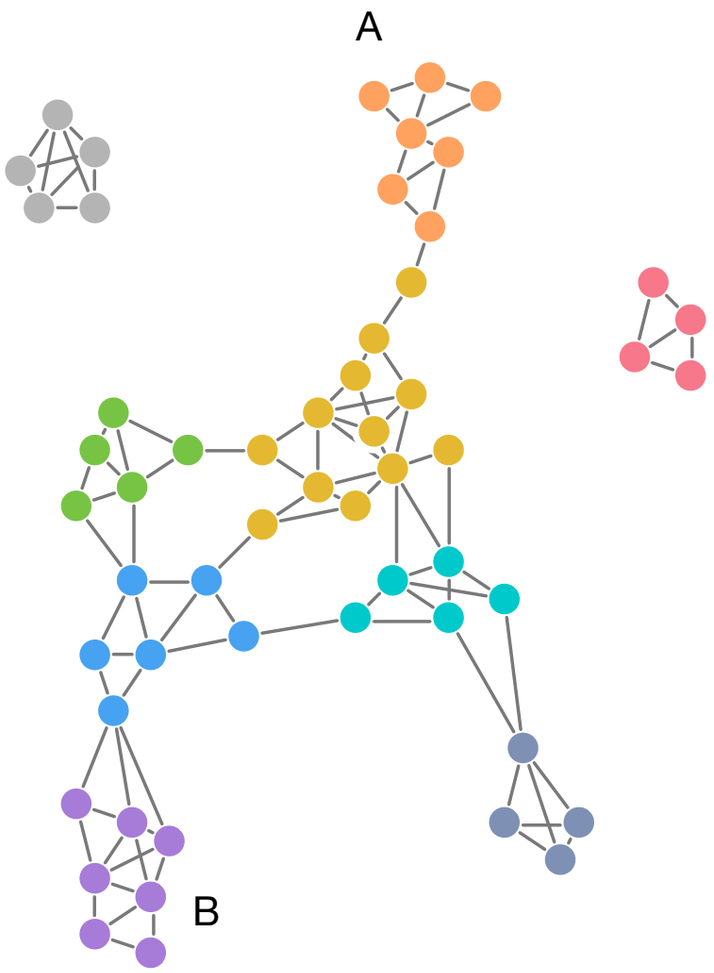
  Amir et al., Nat Biotechn (2013) …



A graph-based coordinate system

Graph Abstraction maps both connected and disconnected structure.

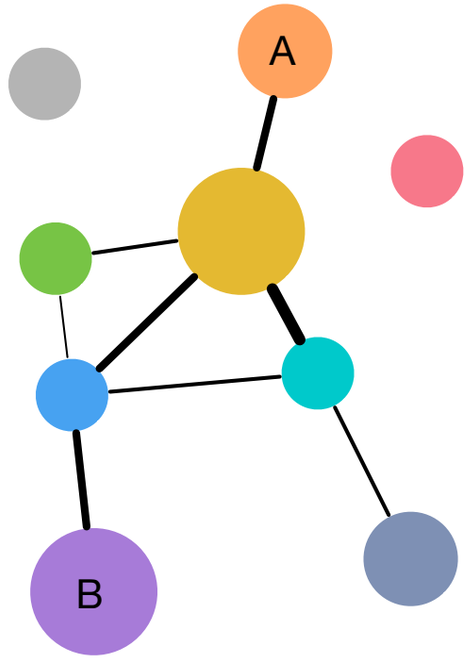
Scanpy 1.0



measure group connectivity

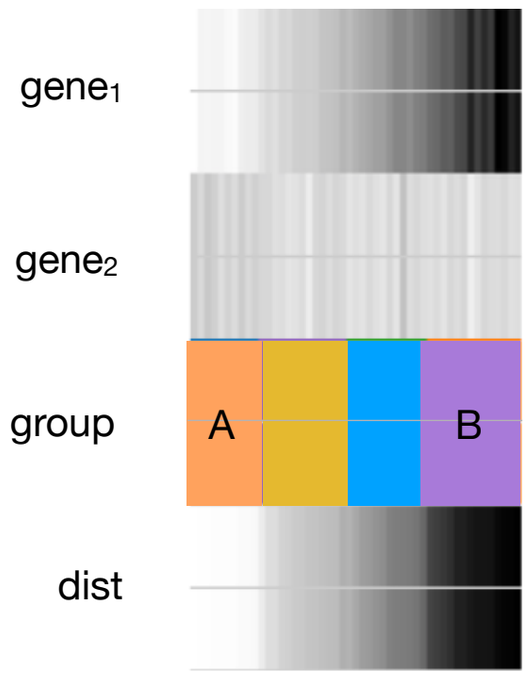
choose paths

compute random-walk based distance, extension of DPT



abstracted graph:

discrete coordinate



paths in abstracted graph:

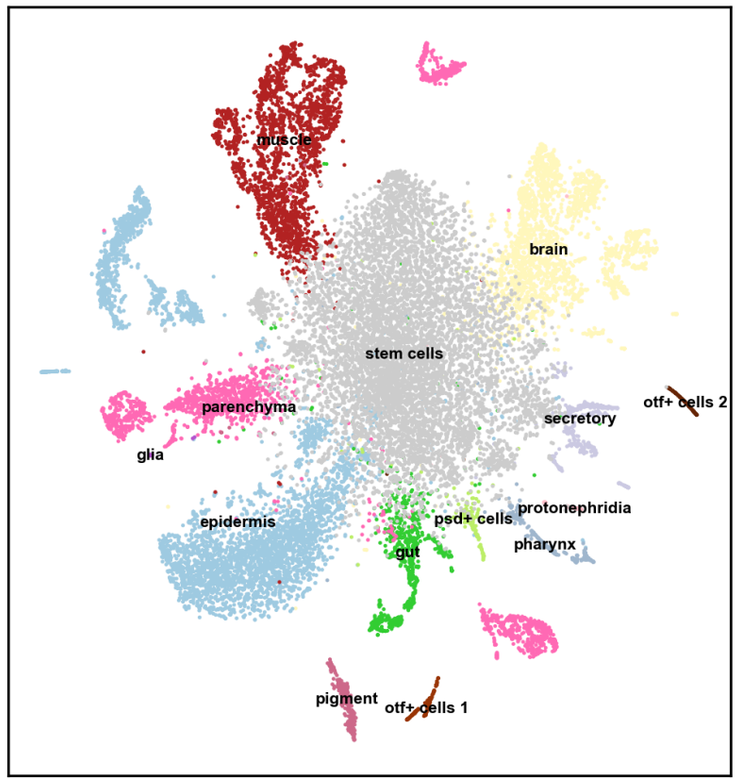
continuous coordinate, single-cell resolved

Wolf *et al.*, bioRxiv (2017)

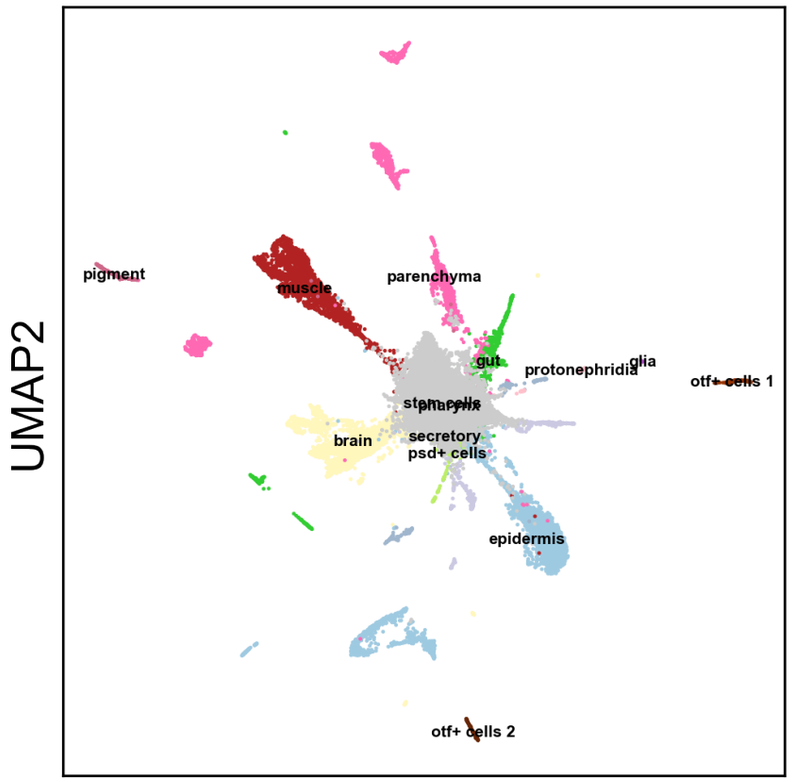
Map of whole organism

Plass *et al.*, unpublished (2017)
Wolf *et al.*, bioRxiv (2017)

