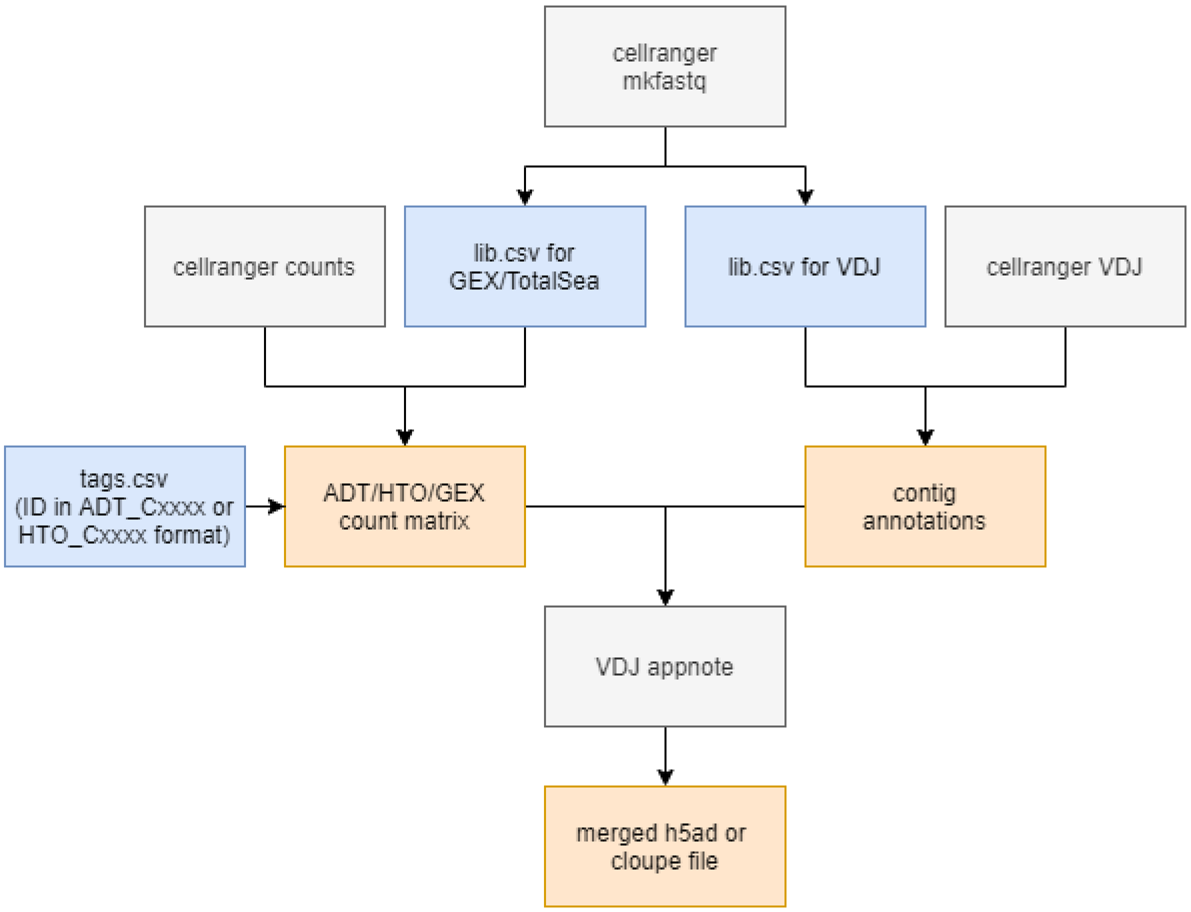
Process overview



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_VDJ \
                   --run=/path/to/bcl \
                   --samplesheet=GEX_VDJ_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
```

Generating contig annotation for VDJ with cellranger VDJ

a library csv file for fastqs generated the the mkfastq is required
Below is an example command of running cellranger counts:

```
cellranger vdj --id=VDJ \
               --sample=fqname \
               --fastqs=path/to/fqfolder \
               --reference=refdata-cellranger-vdj-GRCh38-alts-ensembl-5.0.0
```

Dataprocessing

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

In [1]:

```
import os
import matplotlib
import scipy.io
import csv
import anndata
import numpy as np
import pandas as pd
import scanpy as sc
import matplotlib.pyplot as plt
import VDJana as VDJ
```

In [2]:

```
%matplotlib inline
matplotlib.rcParams['font.size']=16
```

TotalSeq

read data and filtering

We first did some baisc filtering based on cell UMI from raw data matrix

In [3]:

```
sample = 'Sample1'
```

In [4]:

```
ann = sc.read_10x_h5(f'./{sample}/GEX/outs/raw_feature_bc_matrix.h5',gex_only=False)
```

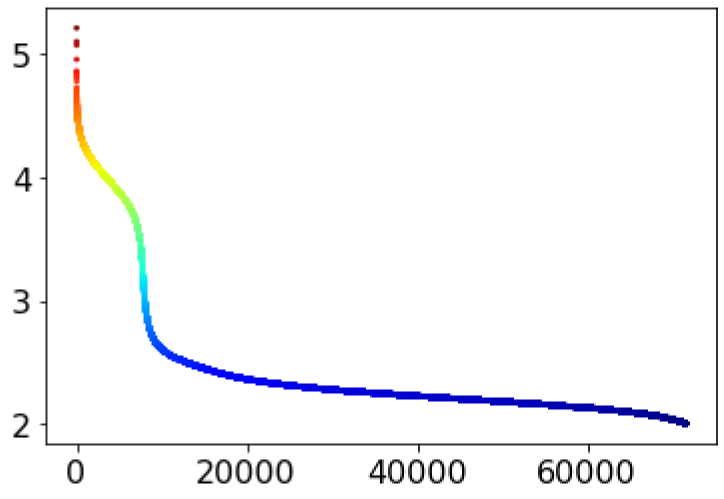
```
Variable names are not unique. To make them unique, call `.var names make unique`.
```

In [5]:

```
umi = np.squeeze(np.asarray(ann.X.sum(axis=1)))
umi_new = sorted(umi[umi>100],reverse=True)
matplotlib.rcParams['font.size']=16
fig,ax=plt.subplots(1,1)
ax.scatter(np.arange(len(umi_new)),np.log10(umi_new),c=np.log10(umi_new),cmap='jet',s=3)
```

Out[5]:

```
<matplotlib.collections.PathCollection at 0x7fad007ba898>
```

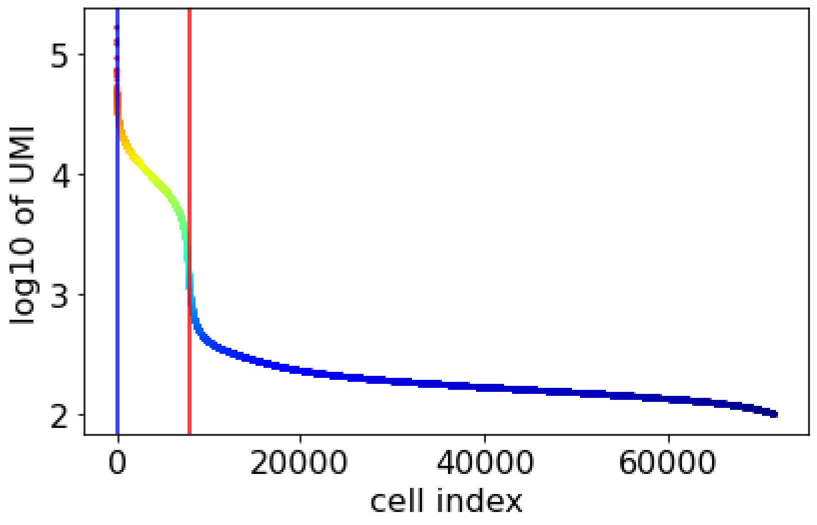


In [6]:

```
th1 = 8000
th2 = 80
```

In [7]:

```
matplotlib.rcParams['font.size']=16
fig,ax=plt.subplots(1,1)
ax.scatter(np.arange(len(umi_new)),np.log10(umi_new),c=np.log10(umi_new),cmap='jet',s=3)
ax.axvline(th2,color='b')
ax.axvline(th1,color='r')
plt.xlabel('cell index')
plt.ylabel('log10 of UMI')
plt.tight_layout()
```



In [8]:

```
ann = ann[np.logical_and(umi<umi_new[th2],umi>umi_new[th1])].copy()
```

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

In [9]:

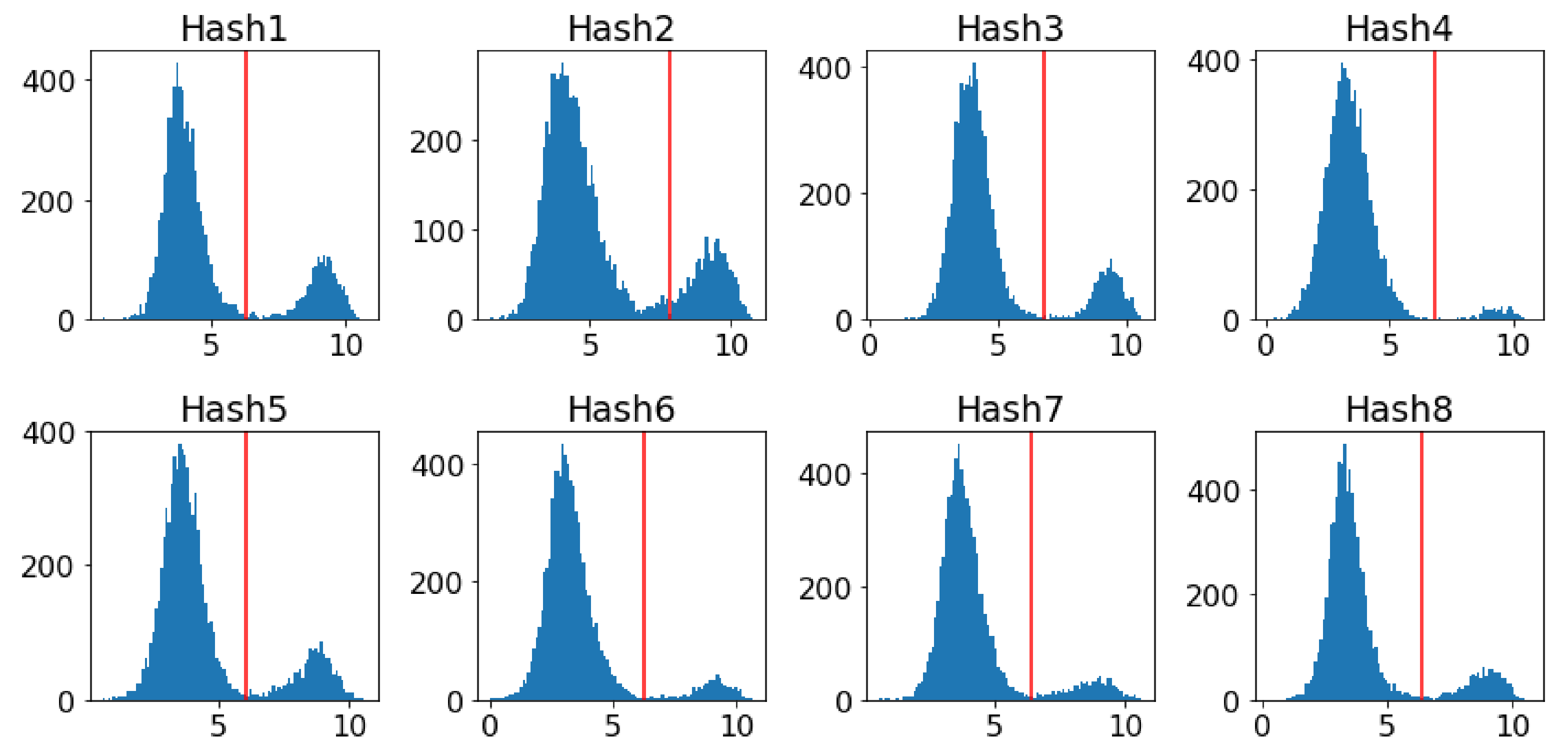
```
ann.var_names_make_unique()
```

demultiplexing

The dataset is further demultiplexed by threshold calling based method
The histogran distribution of hashtags and corresponding threshold is shown as reference.

In [10]:

```
VDJ.get_demux(ann)
```



In [11]:

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

Out[11]:

```
Hash1        1397
Hash2        1338
Hash5        1164
Hash3        1052
Hash8         876
Hash7         643
Doublets      551
Hash6         496
Hash4         248
Negative      154
Name: assignment, dtype: int64
```

HTO UMAP

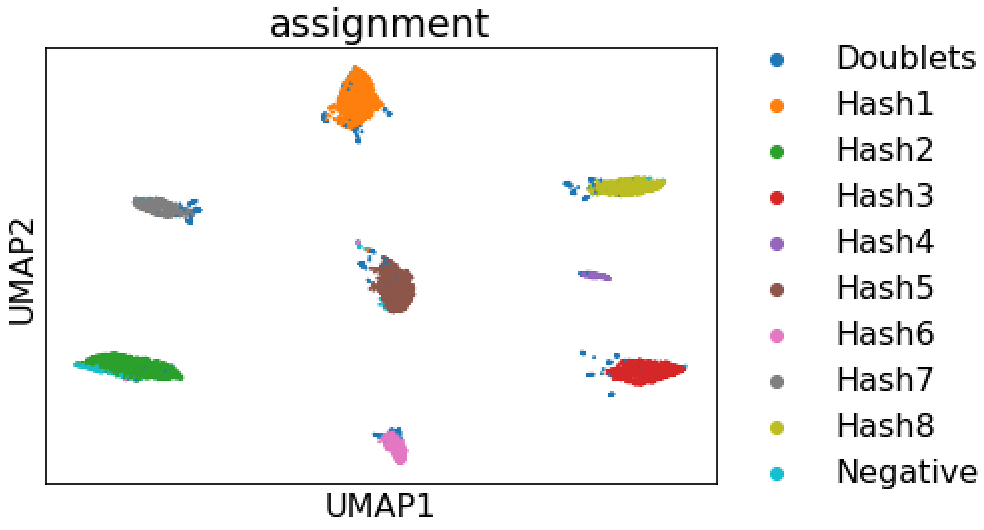
Next, we will generate UMAP based on HTO expression level to see
whether the demultiplxing result looks resonable

In [12]:

```
ann_hto = ann[:,['HTO' in t for t in ann.var['gene_ids']]].copy()
VDJ.asinh_trans(ann_hto)
sc.pp.neighbors(ann_hto, n_neighbors=40, use_rep='X')
sc.tl.umap(ann_hto,min_dist=0.1)
```

```
ann.obsm['X_HTO_umap'] = ann_hto.obsm['X_umap']
sc.pl.umap(ann_hto,color='assignment')
```

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



In [13]:

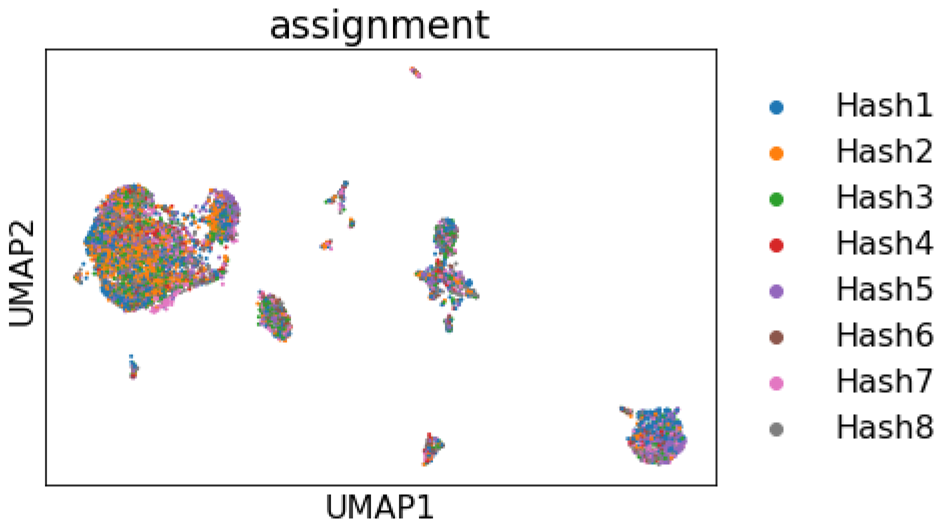
```
ann = ann[ann.obs['assignment'] != 'Negative']
ann = ann[ann.obs['assignment'] != 'Doublets'].copy()
```

add UMAP for ADT

In [14]:

```
ann_adt = ann[:,['ADT' in t for t in ann.var['gene_ids']]].copy()
VDJ.asinh_trans(ann_adt)
sc.pp.neighbors(ann_adt, n_neighbors=40, use_rep='X')
sc.tl.umap(ann_adt,min_dist=0.1)
ann.obsm['X_ADT_umap'] = ann_adt.obsm['X_umap']
sc.pl.umap(ann_adt,color='assignment')
```

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



VDJ on singlets

We will extract the VDJ results from *filtered_contig_annotations.csv*
in the outs folder of Cellranger VDJ

In [15]:

```
vdj_path = f'./{sample}/VDJ/outs'
vdj = pd.read_csv(os.path.join(vdj_path,'filtered_contig_annotations.csv'),index_col=0)

vdj['hashID'] = ann.obs['assignment']
vdj_over_sin = vdj.loc[np.logical_and(vdj.productive == True, vdj.hashID.astype('str') != 
'nan')]
vdj_over_sin.to_csv(f'./{sample}/merge.csv')

# print entry counts for each hashtag
pd.Categorical(vdj_over_sin.hashID).value_counts()
```

Out[15]:

```
Hash1    1431
Hash2    1735
Hash3    1274
Hash4     289
Hash5    1031
Hash6     528
Hash7     526
Hash8     995
dtype: int64
```

map VDJ results to anndata file

the TRA and TRB information is stored as columns of observation dataframe in anndata file

In [16]:

```
ann.obs['VDJ_chain_counts'] = np.zeros(ann.n_obs)
for cb in vdj_over_sin.index:
    chain_info = vdj_over_sin.loc[[cb]]
    ann.obs.loc[cb, 'VDJ_chain_counts'] = len(chain_info)
    if len(chain_info)>2:
        continue
    for ind,row in chain_info.iterrows():
        prefix = row['chain']
        ann.obs.loc[cb,f'{prefix}:v_gene'] = row['v_gene']
        ann.obs.loc[cb,f'{prefix}:d_gene'] = row['d_gene']
        ann.obs.loc[cb,f'{prefix}:j_gene'] = row['j_gene']
    ann.obs.loc[cb,'clonotype'] = row['raw_clonotype_id']
```

In [17]:

```
ann
```

Out[17]:

```
AnnData object with n_obs × n_vars = 7214 × 36646
    obs: 'assignment', 'VDJ_chain_counts', 'TRB:v_gene', 'TRB:d_gene', 'TRB:j_gene', 'clonot
ype', 'TRA:v_gene', 'TRA:d_gene', 'TRA:j_gene'
    var: 'gene_ids', 'feature_types', 'genome', 'thre'
    obsm: 'X_HTO_umap', 'X_ADT_umap'
```

In [18]:

```
ann.obs[['assignment', 'VDJ_chain_counts','clonotype']]
```

Out[18]:

	assignment	VDJ_chain_counts	clonotype
AAACCTGAGCCGGTAA-1	Hash2	1.0	clonotype1099
AAACCTGAGGACTGGT-1	Hash5	2.0	clonotype2662
AAACCTGAGGCAATTA-1	Hash7	2.0	clonotype2719
AAACCTGCAAACCTAC-1	Hash1	2.0	clonotype69
AAACCTGCAATGACCT-1	Hash8	3.0	NaN
...	...	...	...
TTTGTCATCAACACCA-1	Hash5	0.0	NaN
TTTGTCATCCAAAGTC-1	Hash8	2.0	clonotype1460
TTTGTCATCCACGACG-1	Hash3	2.0	clonotype4399
TTTGTCATCGCGCCAA-1	Hash5	2.0	clonotype2043
TTTGTCATCGGCTACG-1	Hash8	0.0	NaN

7214 rows × 3 columns

filter on cells and genes

For each cell, it is expected that there is one TRA chain and one TRB chain
Cells with more than two chains or no chains are considered as doublets and negatives, respectively.
The dataset is further filtered out based on VDJ chain counts.

In [19]:

```
ann =
ann[np.logical_and(ann.obs['VDJ_chain_counts']>0,ann.obs['VDJ_chain_counts']<3)].copy()
```

filter gene down to marker related and VDJ genes

In [20]:

```
vdj_gene_list =  []
for col in ann.obs:
    if ':' in col:
        vdj_gene_list += list(ann.obs[col].unique())
```

In [21]:

```
with open('human_gene_mapping.txt','r') as f:
    human_mapping = [row.strip() for row in f.readlines()]

ann.var['mapping'] = [''] *  ann.n_vars
for tag in ann.var_names[['ADT' in t for t in ann.var['gene_ids']]]:
    feature_id = ann.var.loc[tag, 'gene_ids']
    mapped_gene = human_mapping[int(feature_id[-4:])]
    if mapped_gene == '0':
        ann.var.loc[tag,'mapping'] = 'NA'
    else:
        ann.var.loc[tag,'mapping'] = mapped_gene
        if mapped_gene not in ann.var_names:
            continue
        if len(ann.var.loc[mapped_gene, 'mapping']) == 0:
            ann.var.loc[mapped_gene, 'mapping'] = tag
        else:
            ann.var.loc[mapped_gene, 'mapping'] += f',{tag}'
marker_gene = ann.var_names[ann.var['mapping'] != ''].tolist()
```

In [22]:

```
ann = ann[:,[t in marker_gene+vdj_gene_list for t in ann.var_names]].copy()
```

simplify the clonotype display for cellxgene

In [23]:

```
high_clonotype = list(ann.obs['clonotype'].value_counts().keys()[:99])
```

In [24]:

```
def colon_trans(str_in):
    if str_in in high_clonotype:
        return(str_in[:9]+str_in[9:].zfill(3))
    else:
        return('RareClonotype')

ann.obs['abundant_clonotype'] = [colon_trans(t) for t in ann.obs['clonotype']]
```

generate output for visualization

In [25]:

```
ann.write(f'./{sample}/ann_filter.h5ad')
```

```
... storing 'assignment' as categorical
... storing 'TRB:v_gene' as categorical
... storing 'TRB:d_gene' as categorical
... storing 'TRB:j_gene' as categorical
... storing 'clonotype' as categorical
... storing 'TRA:v_gene' as categorical
... storing 'TRA:j_gene' as categorical
... storing 'abundant_clonotype' as categorical
... storing 'feature_types' as categorical
... storing 'genome' as categorical
... storing 'mapping' as categorical
```

generate metadata csv for loupe browser

get cloupe cells

In [26]:

```
aref =
sc.read_10x_h5(f'./{sample}/lane/outs/filtered_feature_bc_matrix.h5',gex_only=False)
ad = ann[aref.obs_names[[t in ann.obs_names for t in aref.obs_names]]].copy()
```

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

csv for umap

In [27]:

```
pd.DataFrame(ad.obsm['X_ADT_umap'], index=ad.obs_names,
columns=['X','Y']).to_csv(f'./{sample}/ADT_umap.csv')
```

In [28]:

```
pd.DataFrame(ad.obsm['X_HTO_umap'], index=ad.obs_names,
columns=['X','Y']).to_csv(f'./{sample}/HTO_umap.csv')
```

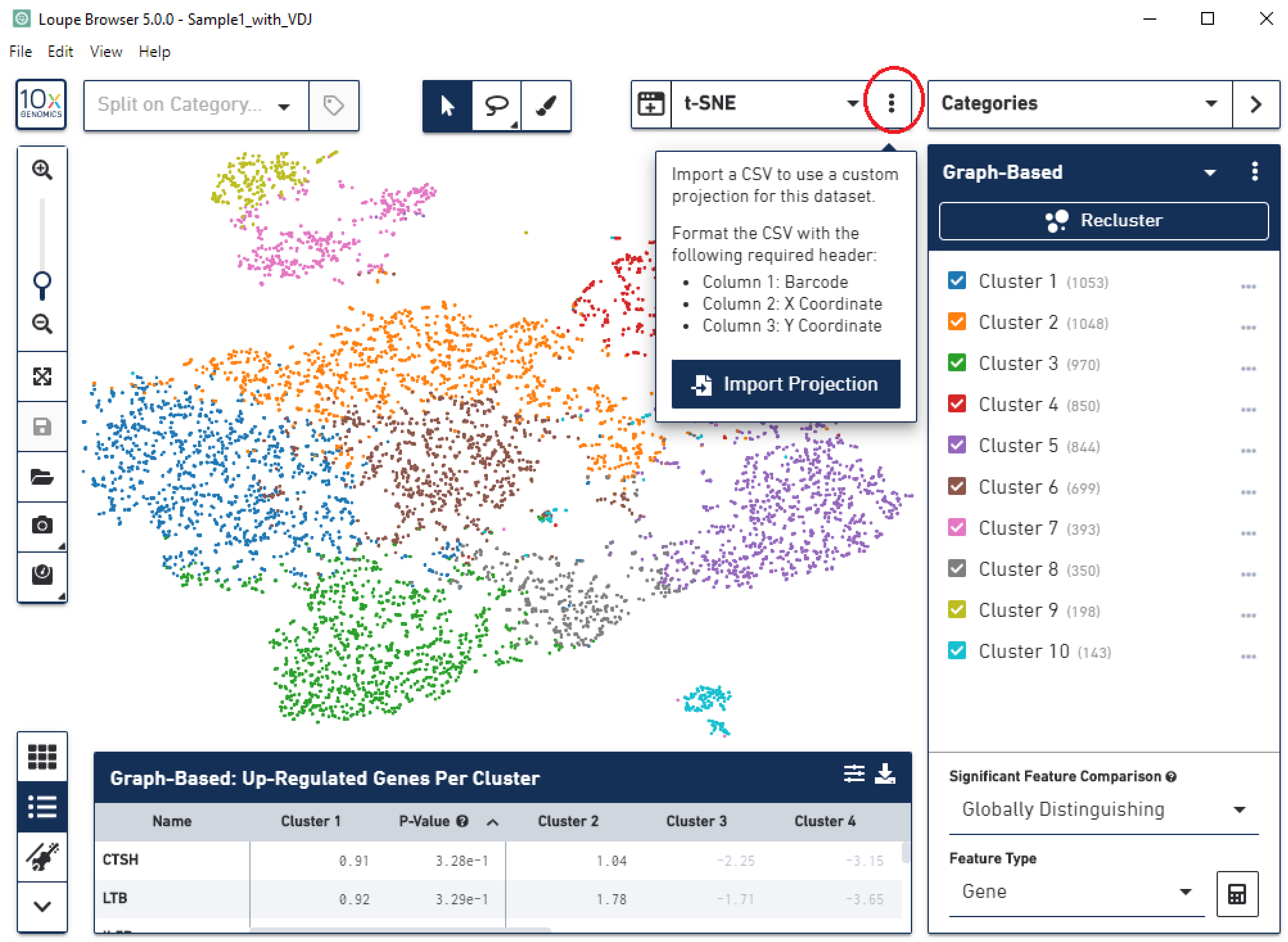
csv for metadata

In [30]:

```
ad.obs[['assignment', 'TRB:v_gene', 'TRB:d_gene', 'TRB:j_gene',
        'TRA:v_gene', 'TRA:d_gene',
'TRA:j_gene','abundant_clonotype']].to_csv(f'./{sample}/cloupe_obs.csv')
```

load csv into cloupe sessions

The umaps for the datasets can be loaded by clicking the "..." button marked in the pic below.
Select the umap.csv file you want to load coordinates for umaps



Import a CSV to use a custom projection for this dataset.

Format the CSV with the following required header:

- Column 1: Barcode
- Column 2: X Coordinate
- Column 3: Y Coordinate

☑ Cluster 1	(1053)
☑ Cluster 2	(1048)
☑ Cluster 3	(970)
☑ Cluster 4	(850)
☑ Cluster 5	(844)
☑ Cluster 6	(699)
☑ Cluster 7	(393)
☑ Cluster 8	(350)
☑ Cluster 9	(198)
☑ Cluster 10	(143)

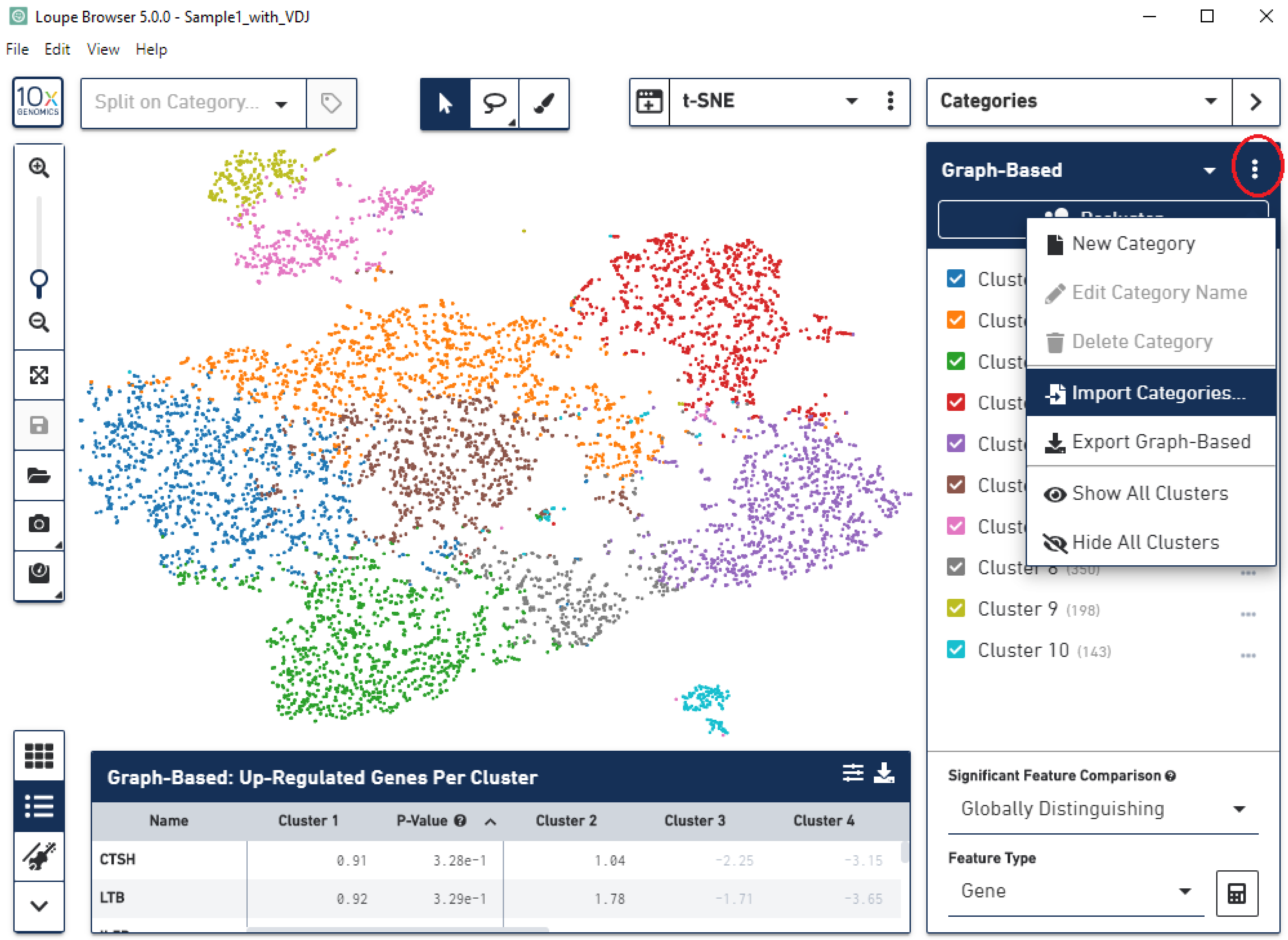
Graph-Based: Up-Regulated Genes Per Cluster

