Integrating data using ingest and BBKNN

The following tutorial describes a simple PCA-based method for integrating data we call ingest and compares it with BBKNN [Polanski19]. BBKNN integrates well with the Scanpy workflow and is accessible through the bbknn function.

The ingest function assumes an annotated reference dataset that captures the biological variability of interest. The rational is to fit a model on the reference data and use it to project new data. For the time being, this model is a PCA combined with a neighbor lookup search tree, for which we use UMAP's implementation [McInnes18]. Similar PCA-based integrations have been used before, for instance, in [Weinreb18].

- As ingest is simple and the procedure clear, the workflow is transparent and fast.
- Like BBKNN, ingest leaves the data matrix itself invariant.
- Unlike BBKNN, ingest solves the label mapping problem (like scmap) and maintains an embedding that might have desired properties like specific clusters or trajectories.

We refer to this *asymmetric* dataset integration as *ingesting* annotations from an annotated reference `adata_ref` into an `adata` that still lacks this annotation. It is different from learning a joint representation that integrates datasets in a symmetric way as BBKNN, Scanorma, Conos, CCA (e.g. in Seurat) or a conditional VAE (e.g. in scVI, trVAE) would do, but comparable to the initiall MNN implementation in scran. Take a look at tools in the external API or at the ecoystem page to get a start with other tools.

[1]:
```
import scanpy as sc
import pandas as pd
import seaborn as sns
```

[2]:
```
sc.settings.verbosity = 1             # verbosity: errors (0), warnings (1), info (2), hints (3)
sc.logging.print_versions()
sc.settings.set_figure_params(dpi=80, frameon=False, figsize=(3, 3), facecolor='white')
```

```
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
```

PBMCs

We consider an annotated reference dataset `adata_ref` and a dataset for which you want to query labels and embeddings `adata`.

[3]:
```
adata_ref = sc.datasets.pbmc3k_processed()  # this is an earlier version of the dataset from the pbmc3k tutorial
adata = sc.datasets.pbmc68k_reduced()
```

To use `sc.tl.ingest`, the datasets need to be defined on the same variables.

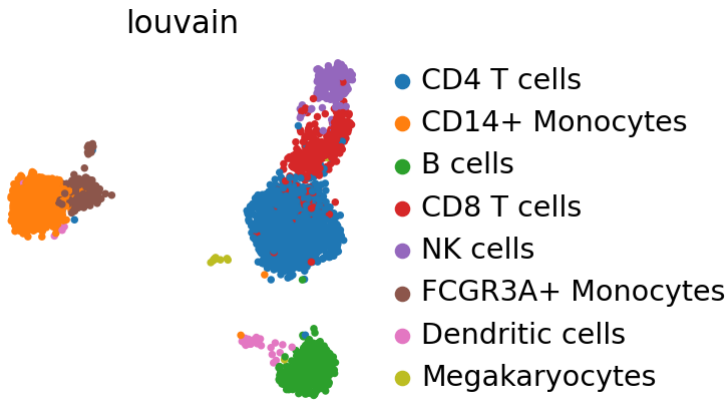
[4]:
```
var_names = adata_ref.var_names.intersection(adata.var_names)
adata_ref = adata_ref[:, var_names]
adata = adata[:, var_names]
```

The model and graph (here PCA, neighbors, UMAP) trained on the reference data will explain the biological variation observed within it.

[5]:
```
sc.pp.pca(adata_ref)
sc.pp.neighbors(adata_ref)
sc.tl.umap(adata_ref)
```

The manifold still looks essentially the same as in the clustering tutorial.