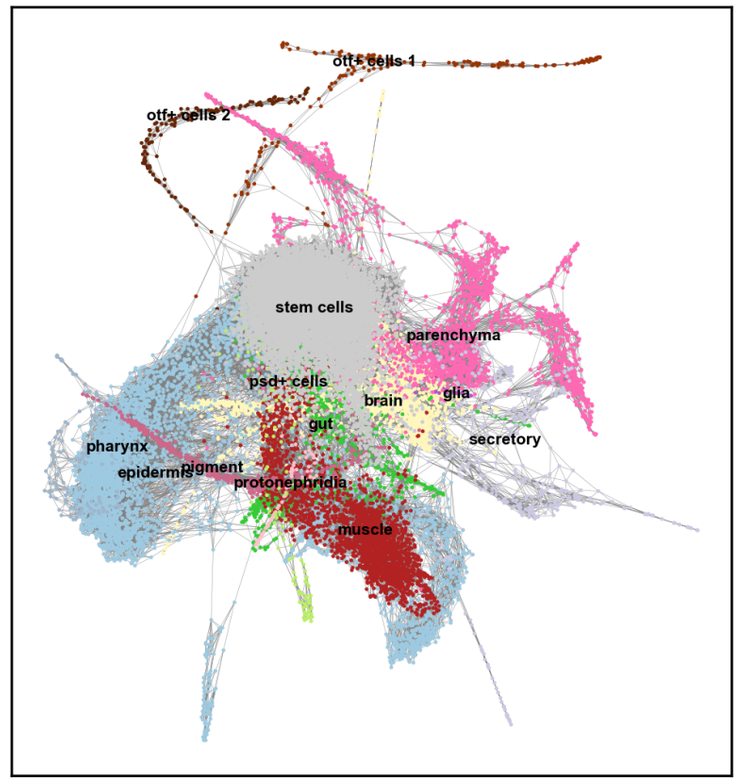
coarse resolution



coarse resolution



coarse resolution



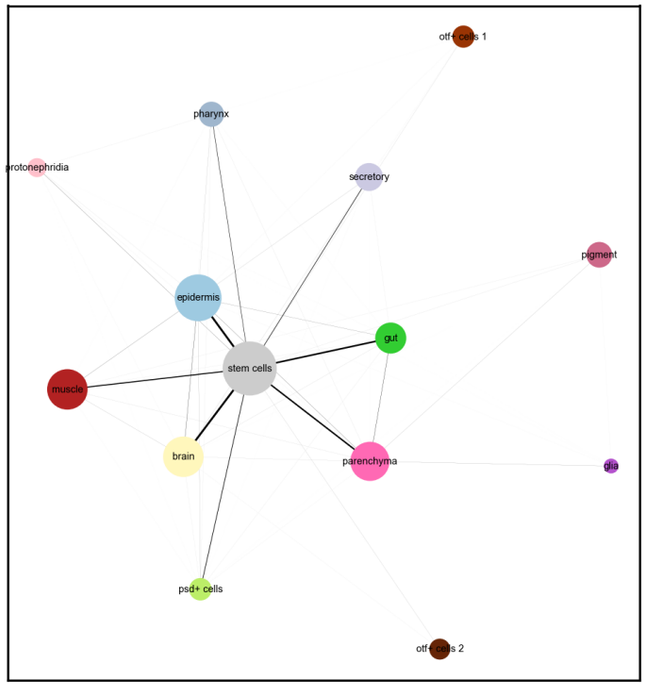
Manifold learning, graph drawing [& topological data analysis]:

- usually violate topological structure
- for large data, provide too much detail, results far from “canonical”
- graph drawing does not scale

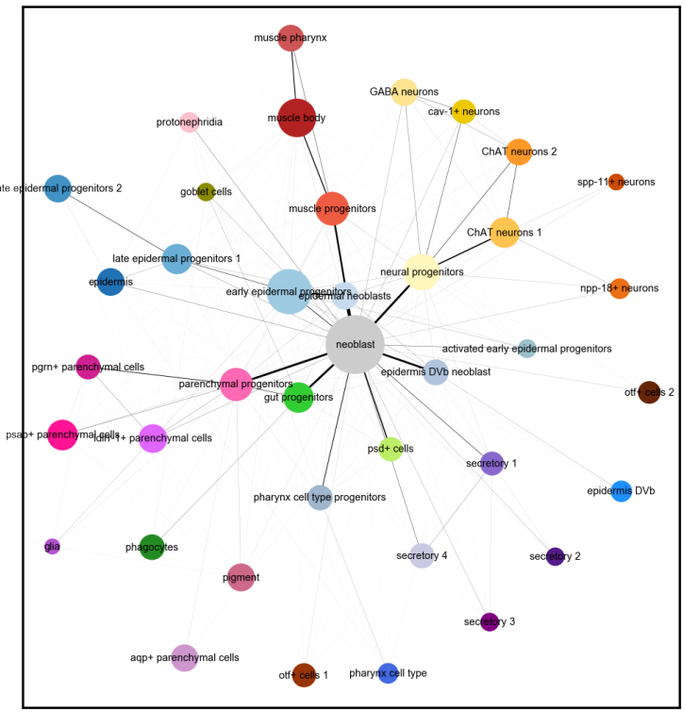
Map of whole organism

Plass *et al.*, unpublished (2017)
Wolf *et al.*, bioRxiv (2017)

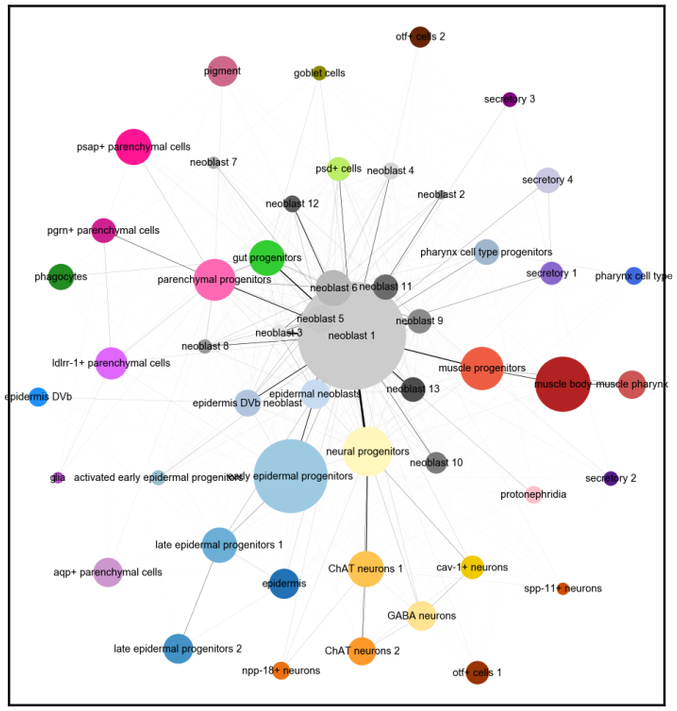
coarse resolution



finer resolution



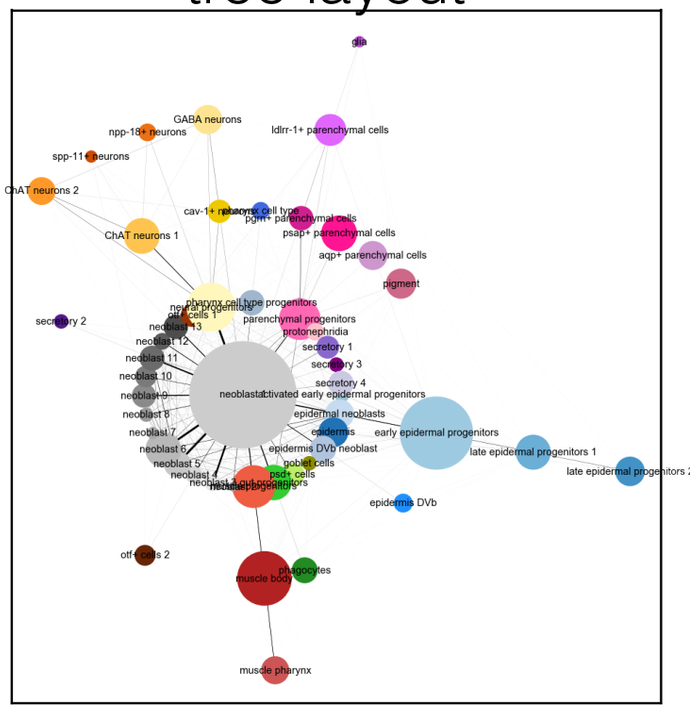
fine resolution



Graph abstraction:

- faithful to topological structure
- provides the wished level of detail
- extremely fast, scales arbitrarily