Name	Cluster 1	P-Value	Cluster 2	Cluster 3	Cluster 4
CTSH	0.91	3.28e-1	1.04	-2.25	-3.15
LTB	0.92	3.29e-1	1.78	-1.71	-3.65

Significant Feature Comparison: Globally Distinguishing

Feature Type: Gene

The metadata for the datasets can be loaded by clicking the "..." button marked in the pic below.
Select the cloupe_obs.csv to add metada to cloupe session



Graph-Based: Up-Regulated Genes Per Cluster

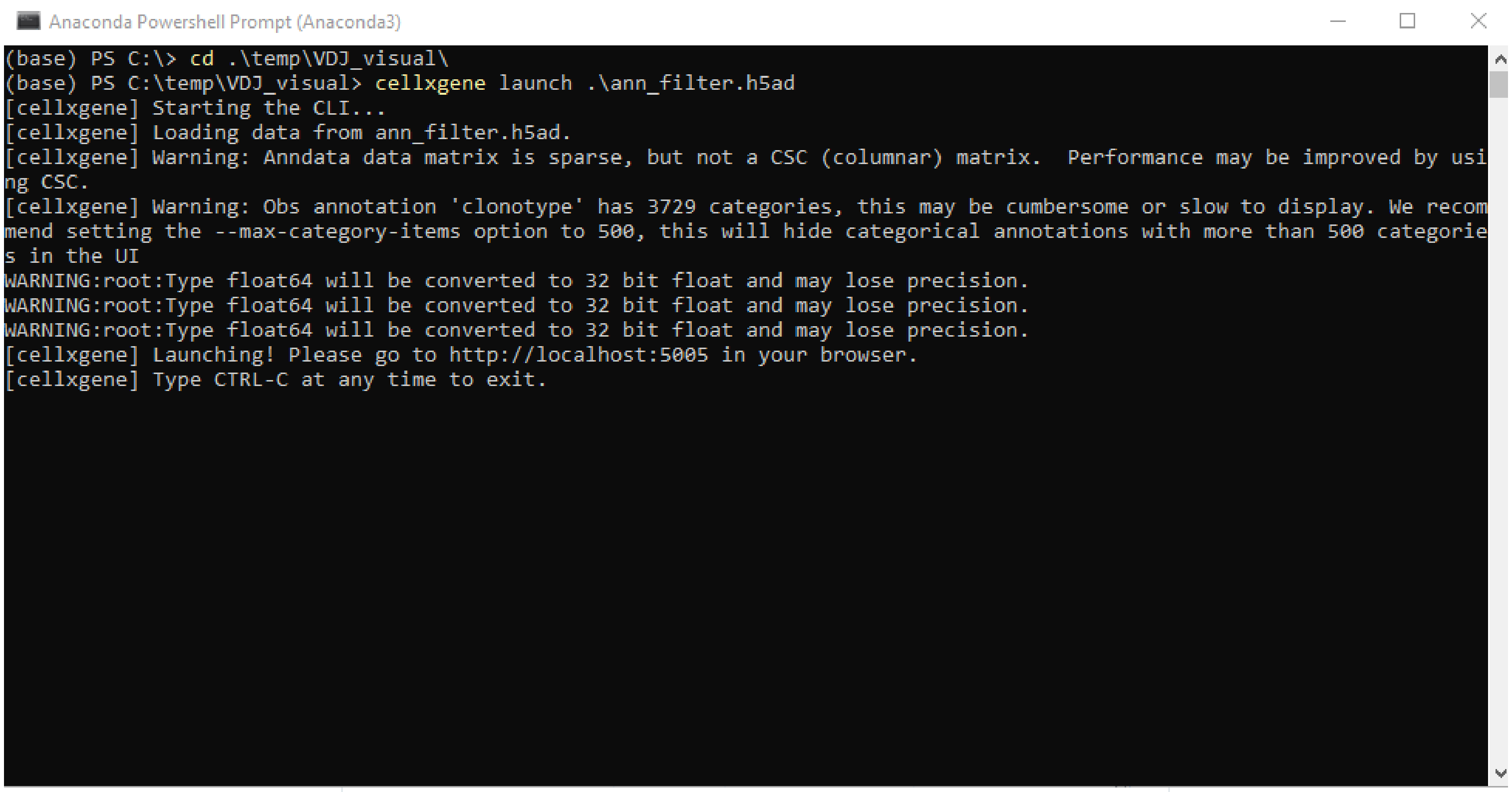
Name	Cluster 1	P-Value	Cluster 2	Cluster 3	Cluster 4
CTSH	0.91	3.28e-1	1.04	-2.25	-3.15
LTB	0.92	3.29e-1	1.78	-1.71	-3.65

Significant Feature Comparison: Globally Distinguishing

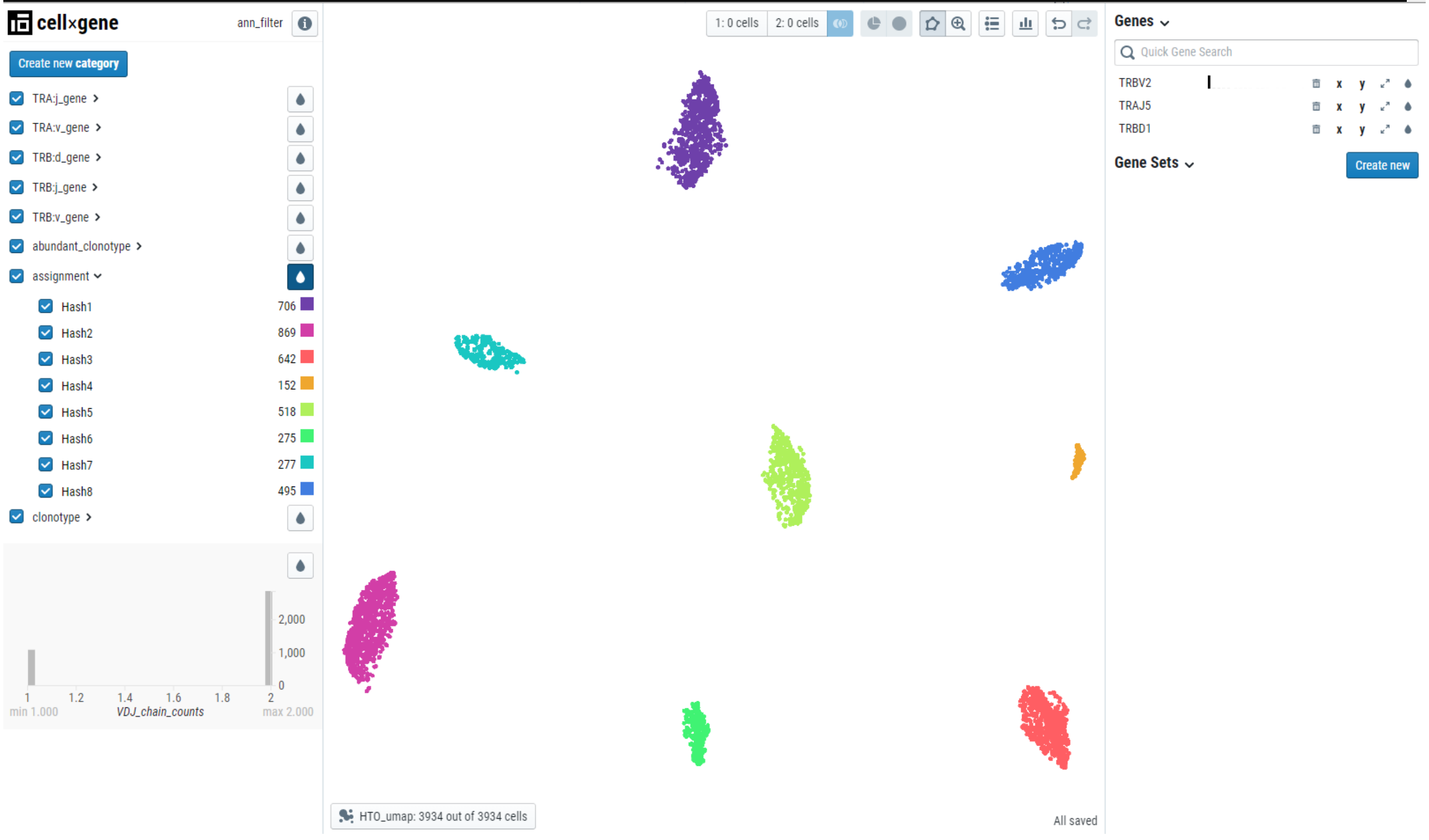
Feature Type: Gene

explore the dataset with cellxgene

navigate to the result folder and load by cellxgene launch command



