Lab 6: scRNA-seq

Skills: scRNA-seq, dimensionality reduction, using Python analysis packages

For this week you'll need to complete the following:

- `CSE185-LAB6-REPORT.ipynb` (90 pts)
- `CSE185-LAB6-README.ipynb` (10 pts)

Similarly to the previous lab, you will complete your report in `CSE185-LAB6-REPORT.ipynb` and should document any code you used to complete the lab in `CSE185-LAB6-README.ipynb`. Note there are no exercises this week in order to give you more time to write your project proposals.

Intro

In this lab, we will analyze scRNA-seq data of human pancreatic cells which were derived from stem cells. By analyzing single-cell data from multiple stages across the differentiation process all the way from stem cells to pancreatic islets, we can also learn about the genes that go up and down across the different stages of development.

We will look at single-cell RNA-seq data generated using 10X Genomics technology. Data is taken from the paper: Functional, metabolic and transcriptional maturation of human pancreatic islets derived from stem cells which it will be helpful for you to refer to (focus on Figure 5) as you go through the lab. The paper produces data for many stages of differentiation of stem cells into pancreas cells, and after those cells are transplanted into mice. We focus on just three of the time points to save computational time:

- Samples "GSM5114461_S6_A11" and "GSM5114464_S7_D20" are taken from two different time points (stages 6 and 7) of in vitro differentiation of stem cells into pancreas cells.
- Sample "GSM5114474_M3_E7" was taken 3 months postimplanation, after the cells were implanted into mice.

In this lab, we'll go through:

- Loading single-cell data into Scanpy.
- Basic filtering and QC
- Correcting for batch effects
- Using dimensionality reduction techniques (PCA), clustering (Leiden) and visualizations (UMAP, t-SNE) to visualize and identify cell type clusters

Summary of data provied

Data for this lab can be found in `~/public/lab6`. You should see the following datasets. which were generated by the 10X Cell Ranger pipeline. The file formats will be discussed during lecture.

- Stage 6 in vitro: `GSM5114461_S6_A11_matrix.mtx.gz`, `GSM5114461_S6_A11_features.tsv.gz`, and `GSM5114461_S6_A11_barcodes.tsv.gz`
- Stage 7 in vitro: `GSM5114464_S7_D20_matrix.mtx.gz`, `GSM5114464_S7_D20_features.tsv.gz`, and `GSM5114464_S7_D20_barcodes.tsv.gz`
- Month 3 postimplantation: `GSM5114474_M3_E7_matrix.mtx.gz`, `GSM5114474_M3_E7_features.tsv.gz`, and `GSM5114474_M3_E7_barcodes.tsv.gz`

Note, to save computational time we are not actually running Cell Ranger ourselves on the raw fastqs, and instead are starting from the counts matrix. Most single-cell papers will make the count matrices available on GEO. The count matrices used in this lab were taken from: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE167880.

Summary of computational tools

The majority of this lab will be completed using a Python library, scanpy, built for performing many types of single-cell analyses.

Another very population package for single-cell analyses is Seurat, which is written in R. You are welcome to use Seurat for this lab if you prefer R.

Most common single-cell analysis procedures are implemented in both of these libraries.

0. Setup

We'll first need to install several python packages we'll be using. In the past, we installed packages for you. Here, you'll instead install the packages on your own. Since you don't have root access to the file server, you'll instead have to install all packages locally, which means

they will only be accessible to you.

We can use pip to easily install python packages:

```
pip install --user scanpy harmonypy leidenalg
```

The option `--user` tells pip to install these packages only for your user. (If you run pip without this option, you will see an error that you do not have root access).

The following packages will be installed:

- scanpy: the main library we'll use for scRNA-seq analysis
- harmonypy: a package for integrating datasets from multiple sources while correcting for batch effects. Note, this is actually a port of a library originally written in R.
- leidenalg: a package for performing graph-based clustering.

We will not call harmonypy or leidenalg directly, but scanpy needs those installed to perform the analyses below.

To make sure scanpy installed correctly, open a Python terminal or Jupyter notebook and check this command runs without an error.

```
# We had to add the install path to sys.path to get these imports to work
import sys
sys.path.append(os.environ["HOME"]+"/.local/lib/python3.9/site-packages")

# Import the libraries we installed
import scanpy as sc
import harmonypy
import leidenalg
```

1. Loading the data

Scanpy stores the data in an AnnData object. Explore what an AnnData object is and the different methods available. The code below shows how to import scanpy, print out what versions of libraries it is using, and load one of the 10X datasets into an AnnData object.

```
import os
import scanpy as sc
sc.logging.print_versions()

DATADIR=os.environ["HOME"]+"/public/lab6"
dataset = sc.read_10x_mtx(DATADIR, prefix="GSM5114461_S6_A11_", cache=True)
```

Read more about the `read_10x_mtx` function here.

Here we will actually want to load all three datasets into a single anndata object. We can do this by "concatenating" multiple anndata objects. The code below shows you how we did this.

```
DATADIR=os.environ["HOME"]+"/public/lab6"
dsets = ["GSM5114461_S6_A11", "GSM5114464_S7_D20", "GSM5114474_M3_E7"]
adatas = {}
for ds in dsets:
    print(ds)
    adatas[ds] = sc.read_10x_mtx(DATADIR, prefix=ds+"_", cache=True)
combined = ad.concat(adatas, label="dataset")
combined.obs_names_make_unique()
```

Typing the following in individual Jupyter cells will print out some helpful info

```
combined # will print out the dimensions of the combined dataset loaded

adatas["GSM5114461_S6_A11"] # will print out dimensions of one of the individual datasets

combined.obs # will print out info about each cell.
             # You should see a "dataset" column indicating which dataset each cell came from
```

Note: `n_obs` gives the number of cells, `n_vars` gives the number of genes.

Question 1 (10 pts) What python library and version did you use to load in the feature barcode matrix? How many datasets did you load and where did they come from? How many genes and total cells were included in the loaded data for each dataset?

I am using scanpy 1.9.3 and anndata 0.9.1 to load in the feature barcode matrix. I also imported harmonypy and leidenalg (0.9.1), I loaded in three datasets. The first is GSM5114461_S6_A11 a sample from stage 6 in vitro differentiation of stem cells into pancreas cells and the second is GSM5114464_S7_D20 a sample from stage 7 in vitro differentiation of stem cells into pancreas cells. The third dataset

is GSM5114474_M3_E7 a sample taken 3 months postimplanation, after the cells were implanted into mice. For the GSM5114461_S6_A11 dataset, there are 4793 cells and 20621 genes. For the GSM5114464_S7_D20 dataset, there are 4910 cells and 20621 genes. Lastly, for the GSM5114474_M3_E7 dataset, there are 4910 cells and 20261 genes.

2. Filtering and normalizing your dataset

2.1 Initial filtering

Before we start the analysis, we need to preprocess the data and filter out the poor quality parts of the matrix. We will perform two levels of filtering:

- Filtering *cells*: We will want to filter cells (columns) that don't look very reliable. For example, one sign a cell didn't get sequenced very well is if not that many genes are expressed (lots of zero counts).
- Filtering *genes*: We will want to filter genes (rows) that are not expressed in at least some of our cells since those won't be very interesting. We will also filter genes that are not expressed highly enough for us to get good data.

