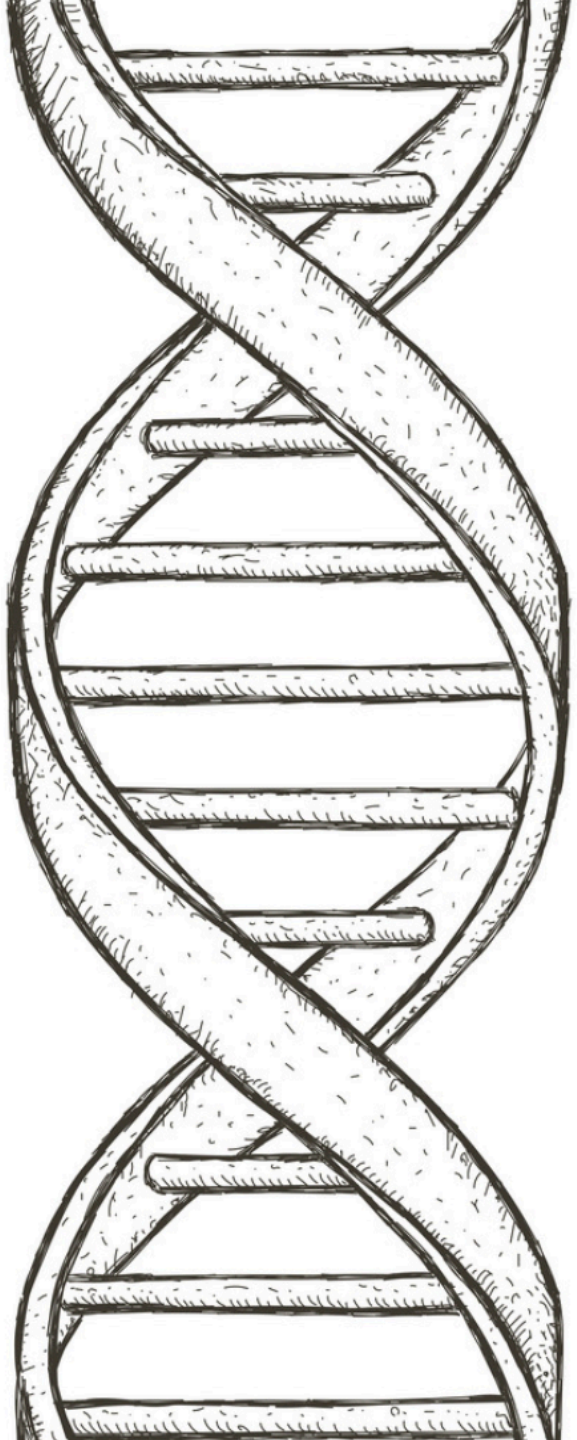


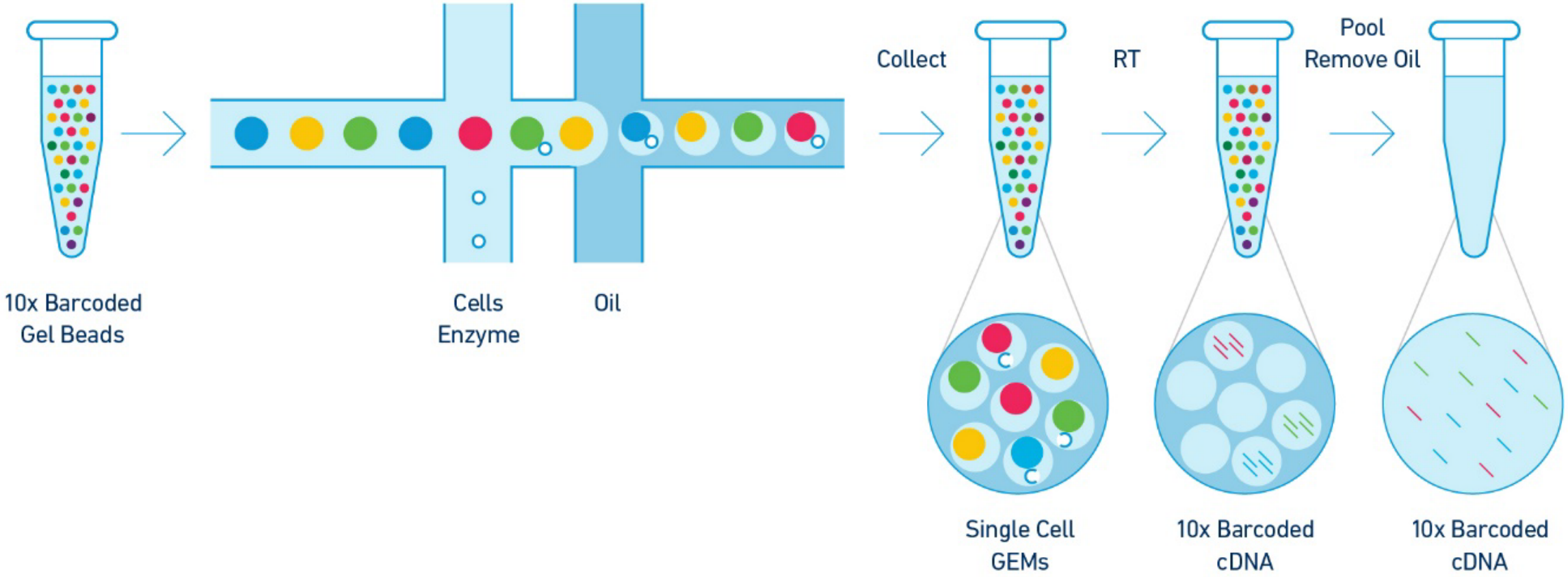
Cell type classification using Single Cell Sequencing data

Caroline Petersen

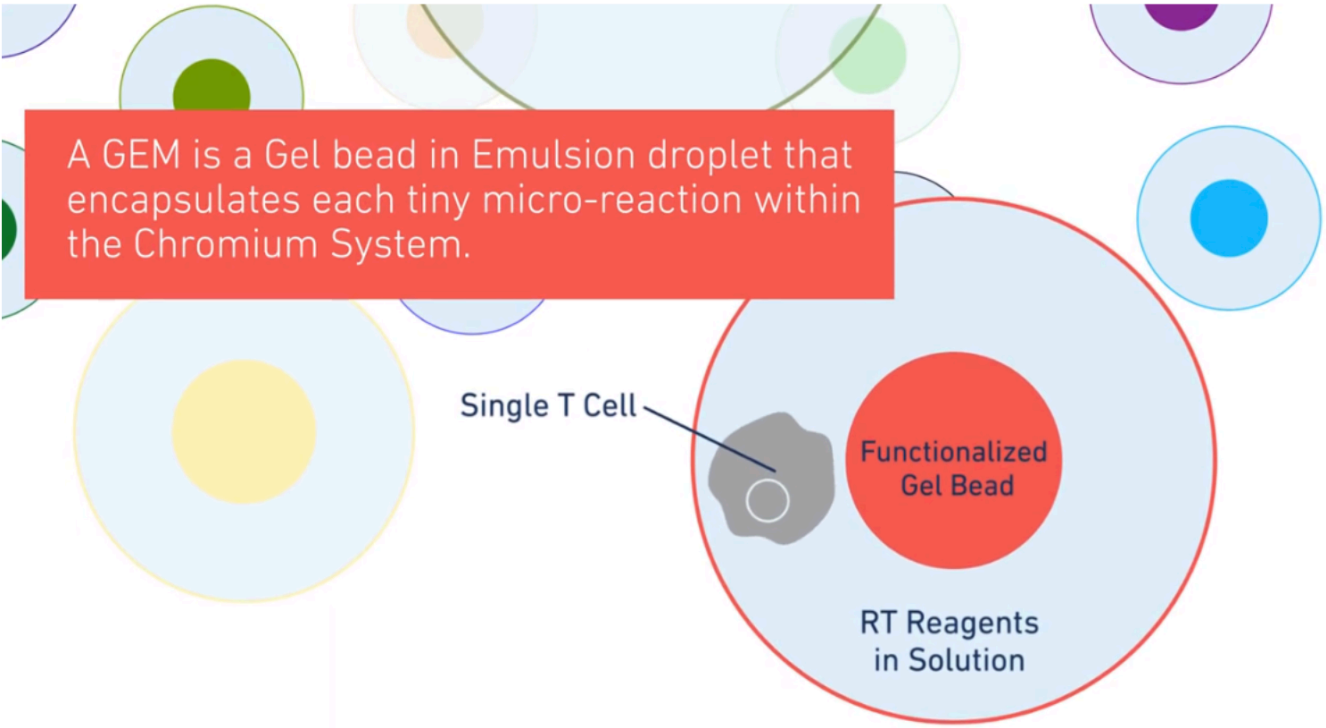
Broad Institute of MIT & Harvard,
Cambridge MA

