Scanpy

Wolf, Angerer & Theis, bioRxiv (2017)

Analysis of large-scale scRNA-seq data

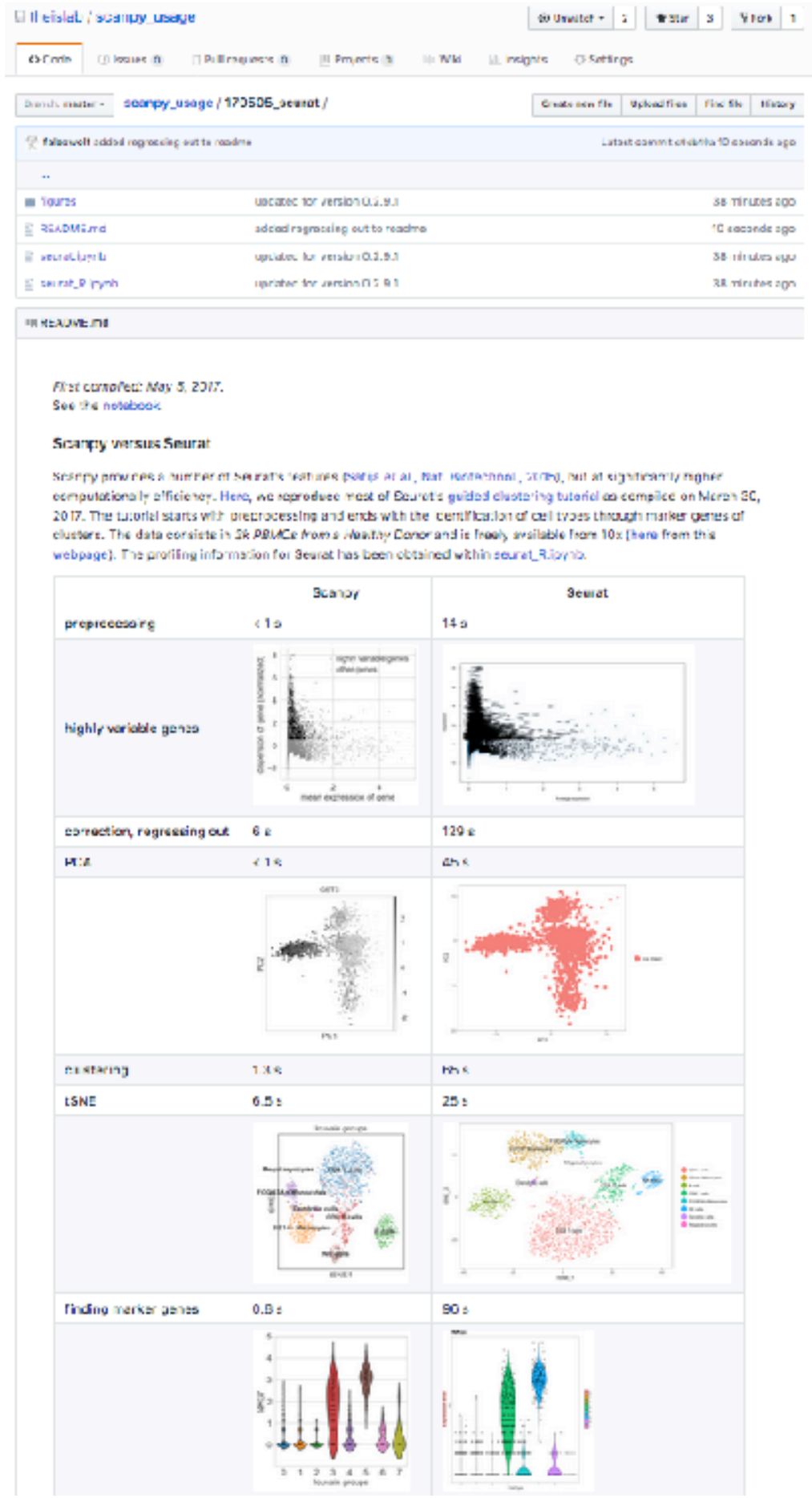
F. Alexander Wolf, Institute of Computational Biology, Helmholtz Munich
November 7, 2017 - Video talk for Regev Lab - Broad Institute

Scanpy vs. Seurat

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

Scanpy is benchmarked with Seurat.

