Analysis and visualization of spatial transcriptomics data

Author: Giovanni Palla

- This tutorial demonstrates how to work with spatial transcriptomics data within Scanpy.
- We focus on 10x Genomics Visium data, and provide an example for MERFISH.

[1]:
```
import scanpy as sc
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
sc.logging.print_versions()
sc.set_figure_params(facecolor="white", figsize=(8, 8))
sc.settings.verbosity = 3
```

```
scanpy==1.5.0 anndata==0.7.1 umap==0.4.2 numpy==1.18.1 scipy==1.4.1 pandas==1.0.3
scikit-learn==0.22.1 statsmodels==0.11.0
```

Reading the data

We will use a Visium spatial transcriptomics dataset of the human lymphnode, which is publicly available from the 10x genomics website: link.

The function datasets.visium_sge() downloads the dataset from 10x Genomics and returns an `AnnData` object that contains counts, images and spatial coordinates. We will calculate standards QC metrics with pp.calculate_qc_metrics and percentage of mitochondrial read counts per sample.

When using your own Visium data, use sc.read_visium() function to import it.

[3]:
```
adata = sc.datasets.visium_sge(sample_id="V1_Human_Lymph_Node")
adata.var_names_make_unique()
adata.var["mt"] = adata.var_names.str.startswith("MT-")
sc.pp.calculate_qc_metrics(adata, qc_vars=["mt"], inplace=True)
```

```
reading /Users/giovanni.palla/Projects/scanpy-
tutorials/spatial/data/V1_Human_Lymph_Node/filtered_feature_bc_matrix.h5
 (0:00:01)
```

This is how the adata structure looks like for Visium data

[4]:
```
adata
```

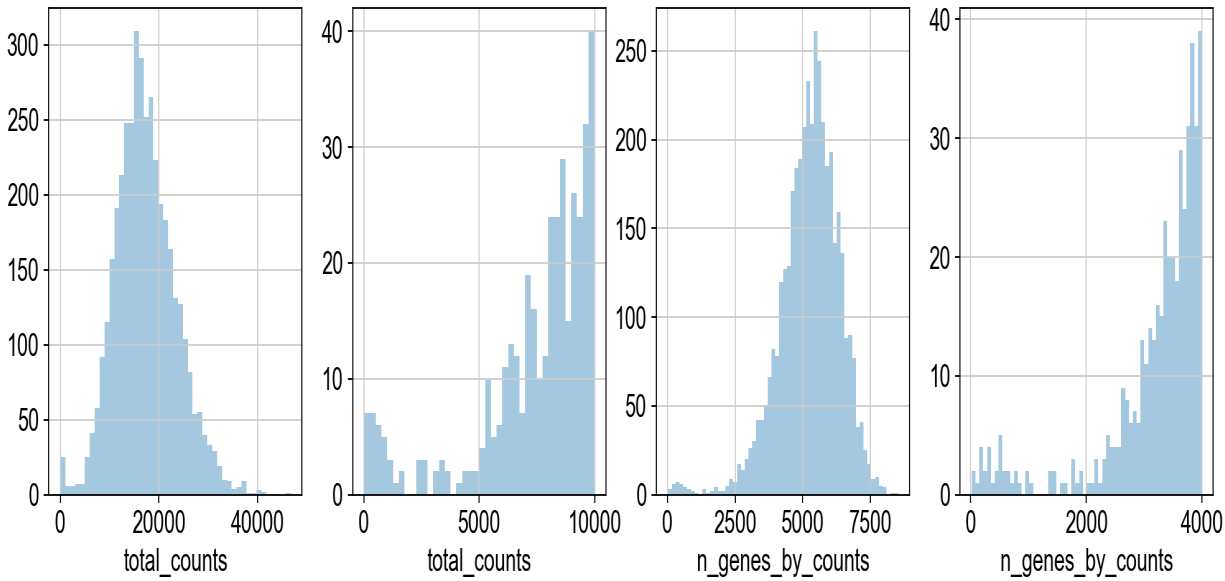
[4]:
```
AnnData object with n_obs × n_vars = 4039 × 33538
    obs: 'in_tissue', 'array_row', 'array_col', 'n_genes_by_counts',
'log1p_n_genes_by_counts', 'total_counts', 'log1p_total_counts',
'pct_counts_in_top_50_genes', 'pct_counts_in_top_100_genes',
'pct_counts_in_top_200_genes', 'pct_counts_in_top_500_genes', 'total_counts_mt',
'log1p_total_counts_mt', 'pct_counts_mt'
    var: 'gene_ids', 'feature_types', 'genome', 'mt', 'n_cells_by_counts',
'mean_counts', 'log1p_mean_counts', 'pct_dropout_by_counts', 'total_counts',
'log1p_total_counts'
    uns: 'spatial'
    obsm: 'spatial'
```