General Assembly Data Science
12/16/19

Single Cell Sequencing

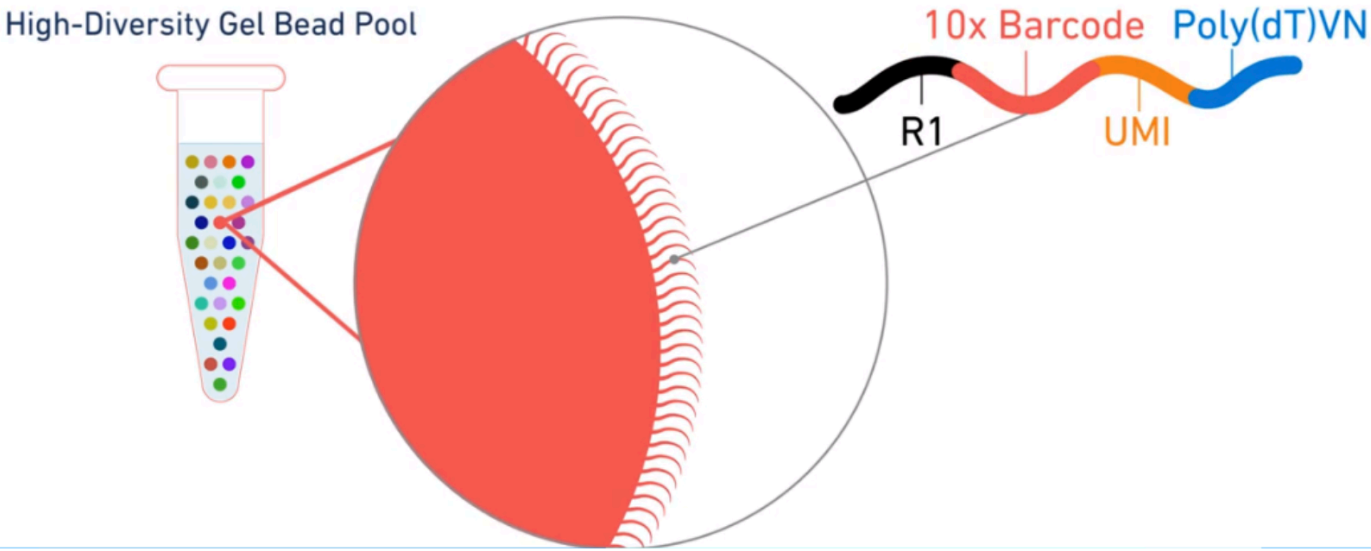


Single Cell Sequencing

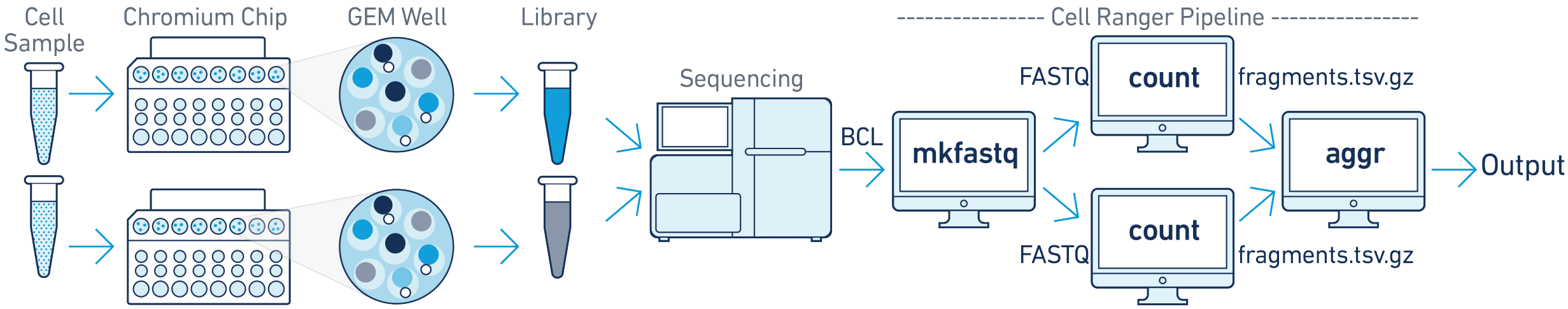


Our “barcode” tells us which cell is which later on

index
AAACCCAAGACAACTA
AAACCCAAGACGGTTG
AAACCCAAGAGAATCT
AAACCCAAGAGCTTTC
AAACCCAAGATACAGT



Single Cell Sequencing – data workflow



8 samples

~10,000 cells per sample

500 M reads per sample

= 4 billion reads total

Output: Raw DNA sequence

Pipeline:

1) De-convolute down to sample, then down to individual cell
2) Align to reference genome
3) Find genes & count how many

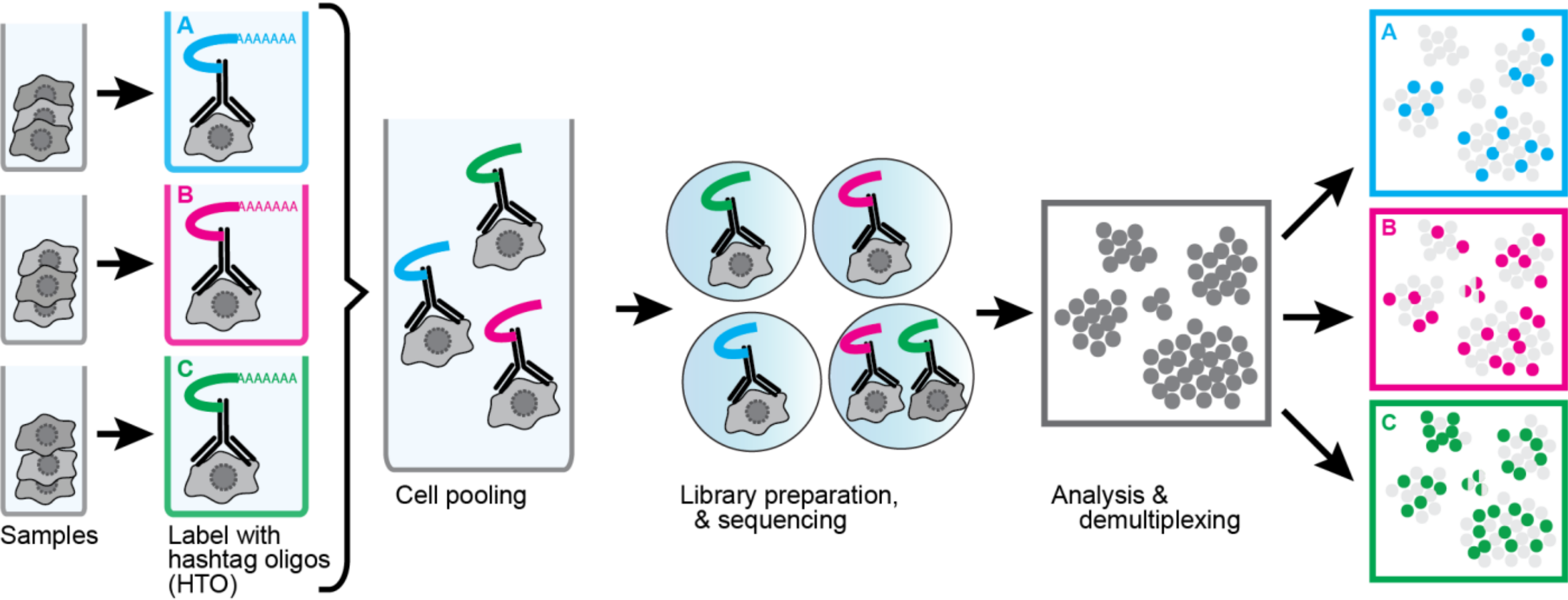
Count matrix

# of genes detected in each cell

Cell Hashing

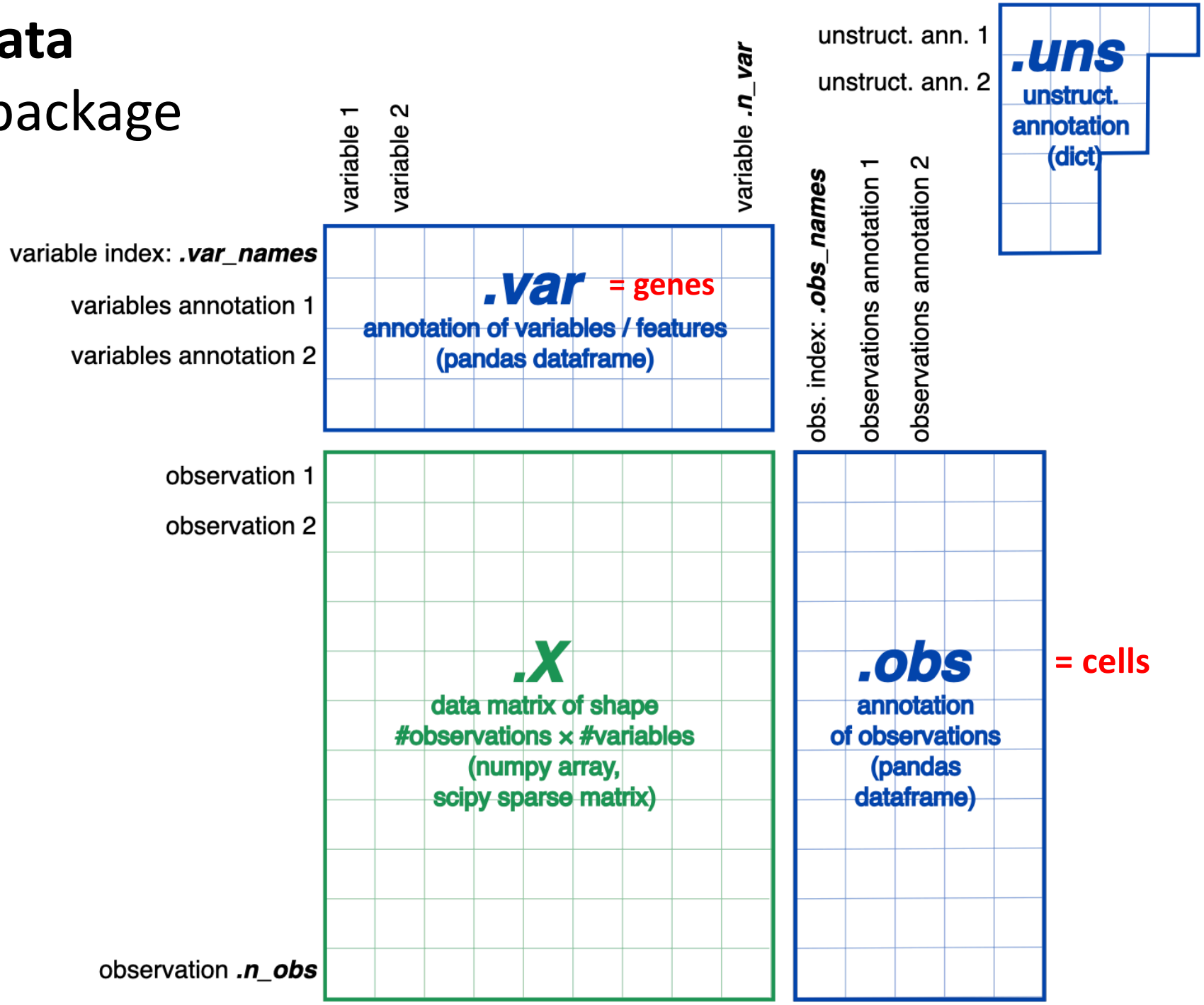
Our experiment:

- 8 samples
- 2 different cell types (4 samples from each)
  - HEK – liver cells
  - A375 – melanoma cells



Data output: AnnData

Part of the ScanPy package



Analysis Overview

Part 1: Using data from Cell Hashing validation
- Clean, clean, clean
- Identify our 8 samples within the data set (confirm that the Cell Hashing worked)
- Cluster the data to distinguish the two cell populations

Part 2: Using data from a separate experiment (same cell type from Part 1)
- Concatenate with our “training” data from Part 1
- Cluster again to determine if our new HEK cells are comparable to Part 1’s HEK cells

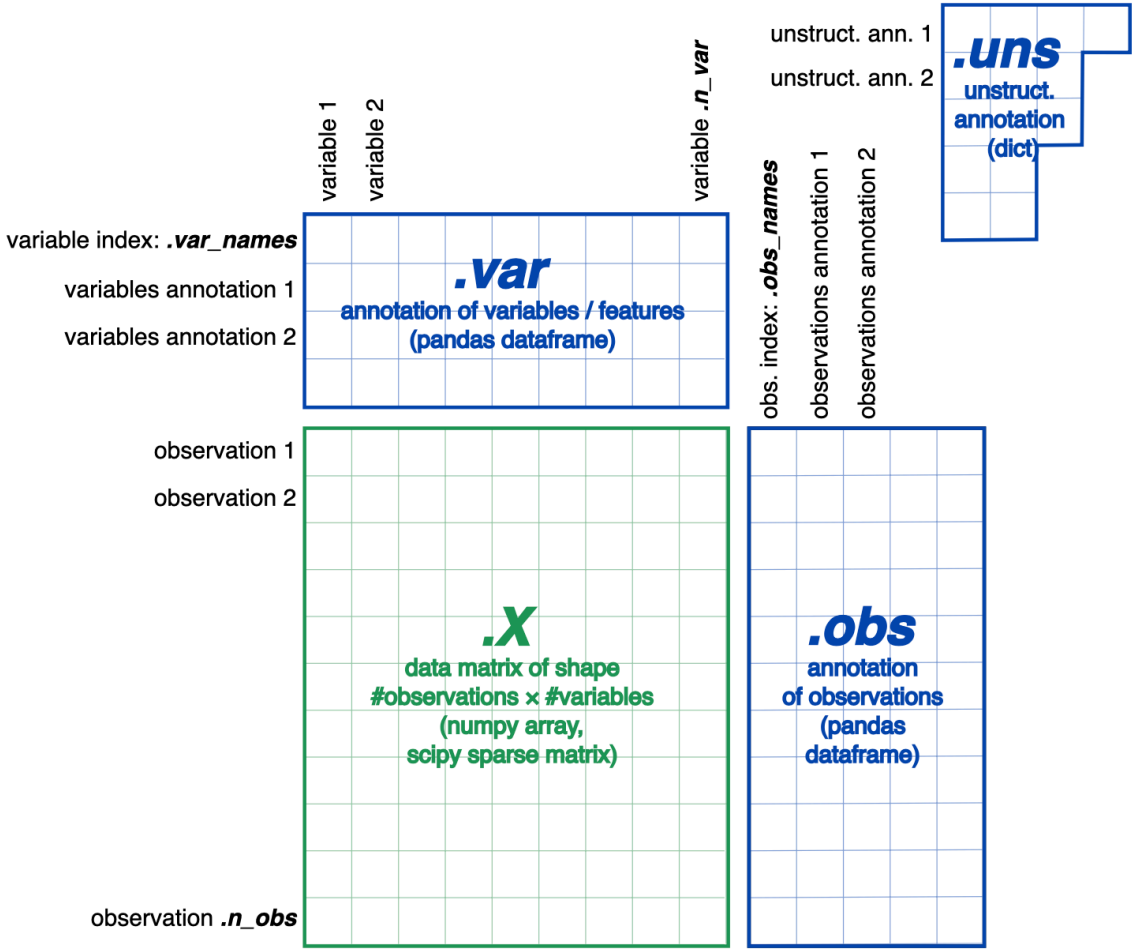
H_0: The gene expression is the same across our cell populations, and the clusters are not distinguishable from each other

H_A: The gene expression varies between our cell populations and they cluster separately from each other

Part 1

The ScanPy Package & AnnData object

```
import scanpy as sc
import scanpy.external as sce
import anndata
```




```
#read in our data
adata=anndata.read_h5ad('./Hashed_secondary_output_Hashed_Hashed_demux.h5ad')
adata
```

```
AnnData object with n_obs × n_vars = 70351 × 33538
    obs: 'Channel', 'n_genes', 'n_counts', 'demux_type', 'assignment'
    var: 'gene_ids', 'robust'
    uns: 'background_probs', 'genome'
    obsm: 'raw_probs'
```

Rows (.obs) = cells
Columns (.var) = genes

Part 1

The “observations” matrix

- **index:** the ”barcode” sequence that identifies each cell
- **channel:** synonymous for sample
- **n_genes:** the number of genes detected per cell
- **n_counts:** the number of reads per cell
- **demux_type:** tells us whether a droplet was a "singlet", "doublet" (containing 2 or more cells), or "unknown”
- **assignment:** our sample name. In this experiment, we have eight: 4 are HEK cells, and 4 are A375 cells.

```
[194]: adata.obs
```

[194]:

index	Channel	n_genes	n_counts	demux_type	assignment
AAACCCAAGACAACTA		368	536.0	doublet	A375_15,A375_13,HEK_10
AAACCCAAGACGGTTG		347	491.0	doublet	HEK_8,A375_15
AAACCCAAGAGAATCT		317	472.0	unknown	
AAACCCAAGAGCTTTC		352	509.0	unknown	
AAACCCAAGATACAGT		369	534.0	unknown	
...	...	...	...	...	...
TTTGTTGTCGTCGACG		257	325.0	unknown	
TTTGTTGTCTCCAATT		285	367.0	unknown	
TTTGTTGTCTCGAGTA		260	330.0	singlet	HEK_10
TTTGTTGTCTTAATCC		276	351.0	unknown	
TTTGTTGTCTTCGACC		253	321.0	doublet	HEK_8,A375_15

70351 rows × 5 columns

Part 1

The “variables” matrix

- **gene_ids:** The abbreviated name for each gene
- **robust:** boolean, tells us whether or not the gene aligns well to the reference gene

```
[193]: adata.var
```

[193]:

index	gene_ids	robust
MIR1302-2HG	ENSG00000243485	True
FAM138A	ENSG00000237613	True
OR4F5	ENSG00000186092	True
AL627309.1	ENSG00000238009	True
AL627309.3	ENSG00000239945	True
...	...	...
AC233755.2	ENSG00000277856	True
AC233755.1	ENSG00000275063	True
AC240274.1	ENSG00000271254	True
AC213203.1	ENSG00000277475	True
FAM231C	ENSG00000268674	True

33538 rows × 2 columns

Part 1

Validate the experiment
– did the Cell Hashing work?

```
[117]: samp.assignment.value_counts().head(25)
```

```
[117]:
                        31871
HEK_8                    5818
HEK_2                    3739
HEK_12                   3639
HEK_10                   3119
A375_14                  2996
A375_13                  2272
A375_15                  1860
A375_9                   1474
HEK_8,A375_13             719
HEK_8,HEK_10              607
HEK_8,A375_14             563
HEK_8,HEK_12              505
A375_14,HEK_8             460
HEK_8,A375_15             458
HEK_10,HEK_8              445
HEK_12,HEK_8              376
A375_13,HEK_8             301
HEK_10,A375_13            264
HEK_12,A375_13            263
HEK_2,HEK_8               236
A375_14,A375_13           230
HEK_8,A375_9              225
A375_15,HEK_8             223
HEK_10,HEK_12             220
Name: assignment, dtype: int64
```

Part 1

Filter

```
#filter out doublets, triplets & unknowns
adata=adata[adata.obs['demux_type']=='singlet']
adata
```