Filter out the cells that have less than 200 genes expressed, cells that have less than 1000 total reads, genes that are detected in less than 5 cells, and genes that have a total count of less than 15. You may find the functions `sc.pp.filter_cells` and `sc.pp.filter_genes` useful (see https://scanpy.readthedocs.io/en/stable/api.html for options available and more on how to use these). For example:

`sc.pp.filter_cells(combined, ....)`
will perform the specified filtering and update the AnnData object `combined` in place.

Question 2 (10 pts) Report any filtering steps you did to remove low quality cells or genes based on the filters described above. Report the number of cells and number of genes you remaining after the filtering steps above.

To remove low quality cells I first called sc.pp.filter_cells on combined dataset with the parameter min_genes = 200 to filter out cells that have less than 200 genes expressed and update combined. I called that function on combined again with the parameter min_counts = 1000 to filter cells that have less than 1000 total reads. I then called the function sc.pp.filter_genes two times on combined dataset with the parameter min_counts = 15 the first time and then parameter min_cells = 5 the second time. This was to filter out genes that are detected in less than 5 cells and that have a total count of less than 15. After the filtering steps, there are 10133 cells and 15779 genes

2.2 Filtering cells with high mitochondria gene expression

Let's next look at the most highly expressed genes in the dataset. You can use `sc.pl.highest_expr_genes(combined, n_top=20)` to plot the top 20 highest expressed genes. This command will show the genes along the y-axis and the distribution of their counts per cell along the x-axis. You should see:

- INS (Insulin) is the most highly expressed gene. This makes sense! We are looking at pancreas-like cells afterall.
- A lot genes with names like "MT-CO1", "MT-CO2", etc.

These genes starting with "MT-" are expressed from mitchondria, which are circular pieces of DNA present in cells at high copy number. High numbers of mitochondrial transcripts are indicators of poor sample quality. This could mean the cell is undergoing apoptosis (dying) or for some reason has higher than normal metabolic activity. For our analysis, this is not the case and we wouldn't want to cluster our cells based off of cells' stress levels. So, we would like to filter cells for which a high percentage of reads are coming from mitochondrial genes.

Follow the steps in this tutorial (or elsewhere online) to:

- Determine the percent of counts in each cell that are from mitochondrial genes.
- Visualize violin and scatter plots of QC metrics including the percent mitochondria per cell, the count number per cell, and the number of genes per cell.
- Filter cells with a high percentage of counts from mitochondrial genes. The paper we got the data from suggested using 25% as a threshold.
- Determine if there is any additional filtering you'd like to do to get rid of outlier cells.

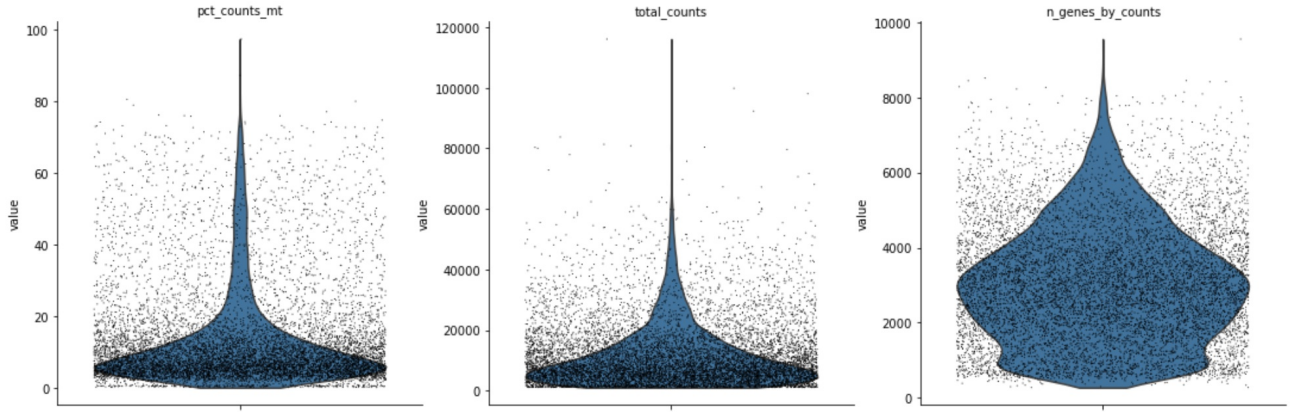
Note, you can use the general syntax below to get a filtered anndata object:

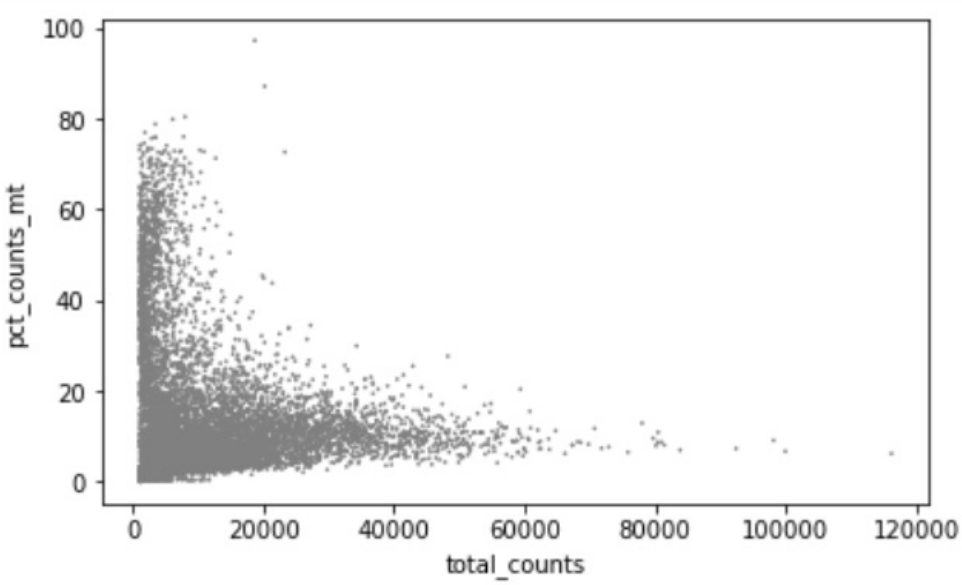
```
#keep cells matching these criteria. keep all genes (":" means all)
adata_filt = combined[(combined.obs[col1]<threshold1) & (combined.obs[col2]<threshold2), :]
```