QC and preprocessing

We perform some basic filtering of spots based on total counts and expressed genes

[5]:
```
fig, axs = plt.subplots(1, 4, figsize=(15, 4))
sns.distplot(adata.obs["total_counts"], kde=False, ax=axs[0])
sns.distplot(adata.obs["total_counts"][adata.obs["total_counts"] < 10000],
kde=False, bins=40, ax=axs[1])
sns.distplot(adata.obs["n_genes_by_counts"], kde=False, bins=60, ax=axs[2])
sns.distplot(adata.obs["n_genes_by_counts"][adata.obs["n_genes_by_counts"] < 4000],
kde=False, bins=60, ax=axs[3])
```

[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x13045d0d0>
```



[6]:
```
sc.pp.filter_cells(adata, min_counts=5000)
sc.pp.filter_cells(adata, max_counts=35000)
adata = adata[adata.obs["pct_counts_mt"] < 20]
print(f"#cells after MT filter: {adata.n_obs}")
sc.pp.filter_genes(adata, min_cells=10)
```

```
filtered out 51 cells that have less than 5000 counts
filtered out 26 cells that have more than 35000 counts
#cells after MT filter: 3962
filtered out 15071 genes that are detected in less than 10 cells
```

We proceed to normalize Visium counts data with the built-in `normalize_total` method from Scanpy, and detect highly-variable genes (for later). Note that there are alternatives for normalization (see discussion in [Luecken19], and more recent alternatives such as SCTransform or GLM-PCA).

[7]:
```
sc.pp.normalize_total(adata, inplace=True)
sc.pp.log1p(adata)
sc.pp.highly_variable_genes(adata, flavor="seurat", n_top_genes=2000)
```

```
normalizing counts per cell
    finished (0:00:00)
extracting highly variable genes
    finished (0:00:01)
--> added
    'highly_variable', boolean vector (adata.var)
    'means', float vector (adata.var)
    'dispersions', float vector (adata.var)
    'dispersions_norm', float vector (adata.var)
```

Manifold embedding and clustering based on transcriptional similarity

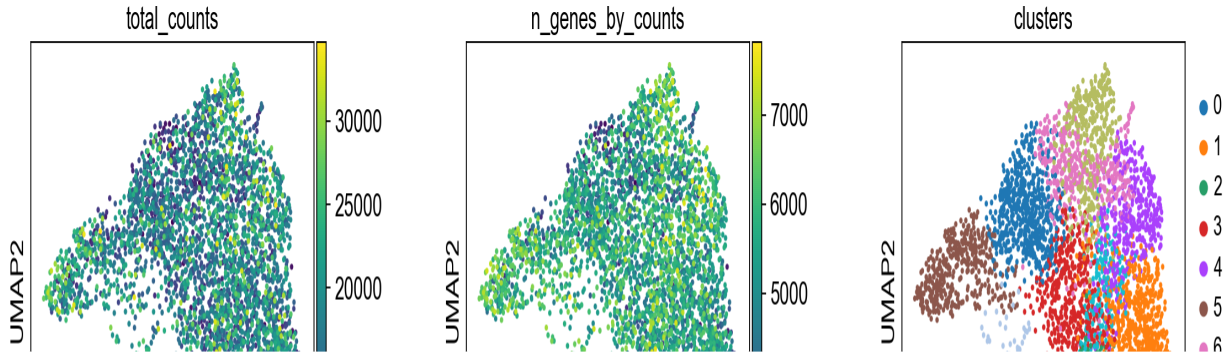
To embed and cluster the manifold encoded by transcriptional similarity, we proceed as in the standard clustering tutorial.

[8]:
```
sc.pp.pca(adata)
sc.pp.neighbors(adata)
sc.tl.umap(adata)
sc.tl.leiden(adata, key_added="clusters")
```

