CSE185 Lab 6 Report - Code Documentation (10 pts)

- Document any commands used or additional analysis steps below!
- You should include enough detail that the instructors (or your future self) could come back to this several months from now and know exactly what you did and why you did it.
- We will not run this notebook, but will look back to see what you did especially if you end up with different answers.

For grading purposes only - Do not copy or edit this cell!

Part 1 - Loading Data

Import scanpy, print out what versions of libraries it is using, load one of the 10X datasets into an AnnData Object.

In [1]:
```
# We had to add the install path to sys.path to get these imports to work
import sys
import os
sys.path.append(os.environ["HOME"]+"/.local/lib/python3.9/site-packages")

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

In [2]:
```
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)
```

```
-----
anndata     0.9.1
scanpy      1.9.3
-----
PIL                 8.3.1
anyio               NA
attr                21.2.0
babel               2.9.1
backcall            0.2.0
beta_ufunc          NA
binom_ufunc         NA
bottleneck          1.3.2
brotli              NA
certifi             2022.12.07
cffi                1.14.6
chardet             4.0.0
charset_normalizer  2.0.0
cloudpickle         1.6.0
colorama            0.4.6
cycler              0.10.0
cython_runtime      NA
cytoolz             0.11.0
dask                2021.06.2
dateutil            2.8.2
debugpy             1.6.7
decorator           5.0.9
defusedxml          0.7.1
entrypoints         0.3
fastjsonschema      NA
fsspec              2021.07.0
google              NA
h5py                3.3.0
harmonypy           NA
idna                3.1
igraph              0.10.4
ipykernel           6.17.1
ipython_genutils    0.2.0
jedi                0.18.0
jinja2              3.0.1
joblib              1.0.1
json5               NA
jsonschema          3.2.0
jupyter_server      1.23.6
jupyterlab_server   2.6.1
kiwisolver          1.3.1
leidenalg           0.9.1
llvmlite            0.36.0
markupsafe          2.0.1
matplotlib          3.4.2
matplotlib_inline   NA
mpl_toolkits        NA
natsort             8.3.1
nbclassic           NA
nbformat            5.8.0
nbinom_ufunc        NA
numba               0.53.1
numexpr             2.7.3
numpy               1.21.1
packaging           21.0
pandas              1.5.3
parso               0.8.2
pexpect             4.8.0
pickleshare         0.7.5
pkg_resources       NA
plotly              5.14.1
prometheus_client   NA
prompt_toolkit      3.0.19
psutil              5.8.0
ptyprocess          0.7.0
pvectorc            NA
pydev_ipython       NA
pydevconsole        NA
pydevd              2.9.5
pydevd_file_utils   NA
pydevd_plugins      NA
pydevd_tracing      NA
pyexpat             NA
pygments            2.9.0
pyparsing           2.4.7
pyrsistent          NA
pytz                2021.1
requests            2.26.0
rfc3339_validator   0.1.4
rfc3986_validator   0.1.1
ruamel              NA
scipy               1.7.0
send2trash          NA
session_info        1.0.0
six                 1.16.0
sklearn             0.24.2
sniffio             1.2.0
socks               1.7.1
storemagic          NA
tblib               1.7.0
terminado           0.10.1
texttable           1.6.7
threadpoolctl       2.2.0
tlz                 0.11.0
toolz               0.11.1
tornado             6.2
traitlets           5.1.1
typing_extensions   NA
urllib3             1.26.6
wcwidth             0.2.5
websocket           0.57.0
yaml                5.4.1
zmq                 25.0.2
-----
IPython             7.25.0
jupyter_client      7.4.9
jupyter_core        4.12.0
jupyterlab          3.0.16
notebook            6.4.0
-----
Python 3.9.5 | packaged by conda-forge | (default, Jun 19 2021, 00:32:32) [GCC 9.3.0]
Linux-3.10.0-1160.88.1.el7.x86_64-x86_64-with-glibc2.31
-----
Session information updated at 2023-05-15 23:47
```

Load all three datasets into one AnnData object by "concatenating" multiple anndata objects.

In [3]:
```
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()
```

