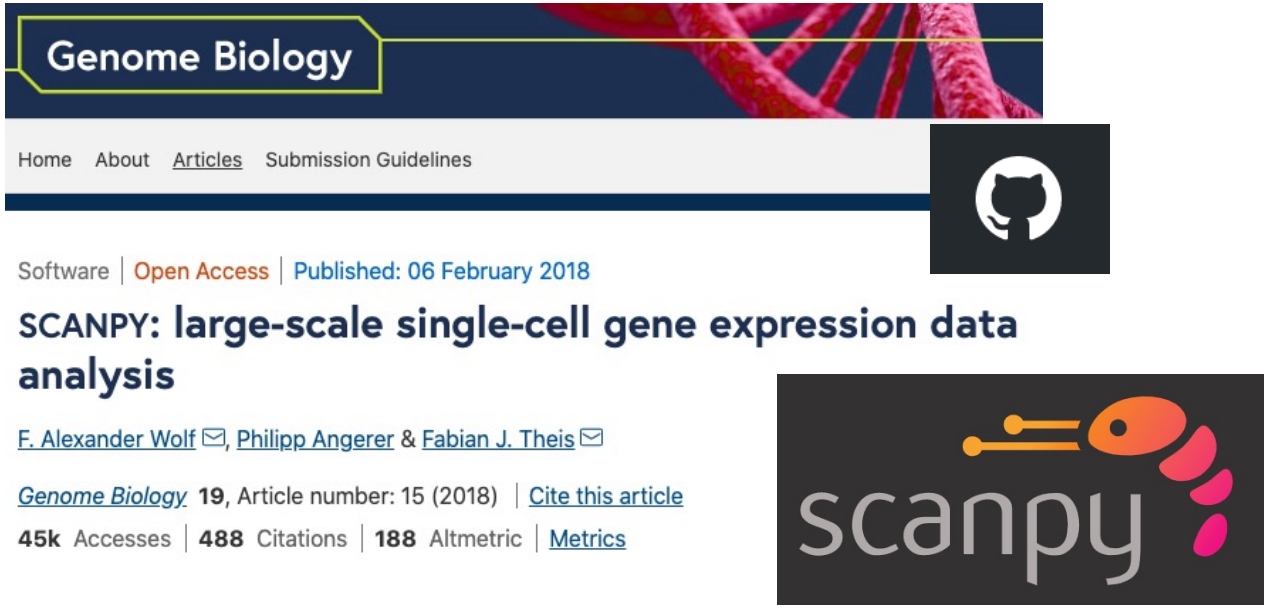
Introduction to scanpy

Single Cell Transcriptomics in Python

Alex Lederer

What is scanpy?

- Highly popular set of Python tools for analysis of single cell datasets (primarily single cell RNA-sequencing data)
- Allows analysis from raw counts through the following steps:
  - Preprocessing and quality control
  - Feature selection
  - Dimensionality reduction
  - Clustering and marker annotation
  - Visualization
- Other related tools for RNA velocity (scvelo), data batch integration, and spatial transcriptomics
- Let's walk through a tutorial!



AnnData objects

- Fundamental unit of scanpy
- Essentially a fancy table with embedding metadata, example:

```
adata
```

```
AnnData object with n_obs × n_vars = 2432 × 2000
    obs: 'n_counts', 'n_genes', 'n_genes_by_counts', 'total_counts', 'total_counts_mt', 'pct_counts_mt', 'leiden', 'l
ouvain'
    var: 'gene_ids', 'n_cells', 'mt', 'n_cells_by_counts', 'mean_counts', 'pct_dropout_by_counts', 'total_counts', 'h
ighly_variable', 'means', 'dispersions', 'dispersions_norm', 'mean', 'std'
    uns: 'log1p', 'hvg', 'pca', 'neighbors', 'umap', 'leiden', 'louvain', 'leiden_colors', 'louvain_colors'
    obsm: 'X_pca', 'X_umap', 'X_tsne'
    varm: 'PCs'
    obsp: 'distances', 'connectivities'
```

rows, columns = 2432 cells, 2000 genes
`adata.obs` = metadata table for the cells (pandas data frame)
`adata.var` = metadata table for the genes (pandas data frame)

- BUT when you first load a file it is pretty empty:

```
adata
```

```
AnnData object with n_obs × n_vars = 2700 × 32738
    var: 'gene_ids'
```

Reading and writing AnnData objects

Essential imports

```
import numpy as np
import pandas as pd
import scanpy as sc
import matplotlib.pyplot as plt
```

Check out the documentation pages for these packages!

Reading a 10X dataset folder

```
adata = sc.read_10x_mtx(
    'data/filtered_gene_bc_matrices/hg19/',  # the directory with the `.mtx` file
    var_names='gene_symbols',                # use gene symbols for the variable names (variables-axis index)
    cache=True)                              # write a cache file for faster subsequent reading
```

Other functions for loading data:

```
sc.read_10x_h5
sc.read_csv
sc.read_h5ad # this function will be used to load any analysis objects you save
sc.read_loom
```

To save your adata object at any step of analysis:

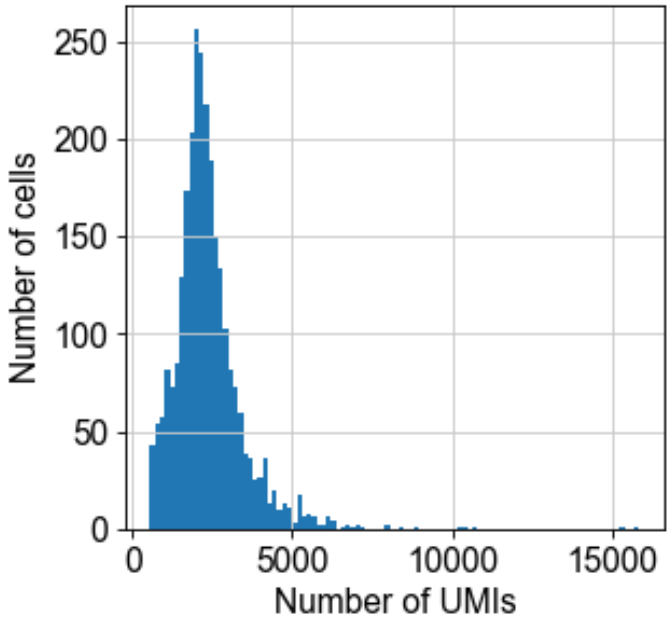
```
adata.write_h5ad("save_file_name.h5ad")
```