[6]:
```
sc.pl.umap(adata_ref, color='louvain')
```



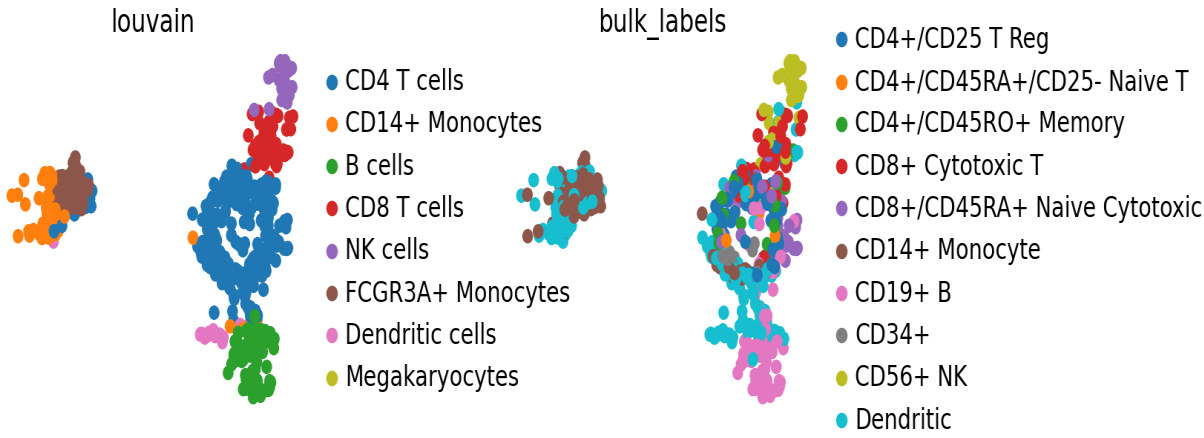
Mapping PBMCs using ingest

Let's map labels and embeddings from `adata_ref` to `adata` based on a chosen representation. Here, we use `adata_ref.obsm['X_pca']` to map cluster labels and the UMAP coordinates.

[7]:
```
sc.tl.ingest(adata, adata_ref, obs='louvain')
```

[8]:
```
adata.uns['louvain_colors'] = adata_ref.uns['louvain_colors']  # fix colors
```

[9]:
```
sc.pl.umap(adata, color=['louvain', 'bulk_labels'], wspace=0.5)
```

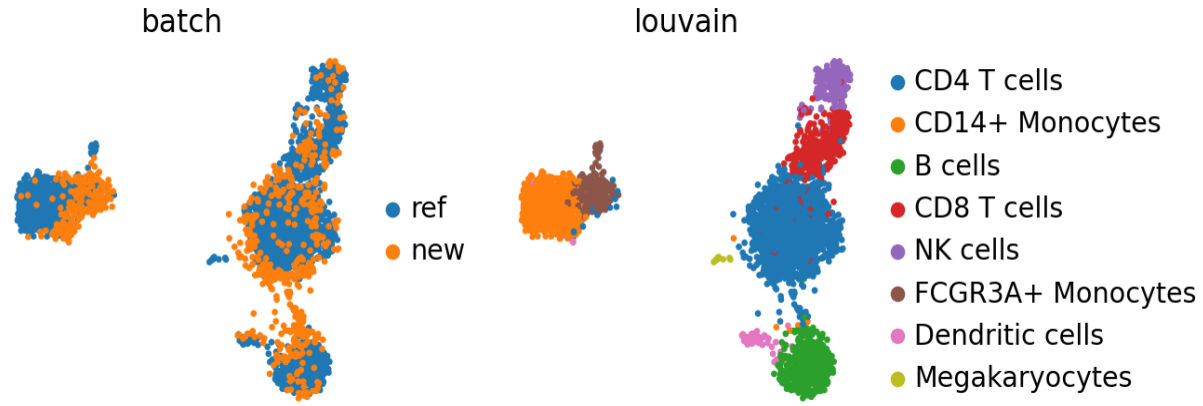


By comparing the 'bulk_labels' annotation with 'louvain', we see that the data has been reasonably mapped, only the annotation of dendritic cells seems ambiguous and might have been ambiiguous in `adata` already.

[10]:
```
adata_concat = adata_ref.concatenate(adata, batch_categories=['ref', 'new'])
```

[11]:
```
adata_concat.obs.louvain = adata_concat.obs.louvain.astype('category')
adata_concat.obs.louvain.cat.reorder_categories(adata_ref.obs.louvain.cat.categories,
inplace=True)  # fix category ordering
adata_concat.uns['louvain_colors'] = adata_ref.uns['louvain_colors']  # fix category
colors
```

[12]:
```
sc.pl.umap(adata_concat, color=['batch', 'louvain'])
```



While there seems to be some batch-effect in the monocytes and dendritic cell clusters, the new data is otherwise mapped relatively homogeneously.