- 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

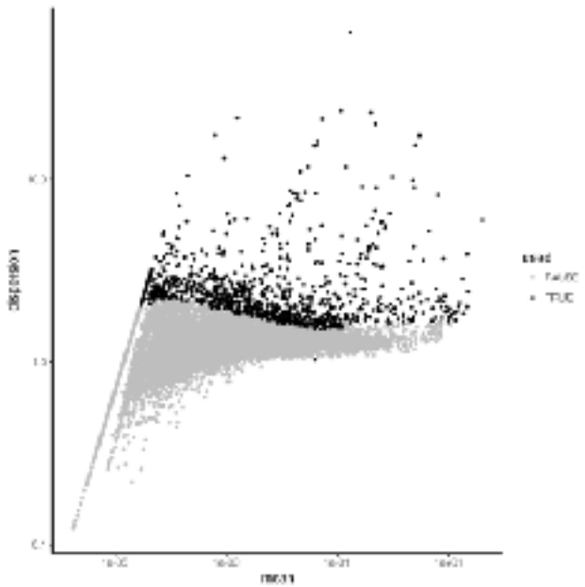


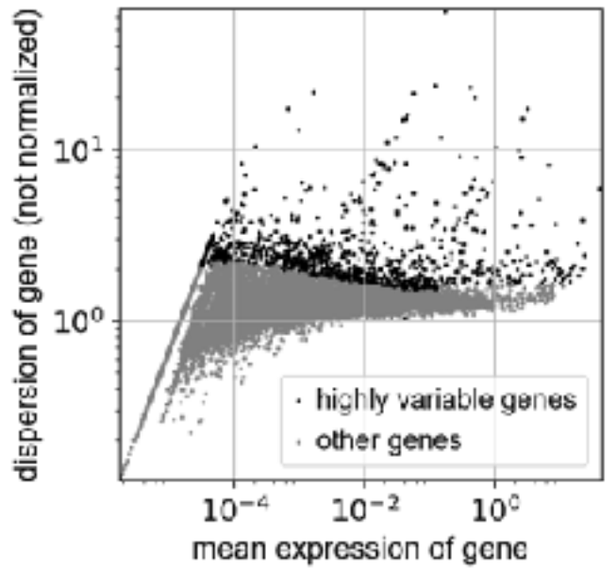
Scanpy vs. Cell Ranger for 68k cells

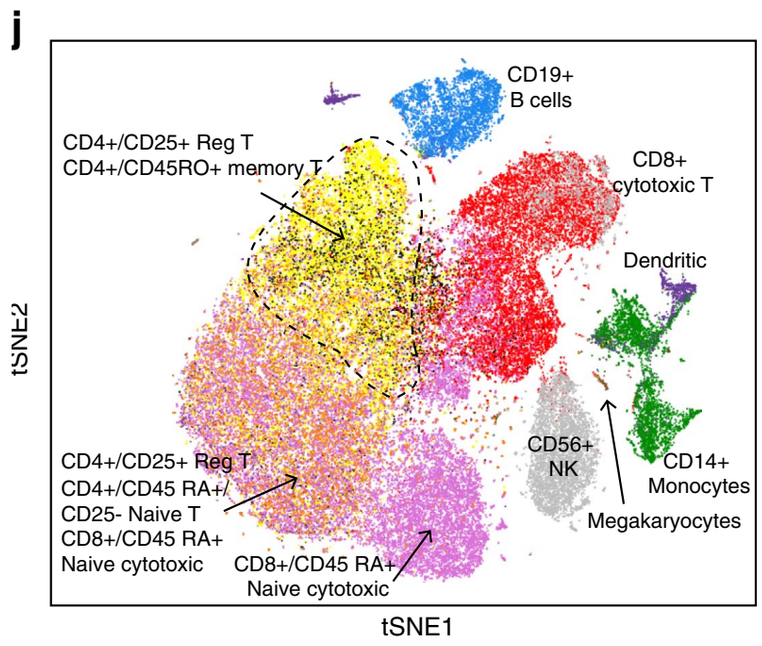
Zheng *et al.*, Nat. Commun. (2017)

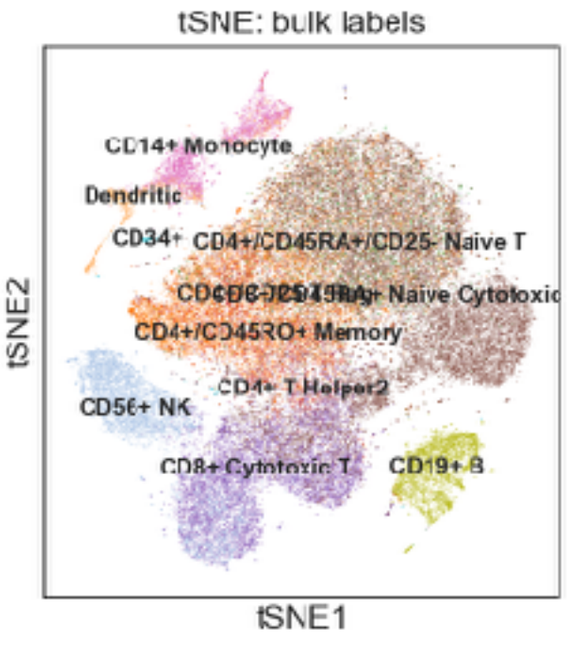
Scanpy is benchmarked with Cell Ranger R kit.