```
computing PCA
    on highly variable genes
    with n_comps=50
    finished (0:00:01)
computing neighbors
    using 'X_pca' with n_pcs = 50
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:02)
computing UMAP
    finished: added
    'X_umap', UMAP coordinates (adata.obsm) (0:00:06)
running Leiden clustering
    finished: found 10 clusters and added
    'clusters', the cluster labels (adata.obs, categorical) (0:00:01)
```

We plot some covariates to check if there is any particular structure in the UMAP associated with total counts and detected genes.

[9]:
```
plt.rcParams["figure.figsize"] = (4, 4)
sc.pl.umap(adata, color=["total_counts", "n_genes_by_counts", "clusters"],
wspace=0.4)
```



Visualization in spatial coordinates

Let us now take a look at how `total_counts` and `n_genes_by_counts` behave in spatial coordinates. We will overlay the circular spots on top of the Hematoxylin and eosin stain (H&E) image provided, using the function sc.pl.spatial.

[10]:
```
plt.rcParams["figure.figsize"] = (8, 8)
sc.pl.spatial(adata, img_key="hires", color=["total_counts", "n_genes_by_counts"])
```

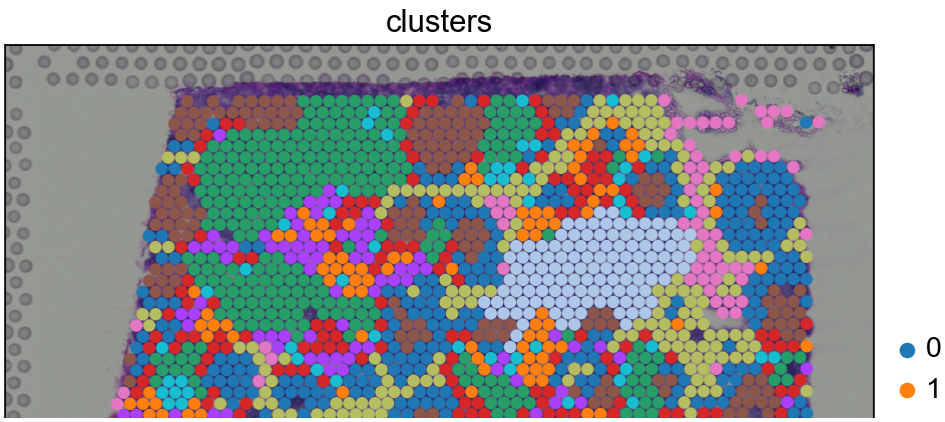
The function sc.pl.spatial accepts 4 additional parameters:

- `img_key`: key where the img is stored in the `adata.uns` element
- `crop_coord`: coordinates to use for cropping (left, right, top, bottom)
- `alpha_img`: alpha value for the transcparency of the image
- `bw`: flag to convert the image into gray scale

Furthermore, in sc.pl.spatial, the `size` parameter changes its behaviour: it becomes a scaling factor for the spot sizes.

Before, we performed clustering in gene expression space, and visualized the results with UMAP. By visualizing clustered samples in spatial dimensions, we can gain insights into tissue organization and, potentially, into inter-cellular communication.

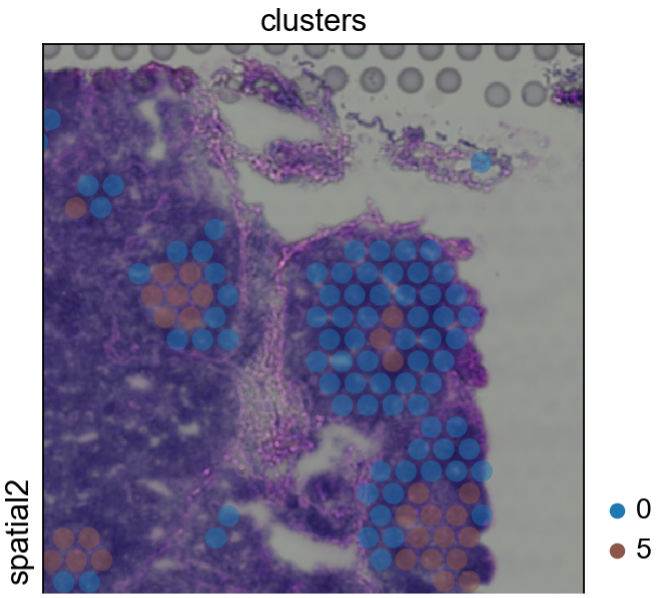
[11]:
```
sc.pl.spatial(adata, img_key="hires", color="clusters", size=1.5)
```



Spots belonging to the same cluster in gene expression space often co-occur in spatial dimensions. For instance, spots belonging to cluster 5 are often surrounded by spots belonging to cluster 0.