The megakaryoctes are only present in `adata_ref` and no cells from `adata` map onto them. If interchanging reference data and query data, Megakaryocytes do not appear as a separate cluster anymore. This is an extreme case as the reference data is very small; but one should always question if the reference data contain enough biological variation to meaningfully accomodate query data.

Using BBKNN

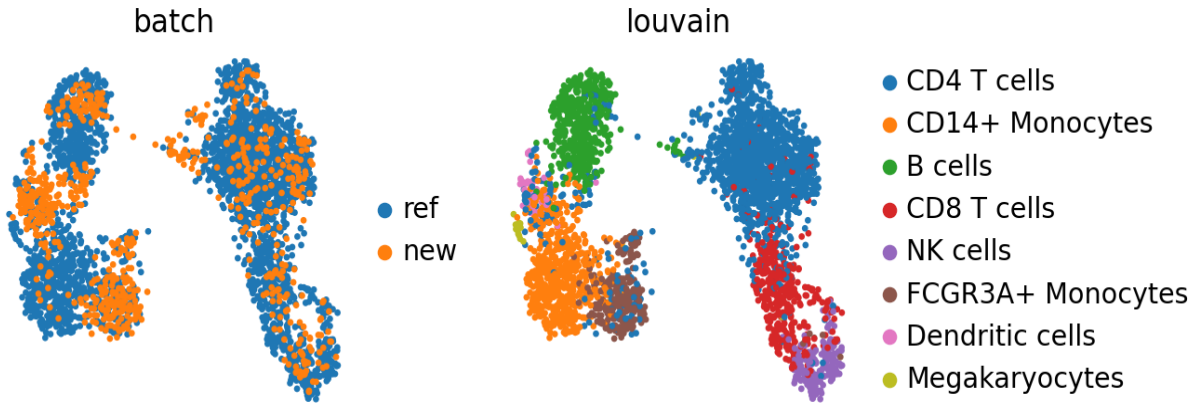
[13]:
```
sc.tl.pca(adata_concat)
```

[14]:
```
%%time
sc.external.pp.bbknn(adata_concat, batch_key='batch')  # running bbknn 1.3.6
```

```
CPU times: user 1.67 s, sys: 749 ms, total: 2.42 s
Wall time: 324 ms
```

[15]:
```
sc.tl.umap(adata_concat)
```

[16]:
```
sc.pl.umap(adata_concat, color=['batch', 'louvain'])
```



Also BBKNN doesn't maintain the Megakaryocytes cluster. However, it seems to mix cells more homogeneously.

Pancreas

The following data has been used in the scGen paper [Lotfollahi19], has been used here, was curated here and can be downloaded from here (the BBKNN paper).

It contains data for human pancreas from 4 different studies (Segerstolpe16, Baron16, Wang16, Muraro16), which have been used in the seminal papers on single-cell dataset integration (Butler18, Haghverdi18) and many times ever since.

[17]:
```
# note that this collection of batches is already intersected on the genes
adata_all = sc.read('data/pancreas.h5ad',
backup_url='https://www.dropbox.com/s/qj1jlm9w10wmt0u/pancreas.h5ad?dl=1')
```

[18]:
```
adata_all.shape
```

[18]:
```
(14693, 2448)
```

Inspect the cell types observed in these studies.

[19]:
```
counts = adata_all.obs.celltype.value_counts()
counts
```

[19]:
```
alpha                    4214
beta                     3354
ductal                   1804
acinar                   1368
not applicable           1154
delta                     917
gamma                     571
endothelial               289
activated_stellate        284
dropped                   178
quiescent_stellate        173
mesenchymal                80
macrophage                 55
PSC                        54
unclassified endocrine     41
co-expression              39
mast                       32
epsilon                    28
mesenchyme                 27
schwann                    13
t_cell                      7
MHC class II                5
unclear                     4
unclassified                2
Name: celltype, dtype: int64
```

To simplify visualization, let's remove the 5 minority classes.