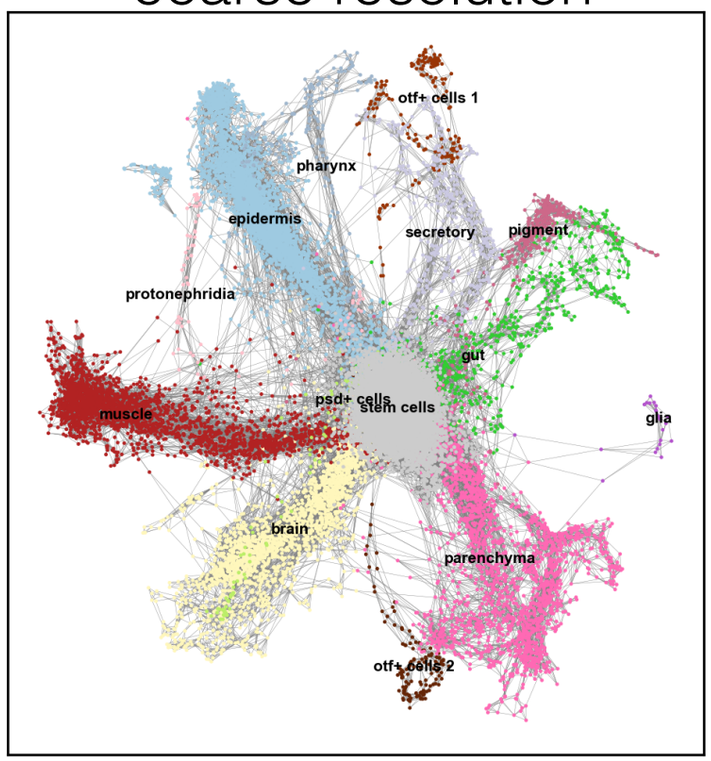
tree layout



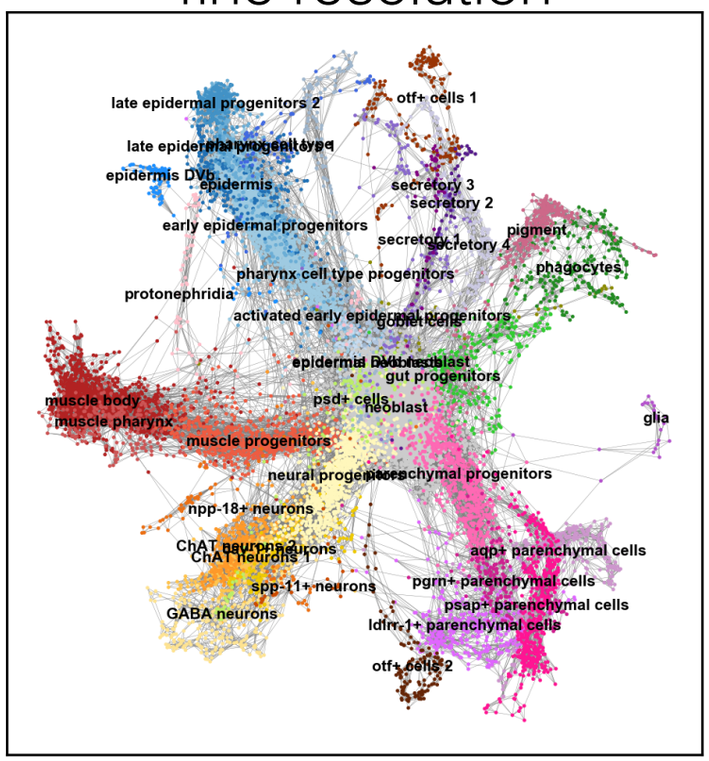
Map of whole organism

Plass *et al.*, unpublished (2017)
Wolf *et al.*, bioRxiv (2017)

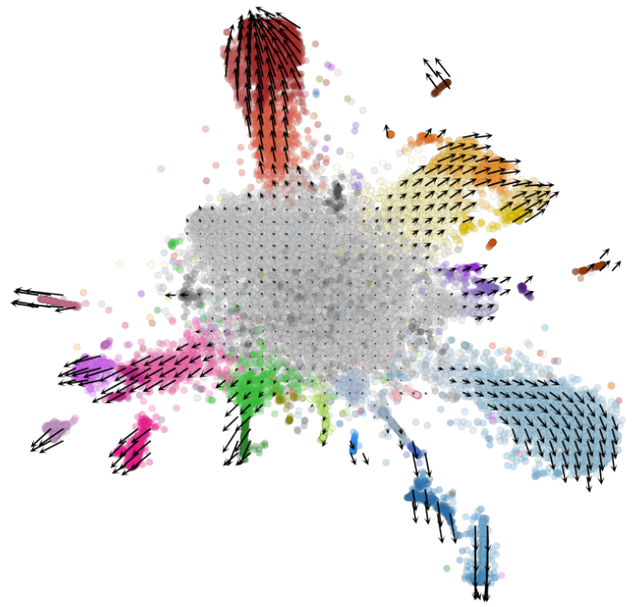
coarse resolution



fine resolution



velocyto

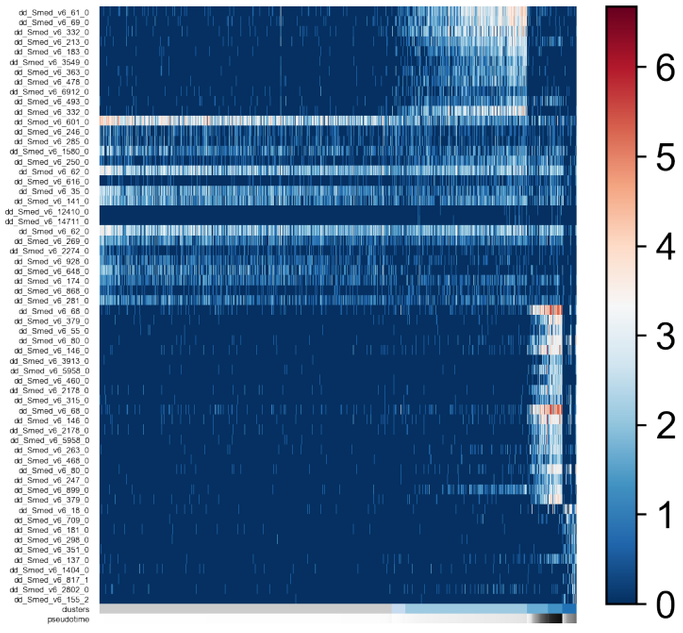


La Manno *et al.*, bioRxiv (2017)

Graph abstraction:

- enables topologically faithful visualisation at single-cell level
- reveals putative lineages, which can be oriented, e.g., with velocyto
- continuous coordinates, extending DPT