```
GSM5114461_S6_A11
GSM5114464_S7_D20
GSM5114474_M3_E7
/opt/conda/lib/python3.9/site-packages/anndata/_core/anndata.py:1830: UserWarning: Observation names are not unique. To make them unique, call `.obs_names_make_unique`.
  utils.warn_names_duplicates("obs")
```

In [4]:
```
combined # will print out the dimensions of the combined dataset loaded
```

Out[4]:
```
AnnData object with n_obs × n_vars = 12357 × 20621
    obs: 'dataset'
```

In [5]:
```
adatas["GSM5114461_S6_A11"] # will print out dimensions of one of the individual datasets
```

Out[5]:
```
AnnData object with n_obs × n_vars = 4793 × 20621
    var: 'gene_ids', 'feature_types'
```

In [6]:
```
adatas["GSM5114464_S7_D20"] # print dimensions of second dataset
```

Out[6]:
```
AnnData object with n_obs × n_vars = 4910 × 20621
    var: 'gene_ids', 'feature_types'
```

In [7]:
```
adatas["GSM5114474_M3_E7"] # print dimensions of third dataset
```

Out[7]:
```
AnnData object with n_obs × n_vars = 2654 × 20621
    var: 'gene_ids', 'feature_types'
```

In [8]:
```
combined.obs # will print out info about each cell.
             # You should see a "dataset" column indicating which dataset each cell came from
```

Out[8]:

	dataset
AAACCTGAGAAGGACA	GSM5114461_S6_A11
AAACCTGAGTAAGTAC	GSM5114461_S6_A11
AAACCTGGTCCCGACA	GSM5114461_S6_A11
AAACCTGTCCTAAGTG	GSM5114461_S6_A11
AAACCTGTCGCCTGTT	GSM5114461_S6_A11
...	...
TTTGTCAAGCGTCTAT	GSM5114474_M3_E7
TTTGTCAAGTCGTACT	GSM5114474_M3_E7
TTTGTCAGTCTGATTG	GSM5114474_M3_E7
TTTGTCATCCCGGATG	GSM5114474_M3_E7
TTTGTCATCGAGAACG	GSM5114474_M3_E7

12357 rows × 1 columns

Part 2 - Filtering and normalizing your dataset

Question 2: Initial Filtering Code

In [9]:
```
sc.pp.filter_cells(combined, min_counts = 1000) # min 1000 counts (> 1000 reads)
sc.pp.filter_cells(combined, min_genes = 200) # min 200 genes (> 200 genes expressed)
sc.pp.filter_genes(combined, min_counts = 15) # min 15 counts (> 15 total count)
sc.pp.filter_genes(combined, min_cells = 5) # min 5 cells (> 5 cells)
```

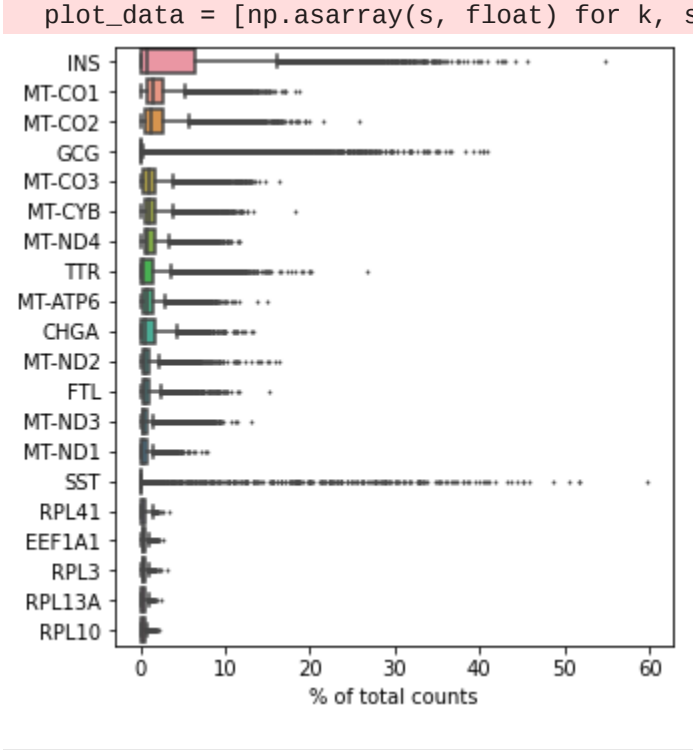
In [10]:
```