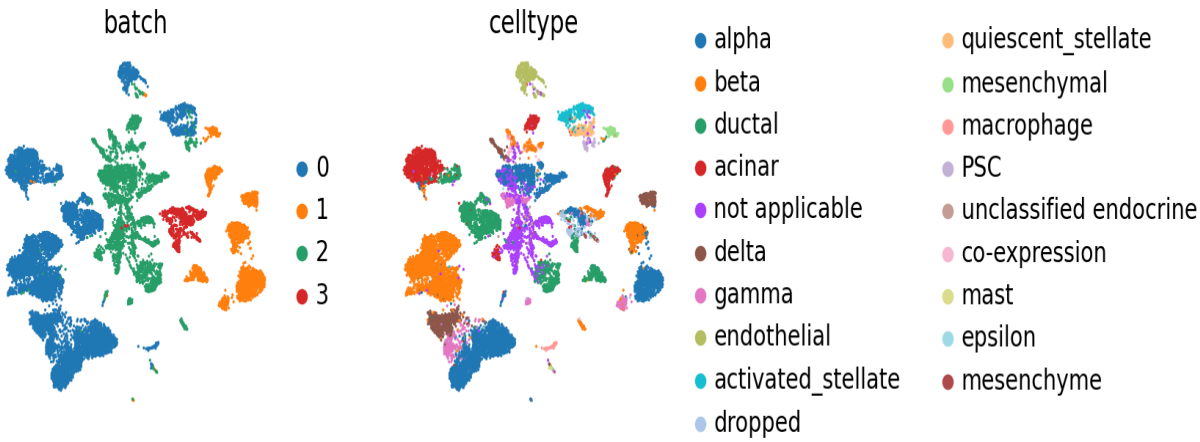
[20]:
```
minority_classes = counts.index[-5:].tolist()        # get the minority classes
adata_all = adata_all[                                # actually subset
    ~adata_all.obs.celltype.isin(minority_classes)]
adata_all.obs.celltype.cat.reorder_categories(        # reorder according to abundance
    counts.index[:-5].tolist(), inplace=True)
```

Seeing the batch effect

[21]:
```
sc.pp.pca(adata_all)
sc.pp.neighbors(adata_all)
sc.tl.umap(adata_all)
```

We observe a batch effect.

[22]:
```
sc.pl.umap(adata_all, color=['batch', 'celltype'],
palette=sc.pl.palettes.vega_20_scanpy)
```



BBKNN

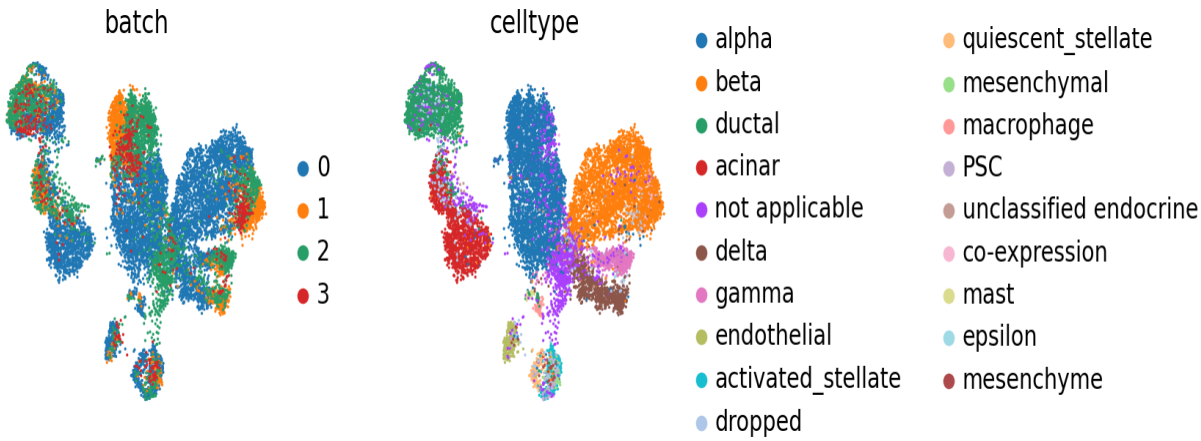
It can be well-resolved using BBKNN [Polanski19].