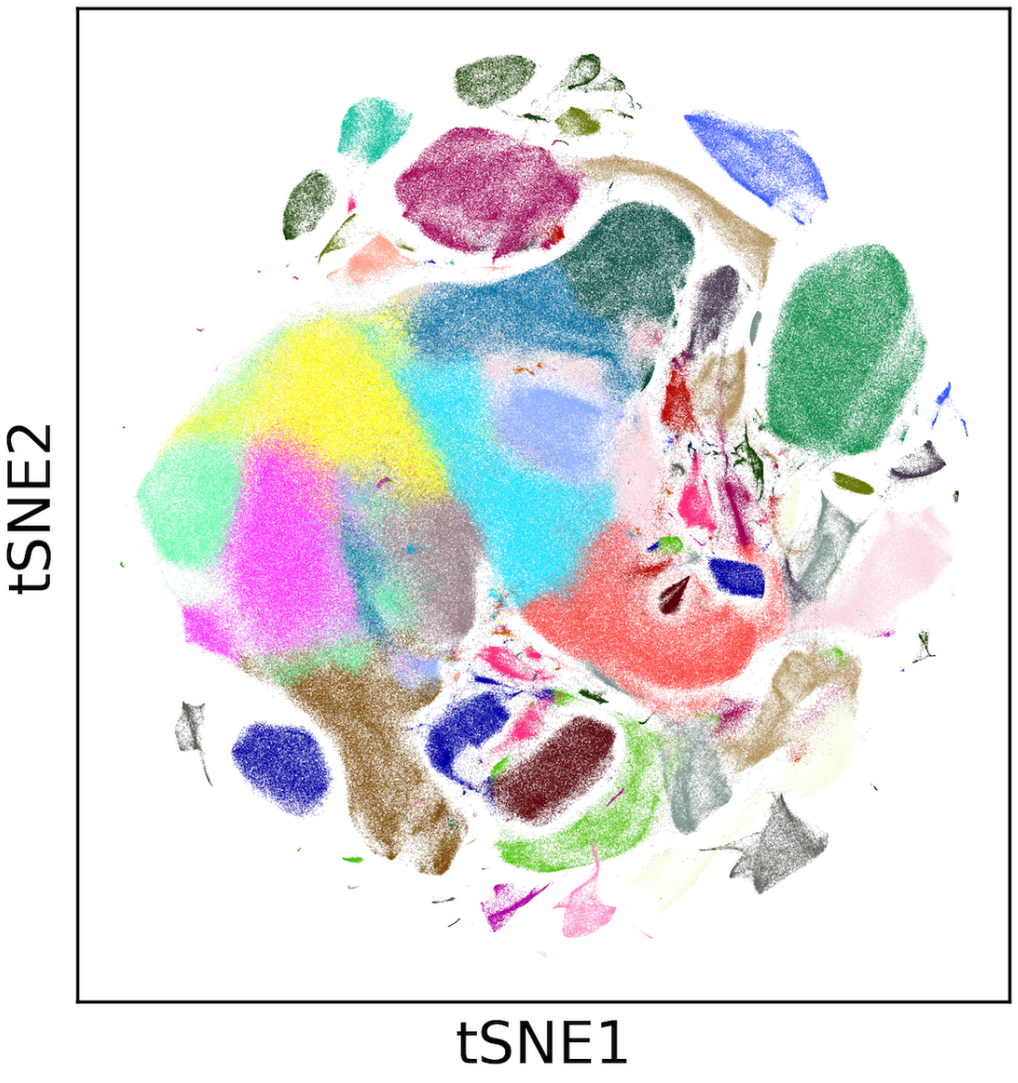
epidermal lineage



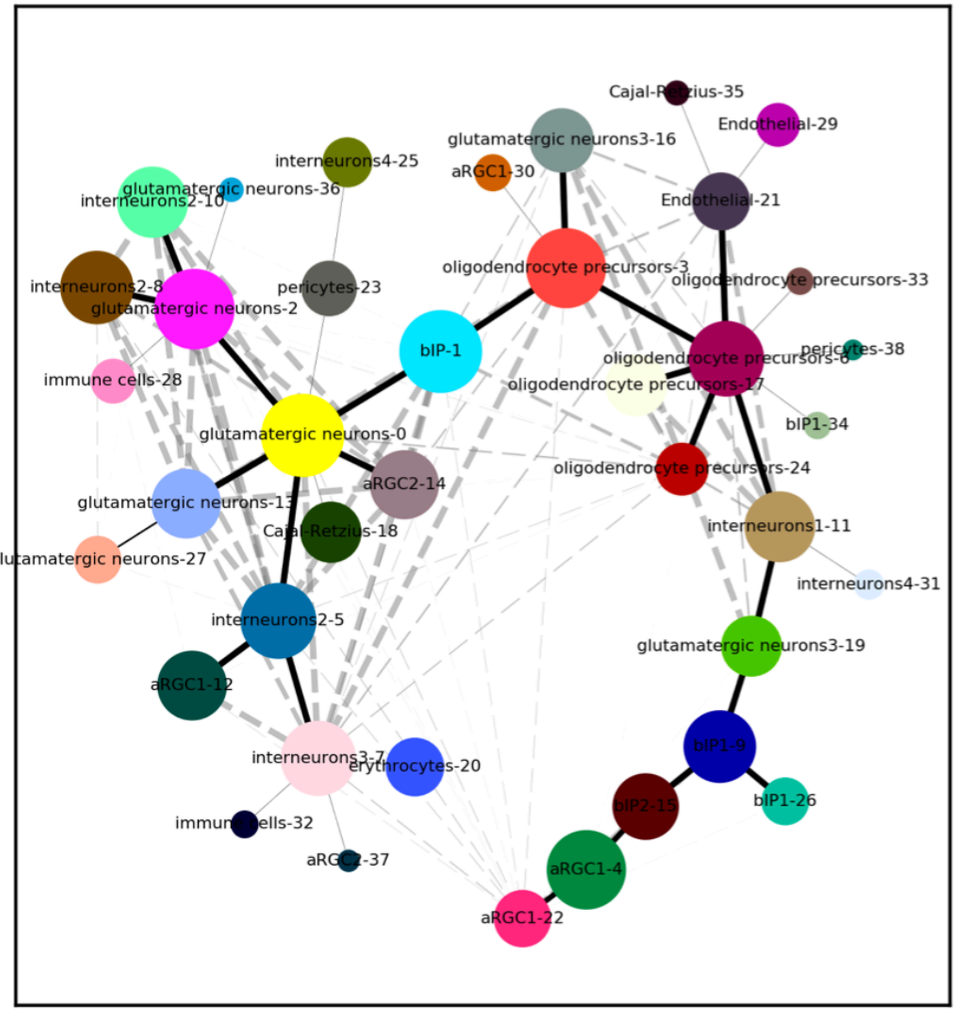
Haghverdi *et al.*, Nat Meth (2016)

1.3M neurons 10x Genomics

louvain groups



abstracted graph

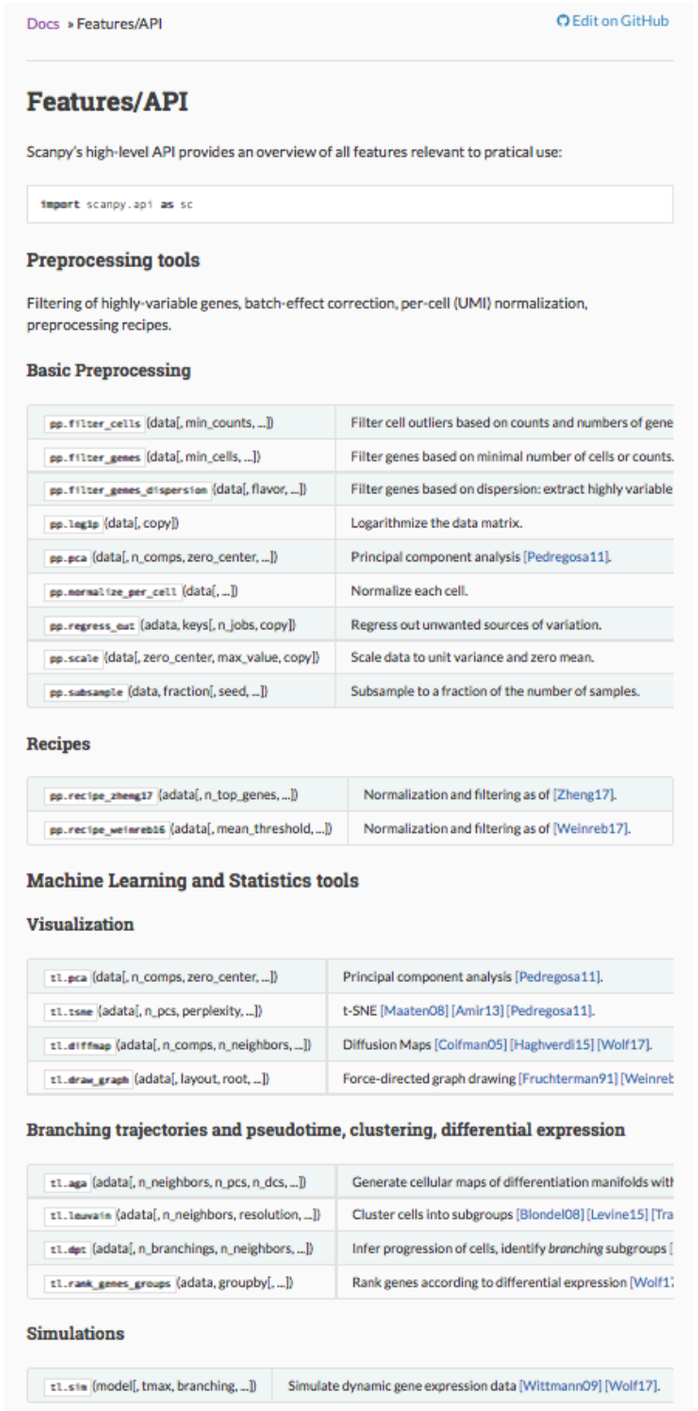


Graph drawing not feasible. Abstracted graph representation takes around 1 h; tSNE around 4 h.

Scanpy's API

Modular API.

- *sc.preprocessing*
- *sc.tools*
- *sc.plotting*
- *sc.datasets*
- *sc.settings*
- *sc.logging*



Docs » Features/API

Edit on GitHub

Features/API

Scanpy's high-level API provides an overview of all features relevant to pratical use:

```
import scanpy.api as sc
```

Preprocessing tools

Filtering of highly-variable genes, batch-effect correction, per-cell (UMI) normalization, preprocessing recipes.

Basic Preprocessing

`pp.filter_cells` (data[, min_counts, ...])	Filter cell outliers based on counts and numbers of gene
`pp.filter_genes` (data[, min_cells, ...])	Filter genes based on minimal number of cells or counts.
`pp.filter_genes_dispersion` (data[, flavor, ...])	Filter genes based on dispersion: extract highly variable
`pp.log1p` (data[, copy])	Logarithmize the data matrix.
`pp.pca` (data[, n_comps, zero_center, ...])	Principal component analysis [Pedregosa11].
`pp.normalize_per_cell` (data[, ...])	Normalize each cell.
`pp.regress_out` (adata, keys[, n_jobs, copy])	Regress out unwanted sources of variation.
`pp.scale` (data[, zero_center, max_value, copy])	Scale data to unit variance and zero mean.
`pp.subsample` (data, fraction[, seed, ...])	Subsample to a fraction of the number of samples.

Recipes

`pp.recipe_zheng17` (adata[, n_top_genes, ...])	Normalization and filtering as of [Zheng17].
`pp.recipe_weinreb16` (adata[, mean_threshold, ...])	Normalization and filtering as of [Weinreb17].