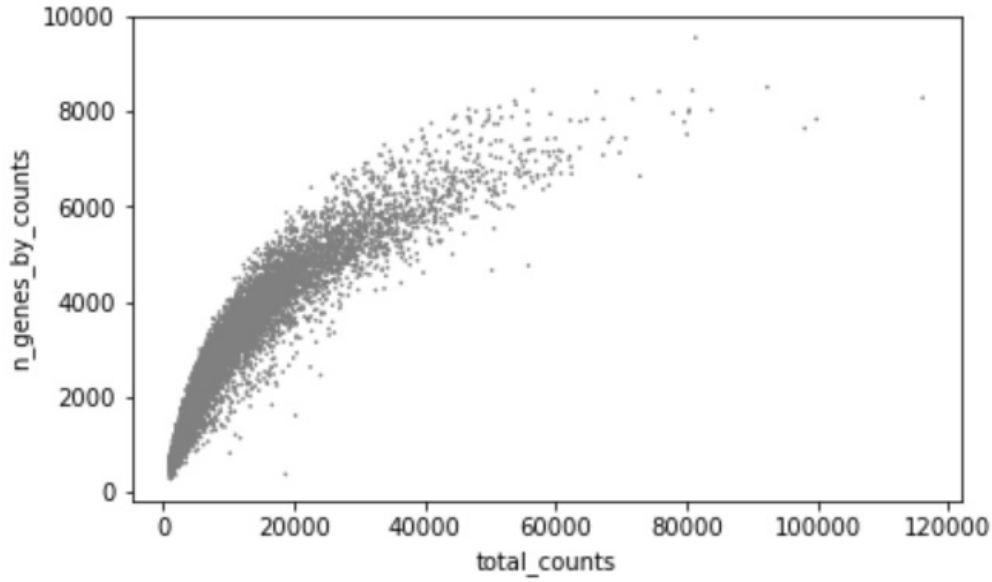
Question 3 (10 pts) Describe the steps you took to filter cells with a high percent of mitochondrial genes. If you did any other filtering, describe that here as well. Report the number of cells and number of genes you remaining after the filtering steps above. Include violin and scatter plots you generated to justify your filtering steps.

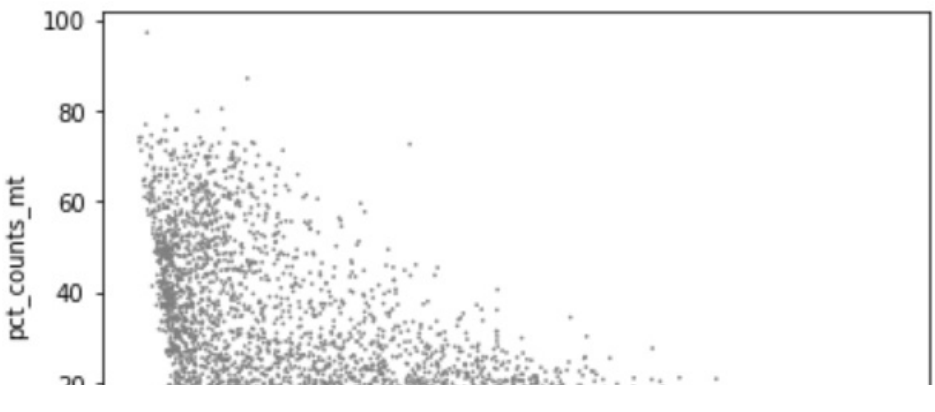
To filter out cells with a high percentage of mitochondrial genes, I first determined the percent of counts in that are mitochondrial genes from each cell by filtered all the variables that start with MT for mitochondrial then I called sc.pp.calculate_qc_metrics() on it. Then, I called sc.pl.violin() & sc.pl.scatter() to create plots for the percent mitochondria per cell, the count number per cell, and the number of

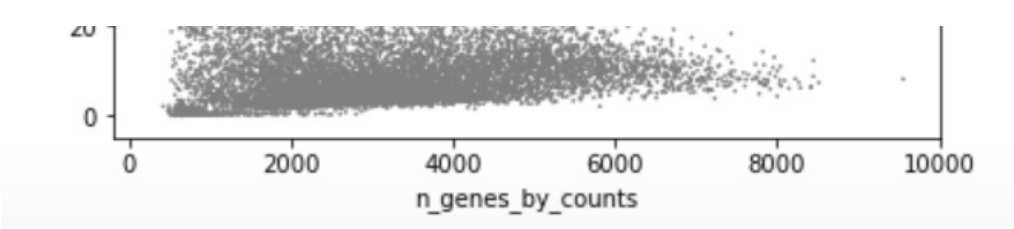
genes per cell. Lastly, I filtered out cells with a percentage of counts from mitochondrial genes of over 25% and I filtered out cells with total counts over 75,000 in order to filter out cells that are low quality.











2.3 Normalizing counts

Finally, we'll need to do some normalization so we can compare expression across cells below. Before normalization, the total reads derived from each cell may differ substantially. We will want to transform the data by dividing each column (cell) of the expression matrix by a “normalization factor,” an estimate of the library size relative to the other cells. It is also standard practice to log transform our data to decrease the variability of our data and transform skewed data to approximately conform to normality.

Total-count normalize (library-size correct) the data matrix to 10,000 reads per cell, so that counts become comparable among cells. Then logarithmize the data. The code below shows how to do this in Scanpy. You should run these steps before proceeding (there are no points for this, but you still need to run the steps below before you move on).

```
sc.pp.normalize_per_cell(adata_filt, counts_per_cell_after=1e4) # normalize to 10,000 reads/cell
sc.pp.log1p(adata_filt) # log transform
```

3. Identifying highly variable genes

In our clustering analysis below, we will want to focus on the genes that are most variable across cells. If a gene is expressed at the same level across all cells, it won't be very interesting.

We will use **dispersion** to quantify the variability of each gene. Dispersion is a measure of how "stretched" or "squeezed" a distribution is and is typically computed as "variance/mean" (other metrics are sometimes used). Higher dispersion means higher variability. In Scanpy, genes are first binned based on mean expression levels. Normalized dispersion for each gene is then computed as the absolute difference between the gene's dispersion and the median dispersion of genes in that bin, divided by the median deviation within each bin. This means that for each bin of mean expression, highly variable genes are selected.

The scanpy function `highly_variable_genes` (see here) is useful for finding genes with highest dispersion. We recommend using that function with the following options:

- `batch_key="dataset"` : This means to select highly variable genes separately within each of our three datasets.
- `n_top_genes=500` : This will select only the top 500 most variable genes. (The paper used more than this, but using fewer genes will make the rest of our analyses go faster).

Note, in addition to `obs` , which contains data *per cell*, the AnnData object has `var` , which has data *per gene*, also in a pandas dataframe. The function above will add several variables to this data frame including:

- `highly_variable` , which is a boolean vector with one entry per gene. It is set to True for the highly variable genes based on the values we used in the function above.
- `dispersions_norm` : normalized dispersion for each gene.

Type `adata_filt.var` to see the data frame.

Question 4 (10 pts) Describe any methods you used to find highly variable genes. How many genes are in your highly variable set? What are the top 5 most variable genes? Why do we only care about the genes that differ between the cells?

To find highly variable genes, I used the method sc.pp.highly_variable_genes() with "dataset" set to the batch_key option and 500 set to the n_top_genes to select only the top 500 most variable genes. I then use adata_fily.var.sort_values() function to sort in descending order then I indexed the first 5 in order to return the top 5 most variable genes. These genes are 'PPY', 'NPY', 'LYZ', 'KRT17', and 'NTS'. We only care about the genes that differ between cells because we are looking at the different gene expression for each cell in order to see any significant differences. If a gene is expressed the same over multiple cell samples then we do not care about this gene.

For the analyses below, we recommend making a new anndata object, which contains only:

- Highly variable genes
- Genes in the set of cell-type specific marker genes used in the paper (see below). We will manually add these back, since we want to analyze them even if they didn't make the cut for being most differentially expressed.