combined # check number of cells and genes after filtering
# n_obs are cells and n_vars are genes
```

Out[10]:
```
AnnData object with n_obs × n_vars = 10133 × 15779
    obs: 'dataset', 'n_counts', 'n_genes'
    var: 'n_counts', 'n_cells'
```

Question 3: Additional Filtering

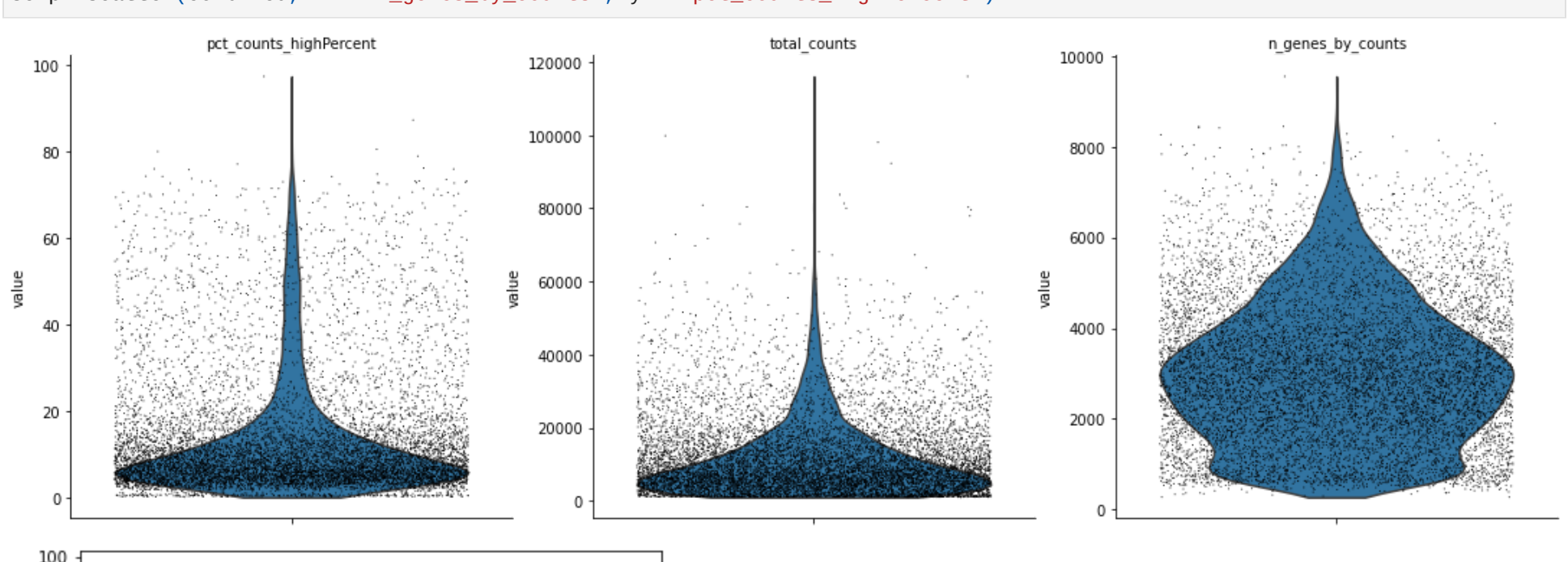
In [11]:
```
sc.pl.highest_expr_genes(combined, n_top=20) # plot 20 highest expressed genes
```

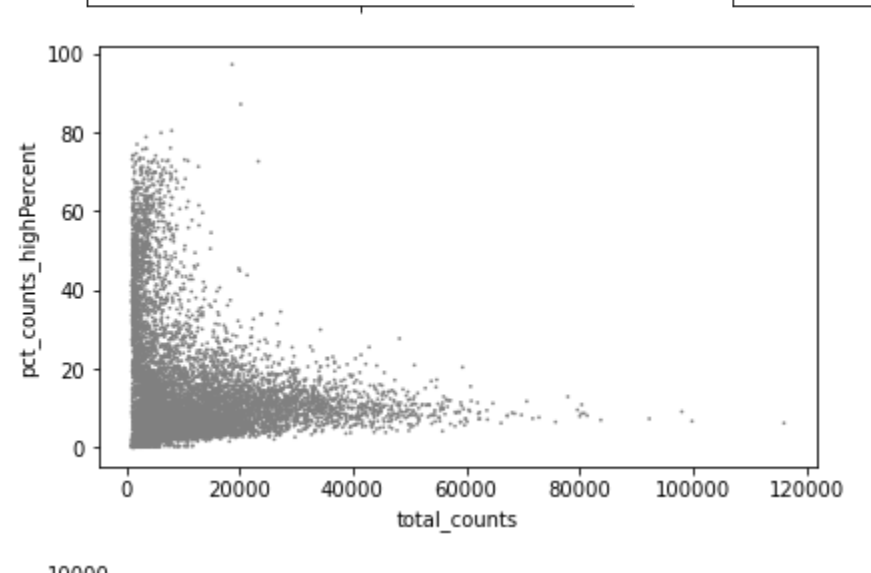
```
/opt/conda/lib/python3.9/site-packages/seaborn/categorical.py:82: FutureWarning: iteritems is deprecated and will be removed in a future version. Use .items instead.
  plot_data = [np.asarray(s, float) for k, s in iter_data]
```

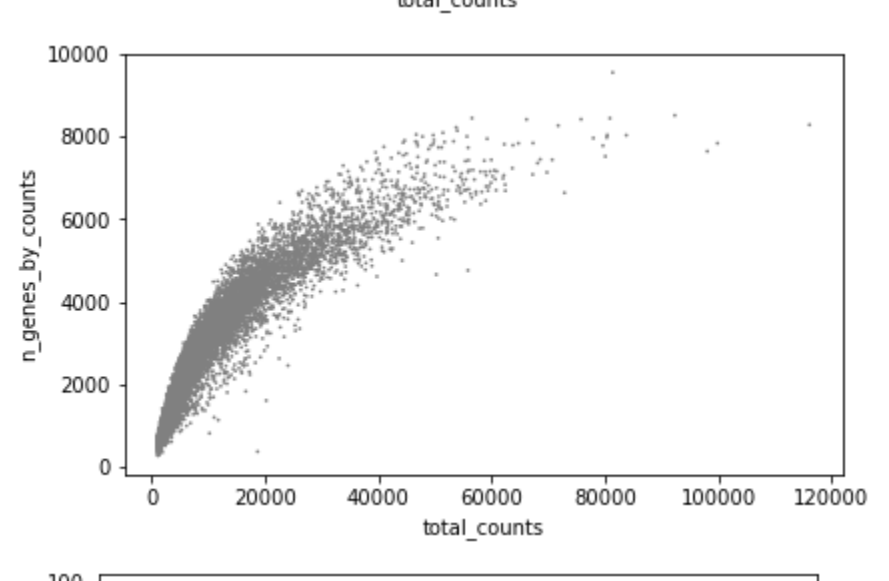


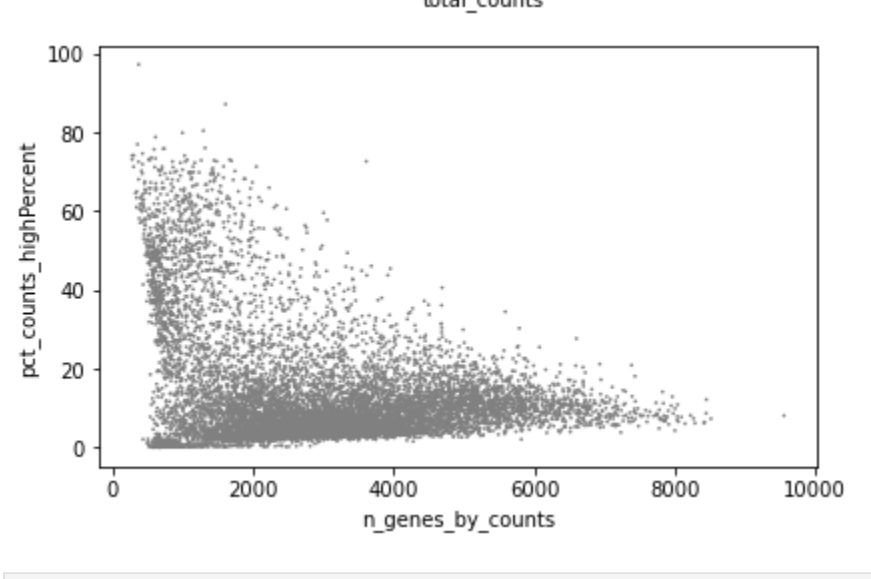
In [12]:
```
# Determine the percent of counts in each cell that are from mitochondrial genes.
combined.var['highPercent'] = combined.var_names.str.startswith('MT')
sc.pp.calculate_qc_metrics(combined, qc_vars = ['highPercent'], percent_top = None, log1p = False, inplace = True)