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



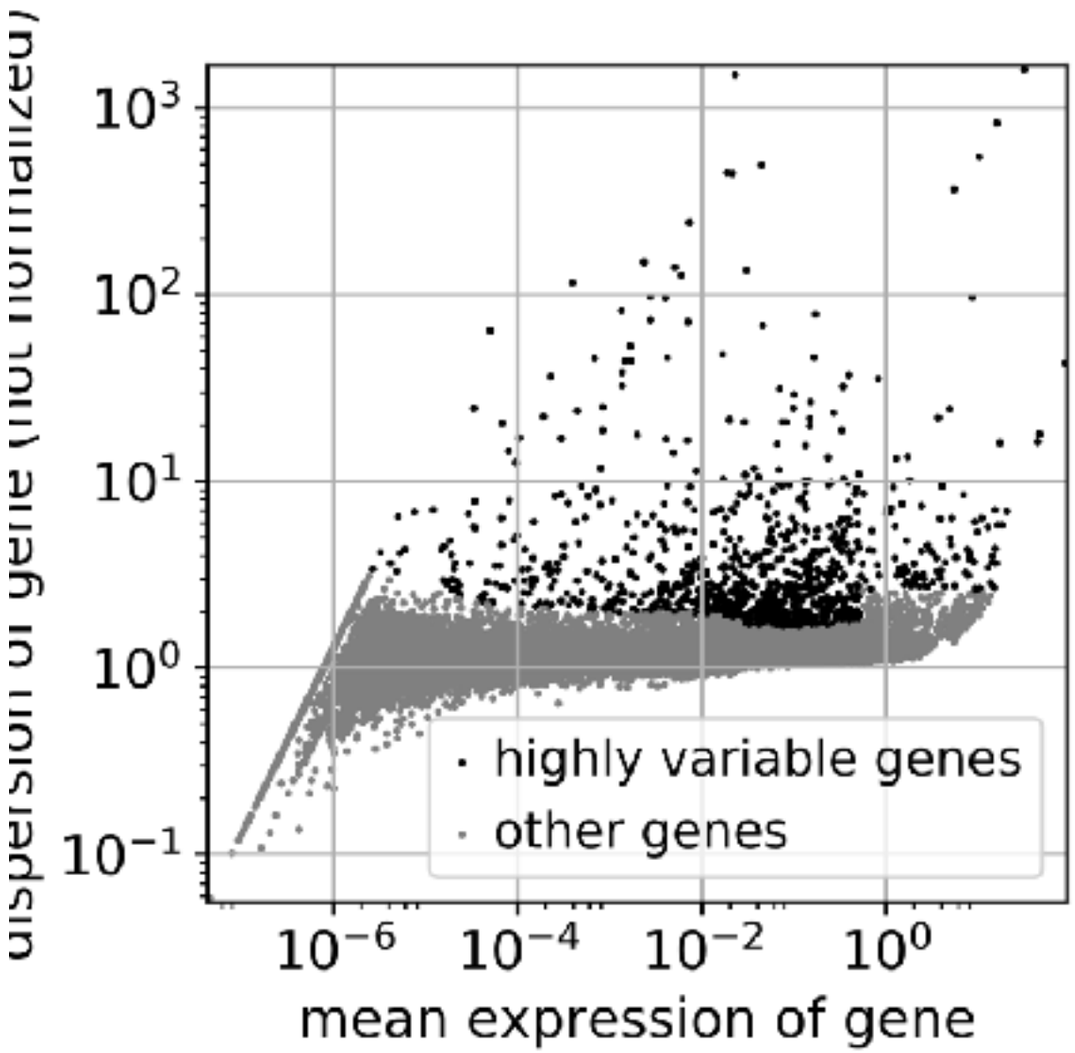


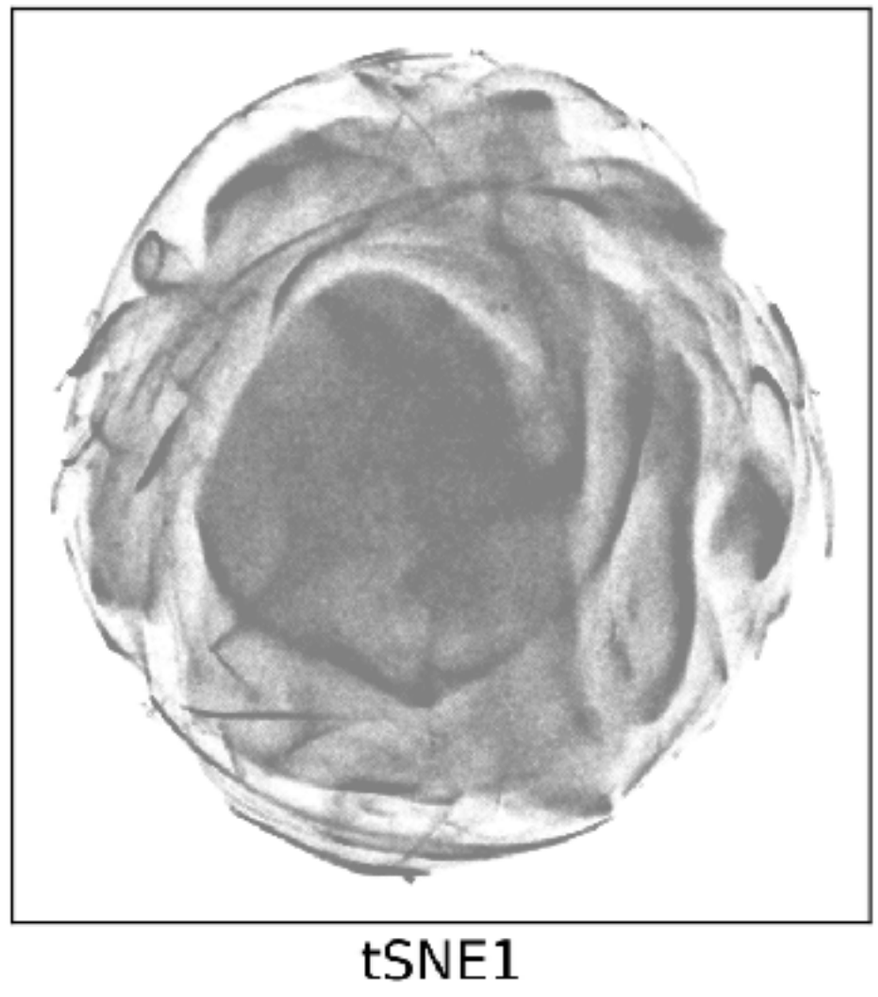




Scanpy scales to >1 million cells

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



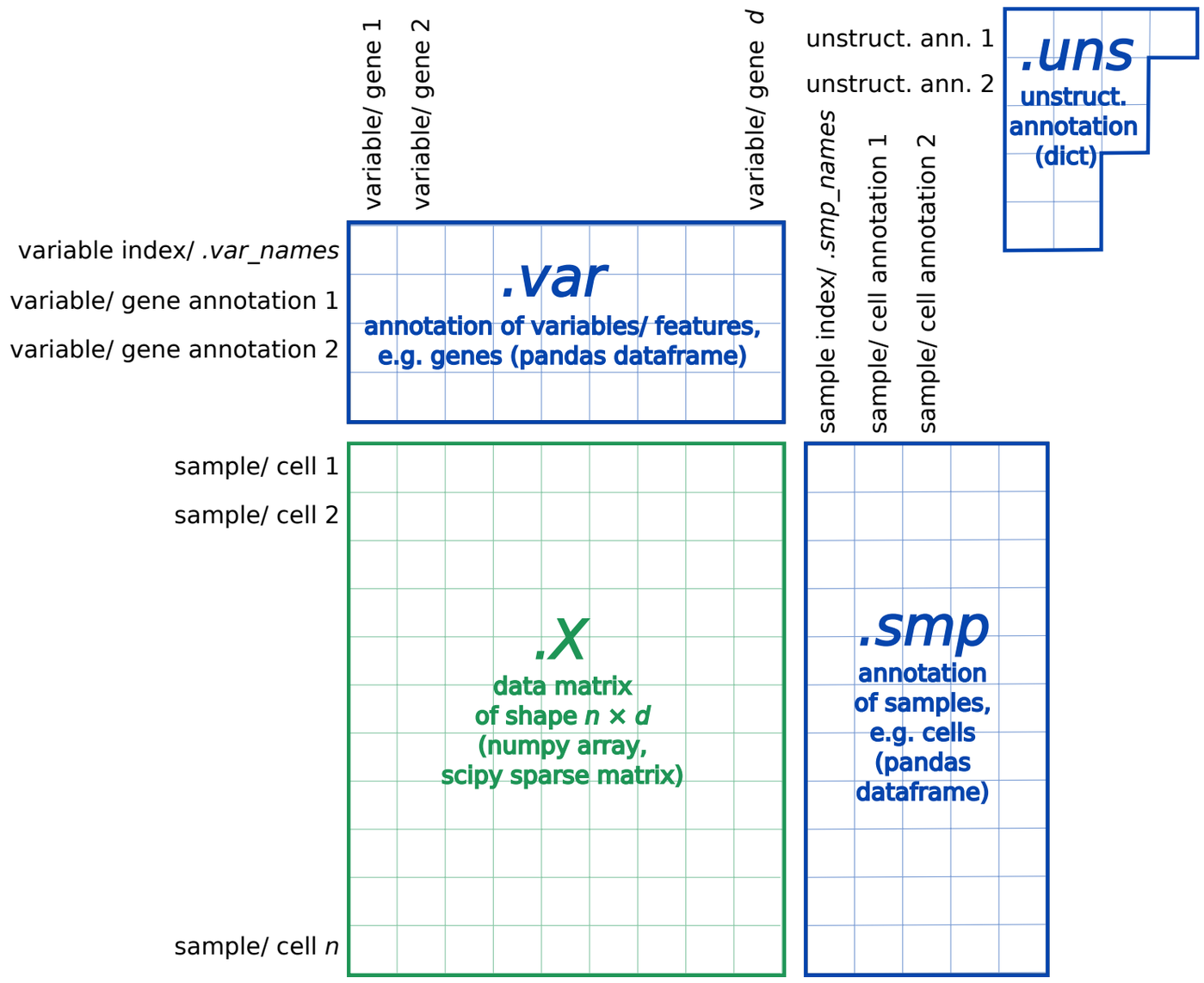


AnnData github.com/theislab/anndata, pypi/anndata

Simple class for a data matrix with most general annotations.