Machine Learning and Statistics tools

Visualization

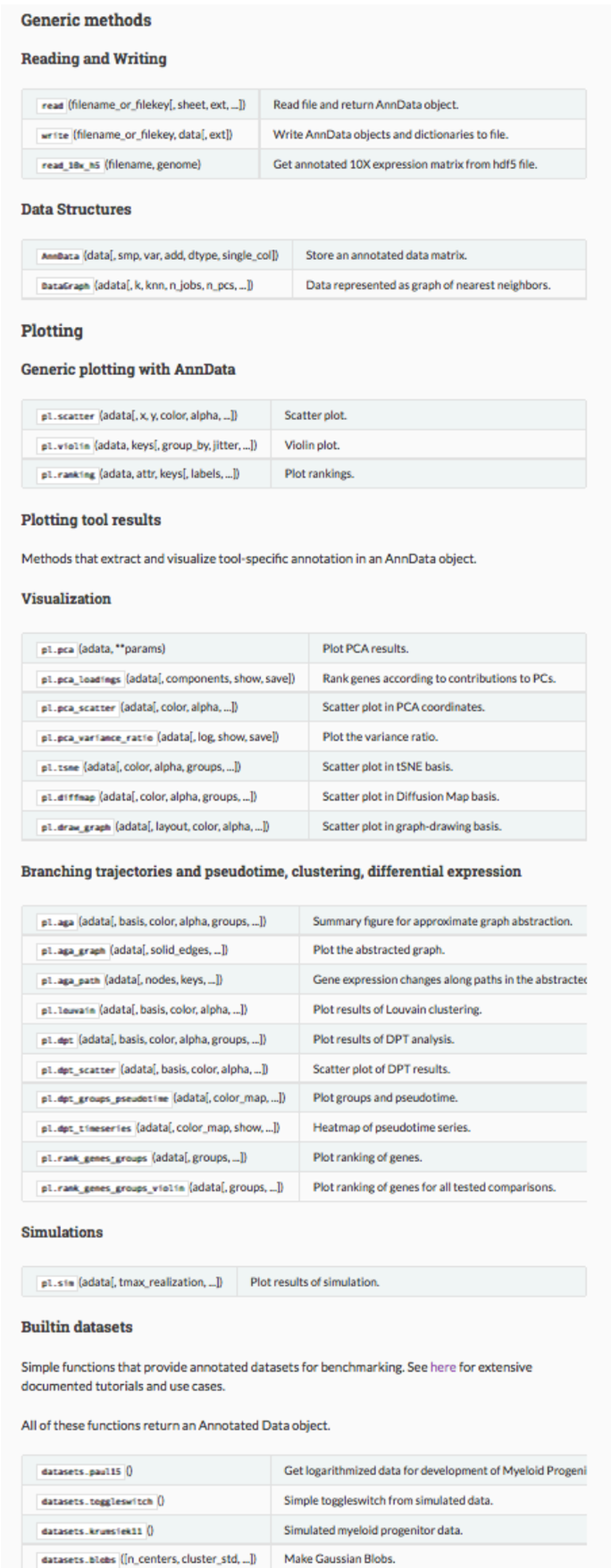
`tl.pca` (data[, n_comps, zero_center, ...])	Principal component analysis [Pedregosa11].
`tl.tsne` (adata[, n_pcs, perplexity, ...])	t-SNE [Maaten08] [Amir13] [Pedregosa11].
`tl.diffmap` (adata[, n_comps, n_neighbors, ...])	Diffusion Maps [Coifman05] [Haghverdi15] [Wolf17].
`tl.draw_graph` (adata[, layout, root, ...])	Force-directed graph drawing [Fruchterman91] [Weinreb

Branching trajectories and pseudotime, clustering, differential expression

`tl.aga` (adata[, n_neighbors, n_pcs, n_dcs, ...])	Generate cellular maps of differentiation manifolds with
`tl.louvain` (adata[, n_neighbors, resolution, ...])	Cluster cells into subgroups [Blondel08] [Levine15] [Tra
`tl.dpt` (adata[, n_branchings, n_neighbors, ...])	Infer progression of cells, identify *branching* subgroups
`tl.rank_genes_groups` (adata, groupby[, ...])	Rank genes according to differential expression [Wolf1

Simulations

`tl.sim` (model[, tmax, branching, ...])	Simulate dynamic gene expression data [Wittmann09] [Wolf17].



Generic methods

Reading and Writing

`read` (filename_or_filekey[, sheet, ext, ...])	Read file and return AnnData object.
`write` (filename_or_filekey, data[, ext])	Write AnnData objects and dictionaries to file.
`read_10x_h5` (filename, genome)	Get annotated 10X expression matrix from hdf5 file.

Data Structures

`AnnData` (data[, smp, var, add, dtype, single_col])	Store an annotated data matrix.
`DataGraph` (adata[, k, knn, n_jobs, n_pcs, ...])	Data represented as graph of nearest neighbors.

Plotting

Generic plotting with AnnData

`pl.scatter` (adata[, x, y, color, alpha, ...])	Scatter plot.
`pl.violin` (adata, keys[, group_by, jitter, ...])	Violin plot.
`pl.ranking` (adata, attr, keys[, labels, ...])	Plot rankings.

Plotting tool results

Methods that extract and visualize tool-specific annotation in an AnnData object.

Visualization

`pl.pca` (adata, **params)	Plot PCA results.
`pl.pca_loadings` (adata[, components, show, save])	Rank genes according to contributions to PCs.
`pl.pca_scatter` (adata[, color, alpha, ...])	Scatter plot in PCA coordinates.
`pl.pca_variance_ratio` (adata[, log, show, save])	Plot the variance ratio.
`pl.tsne` (adata[, color, alpha, groups, ...])	Scatter plot in tSNE basis.
`pl.diffmap` (adata[, color, alpha, groups, ...])	Scatter plot in Diffusion Map basis.
`pl.draw_graph` (adata[, layout, color, alpha, ...])	Scatter plot in graph-drawing basis.

Branching trajectories and pseudotime, clustering, differential expression

`pl.aga` (adata[, basis, color, alpha, groups, ...])	Summary figure for approximate graph abstraction.
`pl.aga_graph` (adata[, solid_edges, ...])	Plot the abstracted graph.
`pl.aga_path` (adata[, nodes, keys, ...])	Gene expression changes along paths in the abstracte
`pl.louvain` (adata[, basis, color, alpha, ...])	Plot results of Louvain clustering.
`pl.dpt` (adata[, basis, color, alpha, groups, ...])	Plot results of DPT analysis.
`pl.dpt_scatter` (adata[, basis, color, alpha, ...])	Scatter plot of DPT results.
`pl.dpt_groups_pseudotime` (adata[, color_map, ...])	Plot groups and pseudotime.
`pl.dpt_timeseries` (adata[, color_map, show, ...])	Heatmap of pseudotime series.
`pl.rank_genes_groups` (adata[, groups, ...])	Plot ranking of genes.
`pl.rank_genes_groups_violin` (adata[, groups, ...])	Plot ranking of genes for all tested comparisons.

Simulations

`pl.sim` (adata[, tmax_realization, ...])	Plot results of simulation.

Builtin datasets

Simple functions that provide annotated datasets for benchmarking. See here for extensive documented tutorials and use cases.

All of these functions return an Annotated Data object.

`datasets.paul15` ()	Get logarithmized data for development of Myeloid Progeni
`datasets.toggleswitch` ()	Simple toggleswitch from simulated data.
`datasets.krumsiek11` ()	Simulated myeloid progenitor data.
`datasets.blobs` ([n_centers, cluster_std, ...])	Make Gaussian Blobs.

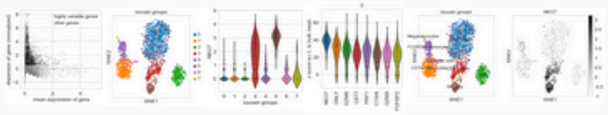
Scanpy's use cases

Docs » Examples Edit on GitHub

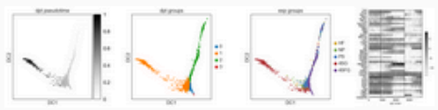
Examples

Good starting points are the following examples, which build on established results from the literature. All examples are versioned on GitHub.

Example 1: Seurat's [Satija15] guided clustering tutorial.



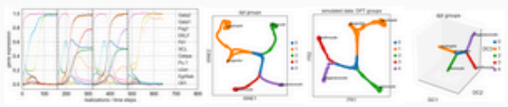
Example 2: The Diffusion Pseudotime (DPT) analyses of [Haghverdi16] for data of [Paul15] and [Moignard15]. Note that DPT has recently been very favorably discussed by the authors of Monocle.



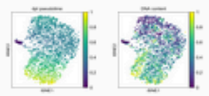
Example 3: Analyzing 68 000 cells from [Zheng17], we find that Scanpy is about a factor 5 to 16 faster and more memory efficient than the Cell Ranger R kit for secondary analysis.

Example 4: Visualizing 1.3 mio brain cells.

Example 5: Simulating single cells using literature-curated gene regulatory networks [Wittmann09]; here, myeloid differentiation [Krumsiek11].



Example 6: Pseudotime-based vs. deep-learning based reconstruction of cell cycle from image data [Eulenberg17].



theislab / graph_abstraction

Unwatch 3 | Star 8 | Fork 1

Code | Issues 2 | Pull requests 0 | Projects 0 | Wiki | Insights | Settings

Generate cellular maps of differentiation manifolds with complex topologies. Edit

Add topics

26 commits | 1 branch | 0 releases | 1 contributor | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

falexwolf updated readme — Latest commit 915f328 12 days ago

deep_learning	updated everything	13 days ago
minimal_examples	matching the preprint	13 days ago
nestorowa16	updated readmes	12 days ago
paul15	updated readmes	12 days ago
pbmcs	removed 33k dataset	13 days ago
planaria	matching the preprint	13 days ago
.gitignore	initial commit	2 months ago
LICENSE	add license	2 months ago
README.md	updated readme	12 days ago

README.md

Graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells

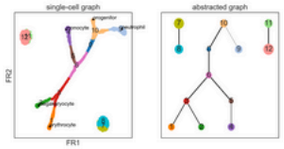
This repository allows to reproduce analyses and figures of the preprint.