[23]:
```
%%time
sc.external.pp.bbknn(adata_all, batch_key='batch')
```

```
CPU times: user 1.89 s, sys: 810 µs, total: 1.9 s
Wall time: 1.89 s
```

[24]:
```
sc.tl.umap(adata_all)
```

[25]:
```
sc.pl.umap(adata_all, color=['batch', 'celltype'])
```



If one prefers to work more iteratively starting from one reference dataset, one can use ingest.

Mapping onto a reference batch using ingest

Choose one reference batch for training the model and setting up the neighborhood graph (here, a PCA) and separate out all other batches.

As before, the model trained on the reference batch will explain the biological variation observed within it.

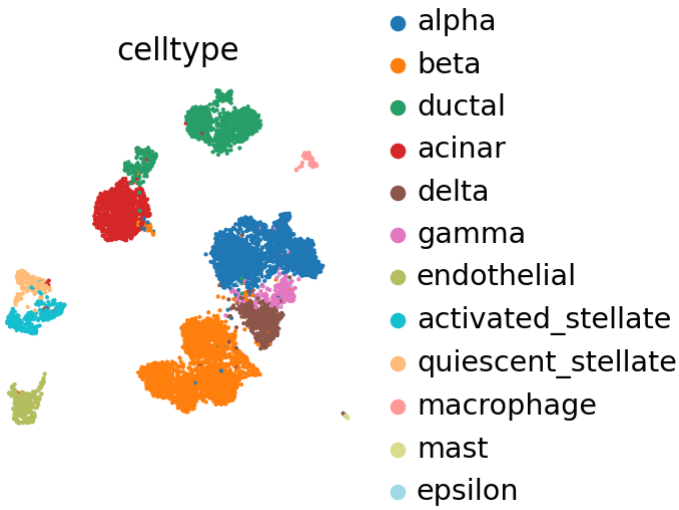
[26]:
```
adata_ref = adata_all[adata_all.obs.batch == '0']
```

Compute the PCA, neighbors and UMAP on the reference data.

[27]:
```
sc.pp.pca(adata_ref)
sc.pp.neighbors(adata_ref)
sc.tl.umap(adata_ref)
```

The reference batch contains 12 of the 19 cell types across all batches.

[28]:
```
sc.pl.umap(adata_ref, color='celltype')
```



Iteratively map labels (such as 'celltype') and embeddings (such as 'X_pca' and 'X_umap') from the reference data onto the query batches.

[29]:
```
adatas = [adata_all[adata_all.obs.batch == i].copy() for i in ['1', '2', '3']]
```

[30]:
```
sc.settings.verbosity = 2  # a bit more logging
for iadata, adata in enumerate(adatas):
    print(f'... integrating batch {iadata+1}')
    adata.obs['celltype_orig'] = adata.obs.celltype  # save the original cell type
    sc.tl.ingest(adata, adata_ref, obs='celltype')
```