A saved h5ad can later be reloaded using the command **`sc.read_h5ad`**

Preprocessing and quality control

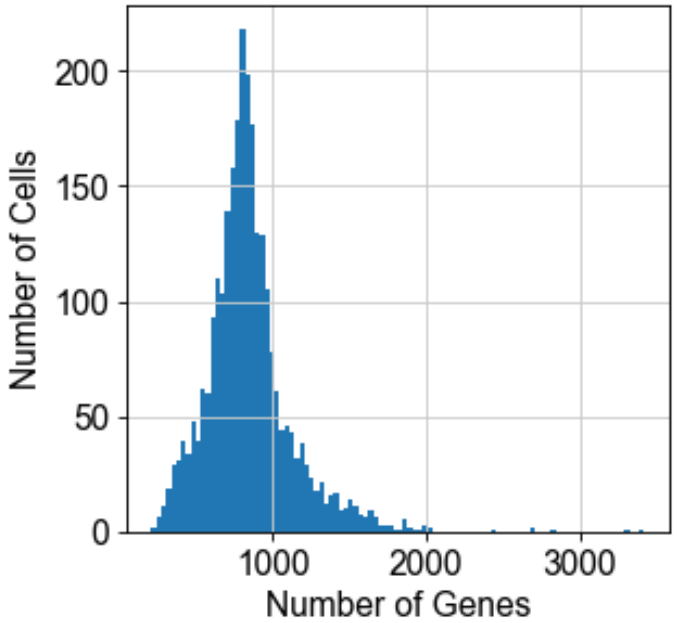
- How many UMIs are there per cell?
  - cells with low num UMIs = low quality
  - cells with high num UMIs = doublets

```
adata.obs['n_counts'] = adata.X.sum(axis=1)
n, bins, *x = plt.hist(adata.obs['n_counts'], bins=100)
plt.xlabel("Number of UMIs")
plt.ylabel("Number of cells")
plt.show()
```



- How many genes are detected per cell?

```
adata.obs['n_genes'] = np.sum(adata.X > 0, 1)
n, bins, *x = plt.hist(adata.obs['n_genes'], bins=100)
plt.xlabel("Number of Genes")
plt.ylabel("Number of Cells")
plt.show()
```



Preprocessing and quality control

Example of filtering criteria:

```
sc.pp.filter_cells(adata, min_counts=1000)
sc.pp.filter_cells(adata, max_counts=5000)
sc.pp.filter_cells(adata, min_genes=250)
sc.pp.filter_cells(adata, max_genes=1500)
```

```
filtered out 153 cells that have less than 1000 counts
filtered out 69 cells that have more than 5000 counts
filtered out 16 cells that have more than 1500 genes expressed
```

```
sc.pp.filter_genes(adata, min_cells=5)
```

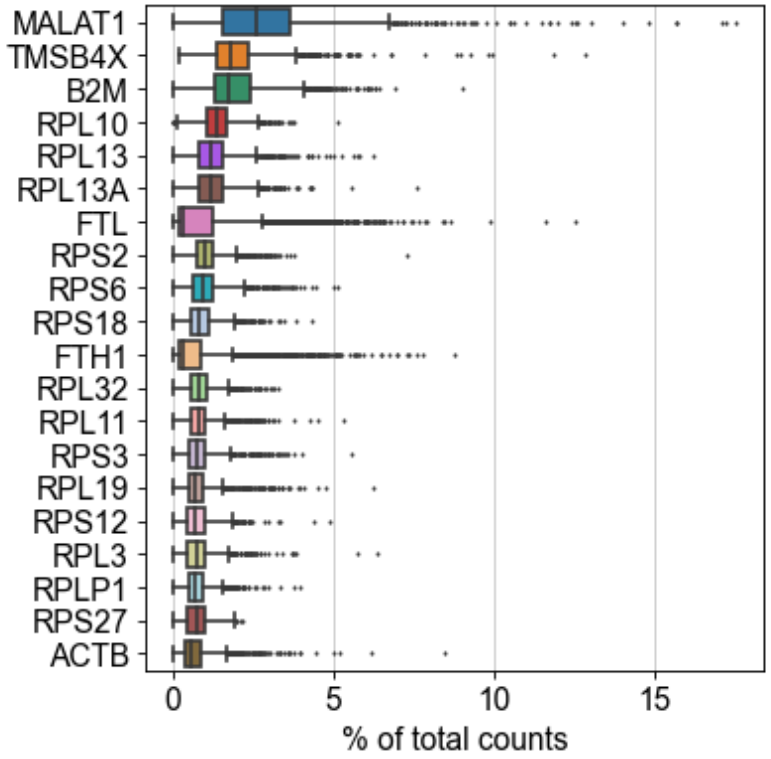
```
filtered out 20443 genes that are detected in less than 5 cells
```

These filtering criteria will depend on the overall sequencing quality and depth of the respective dataset

What are the most highly expressed genes?

```
sc.pl.highest_expr_genes(adata, n_top=20)
```

```
normalizing counts per cell
    finished (0:00:00)
```