Nothing like this in Python.

- *.loom* (Python, merely a file format)
- *VariantDataset* (Python, Java, hail)
- *ExpressionSet* (R)
- “Seurat Object” (R, Seurat)
- *CellDataSet* (R, Monocle)
- *SingleCellExperiment* (R, Scran)



HDF5-backed on disk: cross-platform, cross-language.

Characteristics of single-cell data

Goal

Learn abstractions of biology
(e.g. cellular identities),
associations and mechanisms.

Data

- high-dimensional
- unstructured
- sparse
- noisy
- non-linear

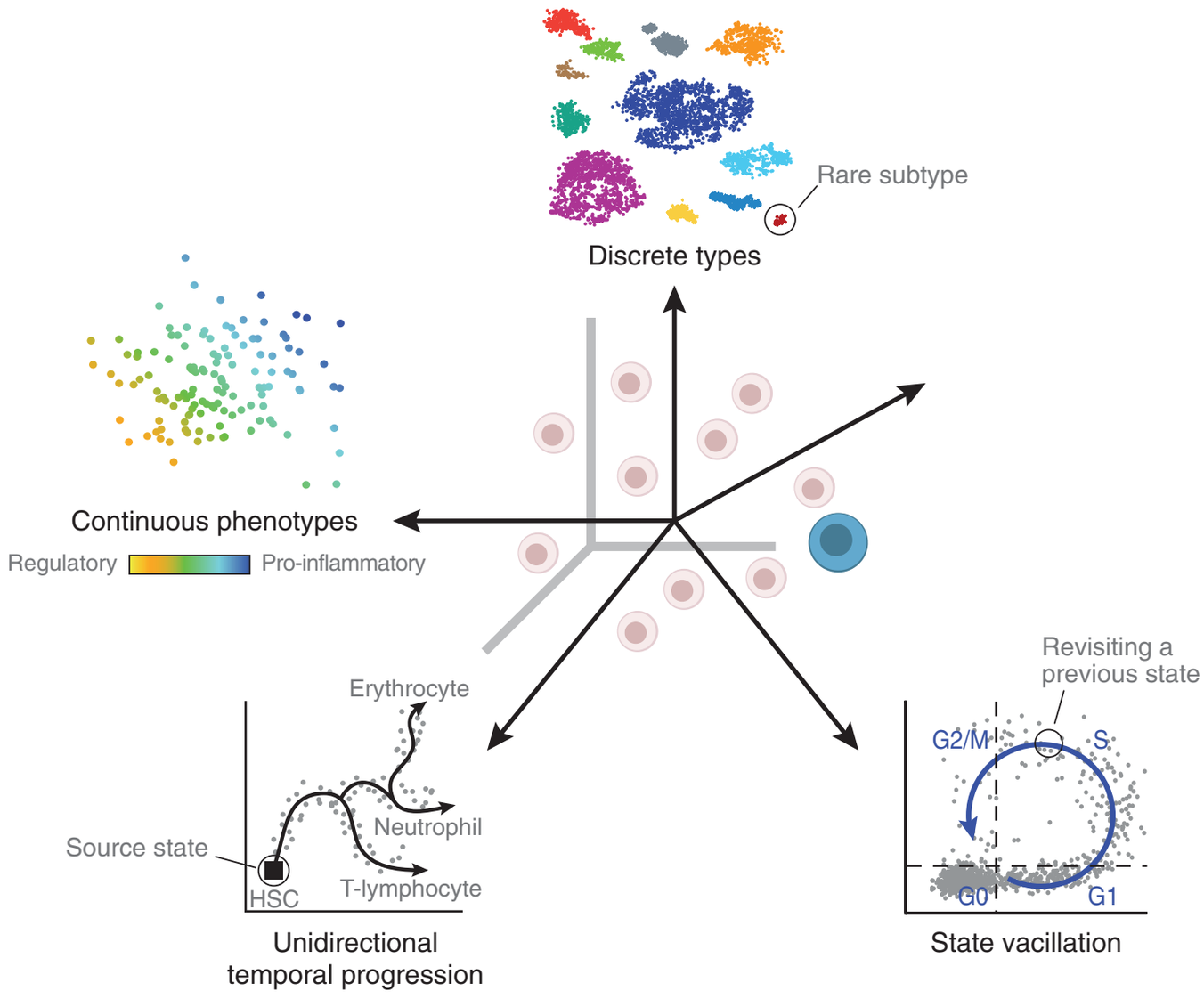


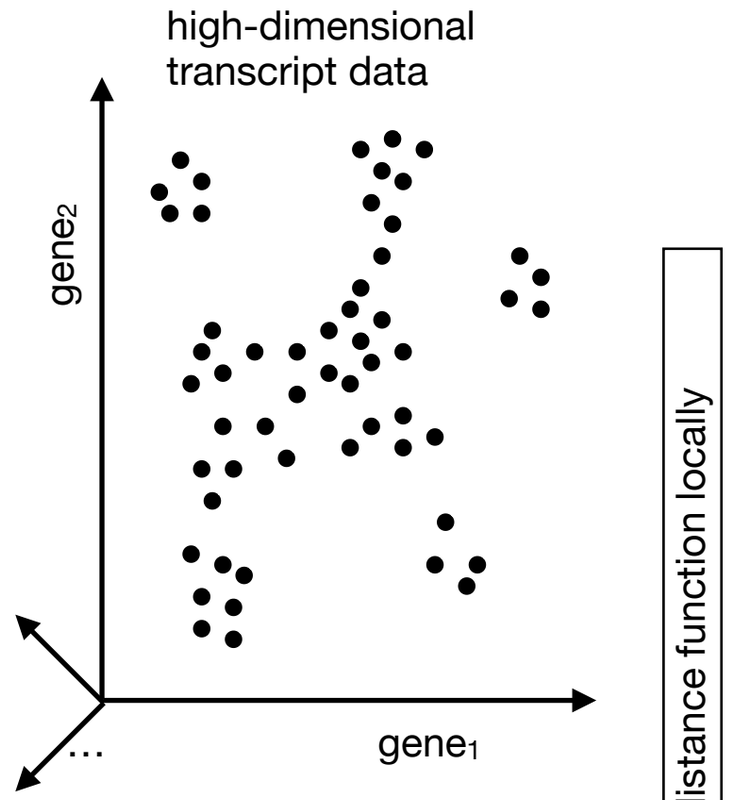
figure from Review Wagner et al., Nat Biotechn (2016)