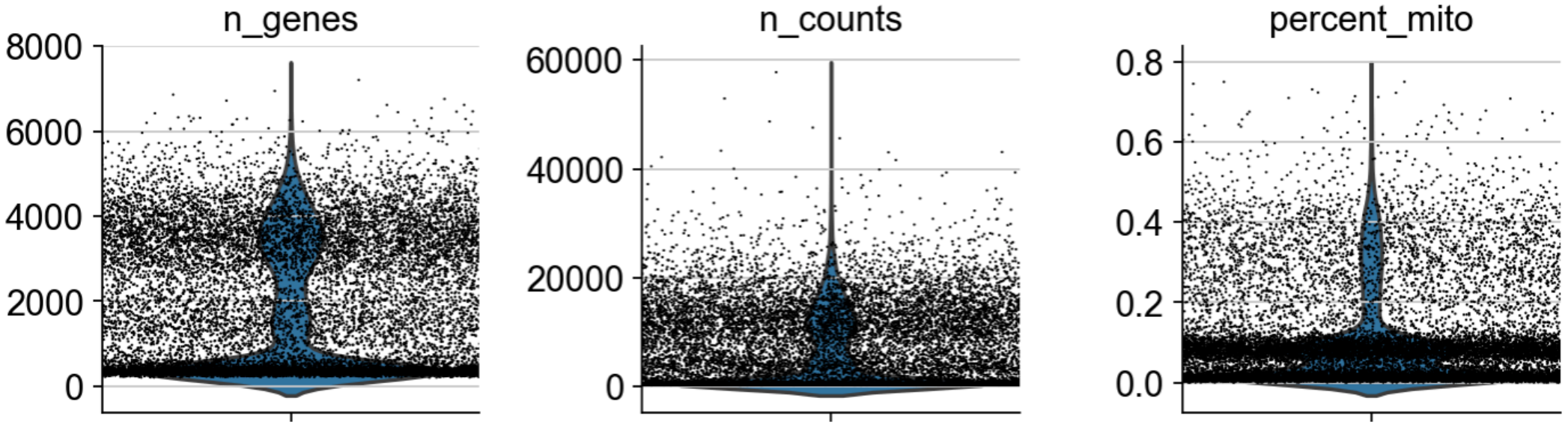
```
View of AnnData object with n_obs × n_vars = 24917 × 33538
    obs: 'Channel', 'n_genes', 'n_counts', 'demux_type', 'assignment'
    var: 'gene_ids', 'robust'
    uns: 'background_probs', 'genome'
    obsm: 'raw_probs'
```

```
#filtering out low genes
sc.pp.filter_cells(adata, min_genes=200)
sc.pp.filter_genes(adata, min_cells=3)
```

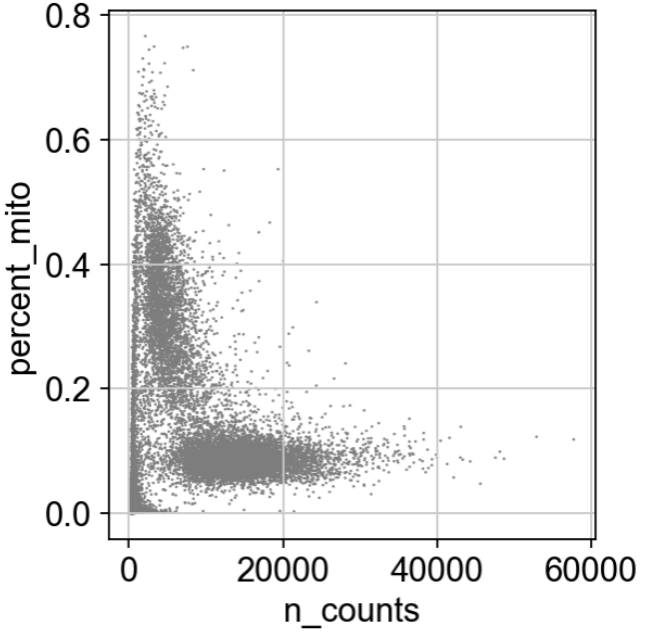
```
filtered out 91 cells that haveless than 200 genes expressed
Trying to set attribute `.obs` of view, making a copy.
filtered out 11892 genes that are detectedin less than 3 cells
```

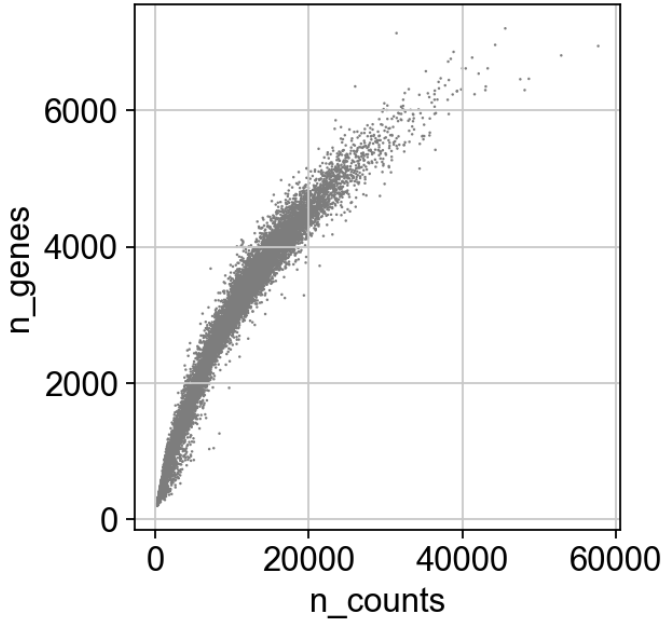
Part 1
Visualize

```
#a violin plot of the computed quality measures
sc.pl.violin(adata, ['n_genes','n_counts','percent_mito'], jitter=0.7, multi_panel=True)
```




```
sc.pl.scatter(adata, x='n_counts', y='percent_mito')
sc.pl.scatter(adata, x='n_counts',y='n_genes')
```



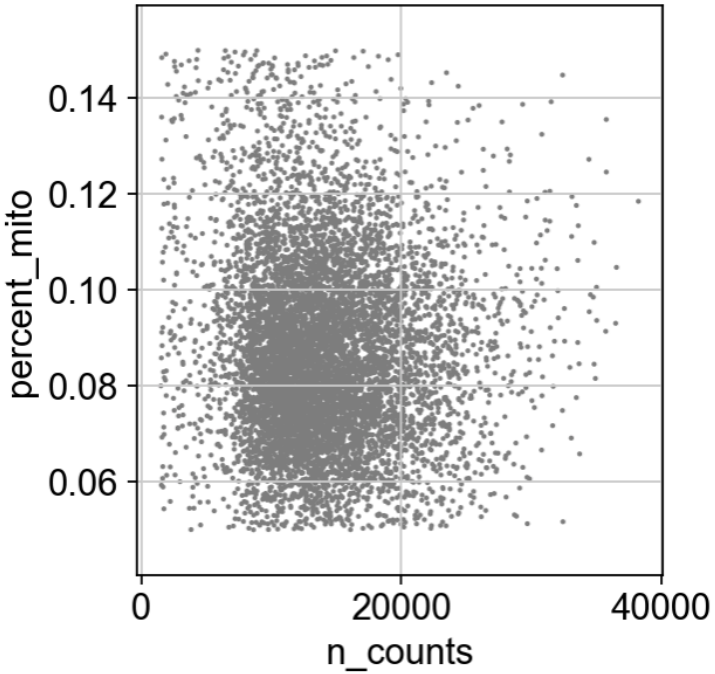


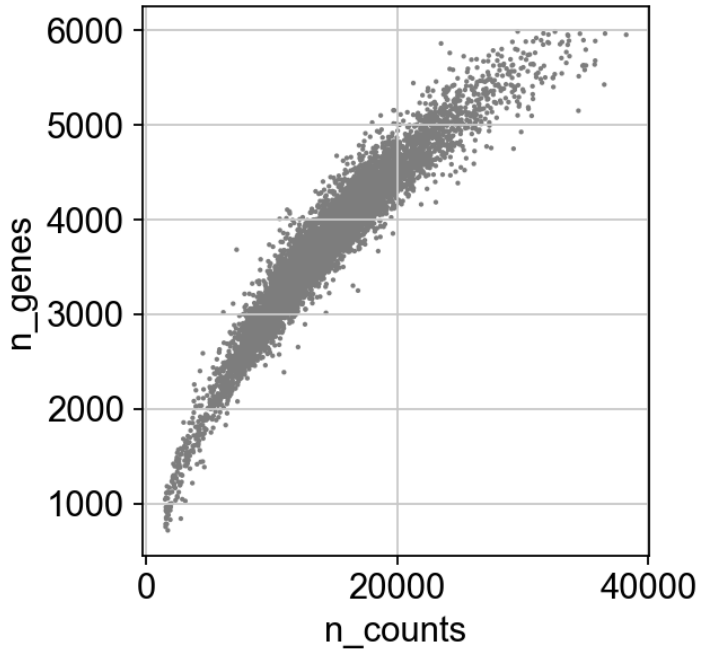
Part 1

Filter again

```
#filter out any cells that have less than 6000 genes (this is based off of our first g
adata=adata[adata.obs['n_genes']<6000, :]
#filter out any cells with over 5% mito (this filters out the very bottom left of the
adata=adata[adata.obs['percent_mito']<0.15, :]
adata=adata[adata.obs['percent_mito']>0.05, :]
#filter out any cells with fewer than 1500 reads per cell
adata=adata[adata.obs['n_counts']>1500, :]
adata
```

```
View of AnnData object with n_obs × n_vars = 8115 × 21646
    obs: 'Channel', 'n_genes', 'n_counts', 'demux_type', 'assignment', 'percent_mito'
    var: 'gene_ids', 'robust', 'n_cells'
    uns: 'background_probs', 'genome'
    obsm: 'raw_probs'
```