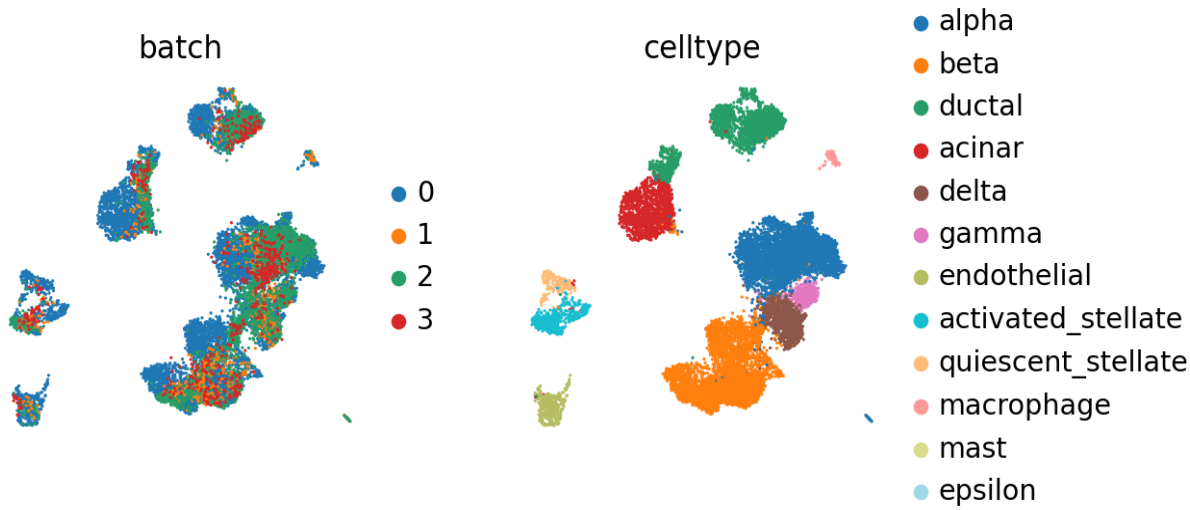
```
... integrating batch 1
running ingest
    finished (0:00:06)
... integrating batch 2
running ingest
    finished (0:00:07)
... integrating batch 3
running ingest
    finished (0:00:03)
```

Each of the query batches now carries annotation that has been contextualized with `adata_ref`. By concatenating, we can view it together.

[31]:
```
adata_concat = adata_ref.concatenate(adatas)
```

[32]:
```
adata_concat.obs.celltype = adata_concat.obs.celltype.astype('category')
adata_concat.obs.celltype.cat.reorder_categories(adata_ref.obs.celltype.cat.categorie
inplace=True)  # fix category ordering
adata_concat.uns['celltype_colors'] = adata_ref.uns['celltype_colors']  # fix
category coloring
```

[33]:
```
sc.pl.umap(adata_concat, color=['batch', 'celltype'])
```



Compared to the BBKNN result, this is maintained clusters in a much more pronounced fashion. If one already observed a desired continuous structure (as in the hematopoietic datasets, for instance), `ingest` allows to easily maintain this structure.

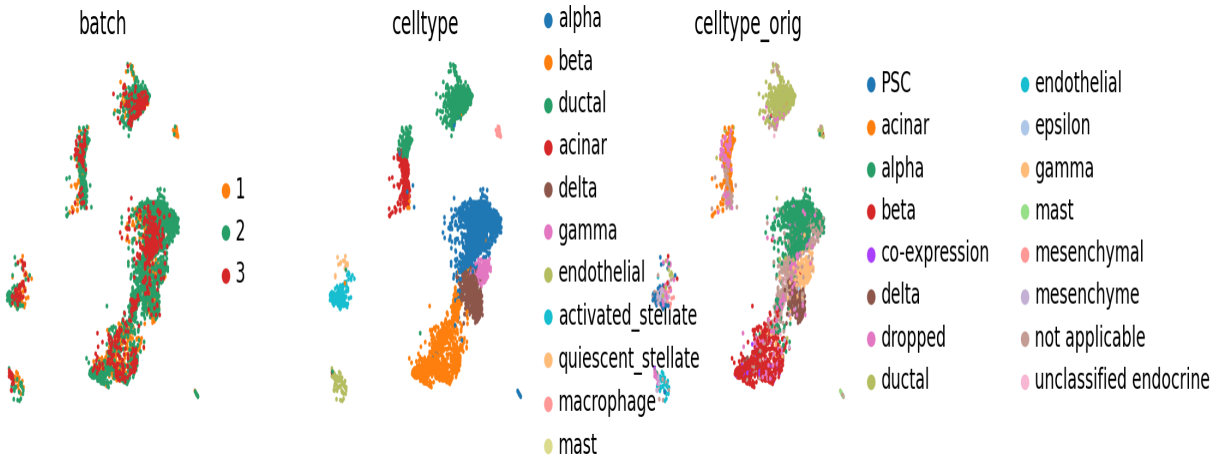
Evaluating consistency

Let us subset the data to the query batches.