You can create a new AnnData object with only these genes using:

```
# We'll manually add these genes to make sure they stay in our
# dataset for the analyses below.
genes = ["GCG", "TTR",  "IAPP",  "GHRL", "PPY", "COL3A1",
    "CPA1", "CLPS", "REG1A", "CTRB1", "CTRB2", "PRSS2", "CPA2", "KRT19", "INS","SST","CELA3A",
"VTCN1"]
```

```
adata_var = adata_filt[:, (adata_filt.var.index.isin(genes) | adata_filt.var["highly_variable"])]
```

4. Removing batch effects

Our dataset above is combined across three separate single-cell experiments. Whenever we combine data from different sources, there is a possibility of introducing "batch" effects, in which there are systematic differences between them due to technical reasons (e.g. they were handled by a different technician, performed on a different machine, collected at different times of day, etc.).

To visualize batch effects, let's first perform principal components analysis (PCA) on our dataset and plot the data along the first two PCs. You can use the function `sc.pp.pca` with option `n_comps=20` to compute only the first 20 PCs, and then `sc.pl.pca(adata_var, color="dataset")` to plot the data, coloring each cell based on the dataset it comes from. You should see some evidence of batch effects in your PCA plot.

Now, we'd like to adjust the count data to control for batch effects. For this, we'll use Harmony, which works by adjusting the PCA embeddings. (So, you must perform the PCA step above before running Harmony). You can run Harmony from within scanpy:

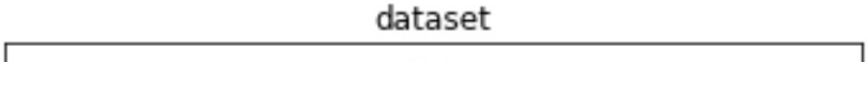
```
# Import the "external" library
import scanpy.external as sce

# Run harmony using suggested params from the paper
sce.pp.harmony_integrate(adata_var, 'dataset', theta=2, nclust=50,  max_iter_harmony = 10,
max_iter_kmeans=10)

# Reset the original PCs to those computed by Harmony
adata_var.obsm['X_pca'] = adata_var.obsm['X_pca_harmony']
```

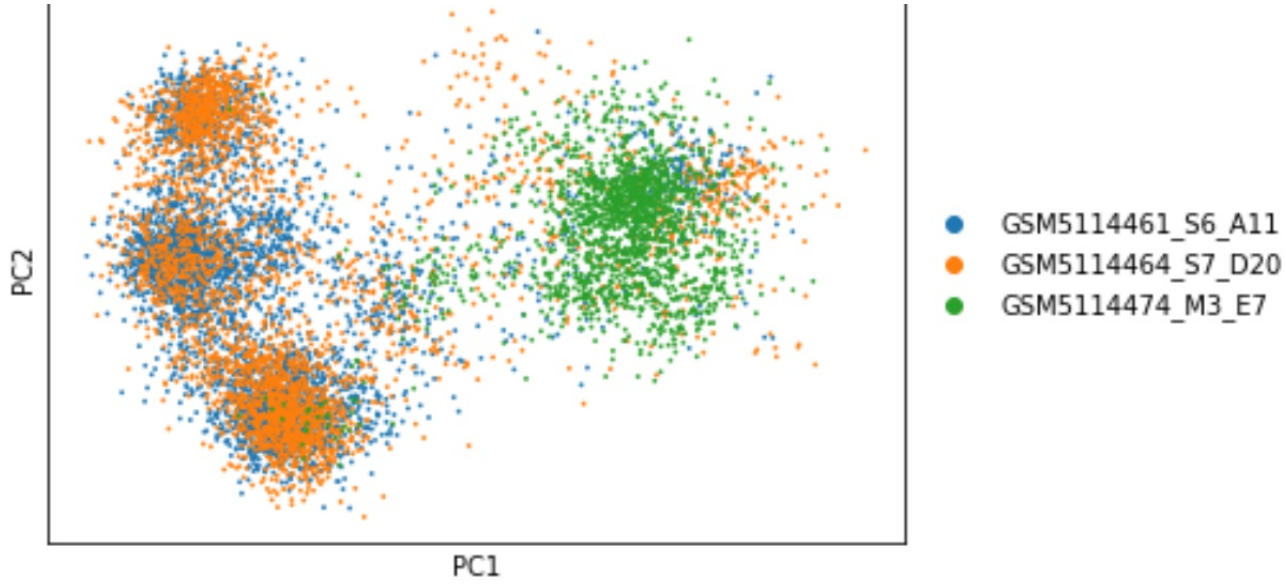
The code above uses Harmony to adjust the PCs, then sets the new PCs to those computed by Harmony. Make a new PCA plot on these adjusted PCs. You should see some of the batch effects seen before are now corrected.

Question 5 (10 pts) Describe how you performed batch correction on your dataset. Which tool did you use? Which version? What parameters did you set and what do they mean? Show the PCA plots before and after batch correction. Describe any overall trends you see in the PCA plot (e.g., is one dataset very different than the rest?)

I performed batch correction on my dataset by first calling sc.pp.pca() and sc.pl.pca() on adata_var with options n_comps=20 and color=dataset. This plot looks like:

I then did batch correction by calling sce.pp.harmony_integrate() on adata_var, 'dataset' to use the dataset column, theta=2 for batch and cell-type balance, nclust=50 for k-means clustering, max_iter_harmony=10 for maximum of 10 harmony iterations, max_iter_kmeans=10 for a maximum of 10 k-means clustering. I then plotted again with these corrections and in conclusion, S6_A11 & S7_D20 overlap each other and seem to be correlated and for the most part are very similar. However, M3_E7 is way more spread out than the other two and doesn't overlap the other two datasets as much. The new corrected plot looks like:





5. Visualizing cell clusters

We will perform clustering on our data to identify individual cell types, and visualize the results using two different methods: t-SNE and UMAP.

To perform clustering (if you're using Scanpy), we'll need the following commands:

```
sc.pp.neighbors(adata_var) # computes neighborhood graphs. Needed to run clustering.
sc.tl.leiden(adata_var) # clusters cells based on expression profiles. This is needed to color
cells by cluster.
```

Defaults for these functions worked ok for us. However, you may wish to play around with parameters to these functions. e.g. the option `n_neighbors` to `sc.pp.neighbors` controls how the nearest neighbor graph is built. There are other parameters you can modify for clustering here.