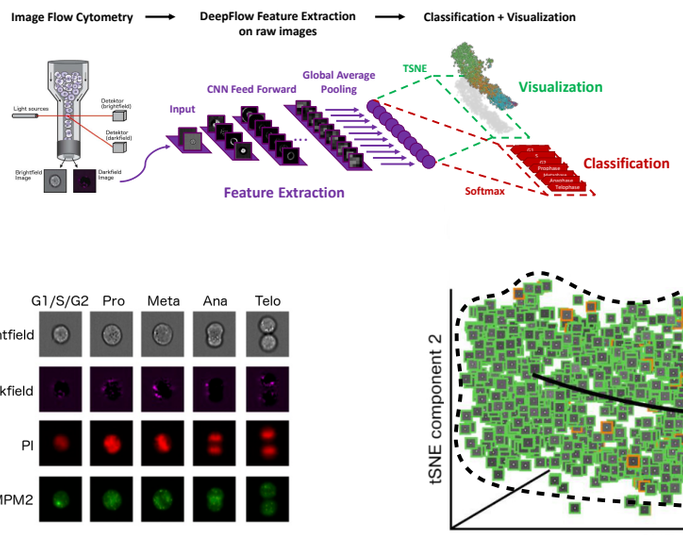
Represent single-cell data

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

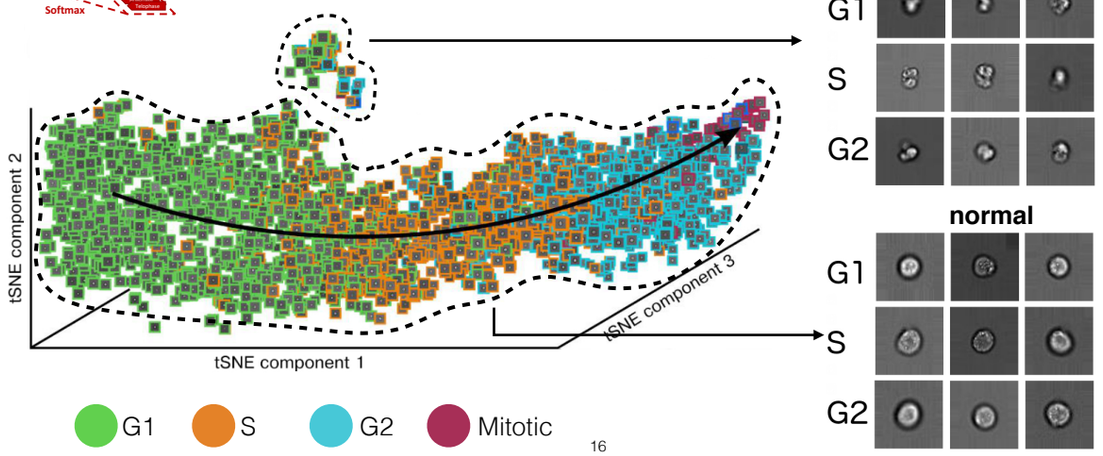
- Typically, obtain d from preprocessing and something like euclidean distance.

- Alternatively, learn *d:*

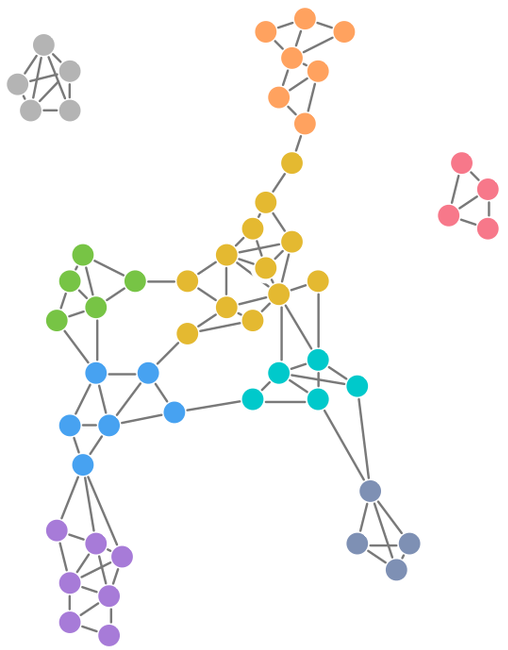




Eulenberg, Köhler et al., Nat. Commun. (2017)



single-cell graph



DataGraph

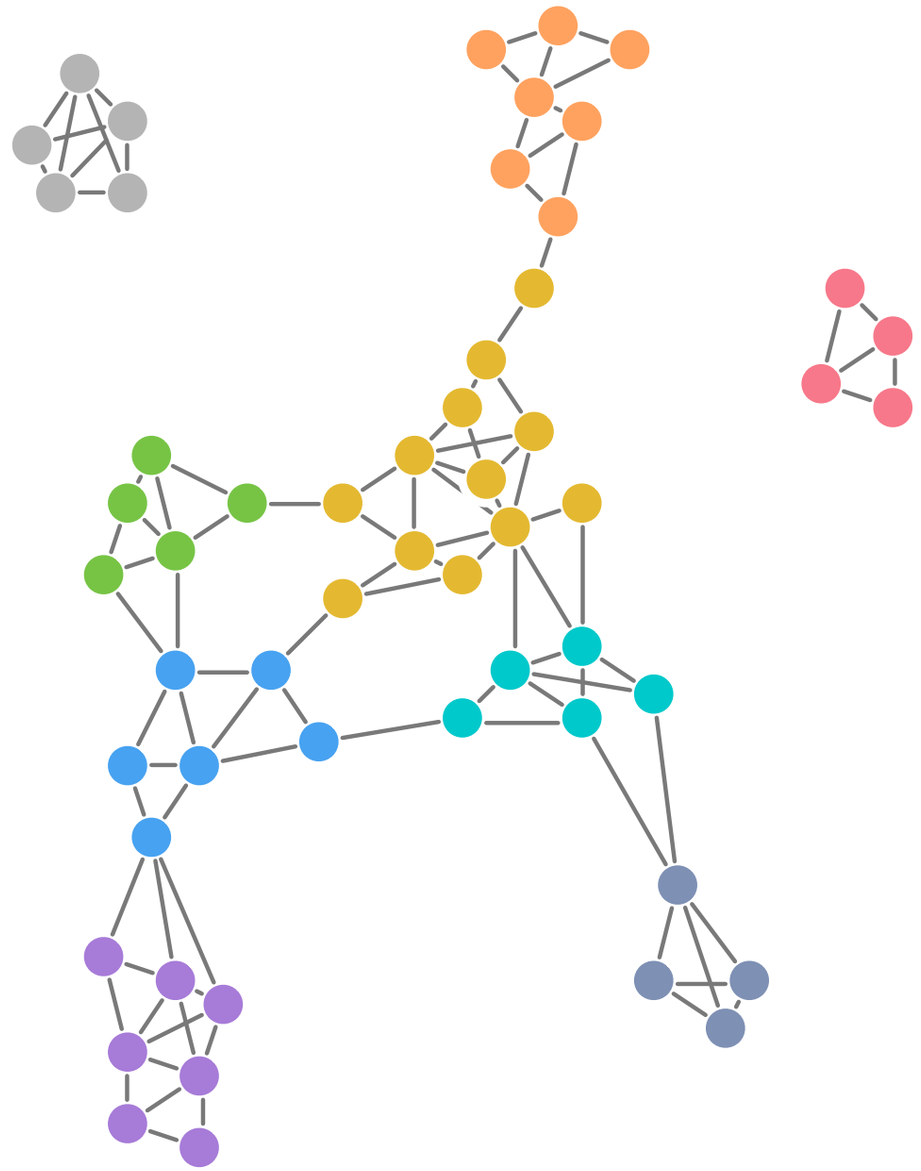
Class for representing data as a graph of neighborhood relations between data points.