# Visualize violin and scatter plots of QC metrics including the percent mitochondria per cell, the count number per cell,
sc.pl.violin(combined, ['pct_counts_highPercent', 'total_counts', 'n_genes_by_counts'], jitter = 0.4, multi_panel = True)
sc.pl.scatter(combined, x = 'total_counts', y = 'pct_counts_highPercent')
sc.pl.scatter(combined, x = 'total_counts', y = 'n_genes_by_counts')
sc.pl.scatter(combined, x = 'n_genes_by_counts', y = 'pct_counts_highPercent')
```









In [13]:
```
# Filter cells with a high percentage of counts from mitochondrial genes. The paper we got the data from suggested using 25
adata_filt = combined[(combined.obs.pct_counts_highPercent < 25) & (combined.obs.total_counts < 70000), :] # Also filter ou
```

In [14]:
```
# Normalizing Counts
sc.pp.normalize_per_cell(combined, counts_per_cell_after=1e4) # normalize to 10,000 reads/cell
sc.pp.log1p(combined) # log transform
```

Question 4:

In [16]:
```
# Find genes with highest dispersion - use batch_key = dataset and n_top_genes = 500
sc.pp.highly_variable_genes(adata_filt, batch_key = 'dataset', n_top_genes = 500)
print(adata_filt.var.sort_values('dispersions_norm', ascending = False).index[0:5]) #sort and print
```