Now, you can use the following functions to visualize your clusters using either UMAP or tSNE:

- UMAP

```
sc.tl.umap(adata_var) # compute UMAP embedding
sc.pl.umap(adata_var, color="leiden") # make the UMAP plot, coloring cells by cluster
```

- tSNE

```
sc.tl.tsne(adata_var)
sc.pl.tsne(data, color=['leiden'], legend_loc='on data', legend_fontsize=10, alpha=0.8,
size=20)
```

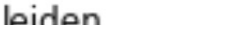
For the plotting functions (`sc.pl.umap` and `sc.pl.tsne`) you can change `color` to color cells by different attributes. For example:

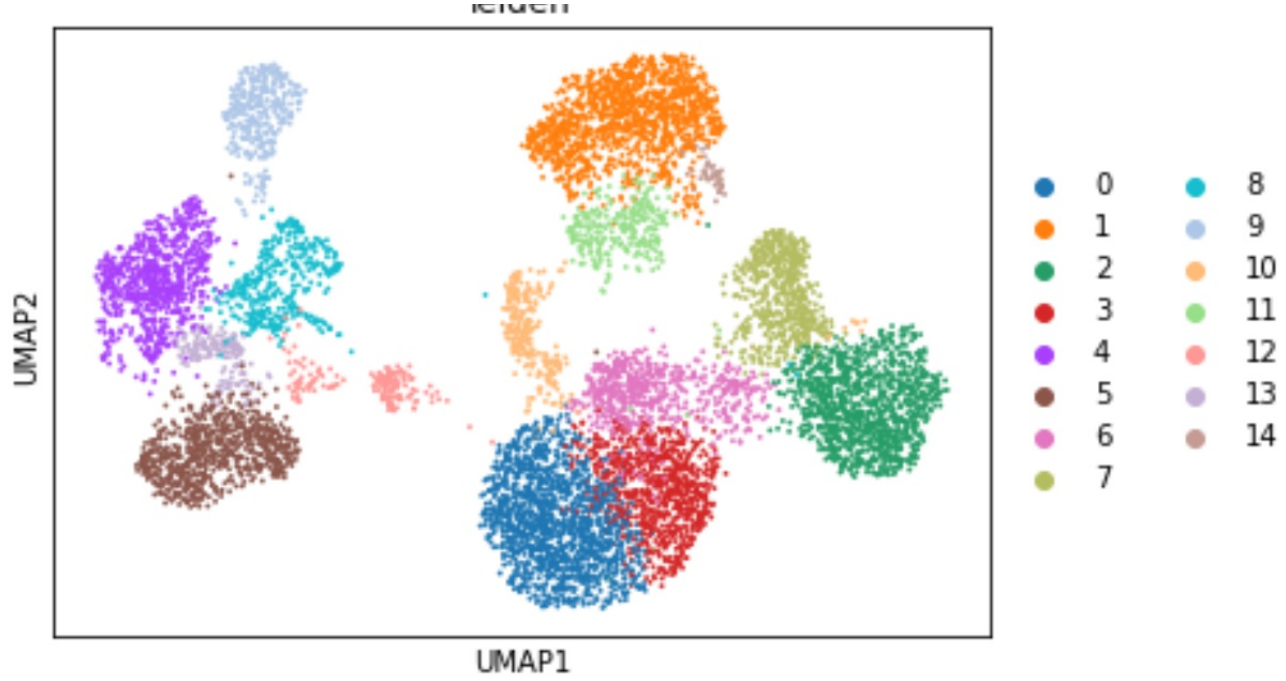
- `color="leiden"` colors cells by their cluster assignment
- `color="dataset"` colors cells by the dataset they came from
- `color="INS"` colors cells by their expression level of the gene INS.

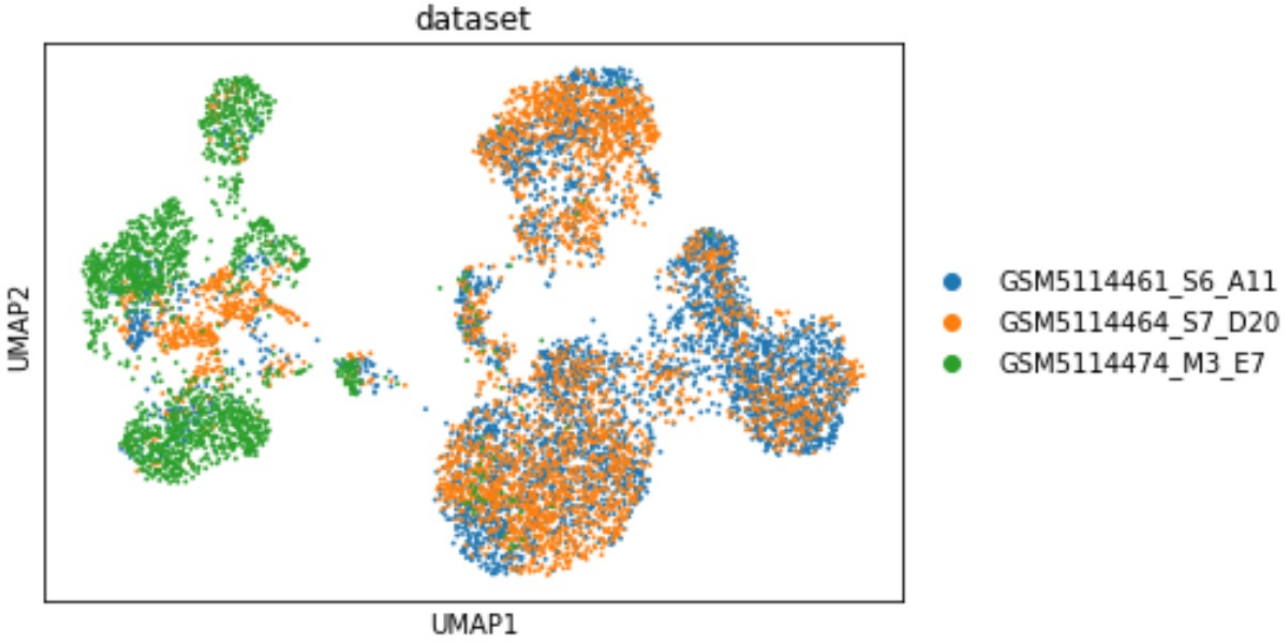
Question 6 (10 pts) Describe how you performed clustering on your dataset. Show UMAP and t-SNE plots colored by cluster assignment vs. colored by the dataset of origin. How do your results compare to Fig. 5 of the paper? (https://www.nature.com/articles/s41587-022-01219-z.pdf) Do any clusters contain cells from multiple datasets? There should be some overlap (i.e., some cell types will be present in more than one dataset). But, you will likely also find the cell types are pretty distinct across datasets, especially for "M3" compared to "S6" and "S7".

I performed clustering on my dataset by first calling sc.pp.neighbors() and sc.tl.leiden() on adata_var which will compute the neighborhood graph then will color cells by cluster and expression profile. Then, I called sc.tl.umap() on adata_var to create a UMAP visualization then callled sc.pl.umap() on adata_var and set color to 'leiden' which colors by cluster. Then, I called sc.pl.umap() again on adata_var but color by dataset. These created my UMAP Visualizations. Then, I repeated the previous steps by with tSNE. So, I called sc.tl.tsne() on adata_var then called sc.pl.tsne() on adata_var which colors by leiden and changes the data to be labeled. Lastly, I called sc.pl.tsne() again but colored by dataset.

