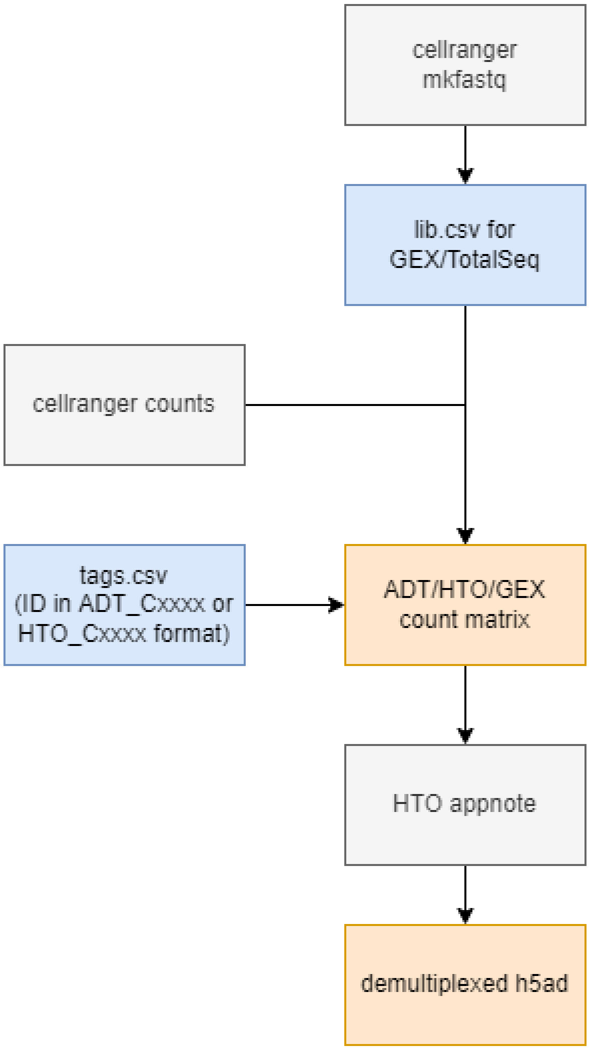
process overview



Generating FASTQs with cellranger mkfastq

The detailed tutorial for generating fastqs with cellranger mkfastq is avaialbe at
Generating FASTQs with cellranger mkfastq
below is an example of mkfastq command:

```
cellranger mkfastq --id=GEX \
                   --run=/path/to/bcl \
                   --samplesheet=GEX_samplesheet.csv
```

Generating count matrix for GEX and TS with cellranger count

a library csv file for fastqs generated the the mkfastq and a tags csv for detailed inforamation of stained markers are required
We recommend users specify the 4-digit TotalSeq ID of markers used in the tags.csv with the format of *HTO_Cxxxx* or *ADT_Cxxxx* for hashtags and antibodies, respectively.
Below is an example command of running cellranger counts:

```
cellranger count --id=GEX \
                 --expect-cells=5000 \
                 --feature-ref=tags.csv \
                 --libraries=libs.csv \
                 --transcriptome=GRCh38-2020-A
```

Dataprocessing

Next we will move on to the data processing part
The count matrix and the contig annotations from cellranger runs are required.

basic setup

import packages

In [1]:

```
%matplotlib inline
import os
import anndata
import matplotlib
import numpy as np
import scanpy as sc
import matplotlib.pyplot as plt
import demultiplexing as demux
```

set up sample name and io path

In [2]:

```
# sample name and cellranger output path
sample = 'TsA_wRNA'
matrix_input = f'{sample}/lane_new/outs'
```

read data
We will use scanpy to read, write and process the data.
Scanpy takes both mtx and h5 format.
For faster reading spead, we will read the filtered h5 file into an anndata object in this demo

In [3]:

```
ann = sc.read_10x_h5(os.path.join(matrix_input,'filtered_feature_bc_matrix.h5'),
gex_only=False)
```

```
Variable names are not unique. To make them unique, call `.var names make unique`.
```

according to the warning message from scanpy, there are duplicate variable names
we need to correct the variable names with the function mentioned

In [4]:

```
ann.var_names_make_unique()
```

In [5]:

```
# check the variable metadata
ann.var
```

Out[5]:

	gene_ids	feature_types	genome
MIR1302-2HG	ENSG00000243485	Gene Expression	GRCh38

FAM138A	ENSG00000237613	Gene Expression	GRCh38
OR4F5	ENSG00000186092	Gene Expression	GRCh38
AL627309.1	ENSG00000238009	Gene Expression	GRCh38
AL627309.3	ENSG00000239945	Gene Expression	GRCh38
...	...	...	...
TsA_A0260	HTO_A0260	Antibody Capture	
TsA_A0262	HTO_A0262	Antibody Capture	
TsA_A0263	HTO_A0263	Antibody Capture	
TsA_A0264	HTO_A0264	Antibody Capture	
TsA_A0265	HTO_A0265	Antibody Capture	

36624 rows × 3 columns

demultiplexing

We will use two ways to dmultiplex the dataset:
demuxEM and demultiplexing based on threshold calling

demultiplexing with demuxEM

In [6]:

```
demux.get_demux(ann, method='demuxEM')
```

```
2021-09-03 15:35:46,562 - demuxEM.tools.demuxEM - INFO - Background probability distribution
 is estimated.
2021-09-03 15:36:14,108 - demuxEM.tools.demuxEM - INFO - Demultiplexing is done.
```

The get_demux function will add a column named ***assignment*** to the obs dataframe
Here we use value_counts() for series to quick check counts of each hashtag assginment:

In [7]:

```
ann.obs['assignment'].value_counts()
```

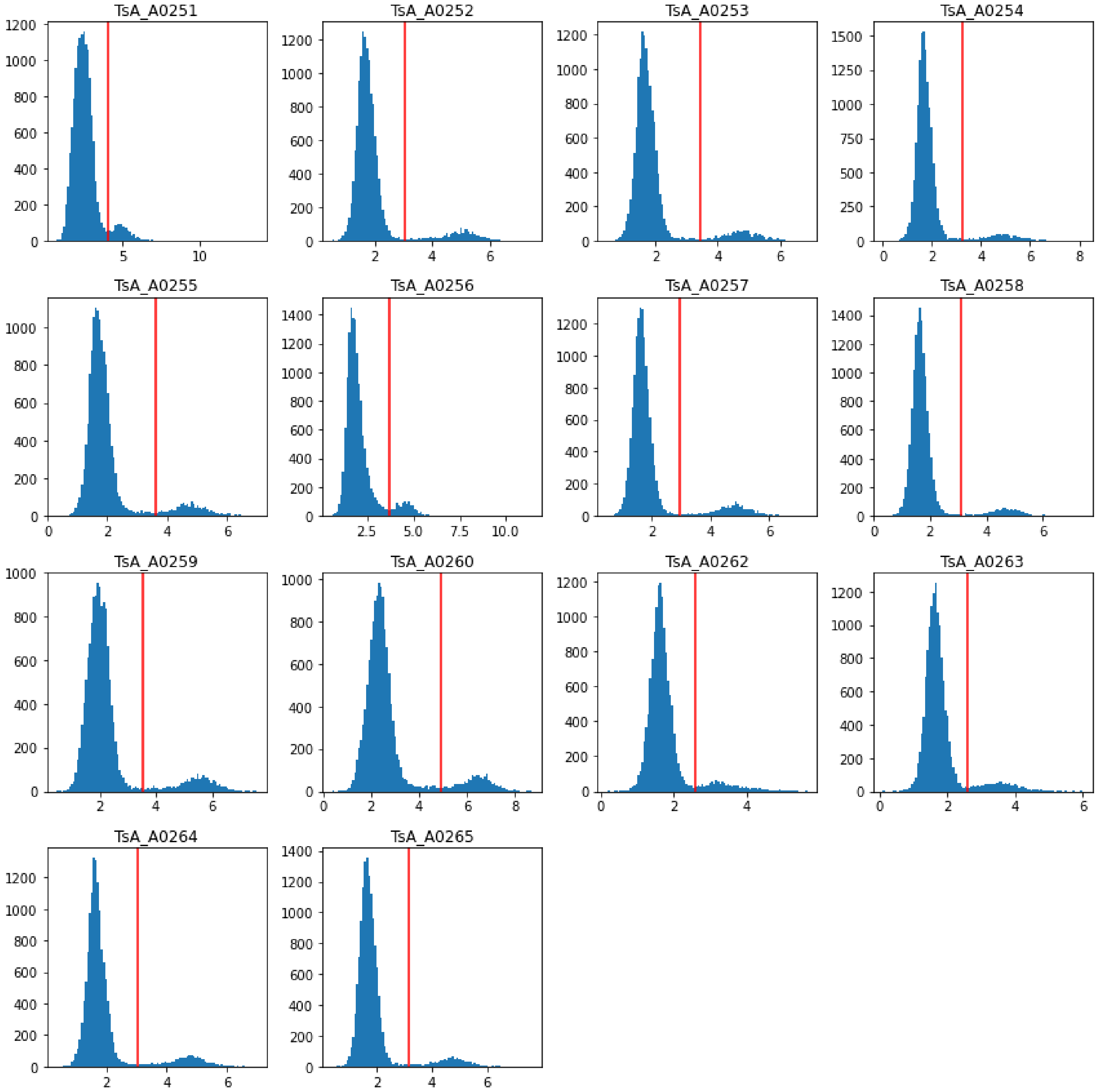
Out[7]:

```
Doublet      4832
TsA_A0260    1057
TsA_A0259     910
TsA_A0257     805
TsA_A0264     729
TsA_A0255     705
TsA_A0252     667
Negative      666
TsA_A0253     654
TsA_A0265     590
TsA_A0251     586
TsA_A0256     551
TsA_A0254     508
TsA_A0258     500
TsA_A0263     228
TsA_A0262     181
Name: assignment, dtype: int64
```