Part 1

More cleaning up…

Normalization based on counts per cell:

```
#total count normalize (library-size correct) the data matrix
sc.pp.normalize_per_cell(adata, counts_per_cell_after=1e4)
```

```
normalizing by total count per cell
    finished (0:00:00): normalized adata.X and added    'n_counts', counts per cell before normalization (adata.obs)
```

Logarithmize:

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

scanpy.api.pp.log1p(*data, copy=False, chunked=False, chunk_size=None*)

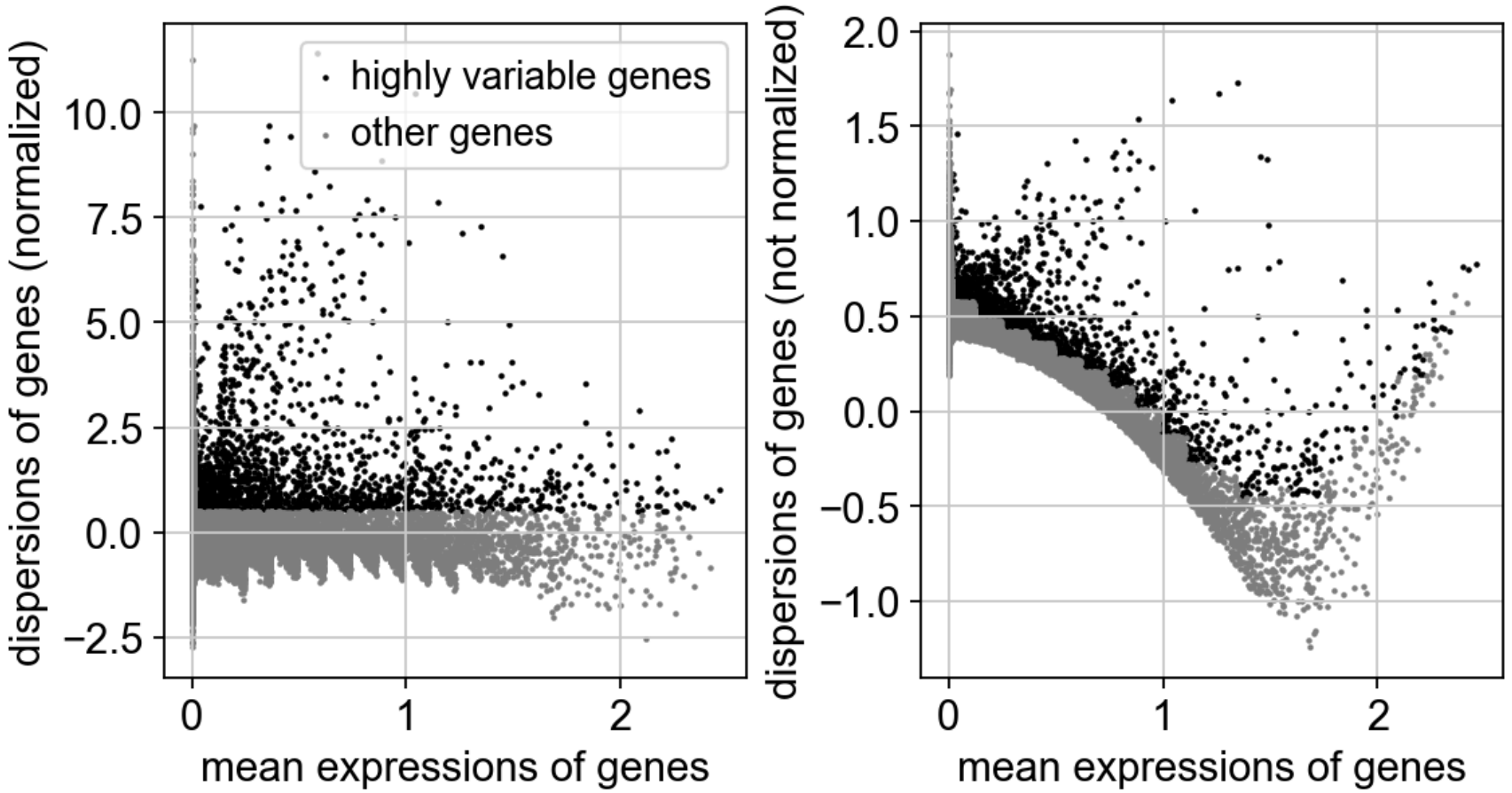
Logarithmize the data matrix.

Computes $X = \log(X + 1)$, where log denotes the natural logarithm.

Part 1

Identify highly variable genes

```
#Identify highly variable genes
sc.pp.highly_variable_genes(adata_conc, min_mean=0.0125, max_mean=3, min_disp=0.5)
```



Part 1

Filter & regress

```
#filter
adata_conc=adata_conc[:, adata_conc.var['highly_variable']]

#regress out effects of n counts and % mito
sc.pp.regress_out(adata_conc, ['n_counts','percent_mito'])

# Clip values exceeding standard deviation 10
sc.pp.scale(adata_conc, max_value=10)
```

scanpy.api.pp.regress_out

`scanpy.api.pp.regress_out(adata, keys, n_jobs=None, copy=False)`

Regress out unwanted sources of variation.

Uses simple linear regression. This is inspired by Seurat's `regressOut` function in R [Satija15].

scanpy.api.pp.scale

`scanpy.api.pp.scale(data, zero_center=True, max_value=None, copy=False)`

Scale data to unit variance and zero mean.

Note

Variables (genes) that do not display any variation (are constant across all observations) are retained and set to 0 during this operation. In the future, they might be set to NaNs.

Part 1

Our data is finally cleaned up!

Before:

```
#read in our data
adata=anndata.read_h5ad('./Hashed_secondary_output_Hashed_Hashed_demux.h5ad')
adata
```

```
AnnData object with n_obs × n_vars = 70351 × 33538
    obs: 'Channel', 'n_genes', 'n_counts', 'demux_type', 'assignment'
    var: 'gene_ids', 'robust'
    uns: 'background_probs', 'genome'
    obsm: 'raw_probs'
```

After:

```
adata
```

```
AnnData object with n_obs × n_vars = 8115 × 2049
    obs: 'Channel', 'n_genes', 'n_counts', 'demux_type', 'assignment', 'percent_mito'
    var: 'gene_ids', 'robust', 'n_cells', 'highly_variable', 'means', 'dispersions', 'dispersions_norm'
    uns: 'background_probs', 'genome'
    obsm: 'raw_probs'
```

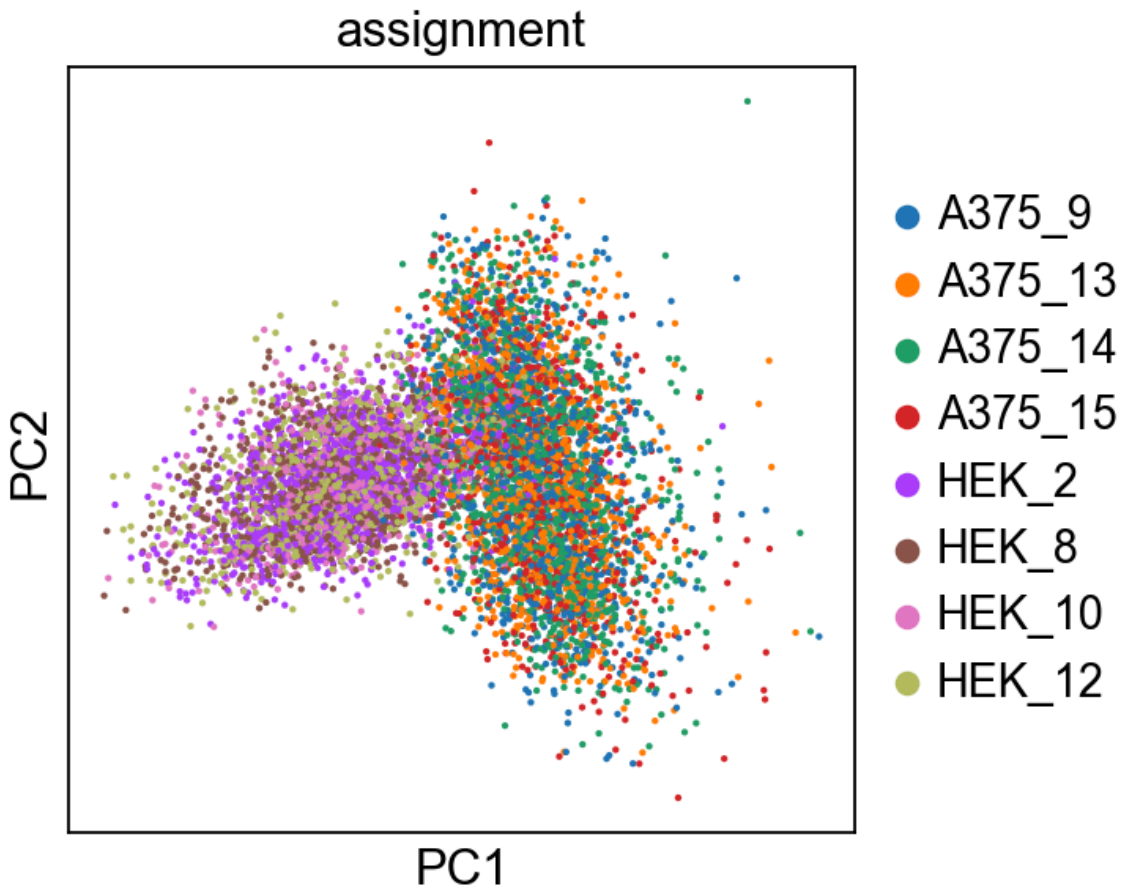
Part 1

Principle Component Analysis

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

```
computing PCA with n_comps = 50
computing PCA on highly variable genes
    finished (0:00:02)
```

```
#scatter plot in the PCA coordinates
sc.pl.pca(adata, color='assignment')
```