Simplest case: knn graph.

- much faster than *sklearn.neighbors*
- much faster than R-wrapped C++

One idea: use matrix-multiplication for submatrices of data matrix in parallel.

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



Scanpy tools operate on *DataGraph*

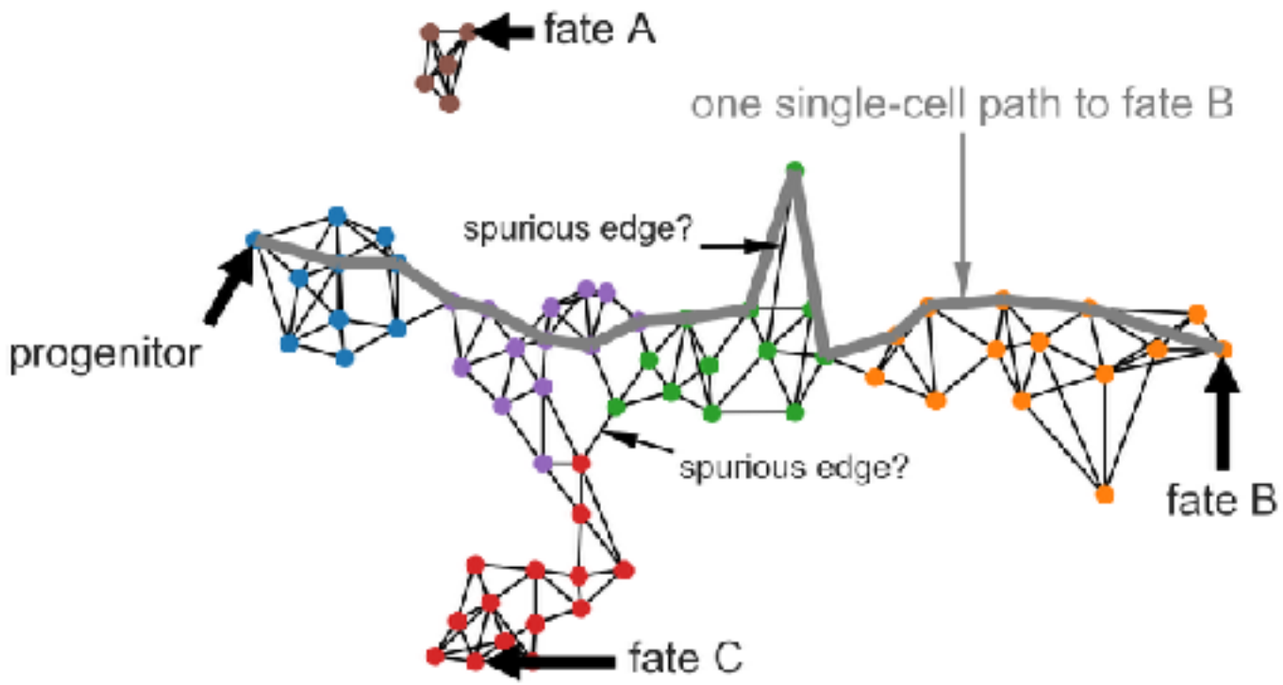
A single framework for common analysis tasks.

- clustering

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

- pseudotime and trajectory inference

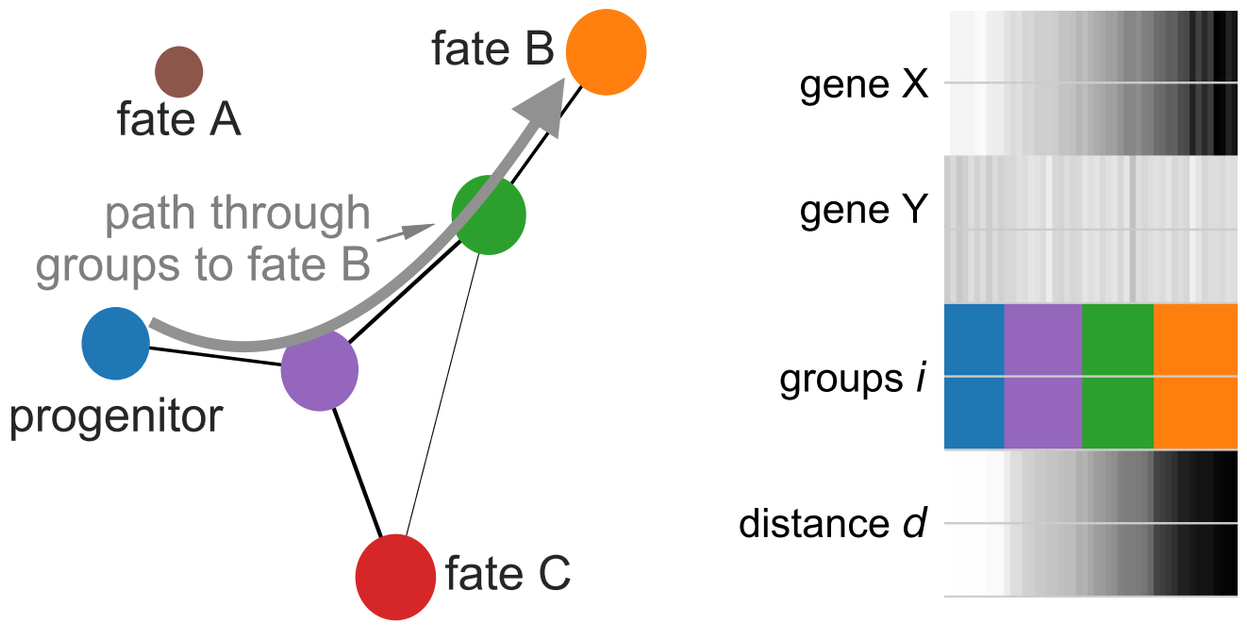
Trapnell et al., Bendall et al. (2014), Haghverdi et al. (2016) …