MALAT1, ribosomal genes (RPL, RPS) are normally the most abundant

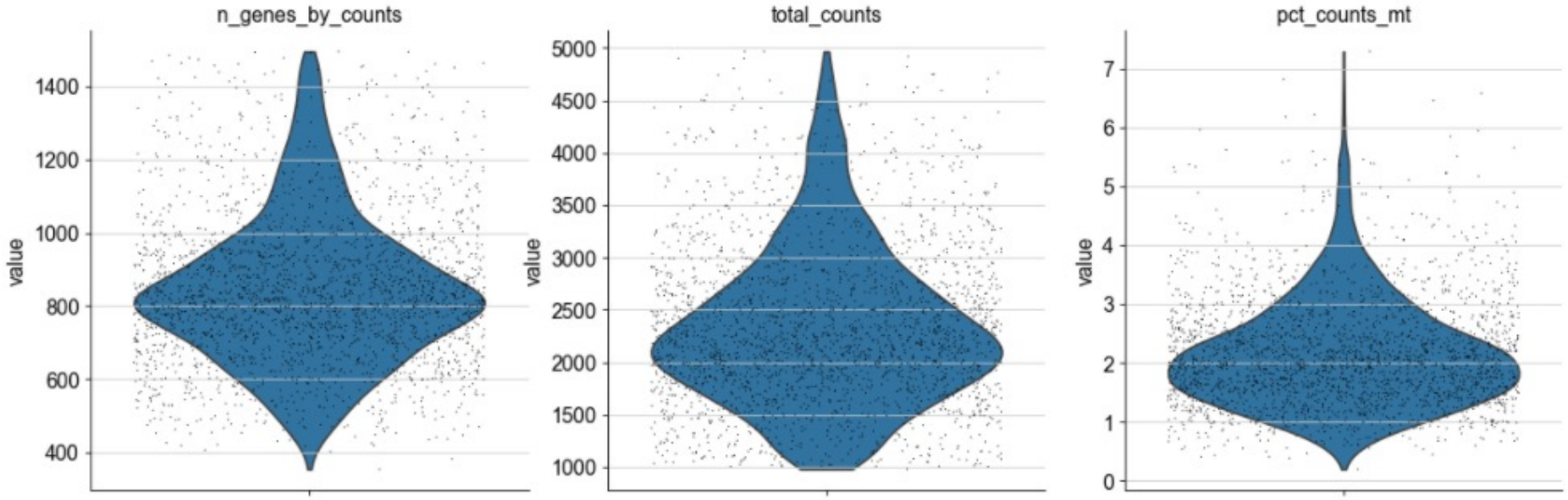
Preprocessing and quality control

Cells with a large percentage of reads from mitochondrial genes are usually of lower quality

```
adata.var['mt'] = adata.var_names.str.startswith('MT-')  # annotate the group of mitochondrial genes as 'mt'
sc.pp.calculate_qc_metrics(adata, qc_vars=['mt'], percent_top=None, log1p=False, inplace=True)
```

'MT' for human datasets!

```
sc.pl.violin(adata, ['n_genes_by_counts', 'total_counts', 'pct_counts_mt'],
             jitter=0.4, multi_panel=True)
```



Can also make scatter plots:

```
sc.pl.scatter(adata, x='total_counts', y='pct_counts_mt')
sc.pl.scatter(adata, x='total_counts', y='n_genes_by_counts')
```

Filtering step:

```
adata = adata[adata.obs.pct_counts_mt < 5, :] # filter cells with >5% MT reads
```

Normalization

- Total-count normalize (library-size correct) the data matrix $\mathbf{X}$ to 10,000 reads per cell, so that counts become comparable among cells

```
sc.pp.normalize_total(adata, target_sum=1e4)
```

```
normalizing counts per cell
    finished (0:00:00)
```

- Logarithmize the data

```
sc.pp.log1p(adata)
```

- Important: save a copy of the raw data file before any gene filtering is performed in the next step!

```
adata_raw = adata.copy()
```

Or: `adata.raw = adata.X`

Finding highly variable genes

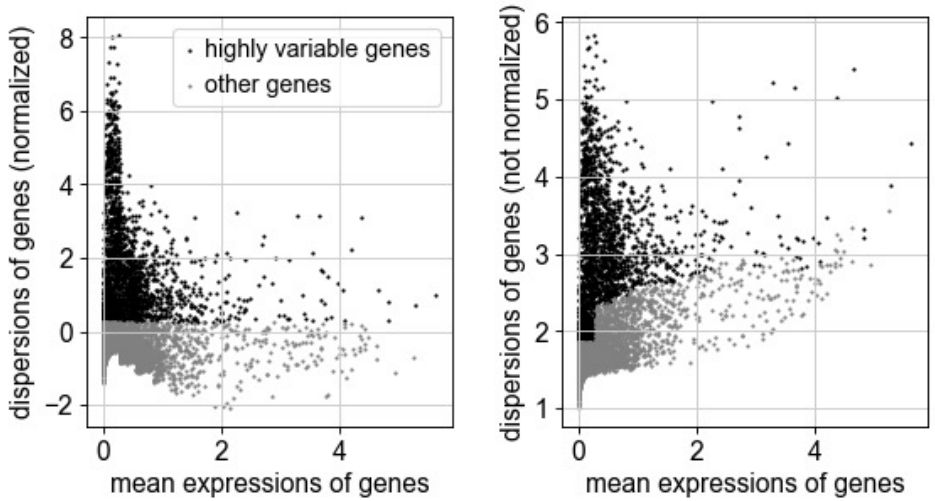
- Select a subset of all genes to use for dimensionality reduction
- Highly variable genes better capture the heterogeneity of the dataset

```
sc.pp.highly_variable_genes(adata, min_mean=0.0125, max_mean=3, min_disp=0.5, n_top_genes=2000)
```

```
If you pass `n_top_genes`, all cutoffs are ignored.
extracting highly variable genes
    finished (0:00:00)
--> added
    'highly_variable', boolean vector (adata.var)
    'means', float vector (adata.var)
    'dispersions', float vector (adata.var)
    'dispersions_norm', float vector (adata.var)
```

- Visualize selected genes

```
sc.pl.highly_variable_genes(adata)
```



- Actually do the gene filtering:

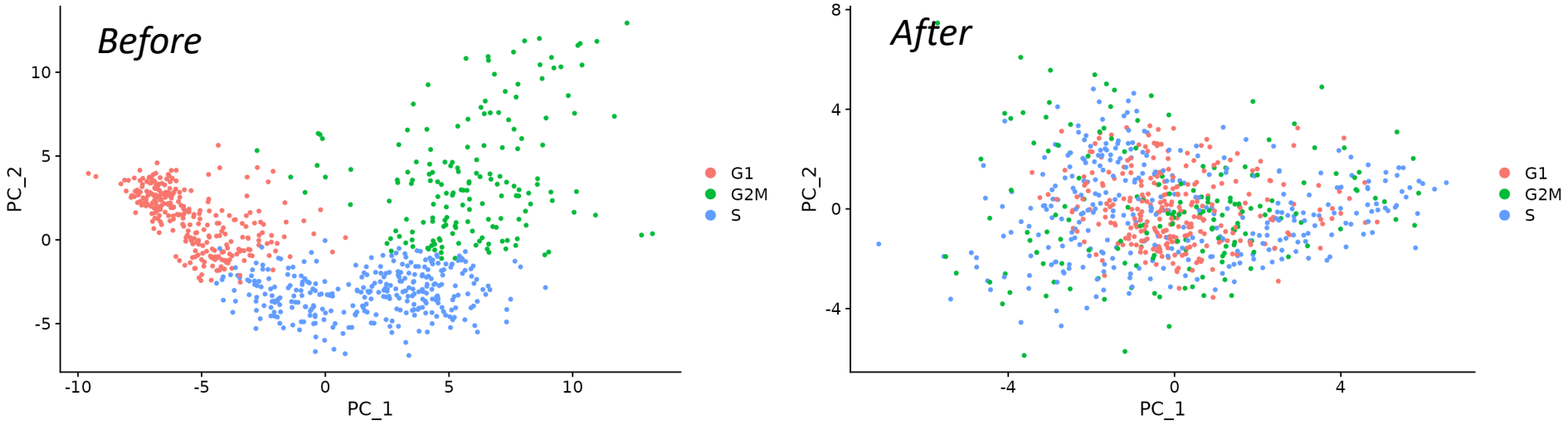
```
adata = adata[:, adata.var.highly_variable]
```

Regression and scaling

- Regress out effects of total counts per cell and the percentage of mitochondrial genes expressed.
- Scale each gene to unit variance. Clip values exceeding standard deviation 10.

```
sc.pp.regress_out(adata, ['total_counts', 'pct_counts_mt'])
```

```
regressing out ['total_counts', 'pct_counts_mt']
    sparse input is densified and may lead to high memory use
    finished (0:00:08)
```



https://satijalab.org/seurat/articles/cell_cycle_vignette.html

Regression and scaling

- Regress out effects of total counts per cell and the percentage of mitochondrial genes expressed.
- Scale each gene to unit variance. Clip values exceeding standard deviation 10.

```
sc.pp.regress_out(adata, ['total_counts', 'pct_counts_mt'])
```

```
regressing out ['total_counts', 'pct_counts_mt']
    sparse input is densified and may lead to high memory use
    finished (0:00:08)
```

```
sc.pp.scale(adata, max_value=10)
```

Center data so that mean=0 and unit variance; clip all values larger than 10

=> Avoids very highly expressed genes having a biased influence on dimensionality reduction steps.

Regression and scaling

- Regress out effects of total counts per cell and the percentage of mitochondrial genes expressed.
- Scale each gene to unit variance. Clip values exceeding standard deviation 10.

```
sc.pp.regress_out(adata, ['total_counts', 'pct_counts_mt'])
```

```
regressing out ['total_counts', 'pct_counts_mt']
    sparse input is densified and may lead to high memory use
    finished (0:00:08)
```

```
sc.pp.scale(adata, max_value=10)
```

Center data so that mean=0 and unit variance; clip all values larger than 10
=> Avoids very highly expressed genes having a biased influence on downstream analysis steps.

- AnnData object continuing to be populated, now only includes 2000 highly variable genes

```
adata
```

```
AnnData object with n_obs × n_vars = 2432 × 2000
    obs: 'n_counts', 'n_genes', 'n_genes_by_counts', 'total_counts', 'total_counts_mt', 'pct_counts_mt'
    var: 'gene_ids', 'n_cells', 'mt', 'n_cells_by_counts', 'mean_counts', 'pct_dropout_by_counts', 'total_counts', 'h
ighly_variable', 'means', 'dispersions', 'dispersions_norm', 'mean', 'std'
    uns: 'log1p', 'hvg'
```

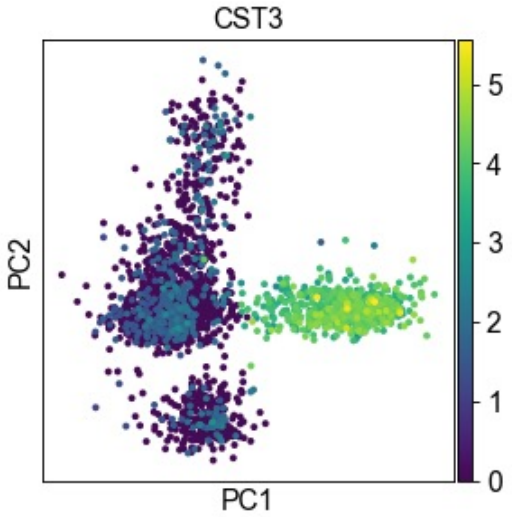
Principal component analysis

- Reduce the dimensionality of the data by running principal component analysis (PCA), which reveals the main axes of variation and denoises the data.

```
sc.tl.pca(adata, svd_solver='arpack')
```

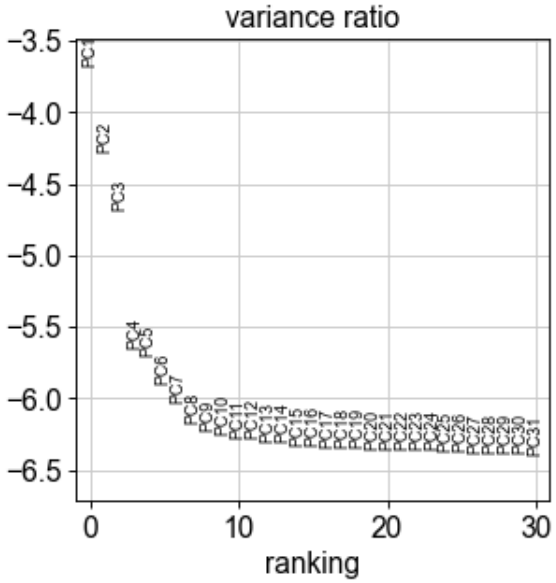
```
computing PCA
    on highly variable genes
    with n_comps=50
    finished (0:00:00)
```

- We can make a scatter plot in the PCA coordinates, but we will not use that later on.