```
(base) PS C:\> cd .\temp\VDJ_visual\
(base) PS C:\temp\VDJ_visual> cellxgene launch .\ann_filter.h5ad
[cellxgene] Starting the CLI...
[cellxgene] Loading data from ann_filter.h5ad.
[cellxgene] Warning: Anndata data matrix is sparse, but not a CSC (columnar) matrix.  Performance may be improved by usi
ng CSC.
[cellxgene] Warning: Obs annotation 'clonotype' has 3729 categories, this may be cumbersome or slow to display. We recom
mend setting the --max-category-items option to 500, this will hide categorical annotations with more than 500 categorie
s in the UI
WARNING:root:Type float64 will be converted to 32 bit float and may lose precision.
WARNING:root:Type float64 will be converted to 32 bit float and may lose precision.
WARNING:root:Type float64 will be converted to 32 bit float and may lose precision.
[cellxgene] Launching! Please go to http://localhost:5005 in your browser.
[cellxgene] Type CTRL-C at any time to exit.
```


cellxgene
ann_filter
Create new category
TRA:j_gene
TRA:v_gene
TRB:d_gene
TRB:j_gene
TRB:v_gene
abundant_clonotype
assignment
Hash1 706
Hash2 869
Hash3 642
Hash4 152
Hash5 518
Hash6 275
Hash7 277
Hash8 495
clonotype
2,000
1,000
0
1
1.2
1.4
1.6
1.8
2
min 1.000
VDJ_chain_counts
max 2.000
1: 0 cells
2: 0 cells
HTO_umap: 3934 out of 3934 cells
All saved
Genes
Quick Gene Search
TRBV2
TRAJ5
TRBD1
Gene Sets
Create new