demultiplexing with threshold calling

In [8]:

```
demux.get_demux(ann, method='threshold')
```



In [9]:

```
ann.obs['assignment'].value_counts()
```

Out[9]:

```
Doublets     2711
TsA_A0257     951
TsA_A0264     915
TsA_A0259     908
TsA_A0260     835
TsA_A0255     831
TsA_A0262     785
TsA_A0253     777
TsA_A0263     775
TsA_A0252     773
TsA_A0265     754
TsA_A0256     695
Negative      641
TsA_A0258     628
TsA_A0251     608
TsA_A0254     582
Name: assignment, dtype: int64
```

manually adjust threshold

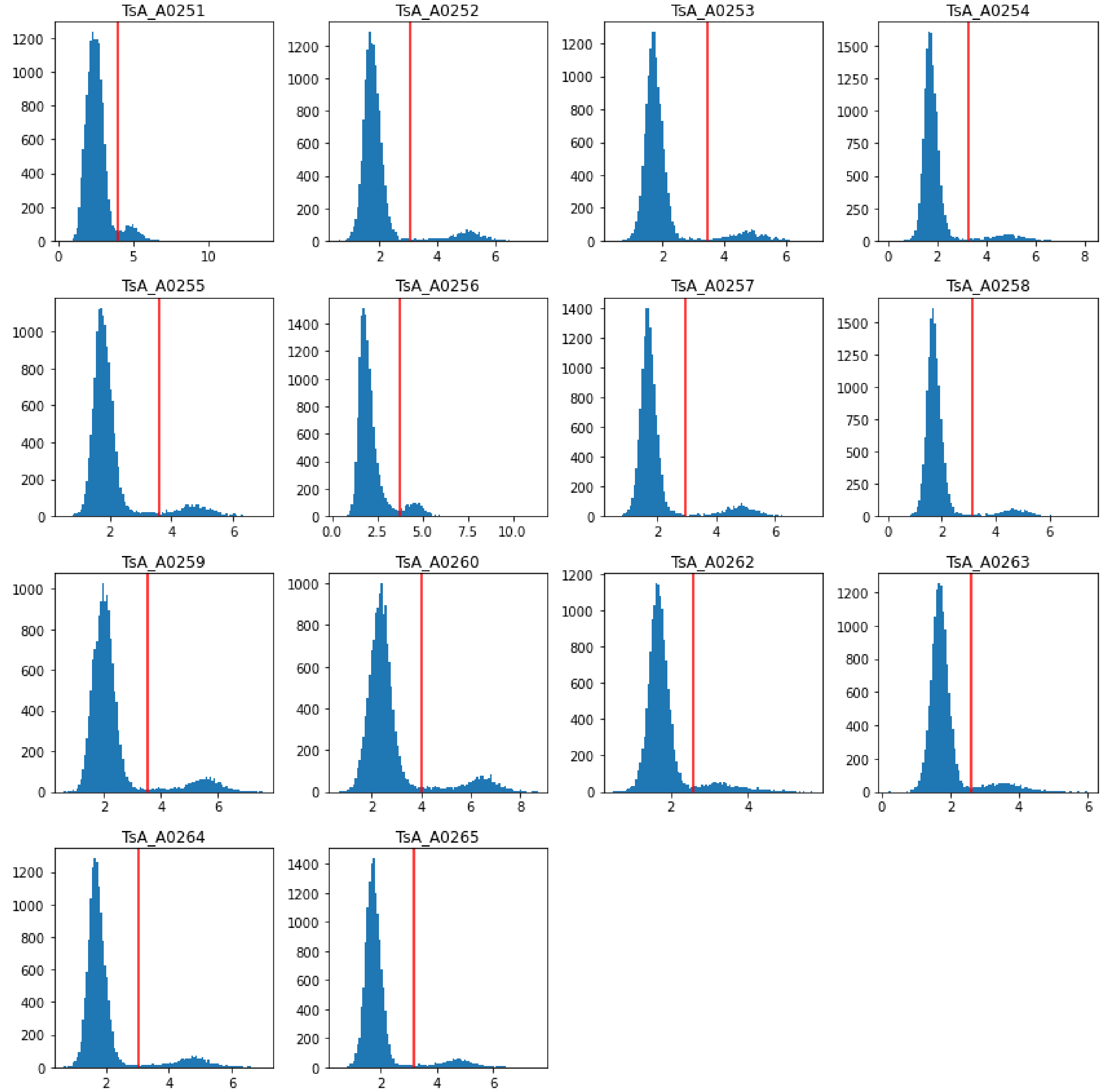
sometimes threshold calling results might be off.
we need to manually correct those thresholds.
The threshold calling results are sotred in **ann.var['thre']**

In [10]:

```
# setup manually assigned threshold in this section
ann.var.loc['TsA_A0260','thre'] = 4
```

In [11]:

```
demux.get_demux(ann, method='update')
```



In [12]:

```
ann.obs['assignment'].value_counts()
```

Out[12]:

```
Doublets      2877
TsA_A0257      935
TsA_A0264      901
TsA_A0259      893
TsA_A0260      855
TsA_A0255      816
TsA_A0262      779
```

```
TsA_A0253     767
TsA_A0252     757
TsA_A0263     757
TsA_A0265     748
TsA_A0256     687
TsA_A0258     621
Negative      608
TsA_A0251     594
TsA_A0254     574
Name: assignment, dtype: int64
```

visualize demultiplexing results

normalization

we use asinh transformed normalization result for further analysis including umap calculation
This method add jittering noise to the expression before asinh transform to get flow like peak distributions
HTO markers are annotated with ids starting with 'HTO'

In [13]:

```
ann_hto = ann[:,['HTO' in t for t in ann.var['gene_ids']]].copy()
demux.asinh_trans(ann_hto)
```

generate HTO ridge plots and umaps

with the normalized data, we can now calcuate umap based on HTO expression level

In [14]:

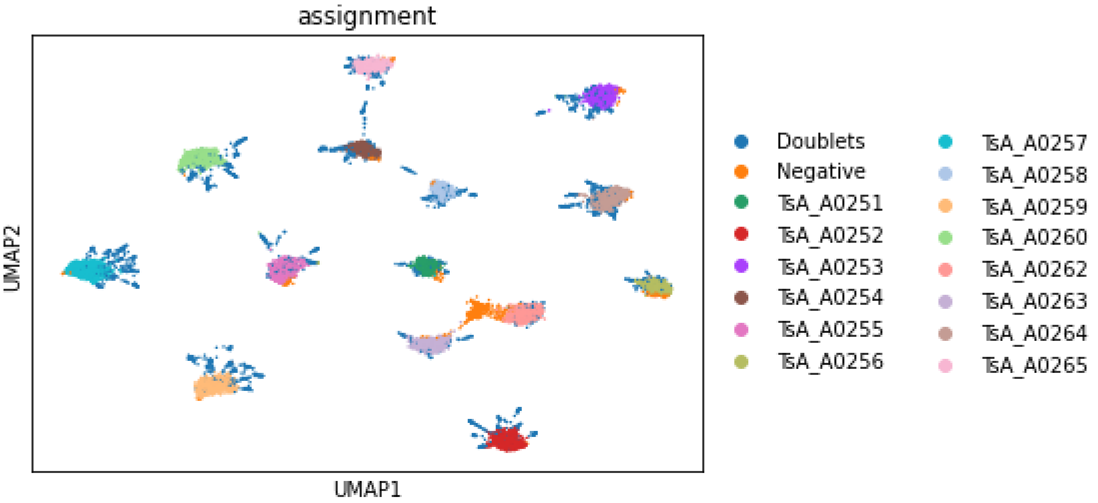
```
# calculate HTO umap and attach it to raw data matrix as metadata
sc.pp.neighbors(ann_hto, n_neighbors=40, use_rep='X')
sc.tl.umap(ann_hto,min_dist=0.1)
```

plot demultplexed result on HTO umaps

In [15]:

```
sc.pl.umap(ann_hto,color='assignment')
```

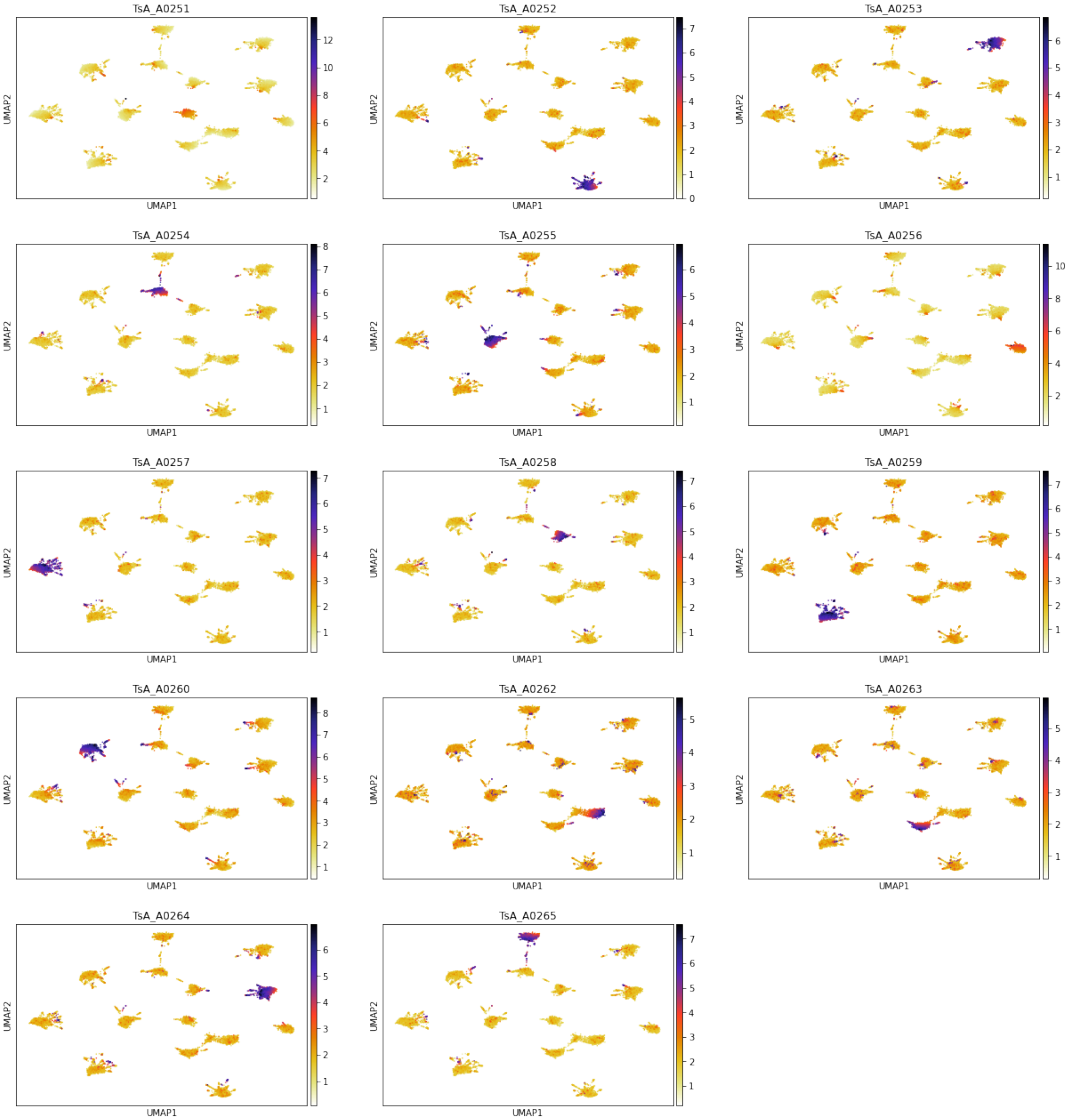
```
... storing 'assignment' as categorical
... storing 'feature_types' as categorical
... storing 'genome' as categorical
```



The umap shows that clusgers of each hashtags are well separated, which indicates good distribution shape of hashtags expression level of each hashtag is also plotted below

In [16]:

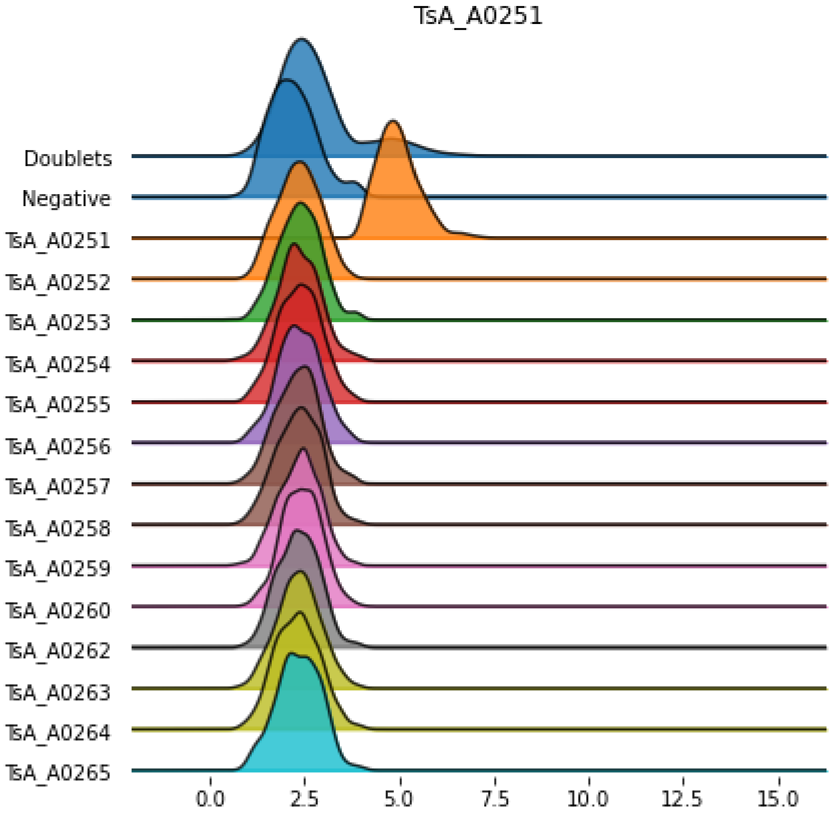
```
sc.pl.umap(ann_hto,color=ann_hto.var_names, color_map='CMRmap_r', ncols=3)
```