Reconciling both:

- graph abstraction

Wolf et al., bioRxiv (2017)

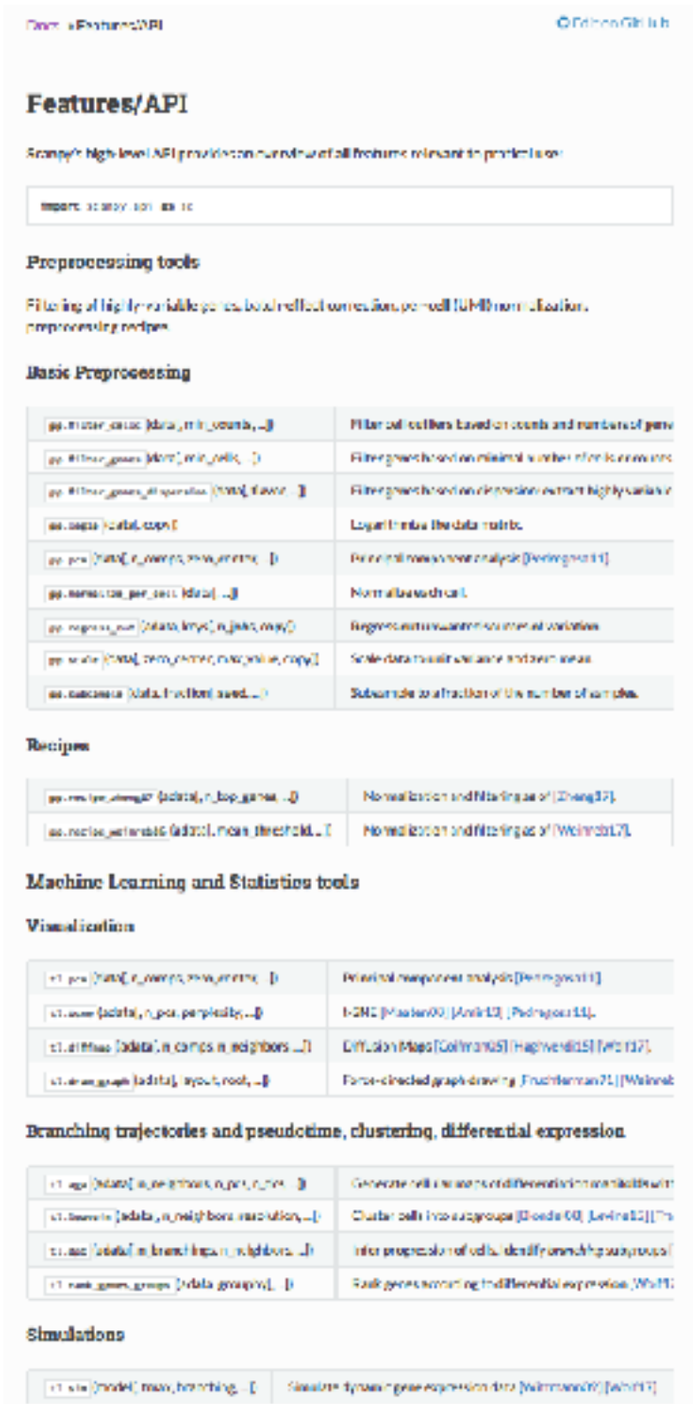


Scanpy's API

Modular and intuitive API.