- We can inspect the contribution of single PCs to the total variance in the data. This gives us information about how many PCs we should consider in order to compute the neighborhood relations of cells.

```
sc.pl.pca_variance_ratio(adata, log=True)
```



Computing and embedding the neighborhood graph

- Compute the neighborhood graph of cells using the PCA representation of the data matrix.

```
sc.pp.neighbors(adata, n_neighbors=20, n_pcs=10)
```

```
computing neighbors
    using 'X_pca' with n_pcs = 10
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:00)
```

- Embedding the graph can be performed using either tSNE or UMAP algorithms

```
In [77]: sc.tl.umap(adata)
```

```
computing UMAP
    finished: added
    'X_umap', UMAP coordinates (adata.obsm) (0:00:03)
```

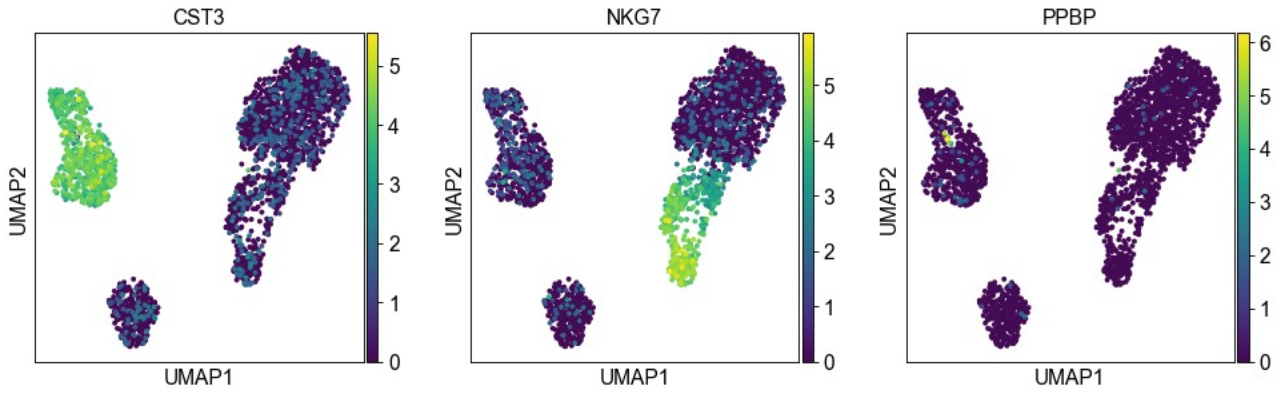
```
In [*]: sc.tl.tsne(adata)
```

```
computing tSNE
    using 'X_pca' with n_pcs = 50
    using the 'MulticoreTSNE' package by Ulyanov (2017)
```

Visualizing the data with tSNE or UMAP

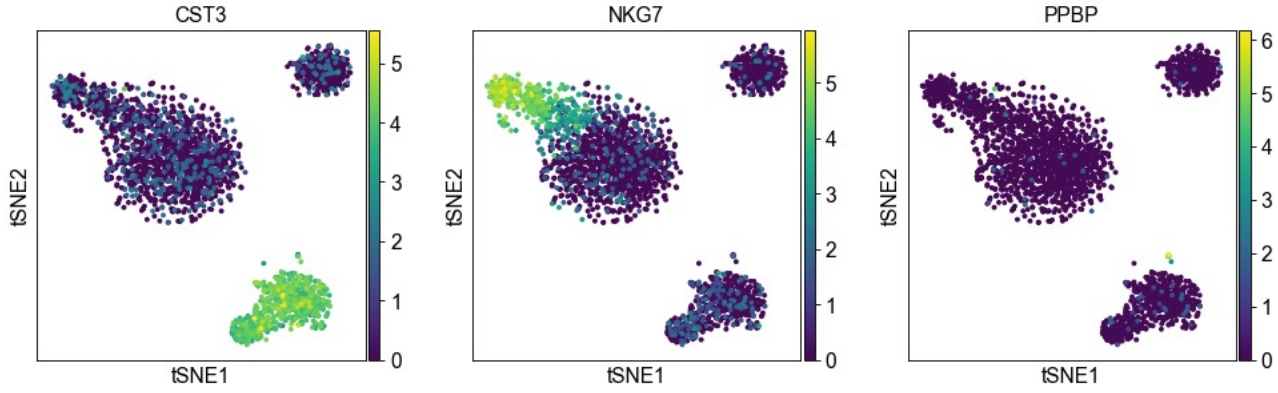
UMAP

```
sc.pl.umap(adata, color=['CST3', 'NKG7', 'PPBP'])
```



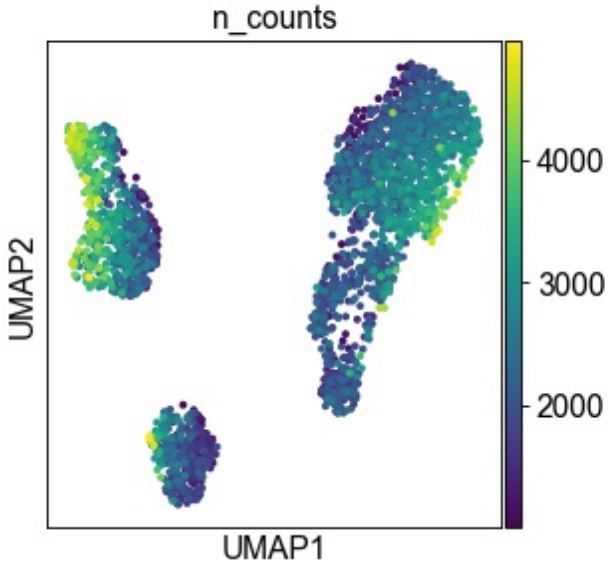
tSNE

```
sc.pl.tsne(adata, color=['CST3', 'NKG7', 'PPBP'])
```



The `color` attribute can be used for any gene in the AnnData object as well as for any metadata features in `adata.obs`

```
sc.pl.umap(adata, color=['n_counts'])
```