We can zoom in specific regions of interests to gain qualitative insights. Furthermore, by changing the alpha values of the spots, we can visualize better the underlying tissue morphology from the H&E image.

[12]:
```
sc.pl.spatial(adata, img_key="hires", color="clusters", groups=["0", "5"],
crop_coord=[1200, 1700, 1900, 1000], alpha=0.5, size=1.3)
```



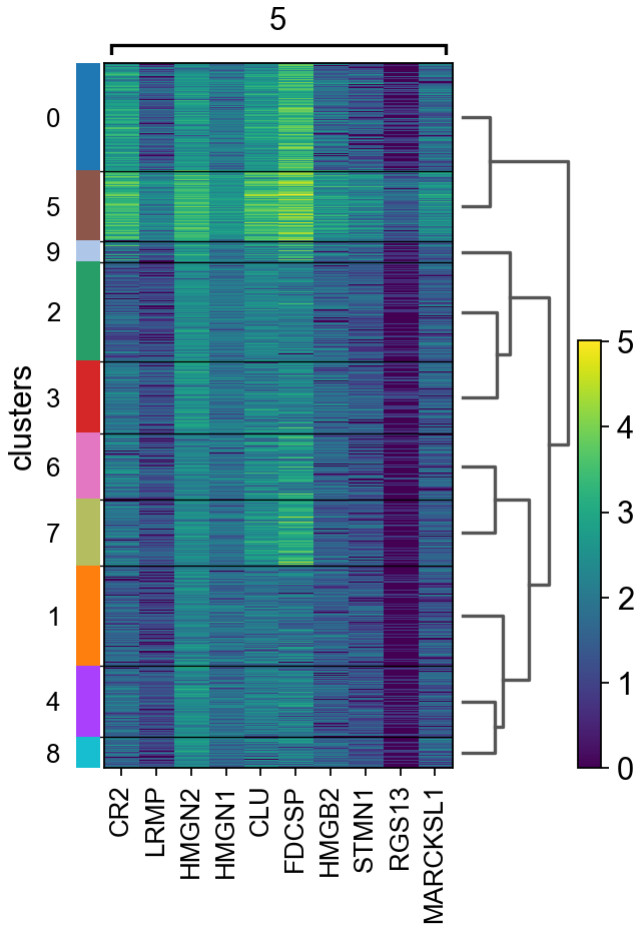
Cluster marker genes

Let us further inspect cluster 5, which occurs in small groups of spots across the image.

Compute marker genes and plot a heatmap with expression levels of its top 10 marker genes across clusters.

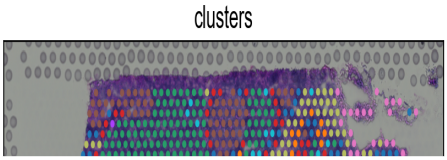
[13]:
```
sc.tl.rank_genes_groups(adata, "clusters", method="t-test")
sc.pl.rank_genes_groups_heatmap(adata, groups="5", n_genes=10, groupby="clusters")
```

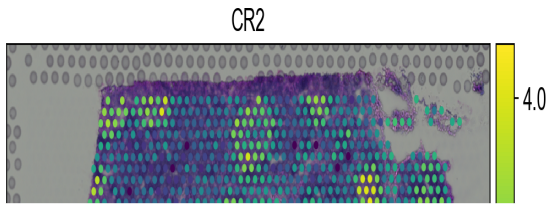
```
ranking genes
    finished: added to `.uns['rank_genes_groups']`
    'names', sorted np.recarray to be indexed by group ids
    'scores', sorted np.recarray to be indexed by group ids
    'logfoldchanges', sorted np.recarray to be indexed by group ids
    'pvals', sorted np.recarray to be indexed by group ids
    'pvals_adj', sorted np.recarray to be indexed by group ids (0:00:01)
WARNING: dendrogram data not found (using key=dendrogram_clusters). Running
`sc.tl.dendrogram` with default parameters. For fine tuning it is recommended to run
`sc.tl.dendrogram` independently.
    using 'X_pca' with n_pcs = 50
Storing dendrogram info using `.uns['dendrogram_clusters']`
WARNING: Groups are not reordered because the `groupby` categories and the
`var_group_labels` are different.
categories: 0, 1, 2, etc.
var_group_labels: 5
```



We see that *CR2* recapitulates the spatial structure.

[14]:
```
sc.pl.spatial(adata, img_key="hires", color=["clusters", "CR2"])
```





Spatially variable genes

Spatial transcriptomics allows researchers to investigate how gene expression trends varies in space, thus identifying spatial patterns of gene expression. For this purpose, we use SpatialDE Svensson18 (code), a Gaussian process-based statistical framework that aims to identify spatially variable genes:

```
pip install spatialde
```

Recently, other tools have been proposed for the identification of spatially variable genes, such as:

- SPARK paper - code
- trendsceek paper - code
- HMRF paper - code

First, we convert normalized counts and coordinates to pandas dataframe, needed for inputs to spatialDE.

Running SpatialDE takes considerable time.

[15]:
```
import SpatialDE
```

[16]:
```
%%time
counts = pd.DataFrame(adata.X.todense(), columns=adata.var_names,
index=adata.obs_names)
coord = pd.DataFrame(adata.obsm['spatial'], columns=['x_coord', 'y_coord'],
index=adata.obs_names)
results = SpatialDE.run(coord, counts)
```

```
CPU times: user 1h 22min 58s, sys: 1min 8s, total: 1h 24min 6s
Wall time: 21min 54s
```

We concatenate the results with the DataFrame of annotations of variables: `adata.var`.

[17]:
```
results.index = results["g"]
adata.var = pd.concat([adata.var, results.loc[adata.var.index.values, :]], axis=1)
```

Then we can inspect significant genes that varies in space and visualize them with `sc.pl.spatial` function.

[18]:
```
results.sort_values("qval").head(10)
```

[18]:

	FSV	M	g	l	max_delta	max_ll	max_mu_hat	max_s2_t_hat
g								
EPHA1	0.092282	4	EPHA1	474.169746	9.649444	-912.478913	0.137565	0.009403
CPVL	0.081288	4	CPVL	474.169746	11.087121	-2683.019176	0.398468	0.021507
PPIA	0.298773	4	PPIA	474.169746	2.302420	-292.511593	3.092615	0.162267
SAMD9L	0.125077	4	SAMD9L	474.169746	6.862115	-3003.385012	0.585090	0.041092
COL1A2	0.090461	4	COL1A2	474.169746	9.863383	-3227.520356	0.740816	0.035660
ARPC1B	0.120060	4	ARPC1B	474.169746	7.189890	-2285.547284	1.971545	0.074483
ATP5MF	0.069258	4	ATP5MF	474.169746	13.183290	-2610.782278	1.671498	0.048891
SYPL1	0.086861	4	SYPL1	474.169746	10.312907	-3329.098692	0.934305	0.039826
PARP12	0.072946	4	PARP12	474.169746	12.467190	-3120.296512	0.678568	0.027263
RARRES2	0.088121	4	RARRES2	474.169746	10.151330	-3289.541156	1.156461	0.045433

[19]:
```
sc.pl.spatial(adata, img_key="hires", color=["COL1A2", "SYPL1"], alpha=0.7)
```

MERFISH example

In case you have spatial data generated with FISH-based techniques, just read the cordinate table and assign it to the `adata.obsm` element.

Let's take a look at the example from Xia et al. 2019.

First, we need to download the coordinate and counts data from the original publication.

[20]:
```
import urllib.request
```

[21]:
```
url_coord =
"https://www.pnas.org/highwire/filestream/887973/field_highwire_adjunct_files/15/pnas
filename_coord = "pnas.1912459116.sd15.xlsx"
urllib.request.urlretrieve(url_coord, filename_coord)
```

[21]:
```
('pnas.1912459116.sd15.xlsx', <http.client.HTTPMessage at 0x12e73a150>)
```

[22]:
```
url_counts =
"https://www.pnas.org/highwire/filestream/887973/field_highwire_adjunct_files/12/pnas
filename_counts = "pnas.1912459116.sd12.csv"
urllib.request.urlretrieve(url_counts, filename_counts)
```

[22]:
```
('pnas.1912459116.sd12.csv', <http.client.HTTPMessage at 0x12e74a4d0>)
```

And read the data in a AnnData object.

[23]:
```
coordinates = pd.read_excel("./pnas.1912459116.sd15.xlsx", index_col=0)
counts = sc.read_csv("./pnas.1912459116.sd12.csv").transpose()
```

[24]:
```
adata_merfish = counts[coordinates.index, :]
adata_merfish.obsm["spatial"] = coordinates.to_numpy()
```

We will perform standard preprocessing and dimensionality reduction.

[25]:
```
sc.pp.normalize_per_cell(adata_merfish, counts_per_cell_after=1e6)
sc.pp.log1p(adata_merfish)
sc.pp.pca(adata_merfish, n_comps=15)
sc.pp.neighbors(adata_merfish)
sc.tl.umap(adata_merfish)
sc.tl.leiden(adata_merfish, key_added="clusters", resolution=0.5)
```