Graph abstraction is available within Scanpy. Central toplevel functions are:

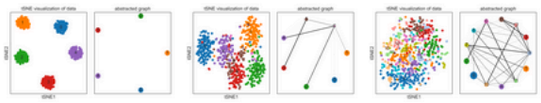
- `scanpy.api.tools.aga`
- `scanpy.api.plotting.aga_graph`
- `scanpy.api.plotting.aga_path`

Minimal examples with known ground truth

In *minimal_examples*, we study clean simulated datasets with known ground truth. In particular, a dataset that contains a tree-like continuous manifold and disconnected clusters...



... and simple datasets that illustrate connectivity patterns of clusters.

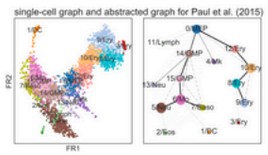


Differentiation manifolds in hematopoiesis

Here, we consider two well-studied datasets on hematopoietic differentiation.

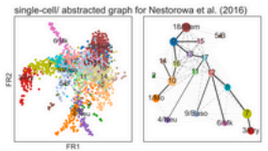
Data from Paul *et al.* (2015)

In *paul15*, we analyze data for myeloid progenitor development. This is the same data, which has served as benchmark for Monocle 2 (Qiu *et al.*, Nat. Meth., 2017) and DPT (Haghverdi *et al.*, Nat. Meth., 2016).



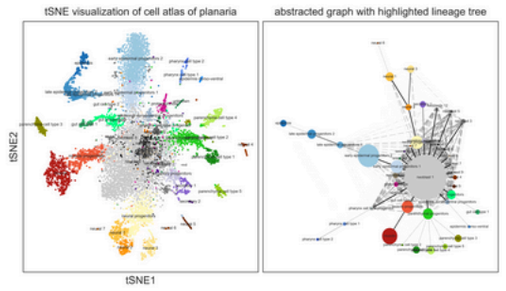
Data from Nestorowa, Hamey *et al.* (2016)

In *nestorowa16*, we analyze data for early hematopoietic differentation.



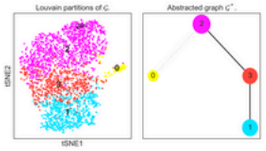
Lineage tree for whole cell atlas of an adult animal

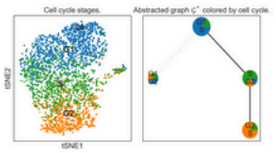
In *planaria*, we reconstruct the lineage tree of the whole cell atlas of planaria (Plass, Jordi *et al.*, submitted, 2017).



Deep Learning

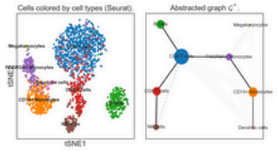
In *deep_learning*, we use deep learning to generate a feature space and, by that, a distance metric, which induces a nearest-neighbor graph. For the problem of reconstructing cell-cycle Eulenberg, Köhler, *et al.*, Nat. Commun. (2017), we find that graph abstraction correctly separates a small cluster of dead cells from the cell evolution through G1, S and G2 phase.

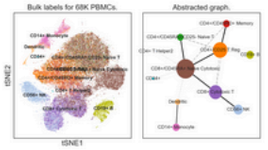




PBMC cells

For all of the following scRNA-seq datasets (3K and 68K PBMC cells, all 10X Genomics), graph abstraction reconstructs correct lineage motifs. As the data is disconnected in large parts, a global lineage tree cannot be inferred.



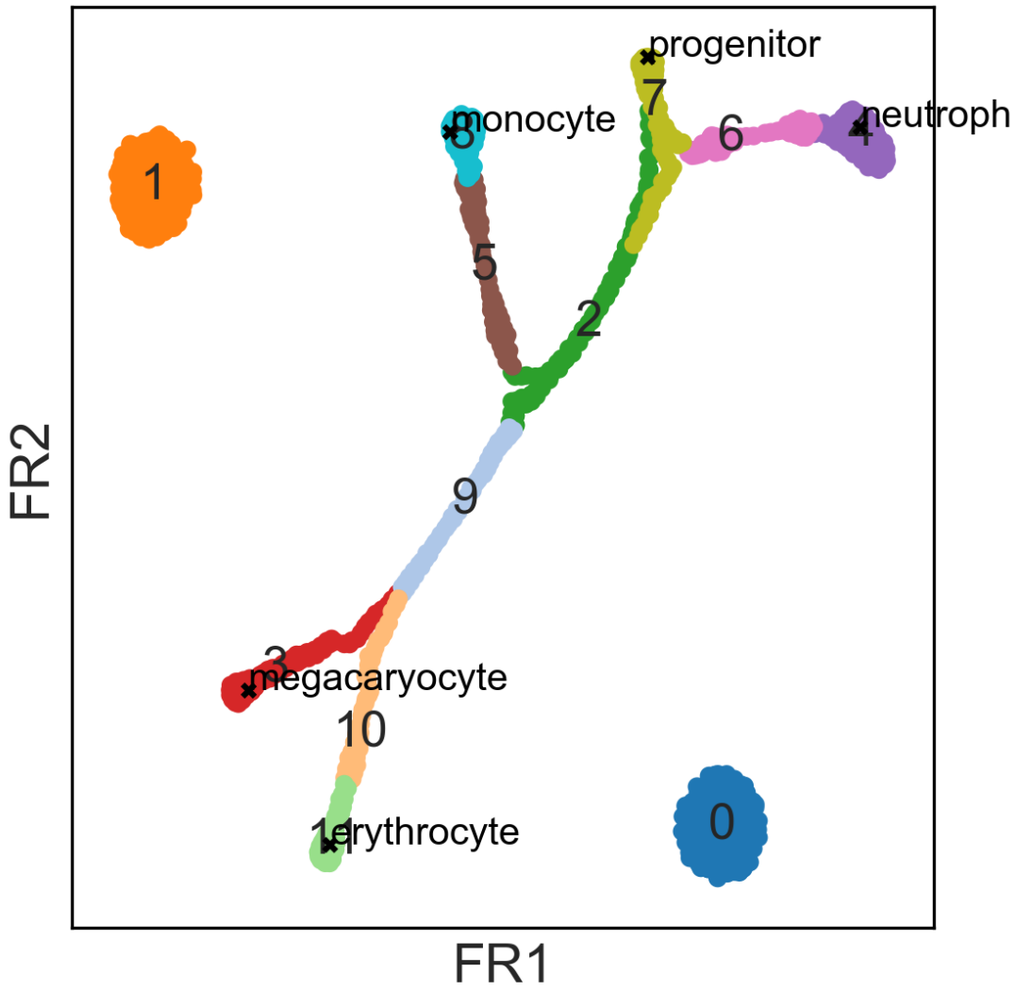


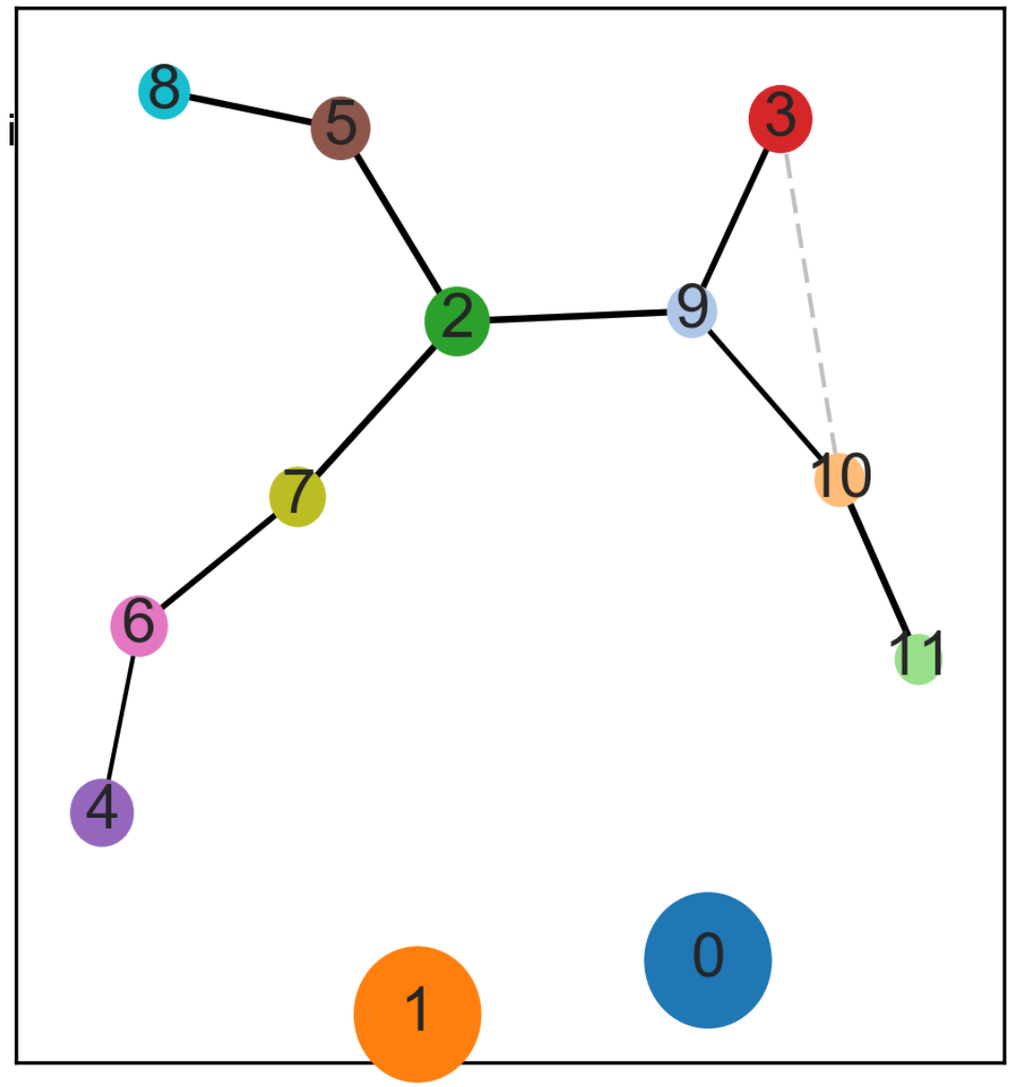
Thanks to

Machine Learning group at Helmholtz Munich, in particular, Philipp Angerer and Fabian Theis.