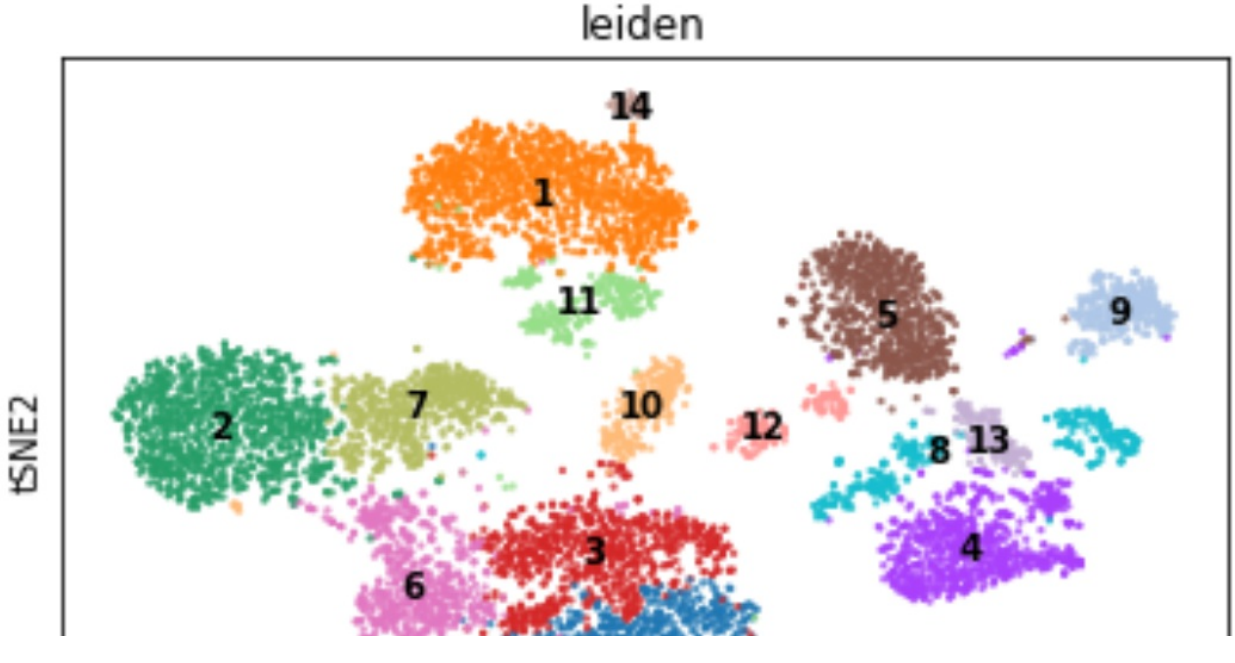
The two UMAP visualizations are:

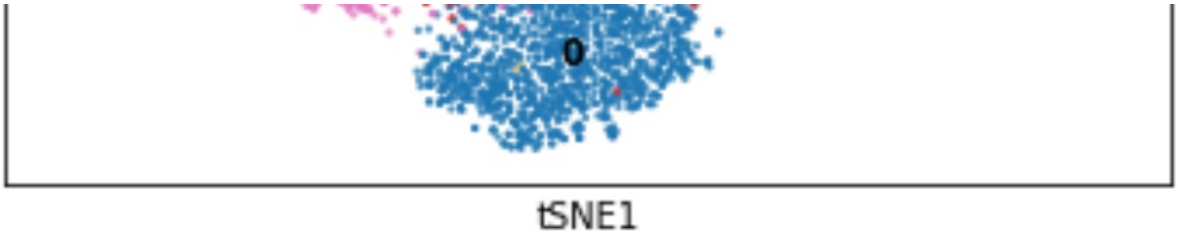


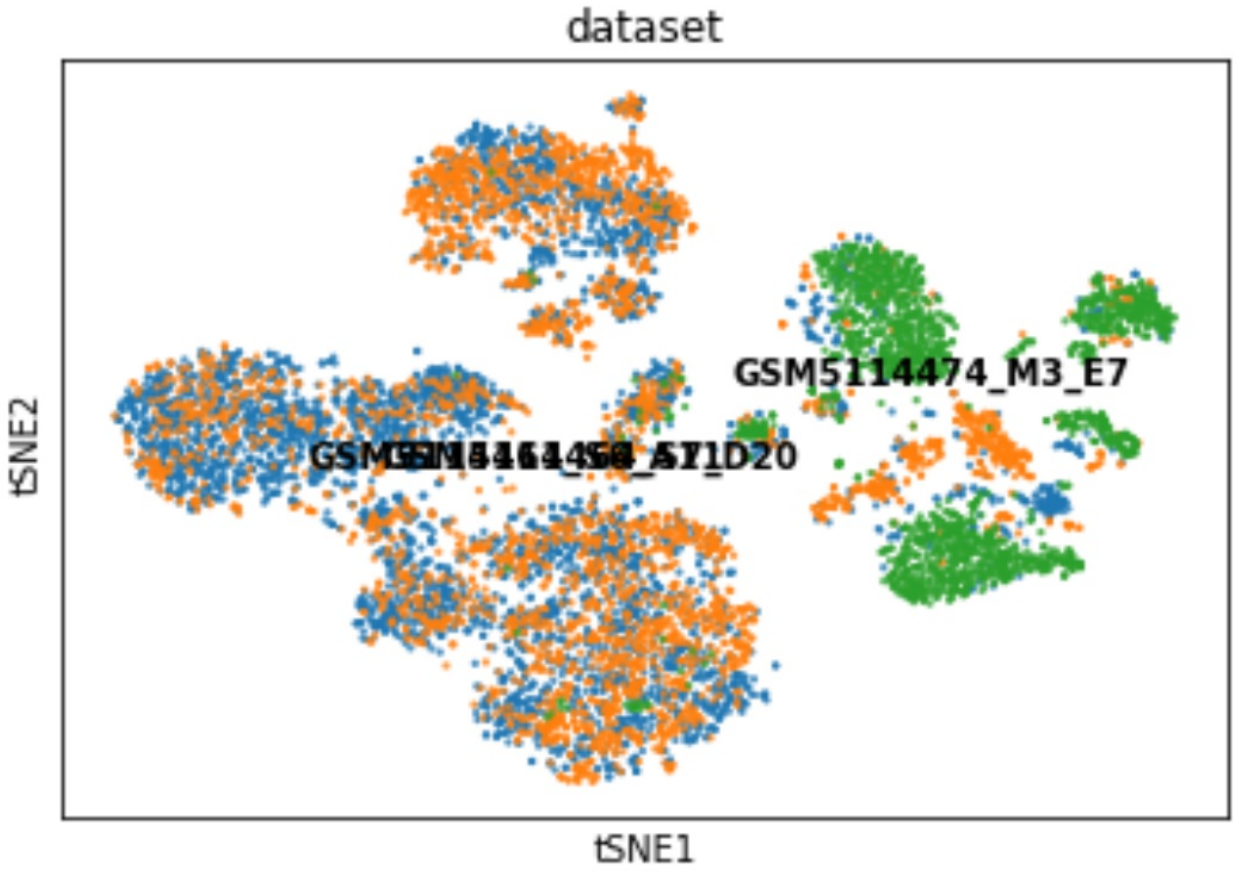




The two tSNE visualizations are:







For the tSNE and UMAP plots that are colored by dataset, S6_A11 & S7_D20 are sort of clustered together while M3_E7 is off to the side by itself. This is similar to the previous question's plots however these visualizations are more spread out as it seems like the M3_E7 is more distinct across the datasets compared to S6_A11 and S7_D20.