```
Index(['PPY', 'NPY', 'LYZ', 'KRT17', 'NTS'], dtype='object')
```

In [17]:
```
adata_filt.var # Look at Dataset
```

Out[17]:

	n_counts	n_cells	highPercent	n_cells_by_counts	mean_counts	pct_dropout_by_counts	total_counts	highly_variable	means	dispersions
SAMD11	3710.0	1470	False	1470	0.366130	85.492944	3710.0	False	0.183694	0.804814
NOC2L	4846.0	3514	False	3514	0.478239	65.321228	4846.0	False	0.380966	0.567331
KLHL17	439.0	417	False	417	0.043324	95.884733	439.0	False	0.030544	0.387182
PLEKHN1	25.0	24	False	24	0.002467	99.763150	25.0	False	0.002365	-0.089249
HES4	7191.0	2114	False	2114	0.709661	79.137472	7191.0	True	0.402617	1.249588
...	...	...	...	...	...	...	...	...	...	...
AC011043.1	10144.0	5064	False	5064	1.001086	50.024672	10144.0	False	0.571877	0.698923
AC007325.4	2477.0	1805	False	1805	0.244449	82.186914	2477.0	False	0.149973	0.578627
AC007325.2	52.0	51	False	51	0.005132	99.496694	52.0	False	0.002825	0.308943
AC004556.3	29.0	29	False	29	0.002862	99.713806	29.0	False	0.003271	0.009244
AC240274.1	560.0	522	False	522	0.055265	94.848515	560.0	False	0.085137	0.600197

15779 rows × 13 columns

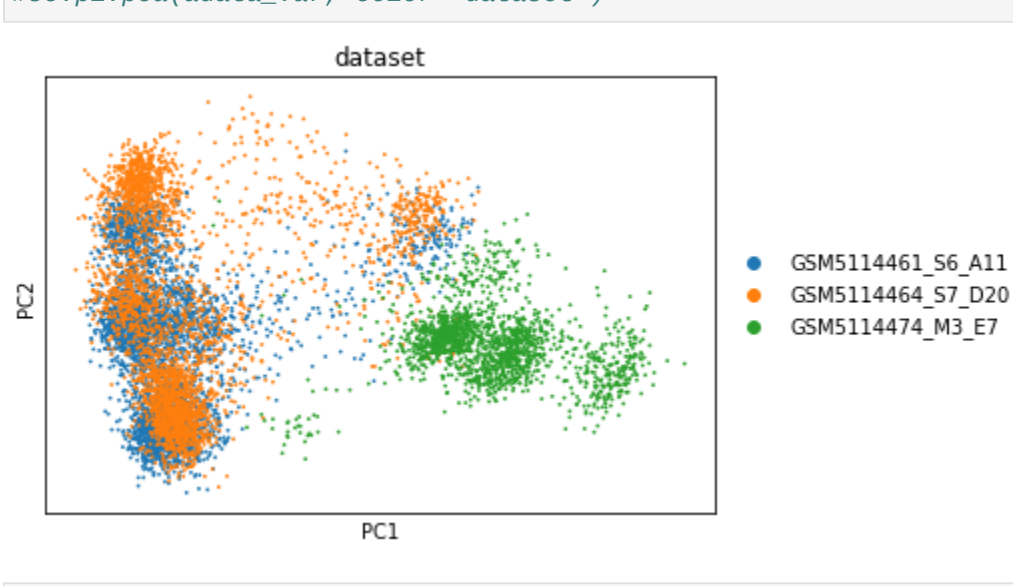
In [18]:
```
# We'll manually add these genes to make sure they stay in our
# dataset for the analyses below.
genes = ["GCG", "TTR",  "IAPP",  "GHRL", "PPY", "COL3A1",
    "CPA1", "CLPS", "REG1A", "CTRB1", "CTRB2", "PRSS2", "CPA2", "KRT19", "INS","SST","CELA3A", "VTCN1"]