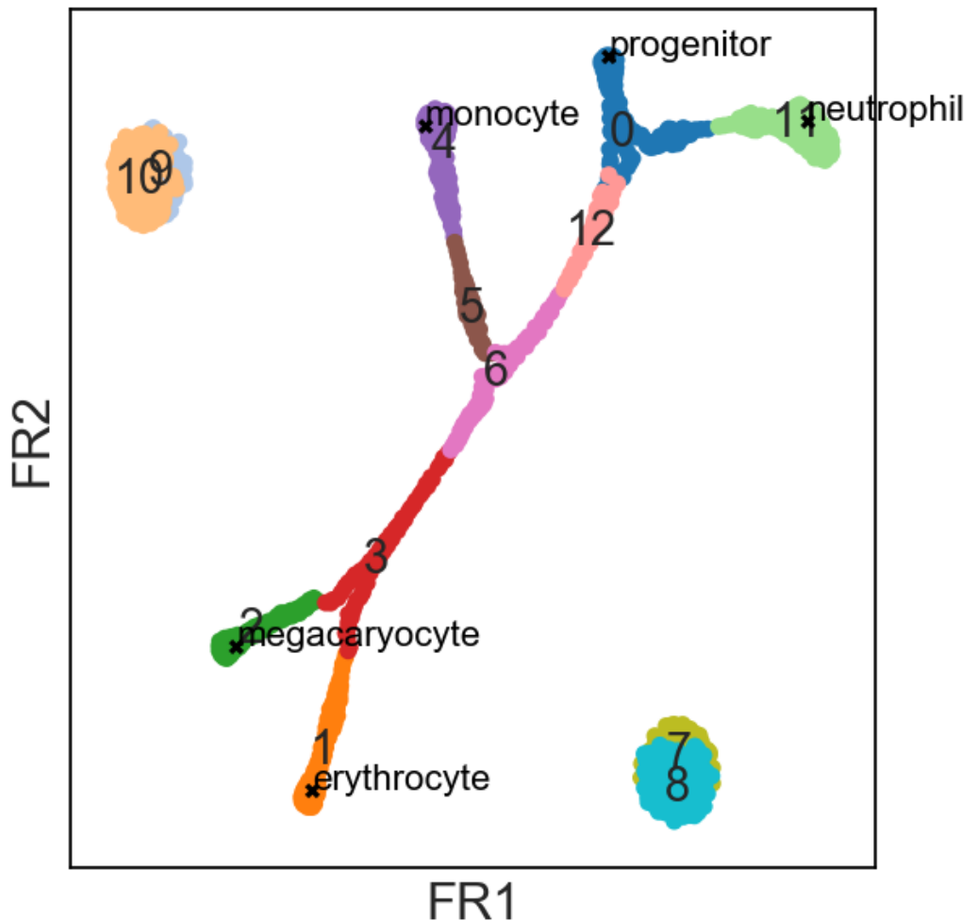
Thank you for your attention!