Clustering the UMAP

- Louvain or Leiden clustering

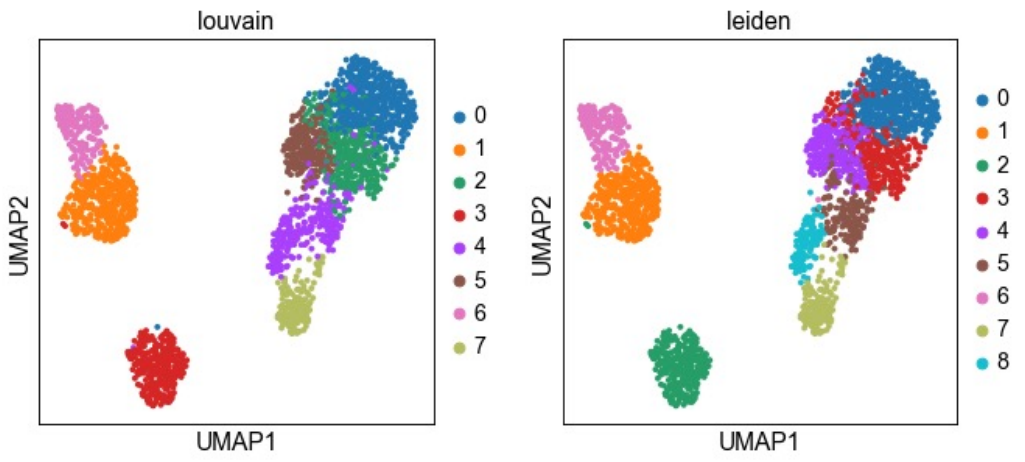
```
sc.tl.louvain(adata, resolution=1)
```

```
running Louvain clustering
    using the "louvain" package of Traag (2017)
    finished: found 8 clusters and added
    'louvain', the cluster labels (adata.obs, categorical) (0:00:00)
```

```
sc.tl.leiden(adata, resolution=1)
```

```
running Leiden clustering
    finished: found 9 clusters and added
    'leiden', the cluster labels (adata.obs, categorical) (0:00:00)
```

```
sc.pl.umap(adata, color=['louvain', 'leiden'])
```



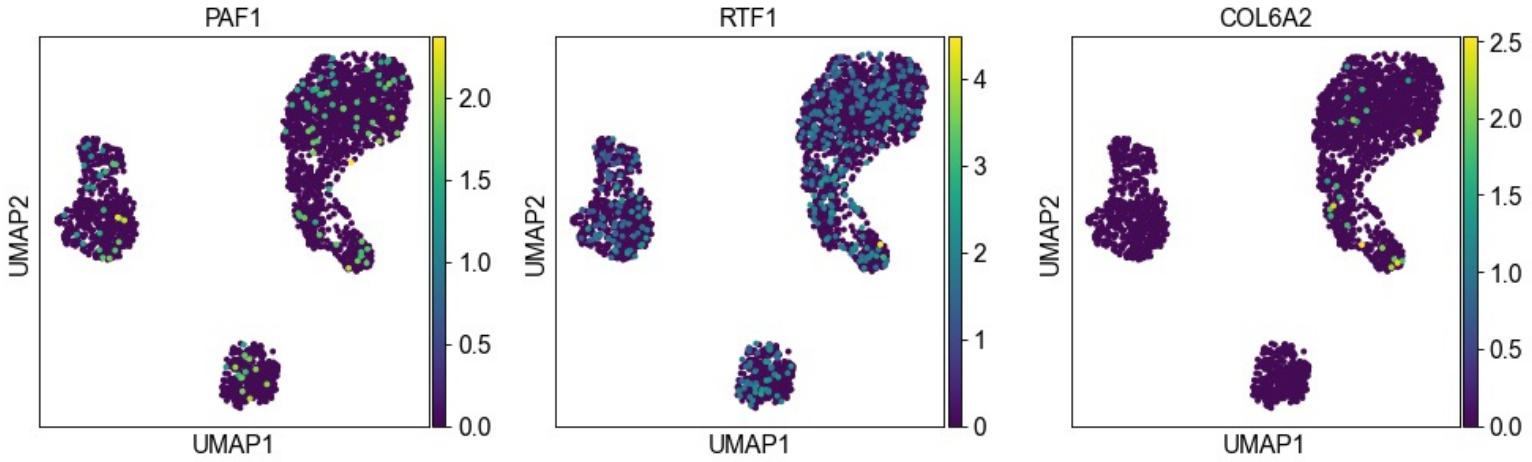
resolution parameter = adjust number of clusters

- Higher resolution = more clusters
- Lower resolution = fewer clusters

Visualizing the data with tSNE or UMAP

- If we want to visualize genes that weren't considered highly variable, we can use our `adata_raw` object
- First we must transfer over the metadata, however.

```
adata_raw.obsm["X_umap"] = adata.obsm["X_umap"]
sc.pl.umap(adata_raw, color=["PAF1", "RTF1", "COL6A2"])
```



- Recommended to use the full list of genes (after initial QC filtering) when looking at differential expression

AnnData objects

- Now the AnnData object is pretty packed with information!

```
adata
```

```
AnnData object with n_obs × n_vars = 2432 × 2000
    obs: 'n_counts', 'n_genes', 'n_genes_by_counts', 'total_counts', 'total_counts_mt', 'pct_counts_mt', 'leiden', 'l
ouvain'
    var: 'gene_ids', 'n_cells', 'mt', 'n_cells_by_counts', 'mean_counts', 'pct_dropout_by_counts', 'total_counts', 'h
ighly_variable', 'means', 'dispersions', 'dispersions_norm', 'mean', 'std'
    uns: 'log1p', 'hvg', 'pca', 'neighbors', 'umap', 'leiden', 'louvain', 'leiden_colors', 'louvain_colors'
    obsm: 'X_pca', 'X_umap', 'X_tsne'
    varm: 'PCs'
    obsp: 'distances', 'connectivities'
```

```
adata.obs
```

	n_counts	n_genes	n_genes_by_counts	total_counts	total_counts_mt	pct_counts_mt	leiden	louvain
AAACATACAACCAC-1	2421.0	781	779	2419.0	73.0	3.017776	3	4
AAACATTGAGCTAC-1	4903.0	1352	1349	4899.0	186.0	3.796693	2	3
AAACATTGATCAGC-1	3149.0	1131	1127	3145.0	28.0	0.890302	4	5
AAACCGTGCTTCCG-1	2639.0	960	958	2637.0	46.0	1.744406	6	6
AAACGCACTGGTAC-1	2164.0	782	780	2155.0	36.0	1.670534	4	5