6. Assigning cell types to clusters

Finally, we'd like to try and assign cell types to some of our clusters. One way to do this is to use a set of known marker genes that are known to be expressed in certain cell types. The authors list the marker genes they used in their methods section, which we have listed below.

```
genes = ["GCG", "TTR",  "IAPP",  "GHRL", "PPY", "COL3A1",
    "CPA1", "CLPS", "REG1A", "CTRB1", "CTRB2", "PRSS2", "CPA2", "KRT19", "INS","SST","CELA3A",
"VTCN1"]
```

For example:

- GCG is a marker for alpha cells, which secrete the hormone glucagon
- SST is a marker for delta cells, which secrete the hormone somatostatin
- INS is a marker for beta cells, which produce insulin and are the most abundant of the islet cells

Other genes in this list are markers for unrelated cell types that might make it into the sample through contamination. e.g. COL3A1 is a collagen gene which is expressed highly in fibroblasts, and KRT19 is a marker gene for epithelial cells.

As we mentioned above, you can use a command like the following to color cells by the expression of a certain gene (or list of genes):

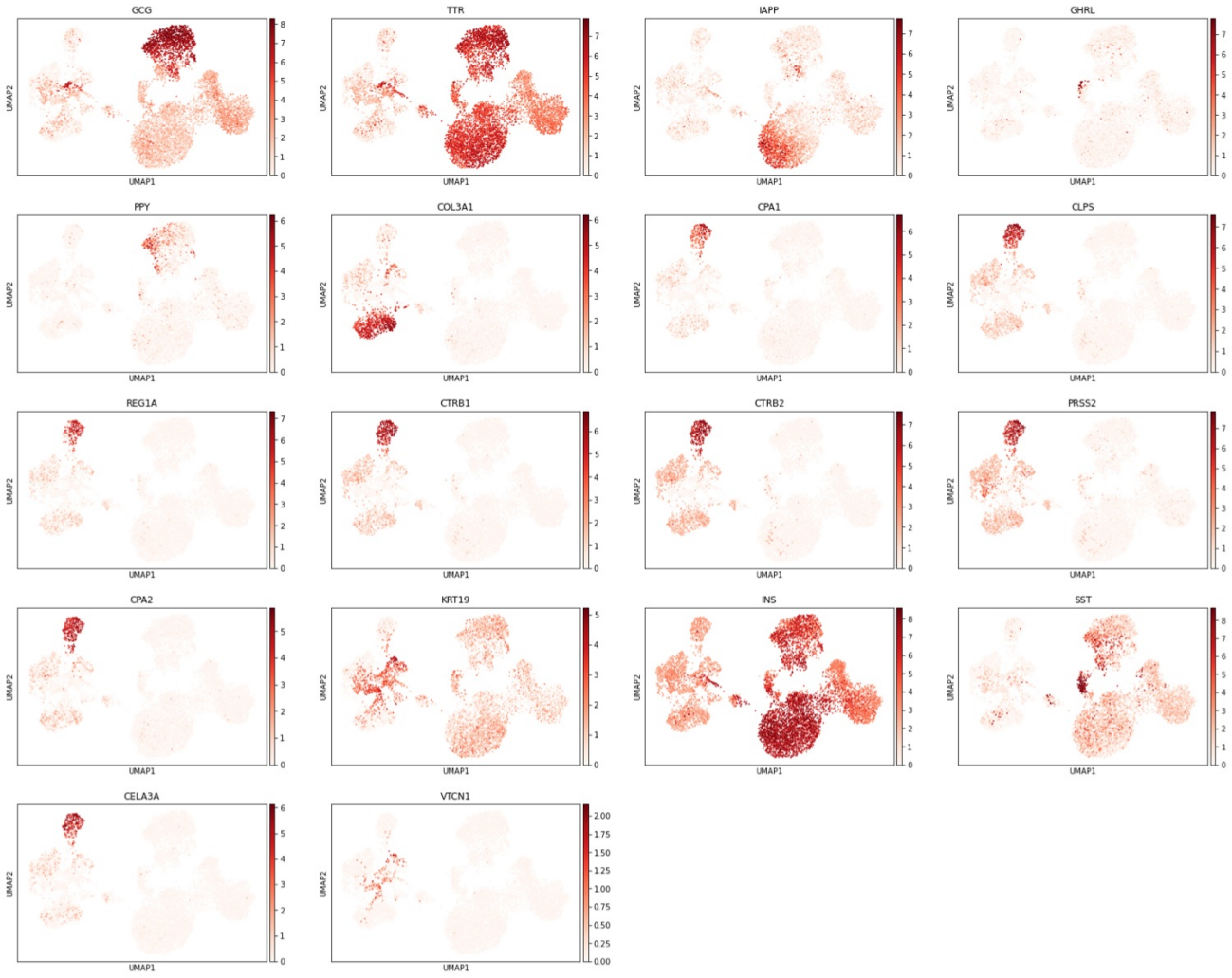
```
sc.pl.umap(adata_var, color=["INS","GCG","SST"], color_map="Reds")
```

You can also make a heatmap to show the expression of a set of genes by dataset or by cluster:

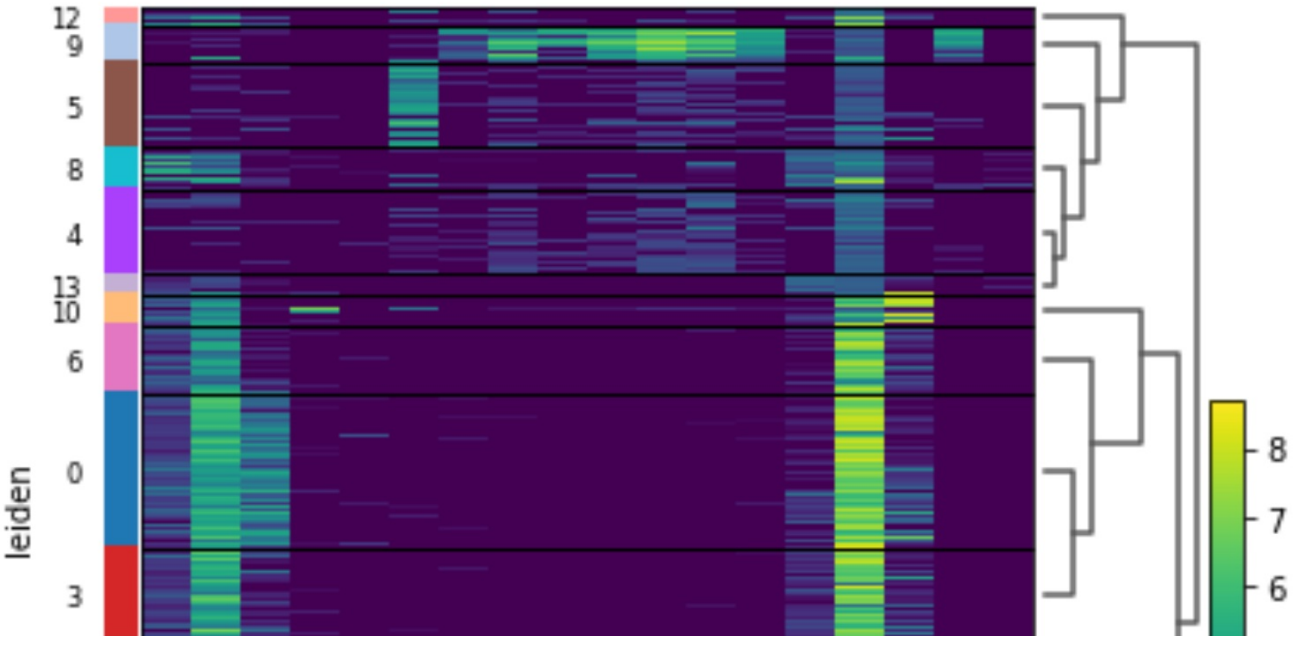
```
sc.pl.heatmap(adata_var, genes, groupby='leiden', dendrogram=True)
sc.pl.heatmap(adata_var, genes, groupby='dataset', dendrogram=True)
```