[34]:
```
adata_query = adata_concat[adata_concat.obs.batch.isin(['1', '2', '3'])]
```

The following plot is a bit hard to read, hence, move on to confusion matrices below.

[35]:
```
sc.pl.umap(
    adata_query, color=['batch', 'celltype', 'celltype_orig'], wspace=0.4)
```



Cell types conserved across batches

Let us first focus on cell types that are conserved with the reference, to simplify reading of the confusion matrix.

[36]:
```
obs_query = adata_query.obs
conserved_categories =
obs_query.celltype.cat.categories.intersection(obs_query.celltype_orig.cat.categories
# intersected categories
obs_query_conserved = obs_query.loc[obs_query.celltype.isin(conserved_categories) &
obs_query.celltype_orig.isin(conserved_categories)]  # intersect categories
obs_query_conserved.celltype.cat.remove_unused_categories(inplace=True)  # remove
unused categoriyes
obs_query_conserved.celltype_orig.cat.remove_unused_categories(inplace=True)  #
remove unused categoriyes
obs_query_conserved.celltype_orig.cat.reorder_categories(obs_query_conserved.celltype
inplace=True)  # fix category ordering
```

[37]:
```
pd.crosstab(obs_query_conserved.celltype, obs_query_conserved.celltype_orig)
```

[37]:

celltype_orig celltype	alpha	beta	ductal	acinar	delta	gamma	endothelial	mast
alpha	1819	3	6	0	1	25	0	6
beta	49	804	4	1	10	21	0	0
ductal	7	5	692	240	0	0	0	0
acinar	2	3	3	168	0	3	0	0
delta	5	4	0	0	305	73	0	0
gamma	1	5	0	0	0	194	0	0
endothelial	2	0	0	0	0	0	36	0
mast	0	0	1	0	0	0	0	1

Overall, the conserved cell types are also mapped as expected. The main exception are some acinar cells in the original annotation that appear as acinar cells. However, already the reference data is observed to feature a cluster of both acinar and ductal cells, which explains the discrepancy, and indicates a potential inconsistency in the initial annotation.

All cell types

Let us now move on to look at all cell types.

[38]:
```
pd.crosstab(adata_query.obs.celltype, adata_query.obs.celltype_orig)
```

[38]:

celltype_orig celltype	PSC	acinar	alpha	beta	co-expression	delta	dropped	ductal	endothelial	epsilon	ga
alpha	0	0	1819	3	2	1	36	6	0	4	
beta	0	1	49	804	35	10	42	4	0	0	
ductal	0	240	7	5	0	0	38	692	0	0	
acinar	0	168	2	3	0	0	25	3	0	0	
delta	0	0	5	4	1	305	13	0	0	4	
gamma	0	0	1	5	0	0	1	0	0	2	
endothelial	1	0	2	0	1	0	7	0	36	0	
activated_stellate	49	1	1	3	0	0	11	8	0	0	

celltype_orig celltype	PSC	acinar	alpha	beta	co-expression	delta	dropped	ductal	endothelial	epsilon	ga
quiescent_stellate	4	0	1	1	0	0	5	1	1	0	
macrophage	0	0	1	1	0	0	0	12	0	0	
mast	0	0	0	0	0	0	0	1	0	0	

We observe that PSC (pancreatic stellate cells) cells are in fact just inconsistently annotated and correctly mapped on 'activated_stellate' cells.

Also, it's nice to see that 'mesenchyme' and 'mesenchymal' cells both map onto the same category. However, that category is again 'activated_stellate' and likely incorrect.

Visualizing distributions across batches

Often, batches correspond to experiments that one wants to compare. Scanpy offers to convenient visualization possibilities for this.