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

Question 5: Batch Correction

Every code below is commented out due to validating time. The lines with characters right after the comment are code and lines with a space then characters are actual comments. like "#code" vs. "# comment"

In [19]:
```
#sc.pp.pca(adata_var, n_comps =20)
#sc.pl.pca(adata_var, color="dataset")
```



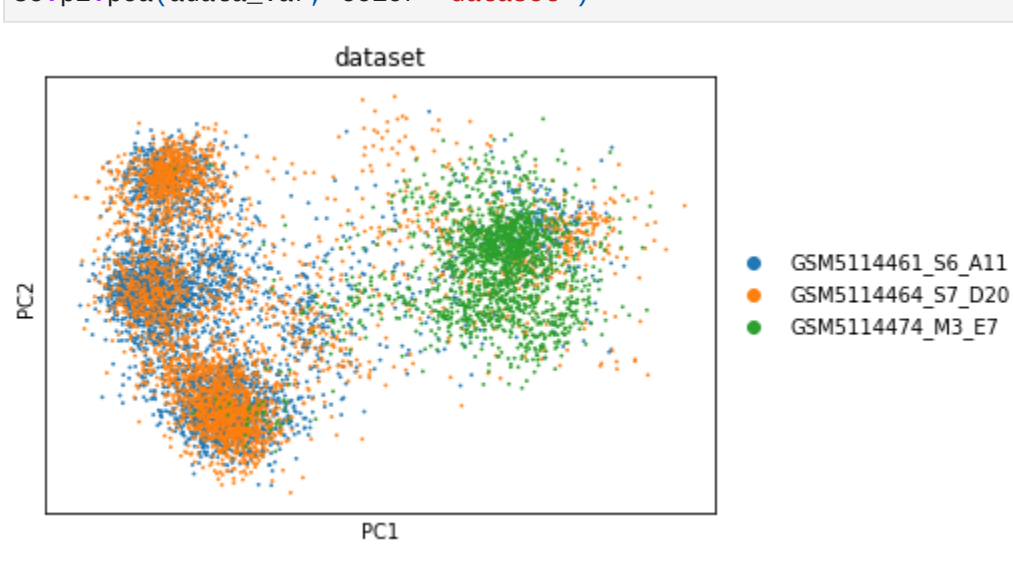
In [20]:
```
# Import the "external" library
#import scanpy.external as sce


# 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']
```

```
2023-05-15 23:49:03,030 - harmonypy - INFO - Computing initial centroids with sklearn.KMeans...
Computing initial centroids with sklearn.KMeans...
2023-05-15 23:49:07,075 - harmonypy - INFO - sklearn.KMeans initialization complete.
sklearn.KMeans initialization complete.
2023-05-15 23:49:07,194 - harmonypy - INFO - Iteration 1 of 10
Iteration 1 of 10
2023-05-15 23:49:14,397 - harmonypy - INFO - Iteration 2 of 10
Iteration 2 of 10
2023-05-15 23:49:23,197 - harmonypy - INFO - Iteration 3 of 10
Iteration 3 of 10
2023-05-15 23:49:32,431 - harmonypy - INFO - Iteration 4 of 10
Iteration 4 of 10
2023-05-15 23:49:39,698 - harmonypy - INFO - Iteration 5 of 10
Iteration 5 of 10
2023-05-15 23:49:46,202 - harmonypy - INFO - Converged after 5 iterations
Converged after 5 iterations
```

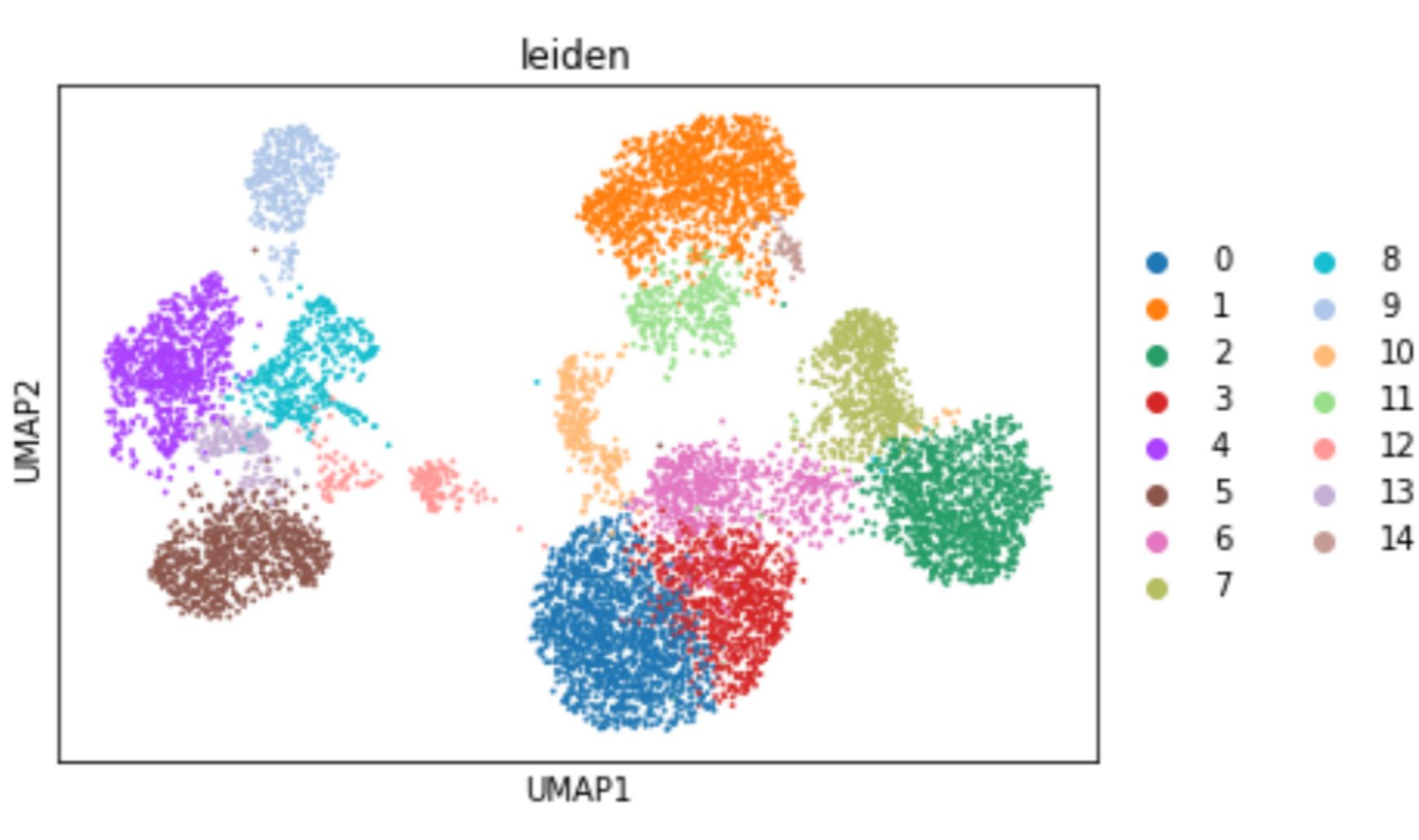
In [21]:
```
# plot again with new corrections after harmony
sc.pl.pca(adata_var, color="dataset")
```