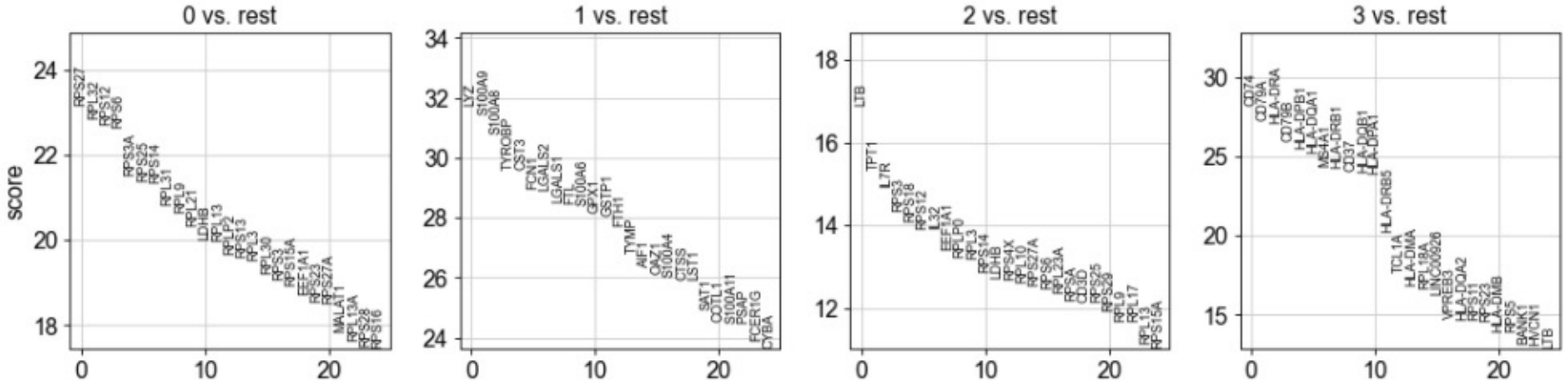
```
adata.var
```

	gene_ids	n_cells	mt	n_cells_by_counts	mean_counts	pct_dropout_by_counts	total_counts	highly_variable	means	dispersions
ISG15	ENSG00000187608	1091	False	1091	1.125914	55.686434	2772.0	True	1.714805	2.908695
TNFRSF18	ENSG00000186891	89	False	89	0.044273	96.385053	109.0	True	0.171695	1.917839
TNFRSF4	ENSG00000186827	149	False	149	0.081641	93.948010	201.0	True	0.284065	2.070605
CPSF3L	ENSG00000127054	181	False	181	0.095045	92.648253	234.0	True	0.390877	4.567900
MRPL20	ENSG00000242485	626	False	626	0.337530	74.573517	831.0	True	0.887434	2.743042

Finding marker genes

- Let us compute a ranking for the highly differential genes in each cluster using the Wilcoxon rank-sum test. For this, by default, the .raw attribute of AnnData is used in case it has been initialized before.

```
sc.tl.rank_genes_groups(adata, 'louvain', method='wilcoxon')
sc.pl.rank_genes_groups(adata, n_genes=25, sharey=False)
```



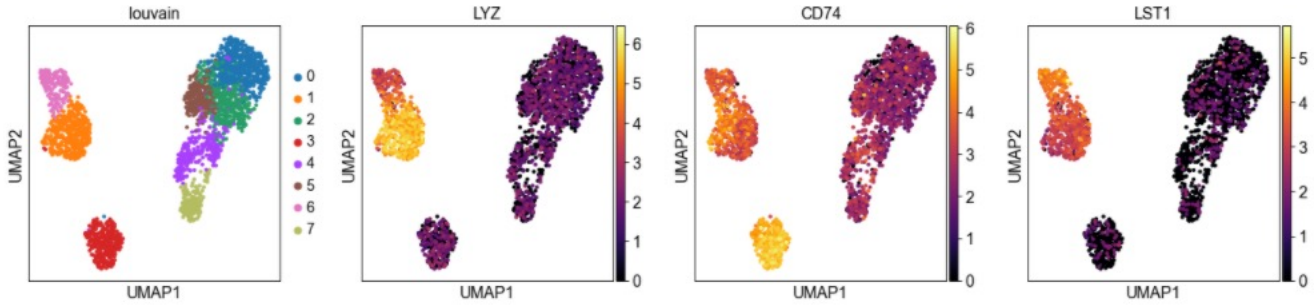
More easily access the marker genes

```
pd.DataFrame(adata.uns["rank_genes_groups"]["names"]).head()
```

	0	1	2	3	4	5	6	7
0	RPS27	LYZ	LTB	CD74	CCL5	IL32	LST1	NKG7
1	RPL32	S100A9	TPT1	CD79A	NKG7	CD3D	COTL1	GNLY
2	RPS12	S100A8	IL7R	HLA-DRA	B2M	LTB	AIF1	GZMB
3	RPS6	TYROBP	RPS3	CD79B	CST7	HLA-A	FCER1G	CTSW
4	RPS3A	CST3	RPS18	HLA-DPB1	GZMA	B2M	FTH1	PRF1

Visualize marker genes on UMAP or tSNE:

```
sc.pl.umap(adata, color=["louvain", "LYZ", "CD74", "LST1"], cmap='inferno')
```