Part 1

Clustering

Compute the neighborhood graph using PCA coordinates & UMAP – similar concept to K-nearest neighbors

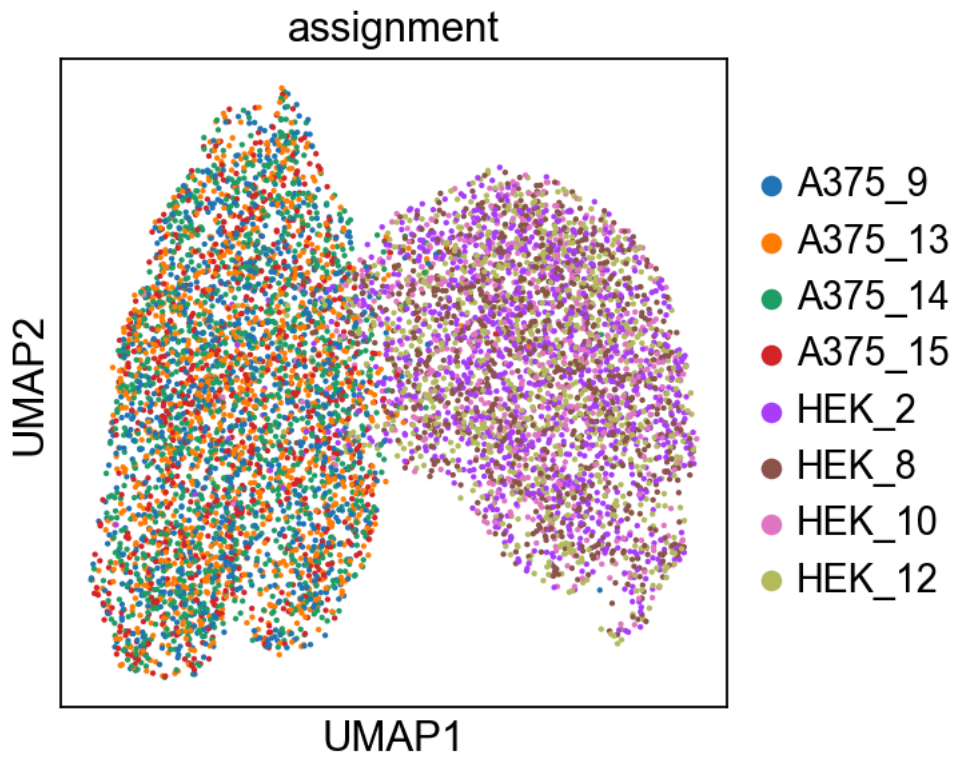
```
sc.pp.neighbors(adata, n_neighbors=10, n_pcs=15)
```

```
computing neighbors
    using 'X_pca' with n_pcs = 15
    finished: added to `.uns['neighbors']`
    'distances', distances for each pair of neighbors
    'connectivities', weighted adjacency matrix (0:00:0
```

```
#Embedding the graph in 2 dimensions using UMAP
sc.tl.umap(adata)
```

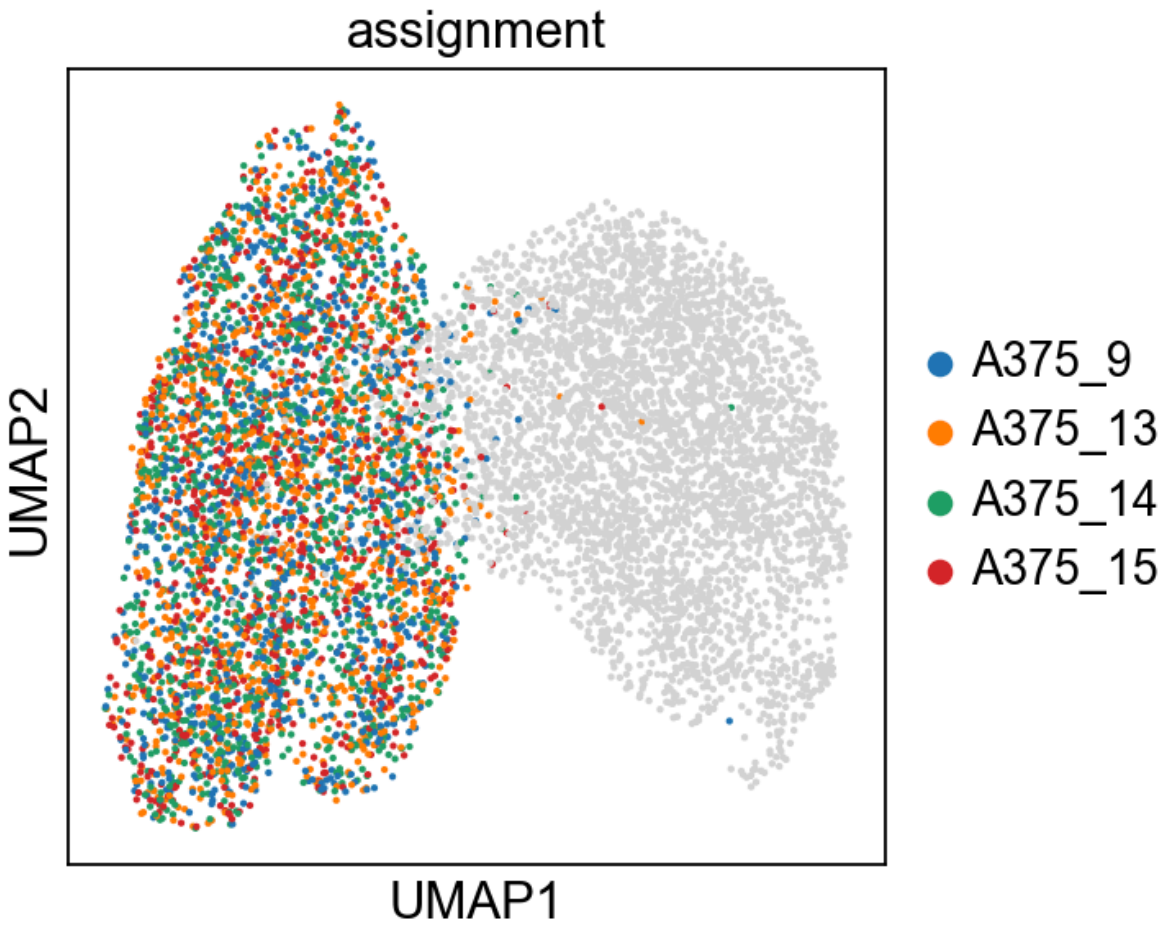
```
computing UMAP
    finished: added
    'X_umap', UMAP coordinates (adata.obsm) (0:00:09)
```