```
normalizing by total count per cell
    finished (0:00:00): normalized adata.X and added    'n_counts', counts per cell
before normalization (adata.obs)
computing PCA
    with n_comps=15
    finished (0:00:00)
computing neighbors
    using 'X_pca' with n_pcs = 15
    finished: added to `.uns['neighbors']`
    `.obsp['distances']`, distances for each pair of neighbors
    `.obsp['connectivities']`, weighted adjacency matrix (0:00:01)
computing UMAP
    finished: added
    'X_umap', UMAP coordinates (adata.obsm) (0:00:01)
running Leiden clustering
    finished: found 6 clusters and added
    'clusters', the cluster labels (adata.obs, categorical) (0:00:00)
```

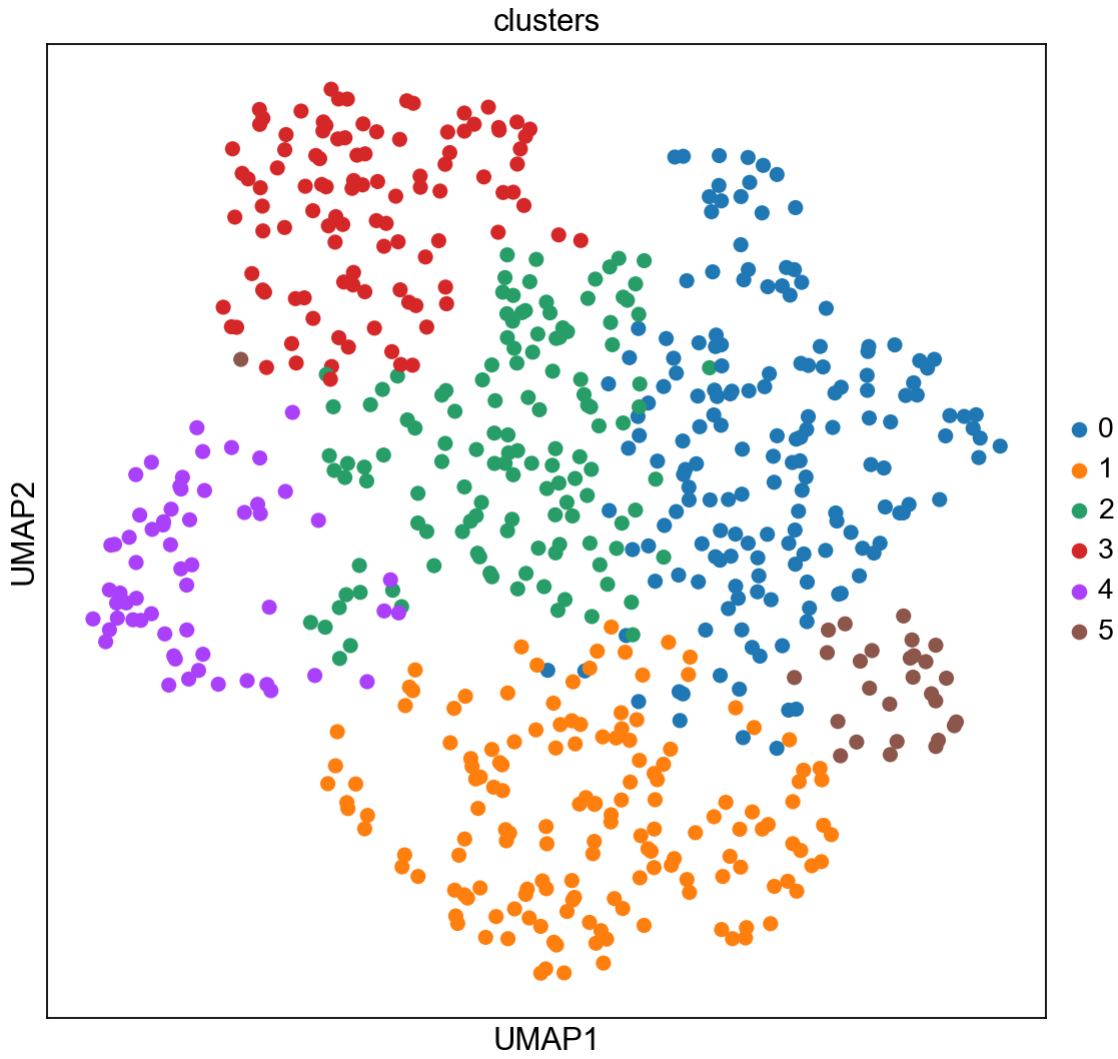
The experiment consisted in measuring gene expression counts from a single cell type (cultured U2-OS cells). Clusters consist of cell states at different stages of the cell cycle. We don't expect to see specific structure in spatial dimensions given the experimental setup.

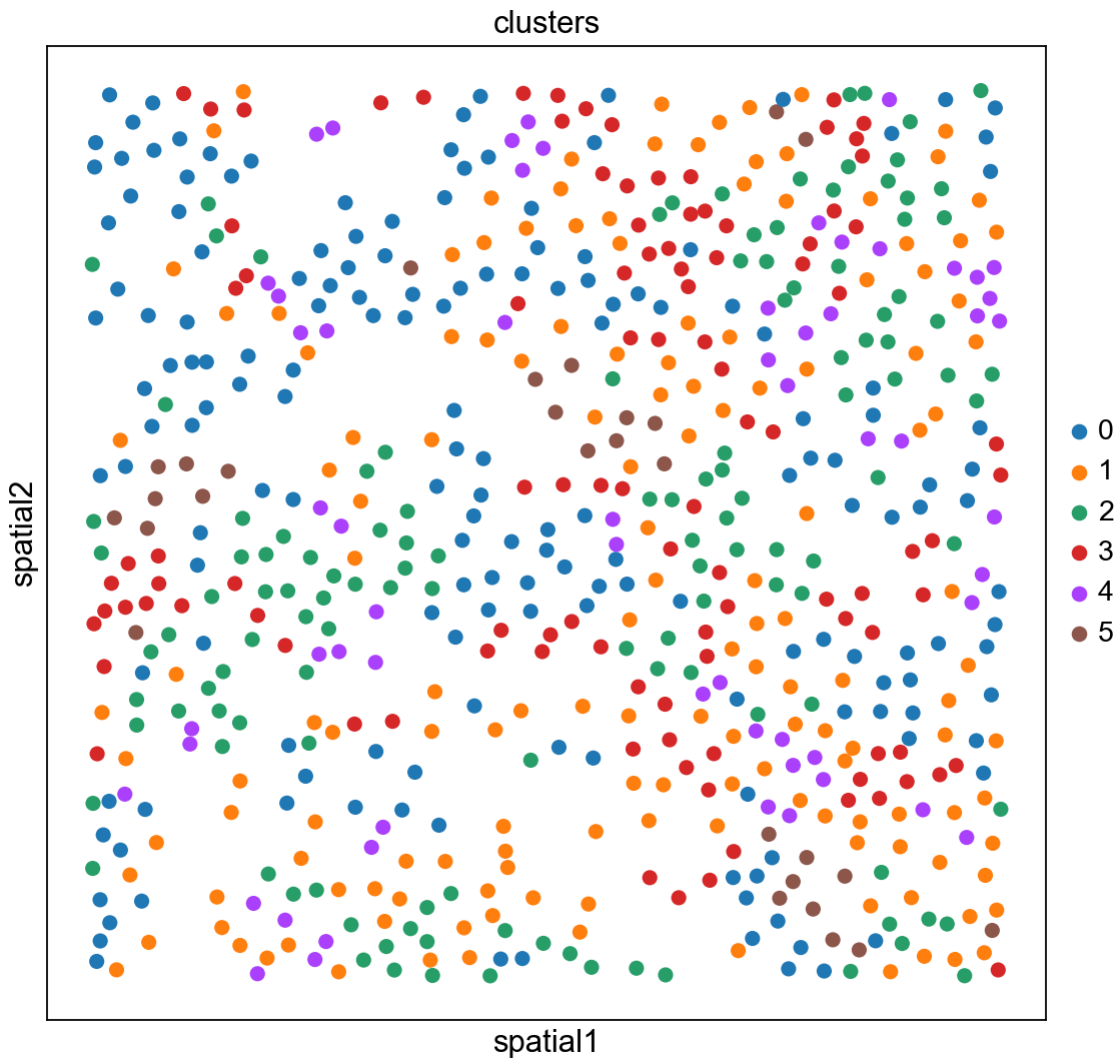
We can visualize the clusters obtained from running Leiden in UMAP space and spatial coordinates like this.

[26]:
```
adata_merfish
```

[26]:
```
AnnData object with n_obs × n_vars = 645 × 12903
    obs: 'n_counts', 'clusters'
    uns: 'log1p', 'pca', 'neighbors', 'umap', 'leiden'
    obsm: 'spatial', 'X_pca', 'X_umap'
    varm: 'PCs'
    obsp: 'distances', 'connectivities'
```

[27]:
```
sc.pl.umap(adata_merfish, color="clusters")
sc.pl.embedding(adata_merfish, basis="spatial", color="clusters")
```


clusters
UMAP2
UMAP1
0
1
2
3
4
5


clusters
spatial2
spatial1
0
1
2
3
4
5

We hope you found the tutorial useful!

Report back to us which features/external tools you would like to see in Scanpy.

We are extending Scanpy and AnnData to support other spatial data types, such as **Imaging Mass Cytometry** and extend data structure to support spatial graphs and additional features. Stay tuned!