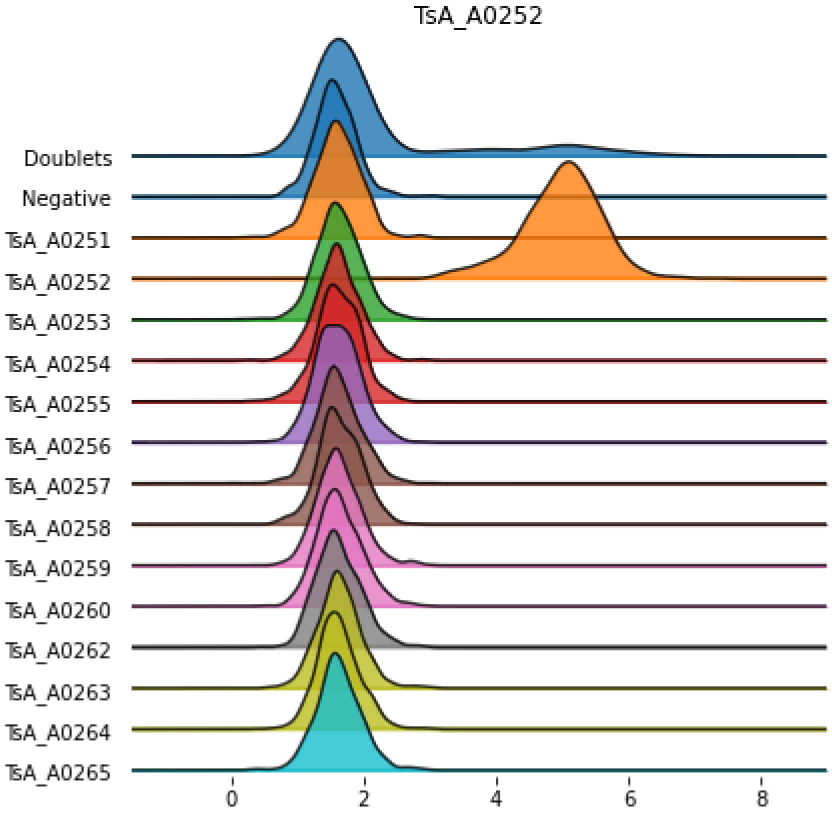


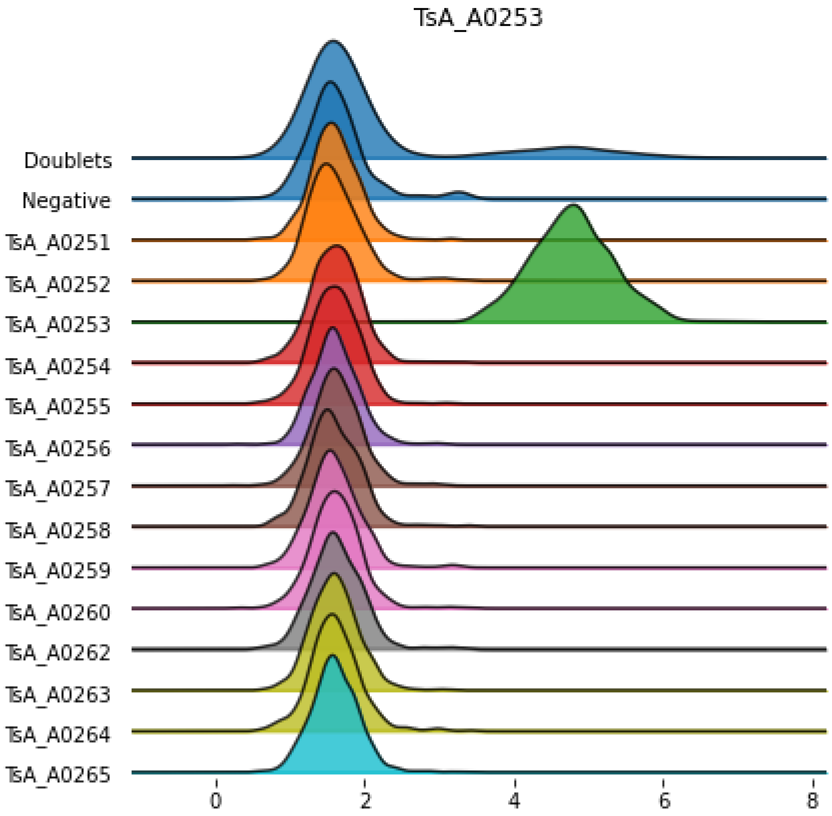
ridge plot will also show the expression level difference between different cell assignment
doublets will have positive counts for multiple hashtags, while negative cells will remain negative for all hashtags

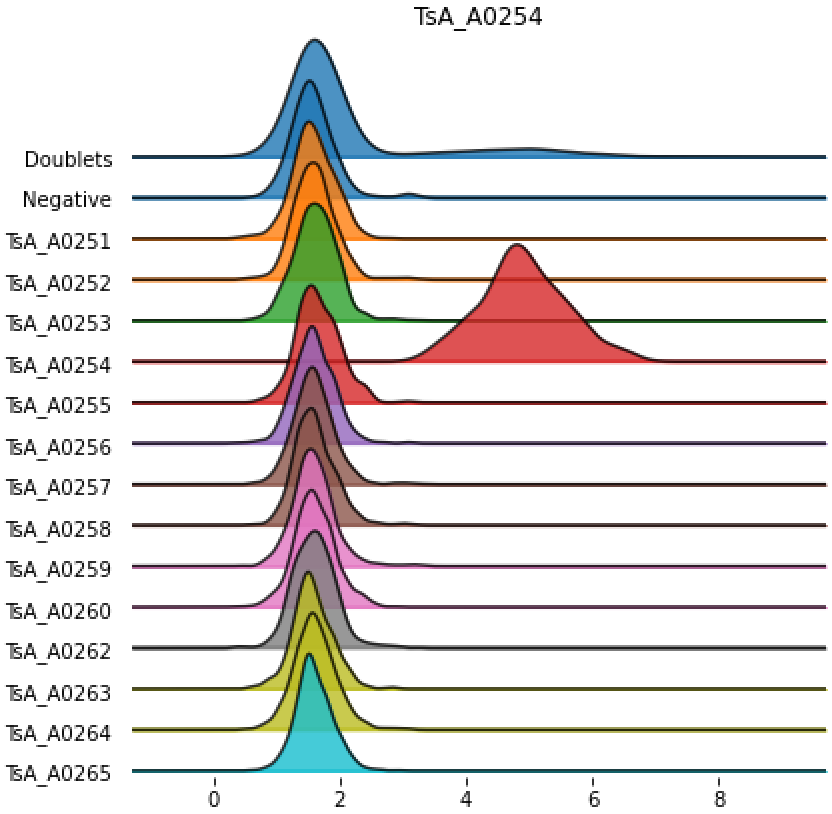
In [23]:

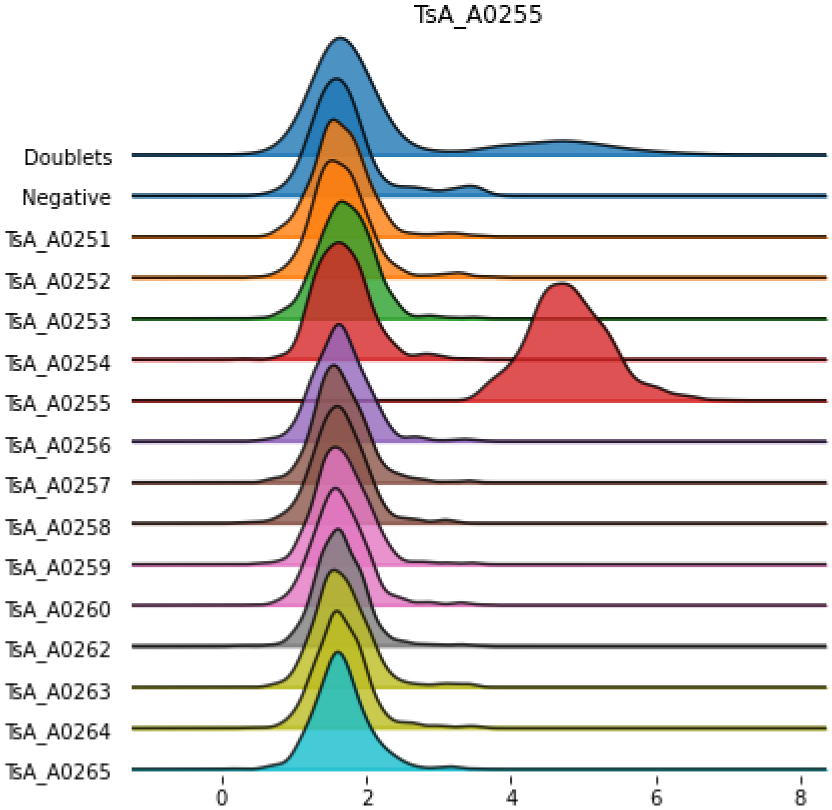
```
demux.get_ridge(ann_hto)
```

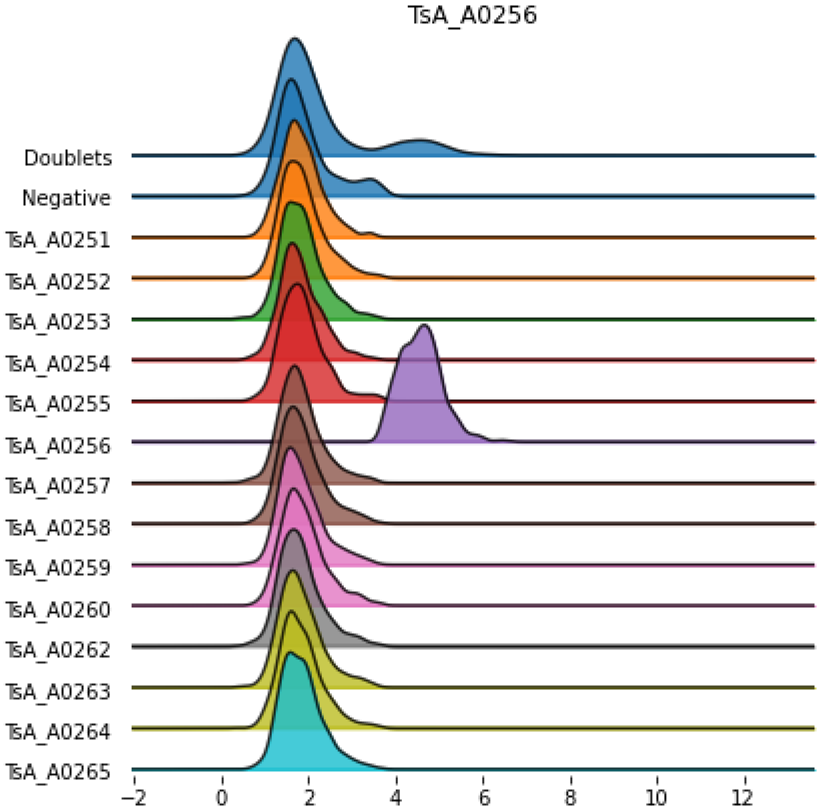


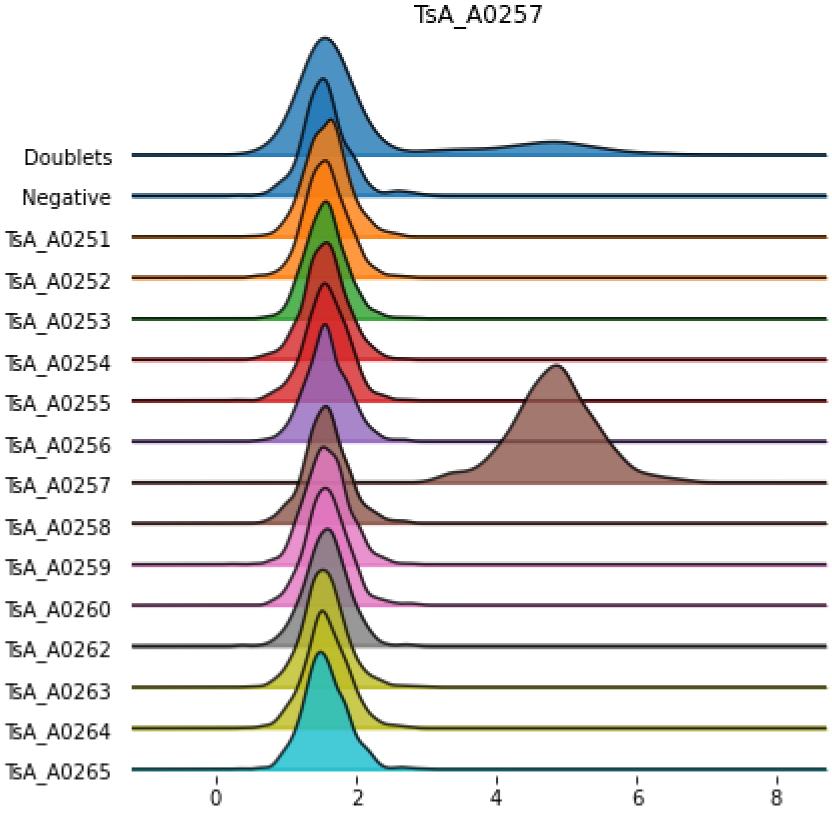


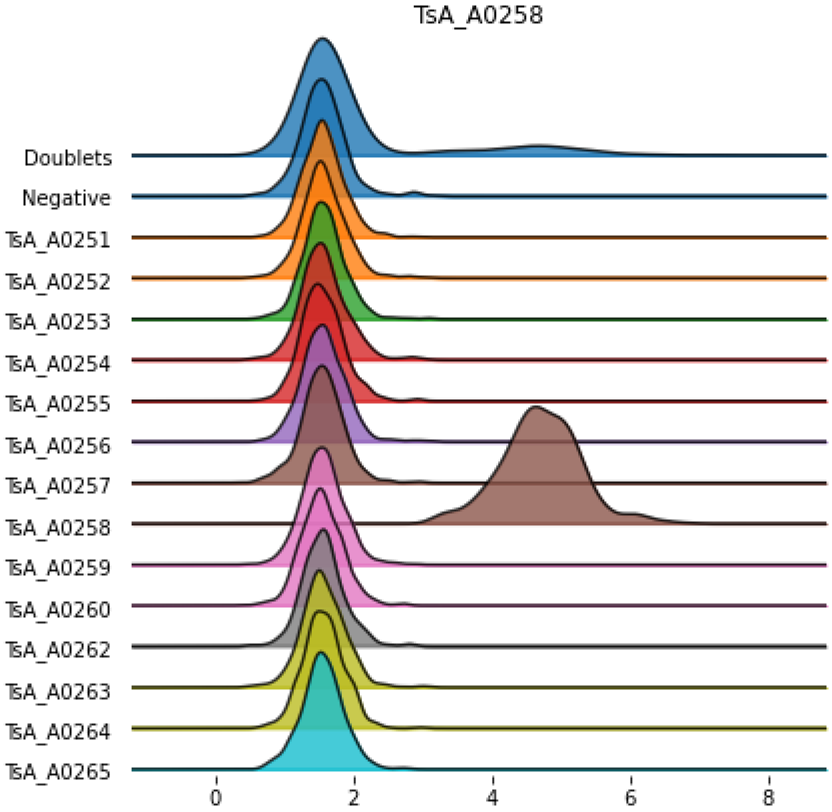

TsA_A0253
Doublets
Negative
TsA_A0251
TsA_A0252
TsA_A0253
TsA_A0254
TsA_A0255
TsA_A0256
TsA_A0257
TsA_A0258
TsA_A0259
TsA_A0260
TsA_A0262
TsA_A0263
TsA_A0264
TsA_A0265
0
2
4
6
8