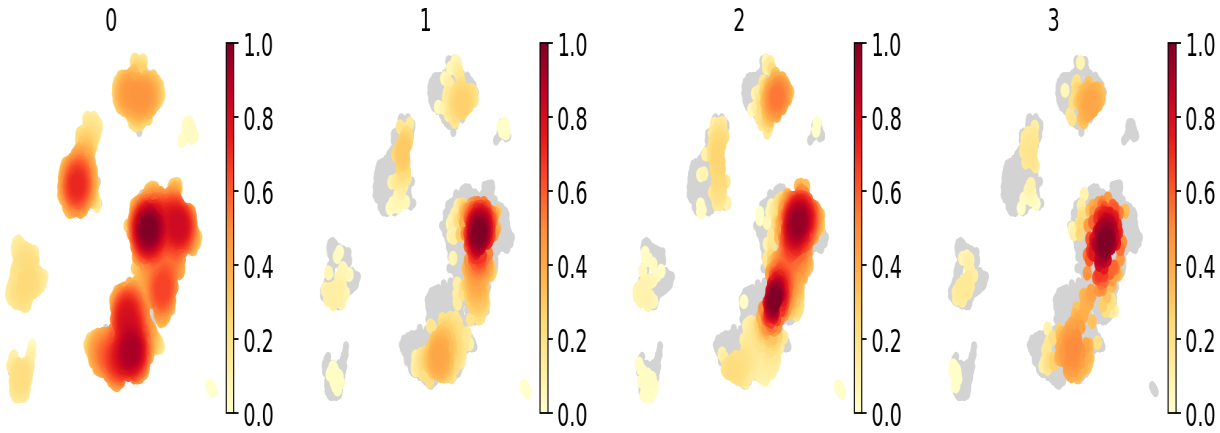
1. a density plot
2. a partial visualization of a subset of categories/groups in an emnbedding

Density plot

[39]:
```
sc.tl.embedding_density(adata_concat, groupby='batch')
```

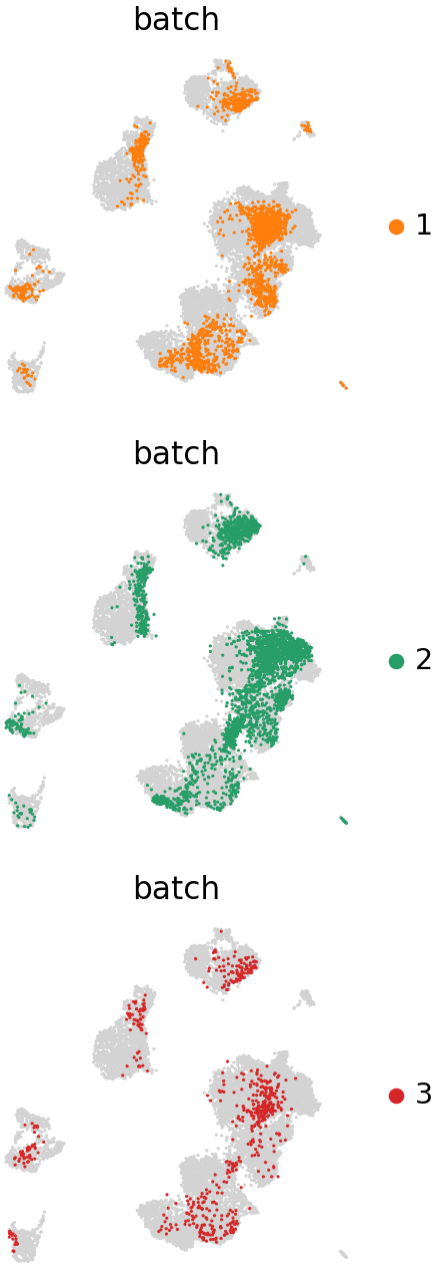
```
computing density on 'umap'
```

[40]:
```
sc.pl.embedding_density(adata_concat, groupby='batch')
```



Partial visualizaton of a subset of groups in embedding

[41]:
```
for batch in ['1', '2', '3']:
    sc.pl.umap(adata_concat, color='batch', groups=[batch])
```


batch
1
batch
2
batch
3