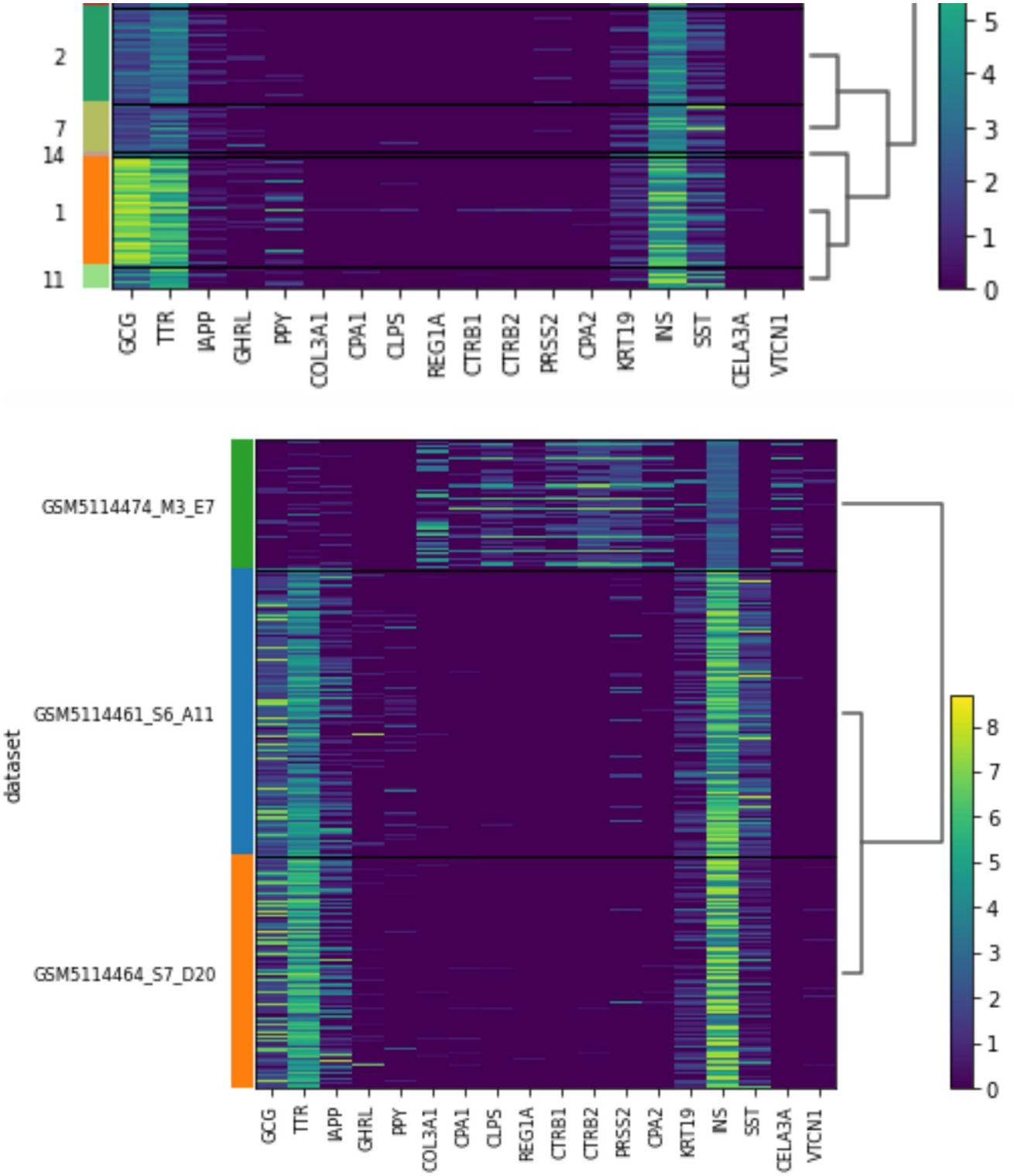
Question 7 (10 pts) Use expression patterns of the marker genes to assign your clusters to individual cell types. You should identify at least three different cell types your clusters correspond to. You may wish to refer to the original paper for more info on which genes are markers for which cell types. Include a plot (tsne or UMAP) where you label the cell types you identified.

To assign my clusters to individual cell types, there are at least three different cell types that my clusters correspond to. These are alpha cell types, beta cell types, and delta cell types. The following is the heatmap I did my analysis on:



Based on the above, there are alpha cells in S6_A11 & S7_D20 due to the high expression of SST & GCG while there are beta cells in S6_A11 & S7_D20 as well due to the high expression of INS. Also, due to the high expression of SST & GCG then there are most likely beta cells too. However, the low expression of SST & GCG show that there are maybe alpha and delta cells in M3_E7. But, there is low expression of M3_E7 in INS so most likely includes beta cells. Lastly, M3_E7 has high expression of CTRB1, CTRB2, and PRSSS2.





7. Discussion questions

Question 8 (7 pts) Summarize overall differences in terms of the cell types you see in the earlier in vitro (S6/S7) vs. later post-implantation (M3) stages.

The overall differences in terms of cell types compared to in-vitro (S6/S7) vs. post-implantation (M3) is that there are more likely to be alpha, beta, delta, and epsilon cells in the earlier in-vitro. However, in the later post-implantation there are more likelt to be alpha beta and sc beta cells.

Question 9 (7 pts) Did you identify any cell types in your clusters that are not related to the pancreas, and thus might have arisen through contamination? Are those more prevalent in the S6/S7 or M3 dataset? Hypothesize why.

One cell type in my clusters that are not related to the pancreas is thee COL3A1. The COL3A1 support cell tissues and makes collagen. Thus, I think it is possible that COL3A1 may have risen through contamination because it is way more expressed in the M3_E7 so it is possible that during the implantation, this cell was contaminated and introduced through the tissue that the implantation occured. Also, since COL3A1 supports cell tissues then it would make sense for it to have entered through its own environment.

Question 10 (6 pts) Read through the methods section of the paper titled "scRNA sequencing analysis". Describe at least two steps the authors took prior to obtaining their final UMAP plot that we did not include in our own analysis. Hypothesize how that might impact the

resulting clusters you identified.

One method that the author took priot to obtaining their final UMAP plot that we did not include was to predict RNA velocities. This could have helped with quality check as some data might have been filtered out which may have improved the quality and accuracy of the UMAP plot. Th

Loading [MathJax]/extensions/Safe.js