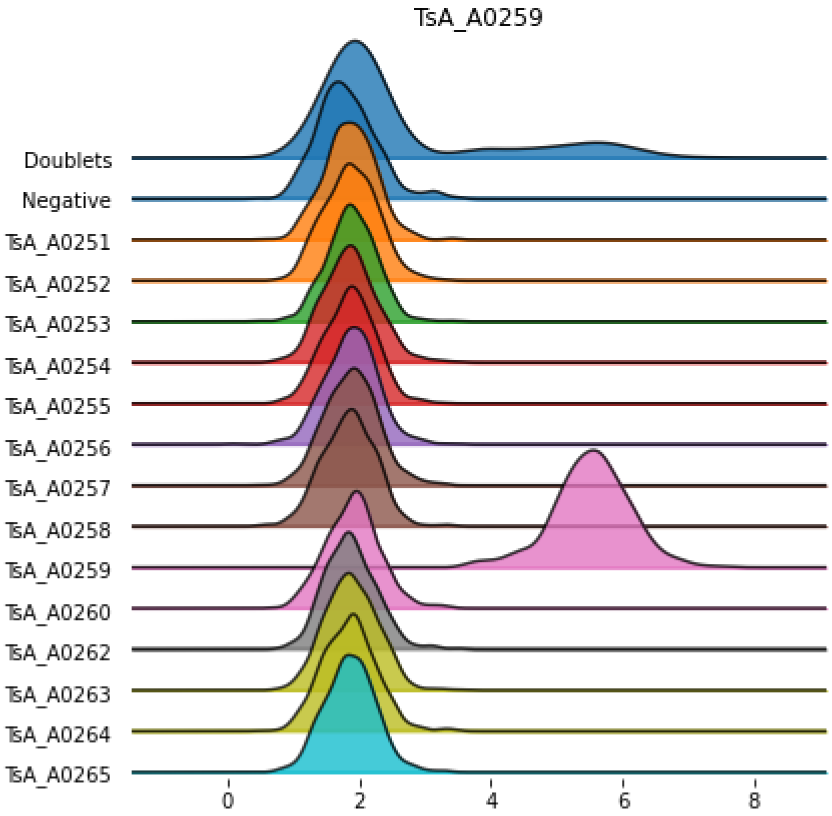

TsA_A0254
Doublets
Negative
TsA_A0251
TsA_A0252
TsA_A0253
TsA_A0254
TsA_A0255
TsA_A0256
TsA_A0257
TsA_A0258
TsA_A0259
TsA_A0260
TsA_A0262
TsA_A0263
TsA_A0264
TsA_A0265
0
2
4
6
8