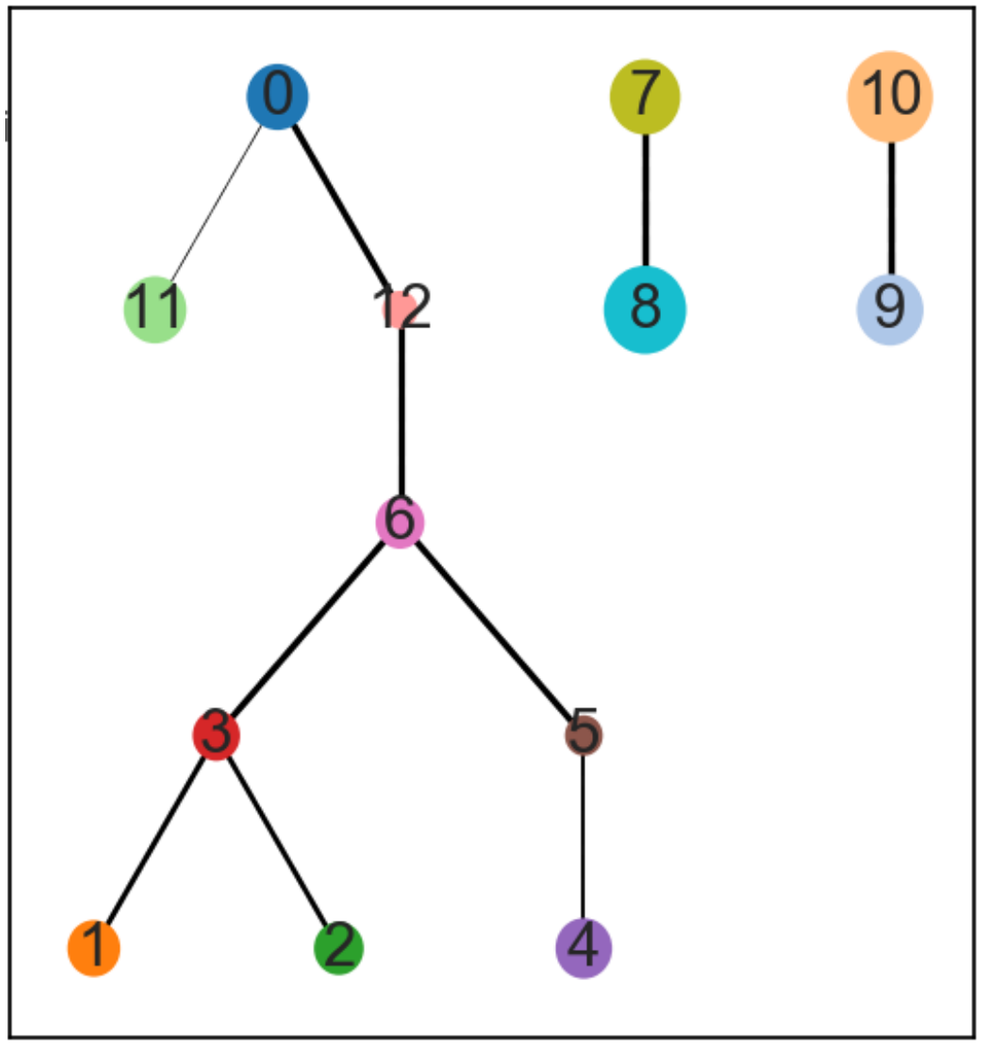
Preservation of topology under zooming



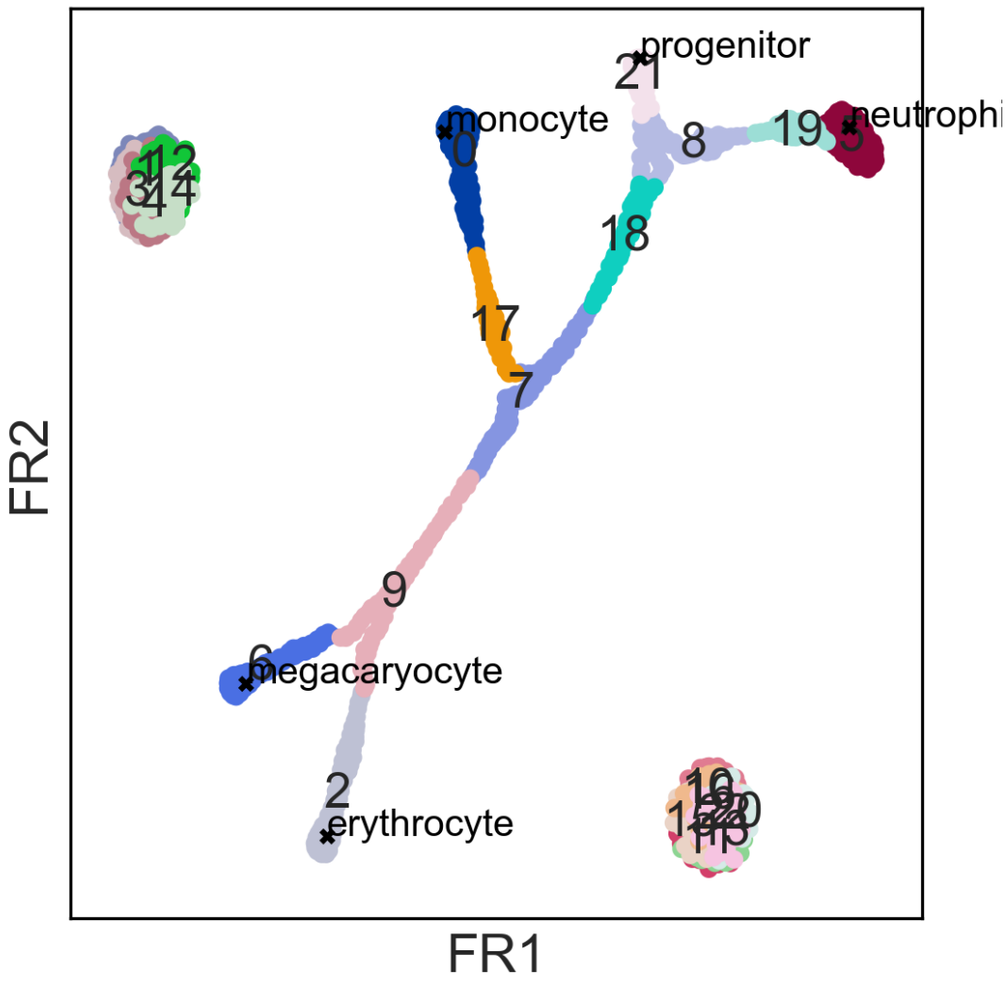


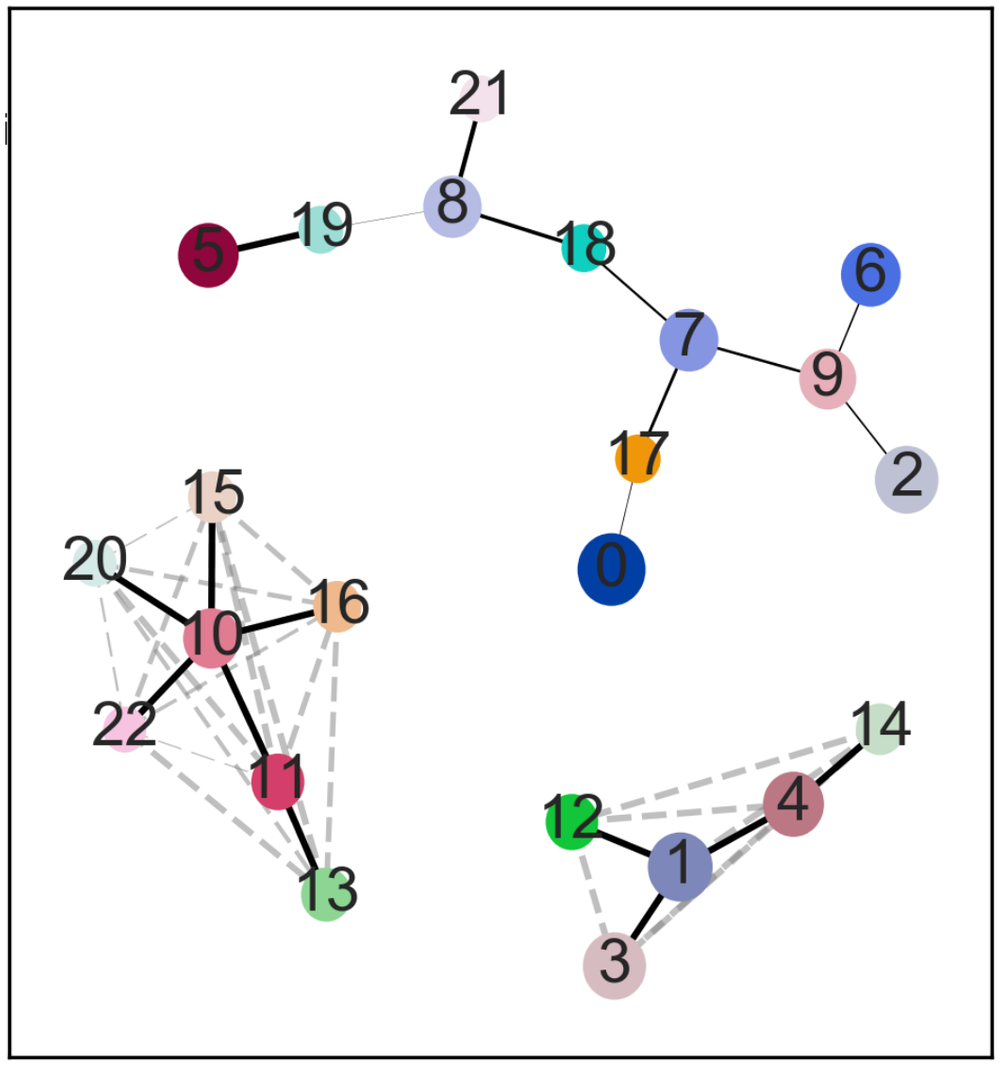
Preservation of topology under zooming





Preservation of topology under zooming





Scanpy code snippets

In [3]:
```
filename_data = './data/pbmc3k_filtered_gene_bc_matrices/hg19/matrix.mtx'
filename_genes = './data/pbmc3k_filtered_gene_bc_matrices/hg19/genes.tsv'
filename_barcodes = './data/pbmc3k_filtered_gene_bc_matrices/hg19/barcodes.tsv'
adata = sc.read(filename_data).transpose()
adata.var_names = np.loadtxt(filename_genes, dtype='S')[:, 1]
adata.smp_names = np.loadtxt(filename_barcodes, dtype='S')
```

```
reading file ./write/data/pbmc3k_filtered_gene_bc_matrices/hg19/matrix.h5
```

Basic filtering.

In [4]:
```
adata.smp['n_counts'] = np.sum(adata.X, axis=1).A1
sc.pp.filter_cells(adata, min_genes=200)
sc.pp.filter_genes(adata, min_cells=3)
```

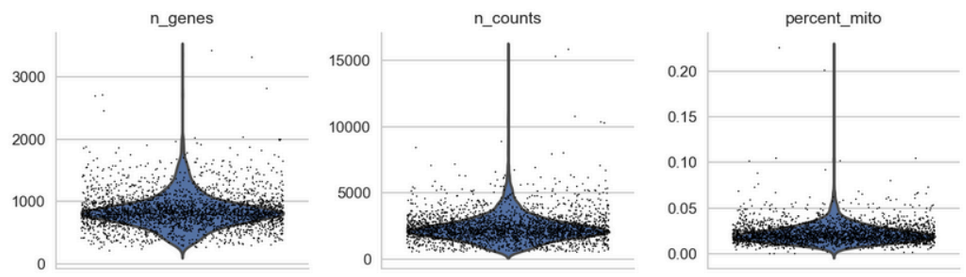
```
... filtered out 0 outlier cells
... filtered out 19024 genes that are detected in less than 3 cells
```

Plot some information about mitochondrial genes, important for quality control

In [5]:
```
mito_genes = np.array([name for name in adata.var_names
                       if bool(re.search("^MT-", name))])
# for each cell compute fraction of counts in mito genes vs. all genes
adata.smp['percent_mito'] = np.sum(adata[:, mito_genes].X, axis=1).A1 / np.sum(adata.X, axis=
# add the total counts per cell as sample annotation to adata
adata.smp['n_counts'] = np.sum(adata.X, axis=1).A1
```

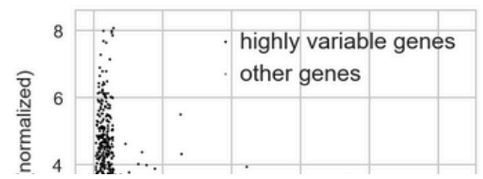
A violin plot of the computed quality measures.

In [6]:
```
sc.pl.violin(adata, ['n_genes', 'n_counts', 'percent_mito'], jitter=0.4, show=True)
```



In [9]:
```
sc.pp.normalize_per_cell(adata, scale_factor=1e4)
result = sc.pp.filter_genes_dispersion(adata.X, log=True,
sc.pl.filter_genes_dispersion(result)
```

```
... filter highly varying genes by dispersion and mean
    using `min_disp`, `max_disp`, `min_mean` and `max_mea
--> set `n_top_genes` to simply select top-scoring genes
```



In [11]:
```
adata_corrected = sc.pp.regress_out(adata,
                                    smp_keys=['n_counts', 'percent_mi
                                    copy=True)
```

```
0:00:00.000 - regress out ['n_counts', 'percent_mito']
... sparse input is densified and may lead to huge memory consumption

0:00:09.418 - finished
```

Compute PCA and make a scatter plot.

In [12]:
```
sc.pp.scale(adata_corrected, max_value=10)
```

```
clipping at max_value 10
```

In [13]:
```
sc.tl.pca(adata_corrected)
adata_corrected.smp['X_pca'] *= -1  # multiply by 1 for correspondenc
sc.pl.pca_scatter(adata_corrected, color='CST3', right_margin=0.2)
```

```
0:00:00.000 - compute PCA with n_comps = 10
0:00:00.668 - finished, added
    the data representation "X_pca" (adata.smp)
    the loadings "PC1", "PC2", ... (adata.var)
    and "pca_variance_ratio" (adata.add)
```