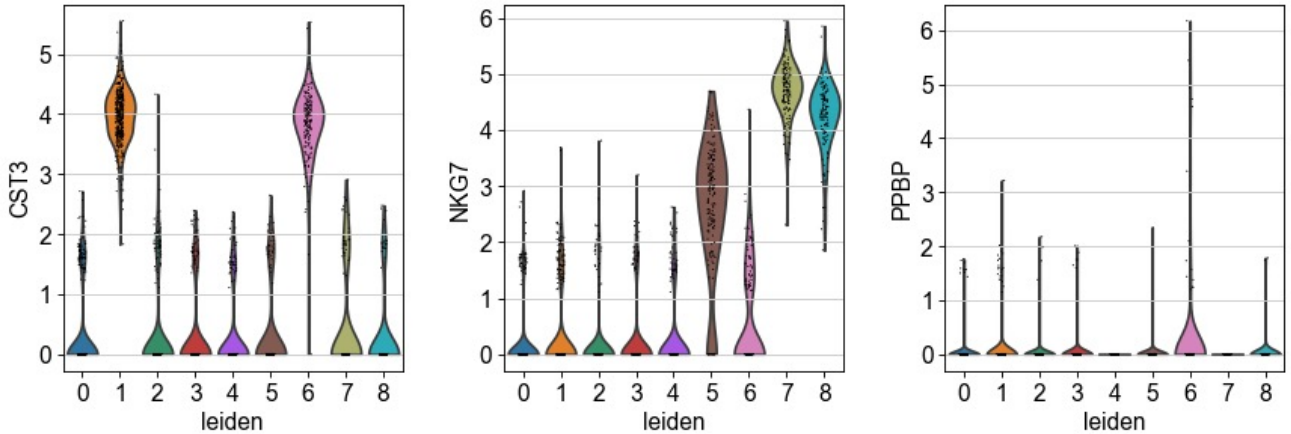


Compare gene expression across cell types

- If you want to compare a certain gene across groups, one can use the following:

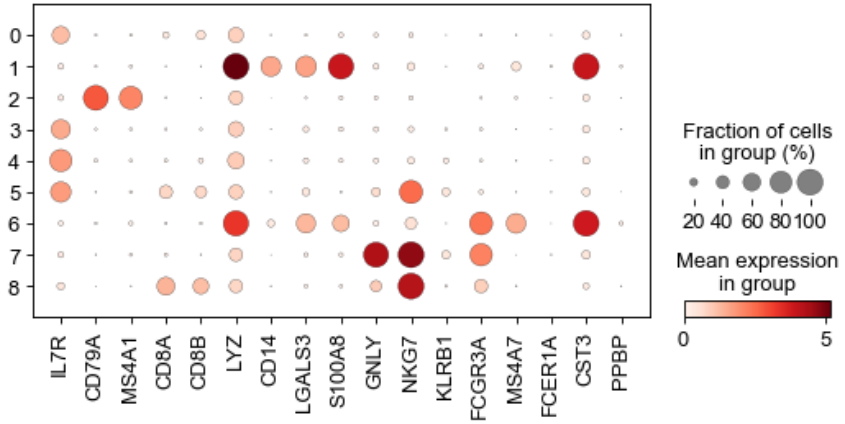
Violin plots

```
sc.pl.violin(adata, ['CST3', 'NKG7', 'PPBP'], groupby='leiden')
```



Dot plots

```
marker_genes = ['IL7R', 'CD79A', 'MS4A1', 'CD8A', 'CD8B', 'LYZ', 'CD14',
                'LGALS3', 'S100A8', 'GNLY', 'NKG7', 'KLRB1',
                'FCGR3A', 'MS4A7', 'FCER1A', 'CST3', 'PPBP']
sc.pl.dotplot(adata, marker_genes, groupby='leiden');
```



Assigning cell types to Louvain clusters

- Almost the most challenging step!
- Use the literature to annotate marker genes for each cluster and obtain cell type estimates:
  - Google search of gene names is often the most useful for finding relevant papers!
  - Online tools: GeneCards, EnrichR, Gene Ontology
  - **Machine learning based approach: Celltypist**

Don’t forget to save your analysis file for later use!

```
adata.write("output_file_name.h5ad")
```

Assigning cell types to Louvain clusters

- Almost the most challenging step!
- Use the literature to annotate marker genes for each cluster and obtain cell type estimates:
  - Google search of gene names is often the most useful for finding relevant papers!
  - Online tools: GeneCards, EnrichR, Gene Ontology
  - **Machine learning based approach: Celltypist**

Don't forget to save your analysis file for later use!

```
adata.write("output_file_name.h5ad")
```

Sub-clustering after the initial analysis

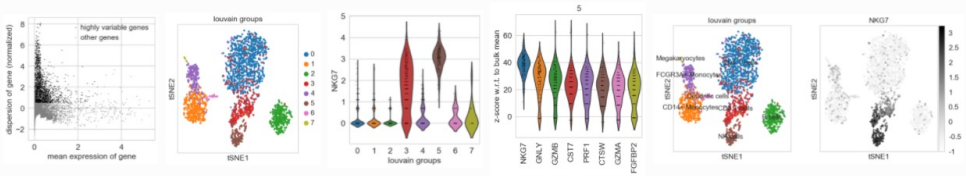
- Additional heterogeneity can sometimes be uncovered by sub-clustering
- Given the cluster annotations from the initial analysis, select the cells from a single cluster
- Using that single cluster, repeat scanpy analysis

Next steps:

- Visit the scanpy website and practice with their tutorials!
https://scanpy.readthedocs.io/en/stable/tutorials.html#

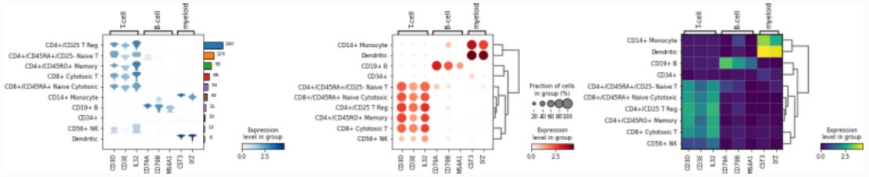
Clustering (covered in depth in these slides)

For getting started, we recommend Scanpy's reimplementation → tutorial: pbmc3k of Seurat's [Satija15] clustering tutorial for 3k PBMCs from 10x Genomics, containing preprocessing, clustering and the identification of cell types via known marker genes.



Visualization

This tutorial shows how to visually explore genes using scanpy. → tutorial: plotting/core



Integrating datasets

Map labels and embeddings of reference data to new data: → tutorial: integrating-data-using-ingest