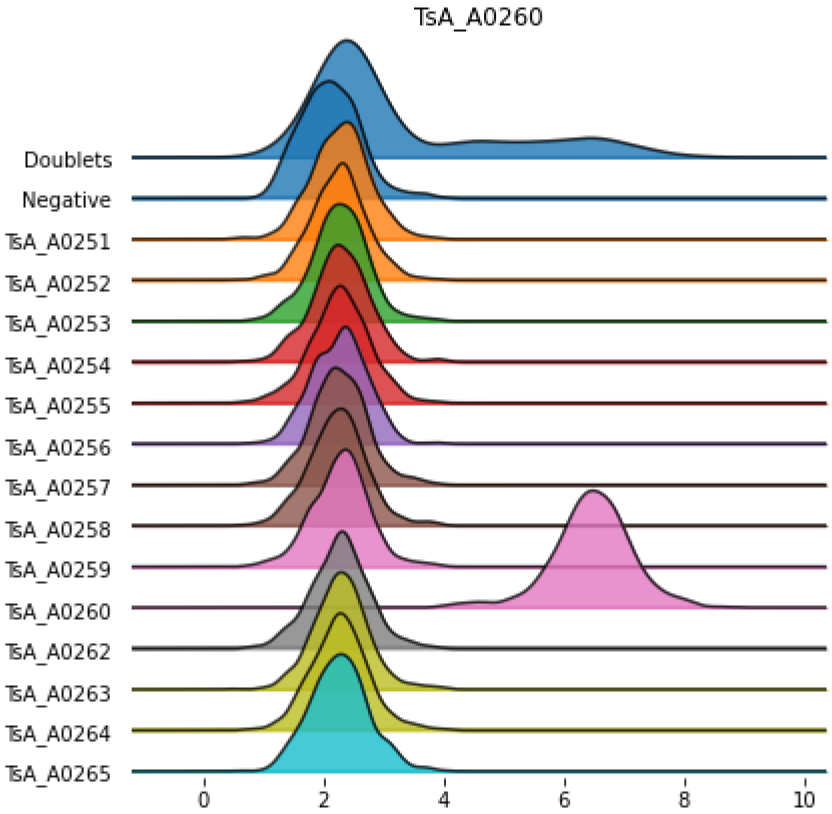

TsA_A0255

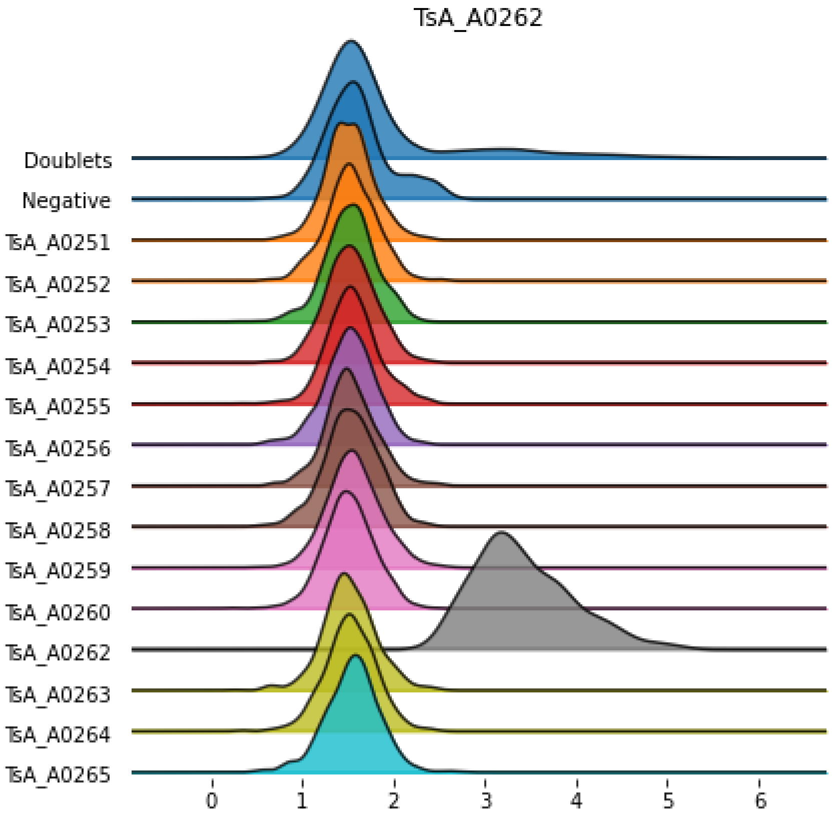

TsA_A0256

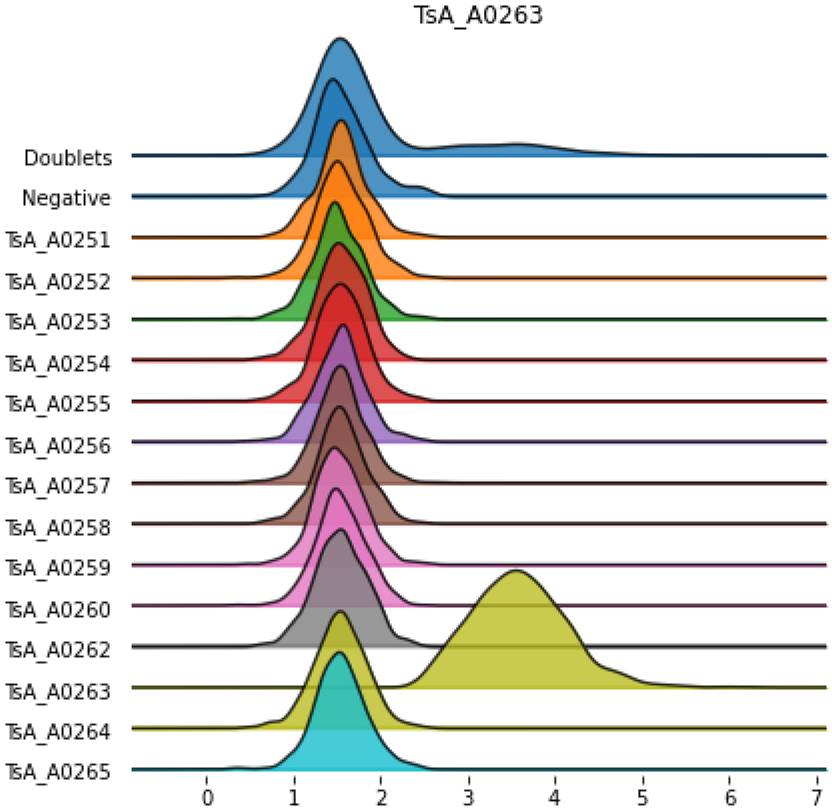

TsA_A0257
Doublets
Negative
TsA_A0251
TsA_A0252
TsA_A0253
TsA_A0254
TsA_A0255
TsA_A0256
TsA_A0257
TsA_A0258
TsA_A0259
TsA_A0260
TsA_A0262
TsA_A0263
TsA_A0264
TsA_A0265
0
2
4
6
8

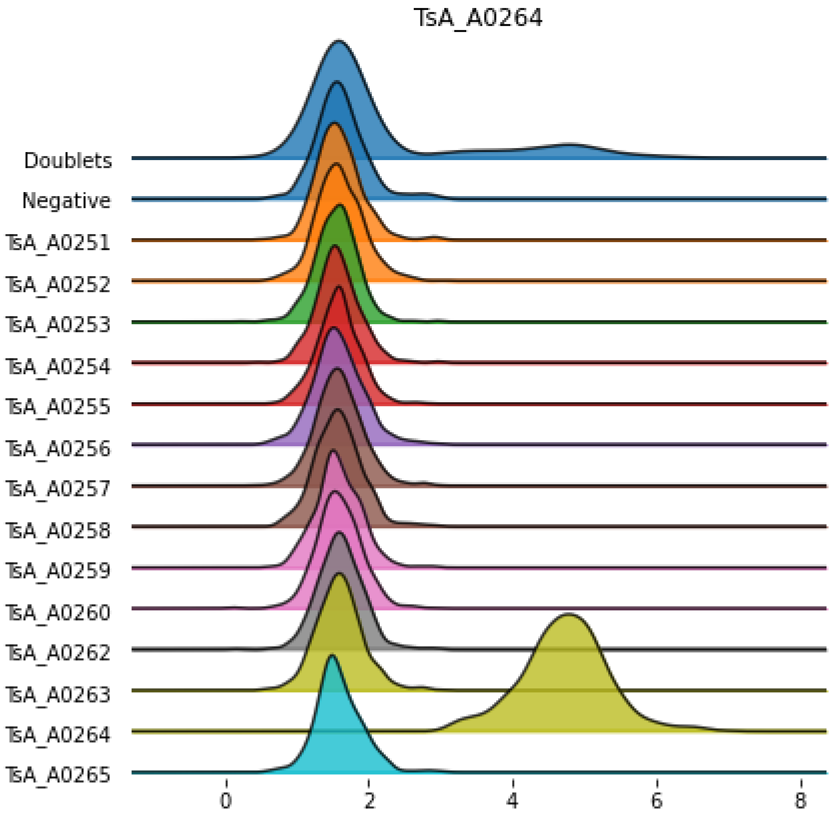

TsA_A0258
Doublets
Negative
TsA_A0251
TsA_A0252
TsA_A0253
TsA_A0254
TsA_A0255
TsA_A0256
TsA_A0257
TsA_A0258
TsA_A0259
TsA_A0260
TsA_A0262
TsA_A0263
TsA_A0264
TsA_A0265
0
2
4
6
8


TsA_A0259
Doublets
Negative
TsA_A0251
TsA_A0252
TsA_A0253
TsA_A0254
TsA_A0255
TsA_A0256
TsA_A0257
TsA_A0258
TsA_A0259
TsA_A0260
TsA_A0262
TsA_A0263
TsA_A0264
TsA_A0265
0
2
4
6
8

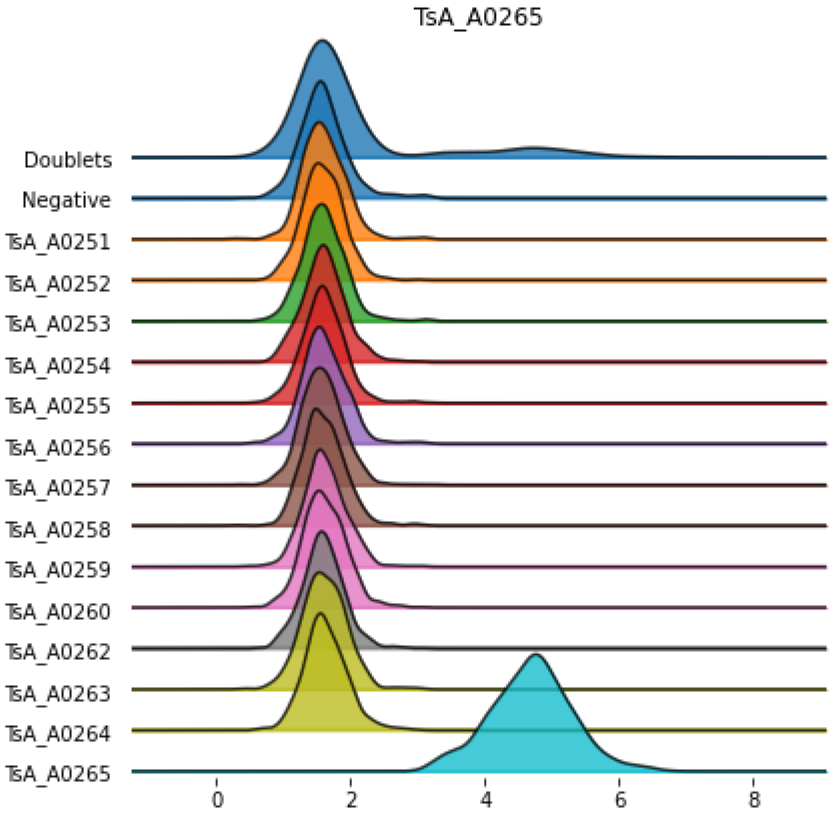

TsA_A0260
Doublets
Negative
TsA_A0251
TsA_A0252
TsA_A0253
TsA_A0254
TsA_A0255
TsA_A0256
TsA_A0257
TsA_A0258
TsA_A0259
TsA_A0260
TsA_A0262
TsA_A0263
TsA_A0264
TsA_A0265
0
2
4
6
8
10