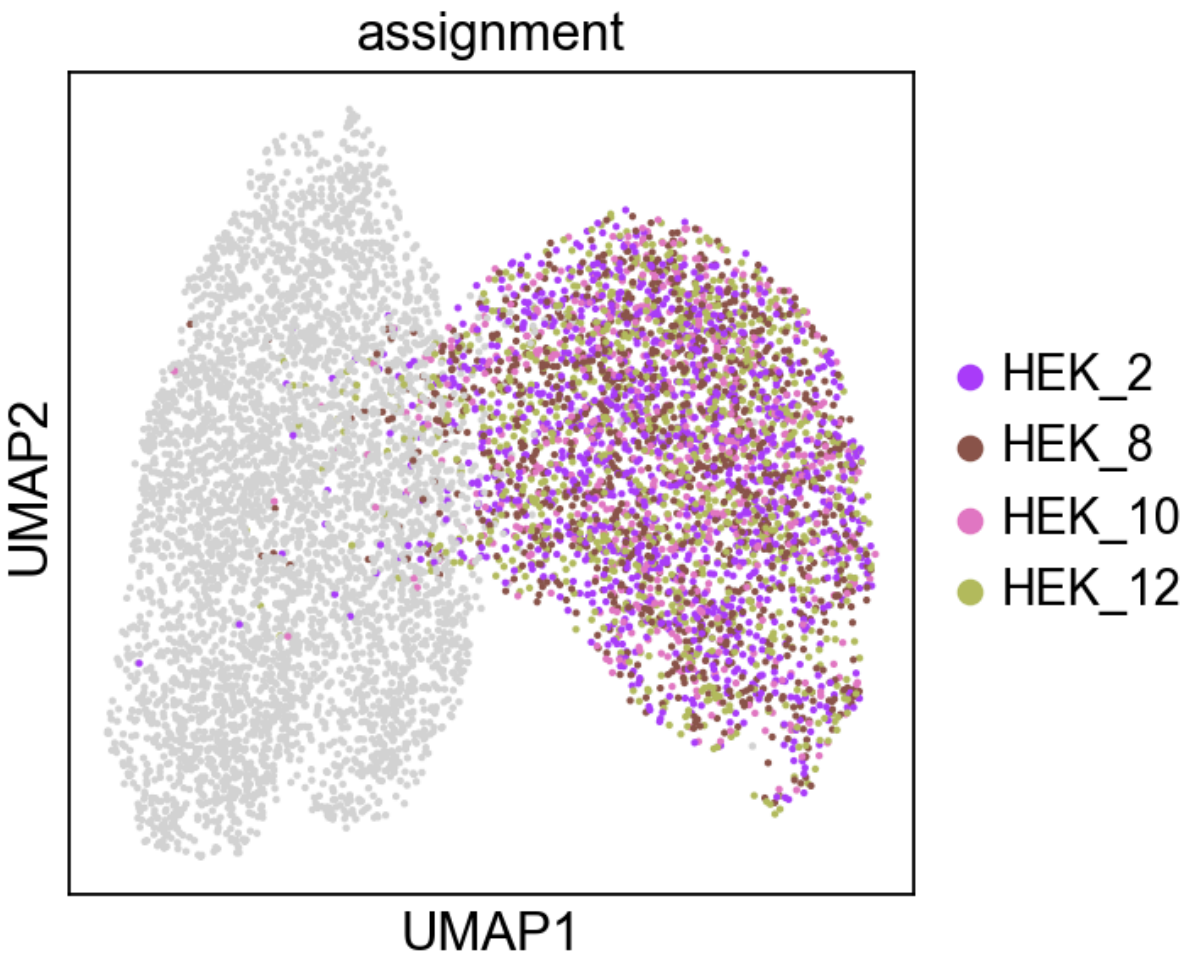
```
#visualize our clusters
sc.pl.umap(adata, color=['assignment'])
```



Part 1

Clustering continued



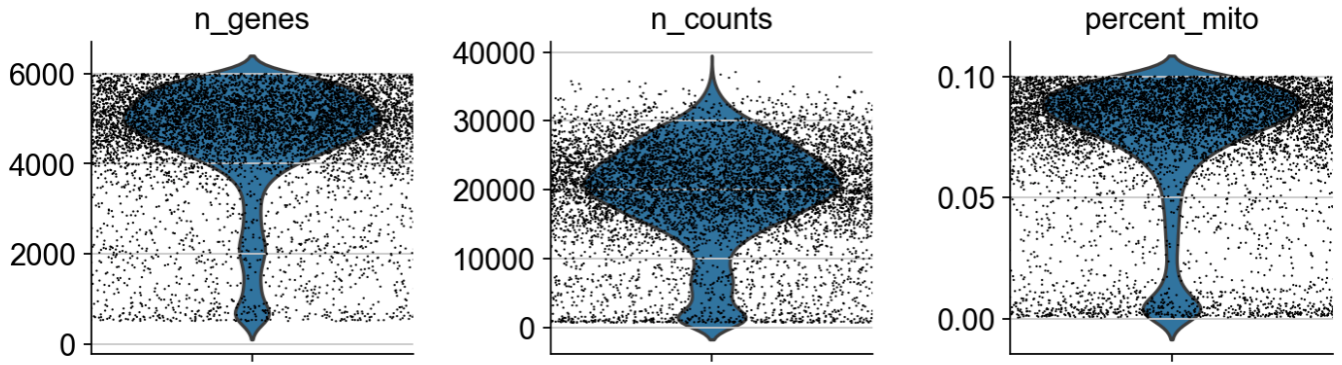


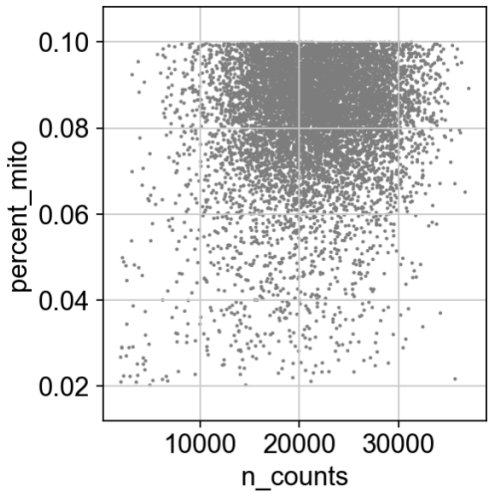
Part 2

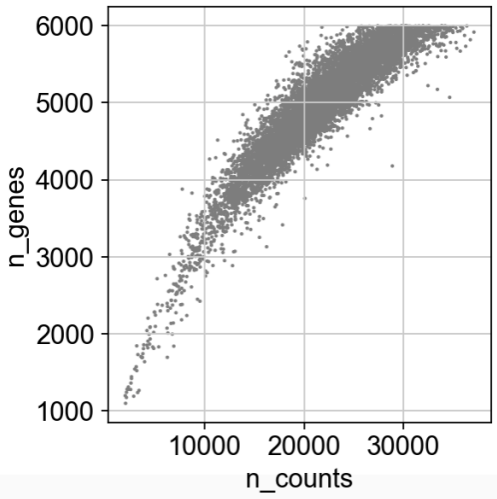
New data – HEK cells from a different experiment

```
#read in our new data
adataA=anndata.read_h5ad('/Users/caroline/Desktop/cumulus_output.h5ad')
adataA
```

```
AnnData object with n_obs × n_vars = 23651 × 27623
    obs: 'Channel', 'passed_qc', 'n_genes', 'n_counts', 'percent_mito', 'louvain_labels'
    var: 'gene_ids', 'n_cells', 'percent_cells', 'robust', 'highly_variable_features', 'mean', 'var', 'hvf_loess', 'hvf_rank'
    uns: 'Channels', 'PCs', 'W_pca', 'genome', 'pca', 'pca_knn_distances', 'pca_knn_indices'
    obsm: 'X_fitsne', 'X_pca'
```