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

Command-line interface…



Features/API

Preprocessing tools

Basic Preprocessing

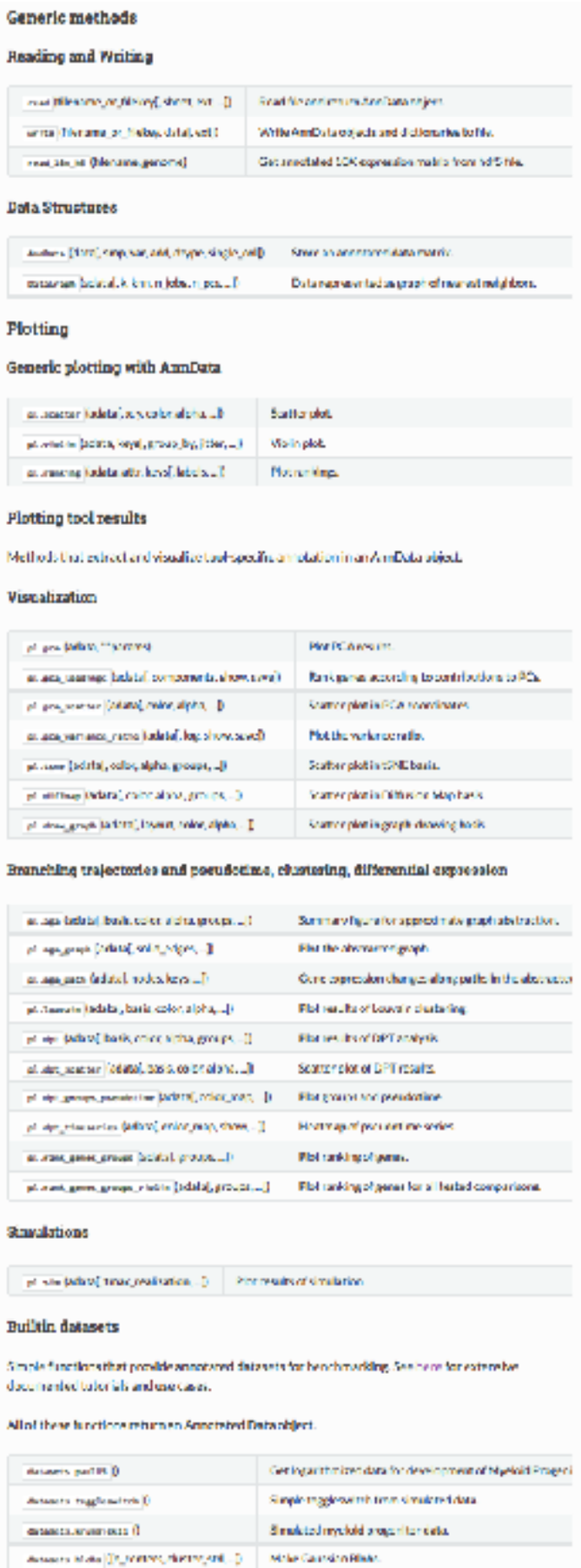
Recipes

Machine Learning and Statistics tools

Visualization

Branching trajectories and pseudotime, clustering, differential expression

Simulations



Generic methods

Reading and Writing

Data Structures

Plotting

Generic plotting with AnnData

Plotting tool results

Visualization

Branching trajectories and pseudotime, clustering, differential expression

Simulations

Builtin datasets

Scanpy's use cases

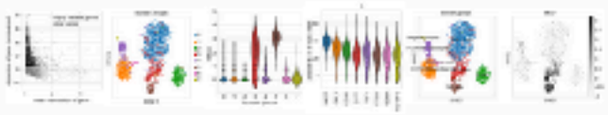
Docs » Examples

Edit on GitHub

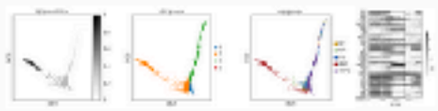
Examples

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

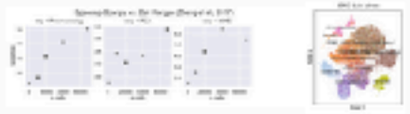
Example 1: Seurat's [Satija15] guided clustering tutorial



Example 2: The Cellrank on Petropoulos et al. [2016] analyses of [Haghverdi16] for data of [Paul15] and [Moignard15]. Note that DPT has recently been very favorably discussed by the authors of Monocle.

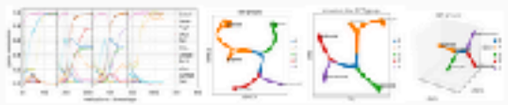


Example 3: Analyzing 1.3M E18 cells from [Zheng17], we find that Scanpy is about a factor 130 faster and more memory efficient than the Cell Ranger R kit for secondary analysis.

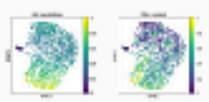


Example 4: Visualizing 1.3 mio brain cells.

Example 5: Simulating single cell data (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

Generate cellular maps of differentiation manifolds with complex topologies.

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
pbmc	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

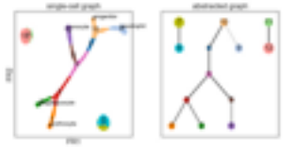
This repository allows to reproduce analyses and figures of the preprint.

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

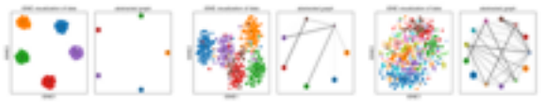
- `scanpy.api.tools.aga`
- `scanpy.api.plotting.aga_graph`
- `scanpy.api.plotting.aga_path`

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.

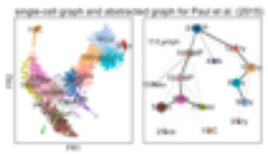


Differentiation manifolds in hematopoiesis

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

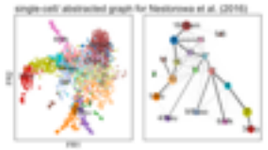
Data from Paul et al. (2015)

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).



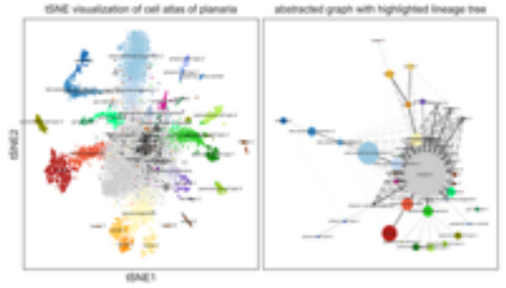
Data from Nestorowa, Hamey et al. (2016)

In nestorowa16, we analyze data for early hematopoietic differentiation.



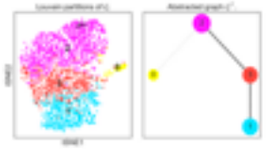
Lineage tree for whole cell atlas of an adult animal

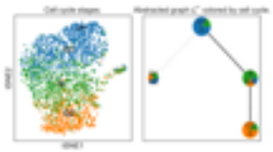
In planaria, we reconstruct the lineage tree of the whole cell atlas of planaria (Plass, Jordi et al., submitted, 2017).



Deep Learning

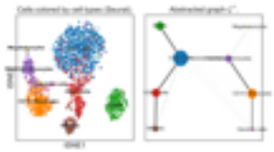
In deep_learning, we use deep learning to generate a feature space and, by that, a distance metric, which induces a nearest-neighbor graph. For the problem of reconstructing cell-cycle Eulenberg, Köhler, et al., Nat. Commun. (2017), we find that graph abstraction correctly separates a small cluster of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

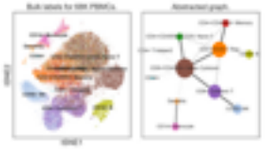




PBMC cells

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.





Outlook

Very-short term

- common file format for backing AnnData
- AnnData based on pandas dataframes instead of structured arrays

Mid-term

- aggregation of datasets
- better correction for confounders
- include any standard, canonical analysis method…
- module-wise installation (reduce dependencies?)

Long-term

- mini-batch learning

Thanks to

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

Thank you for your attention!

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)
```

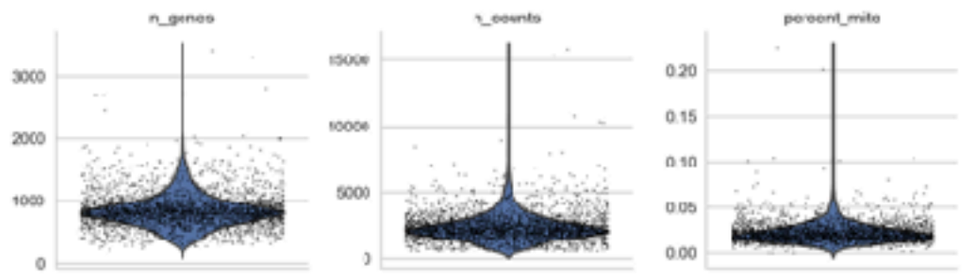
```
... filtered out 0 outlier cells
... filtered out 19024 genes that are detected in less than 3 cells
```

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
```

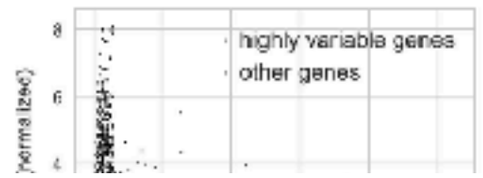
A violin plot of the computed quality measures.

```
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)
```