TsA_A0262


TsA_A0263



TsA_A0264

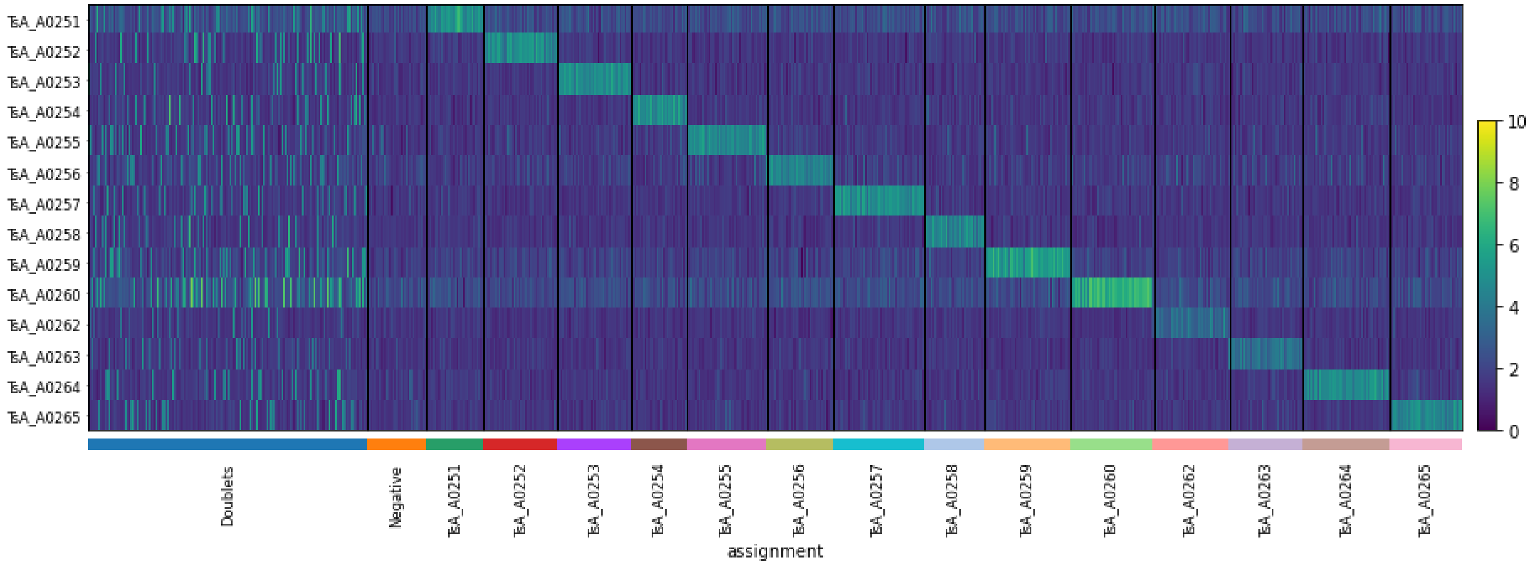


TsA_A0265

heatmap of expression level:
A0260, A0259, and A0251 have relatively higher background according to the heatmap

In [24]:

```
sc.pl.heatmap(ann_hto,groupby='assignment',
                var_names=sorted(ann_hto.var_names),
                swap_axes=True, vmax=10,figsize=(15,5))
```



filtering out doublets and negatives

the doublets and negatives are filteres out for further analysis, including umap based on ADT expression level like HTO markers, ADT markers are annotated with ids starting with 'ADT'

In [25]:

```
# filter out doublets and negatives
ann_single = ann[ann.obs['assignment']!='Negative',:]
ann_single = ann_single[ann_single.obs['assignment']!='Doublet',:].copy()
```

In [26]:

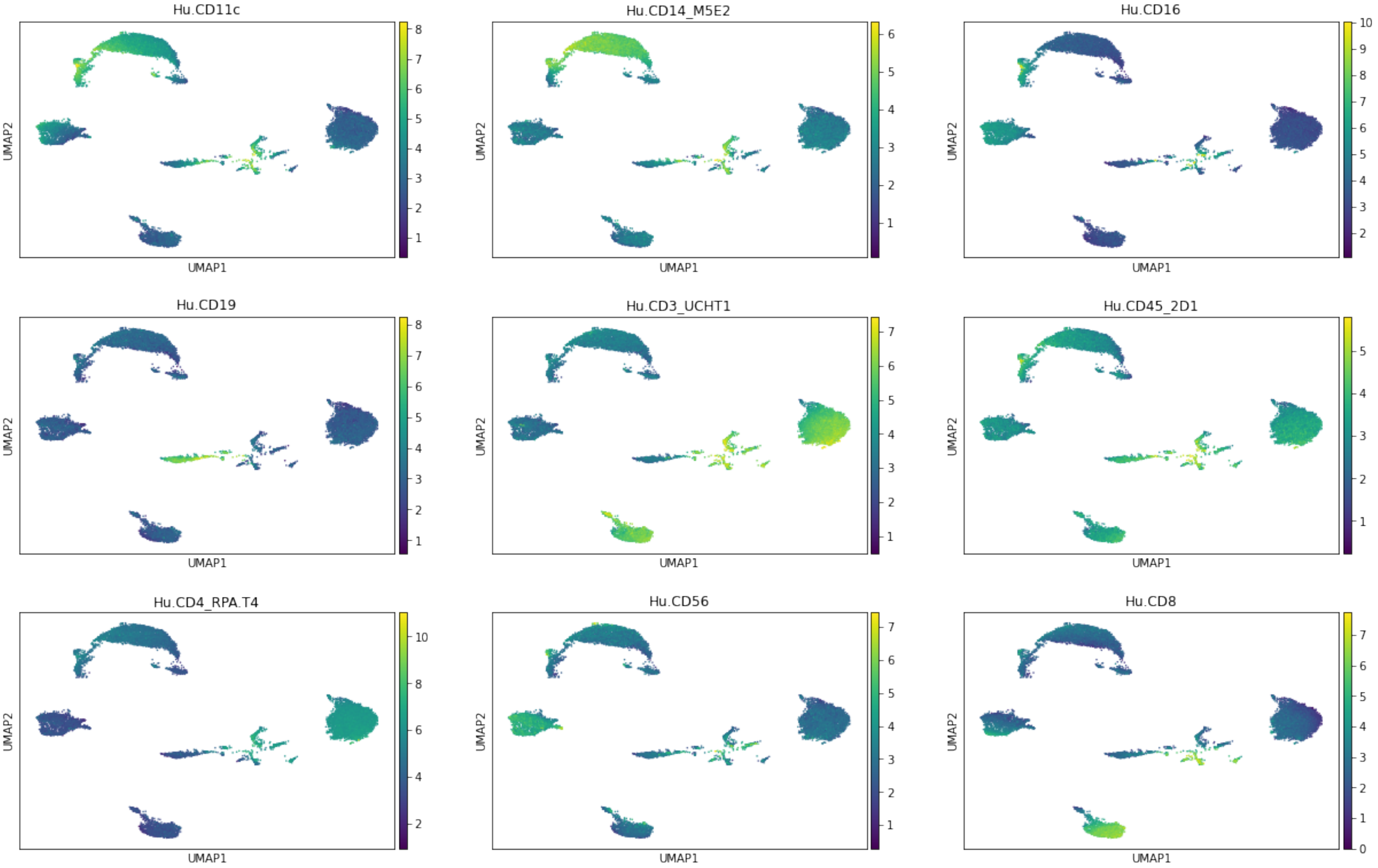
```
# calculate umaps
ann_adt = ann_single[:,['ADT' in t for t in ann_single.var['gene_ids']]].copy()
demux.asinh_trans(ann_adt)
sc.pp.neighbors(ann_adt, n_neighbors=40, use_rep='X')
sc.tl.umap(ann_adt,min_dist=0.1)
```

ADT expression level plotted on ADT umaps are shown below,
different cell type clusters are well separated

In [27]:

```
sc.pl.umap(ann_adt, color=ann_adt.var_names, ncols=3)
```

```
... storing 'assignment' as categorical
... storing 'feature_types' as categorical
... storing 'genome' as categorical
```


Hu.CD11c
UMAP2
UMAP1
1 2 3 4 5 6 7 8
Hu.CD14_M5E2
UMAP2
UMAP1
1 2 3 4 5 6
Hu.CD16
UMAP2
UMAP1
2 3 4 5 6 7 8 9 10
Hu.CD19
UMAP2
UMAP1
1 2 3 4 5 6 7 8
Hu.CD3_UCHT1
UMAP2
UMAP1
1 2 3 4 5 6 7
Hu.CD45_2D1
UMAP2
UMAP1
1 2 3 4 5
Hu.CD4_RPA.T4
UMAP2
UMAP1
2 4 6 8 10
Hu.CD56
UMAP2
UMAP1
1 2 3 4 5 6 7
Hu.CD8
UMAP2
UMAP1
0 1 2 3 4 5 6 7

```
In [ ]:
```