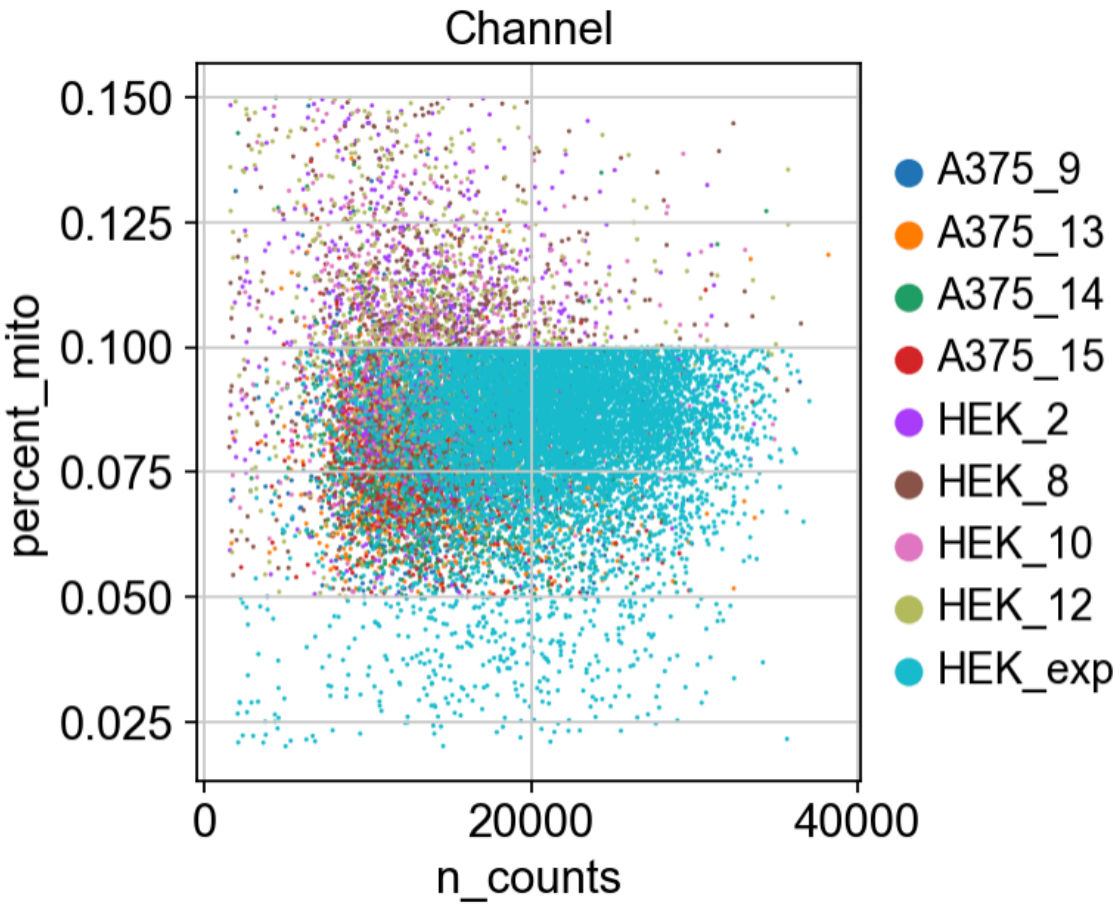


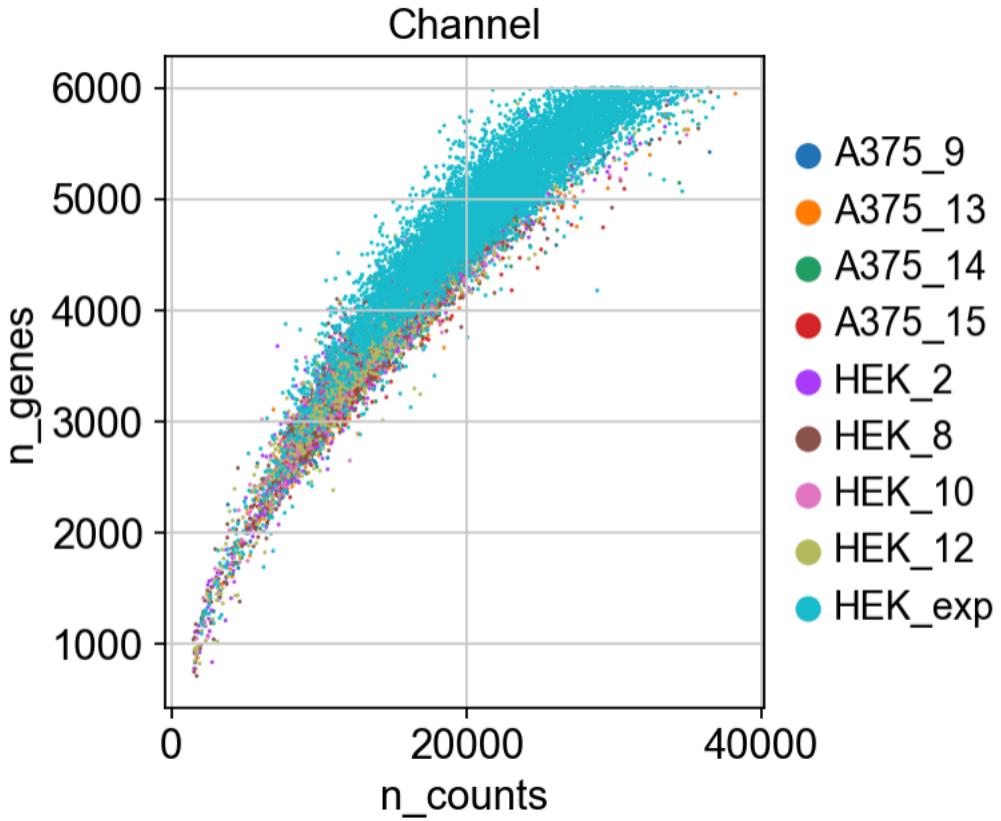




Part 2

Concatenate Part I & Part II

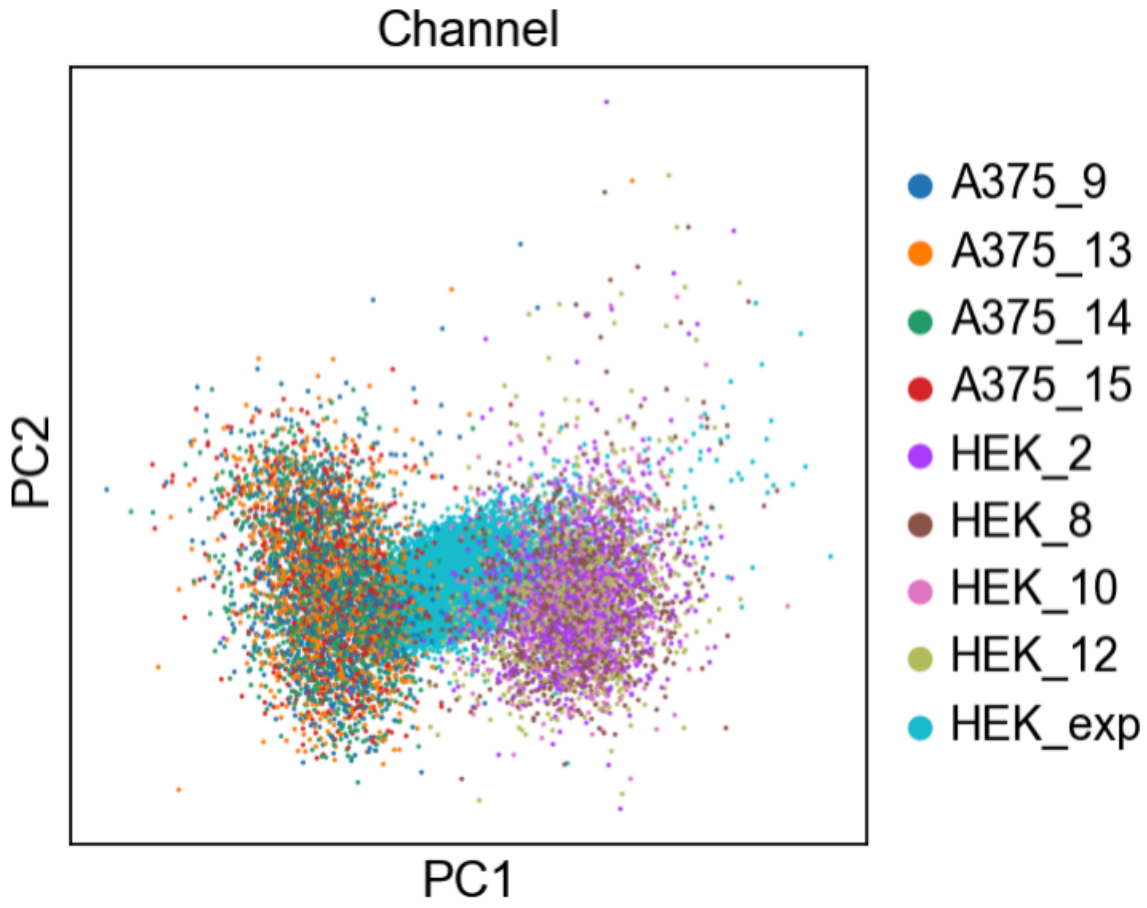


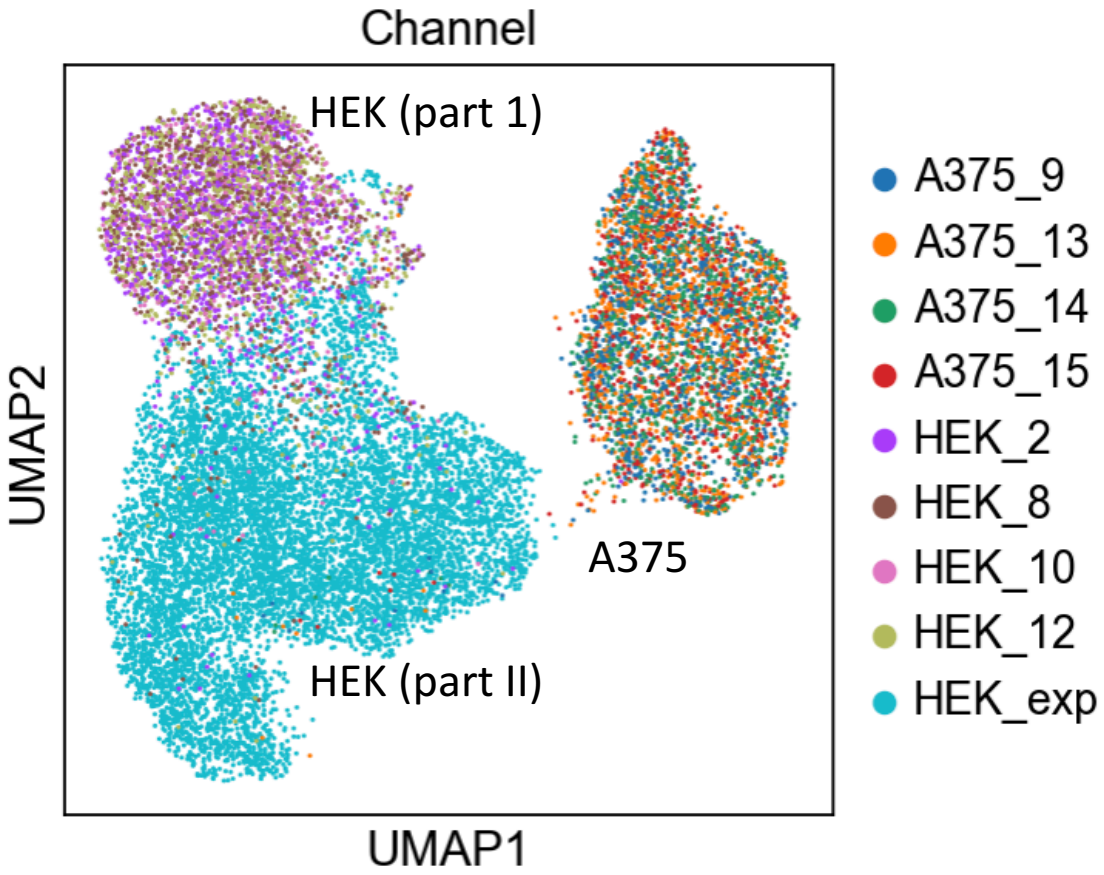


Clean up, normalize & scale as before

Part 2

PCA & Clustering





Conclusions

- HEK cell populations ~sort of~ cluster together, at least more than to A375 cells
- Model works as proof-of-concept but shouldn’t actually be used to classify unknown cells; training data was not ideal

Challenges:

- Pipeline duration – took several days to get the final count matrix
- Manually processing Part II data - different pipeline
- Logic with multi-dimensional matrices

Questions?