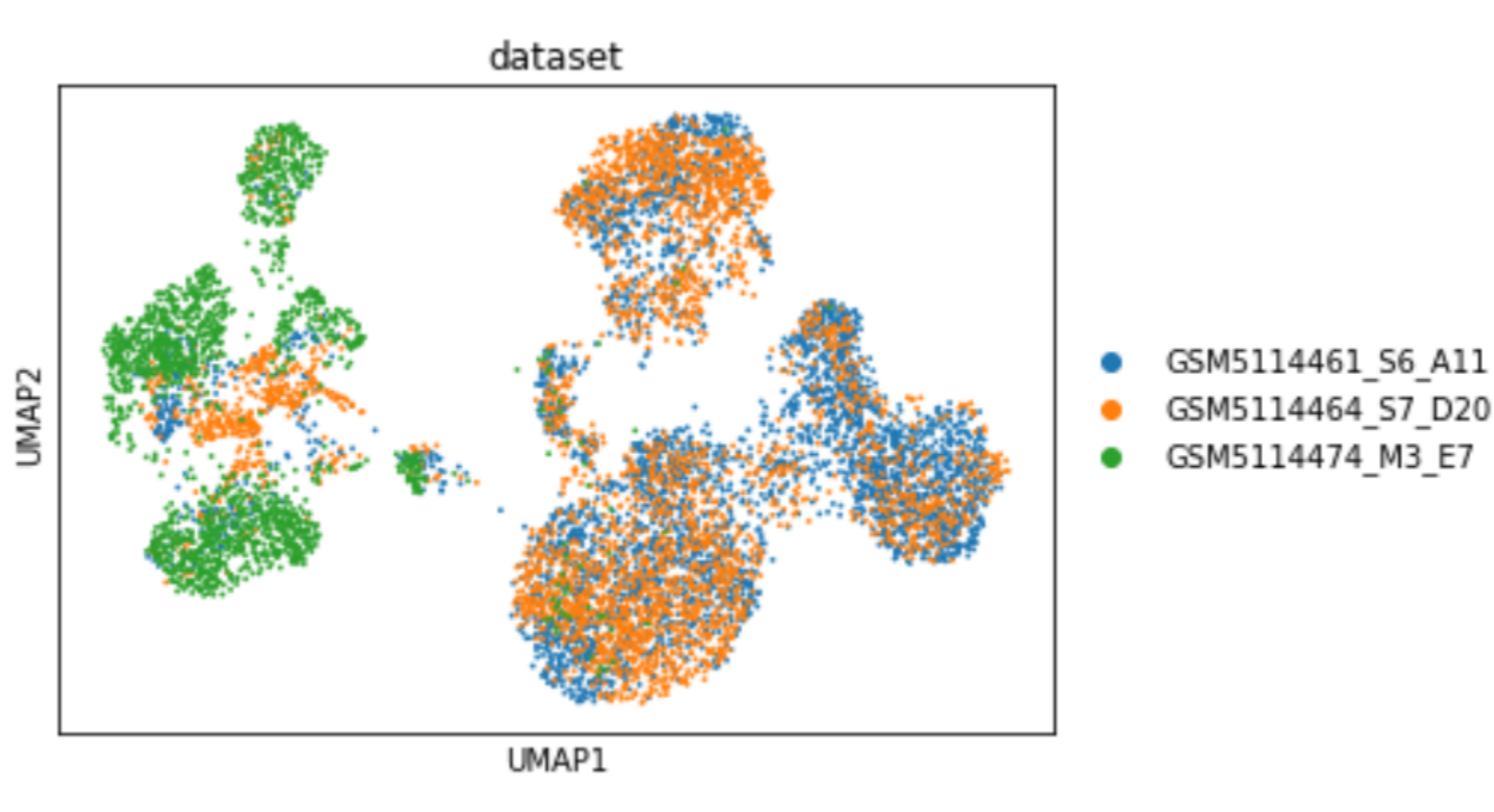
Question 6: Performing Clustering

In [22]:
```
#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.
```

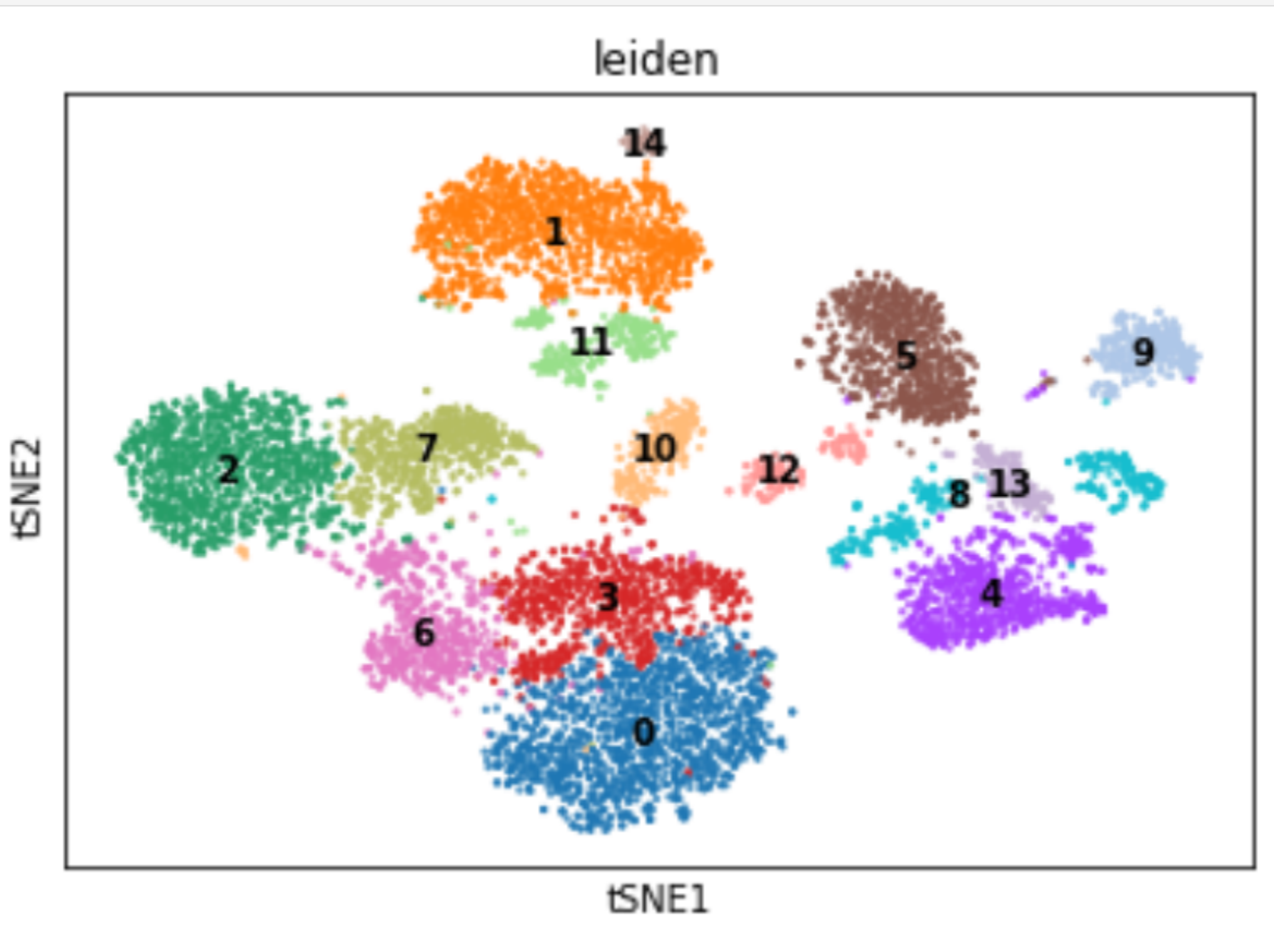
In [ ]:
```
#sc.tl.umap(adata_var) # compute UMAP embedding
# UMAP by leiden
#sc.pl.umap(adata_var, color="leiden") # make the UMAP plot, coloring cells by cluster
```



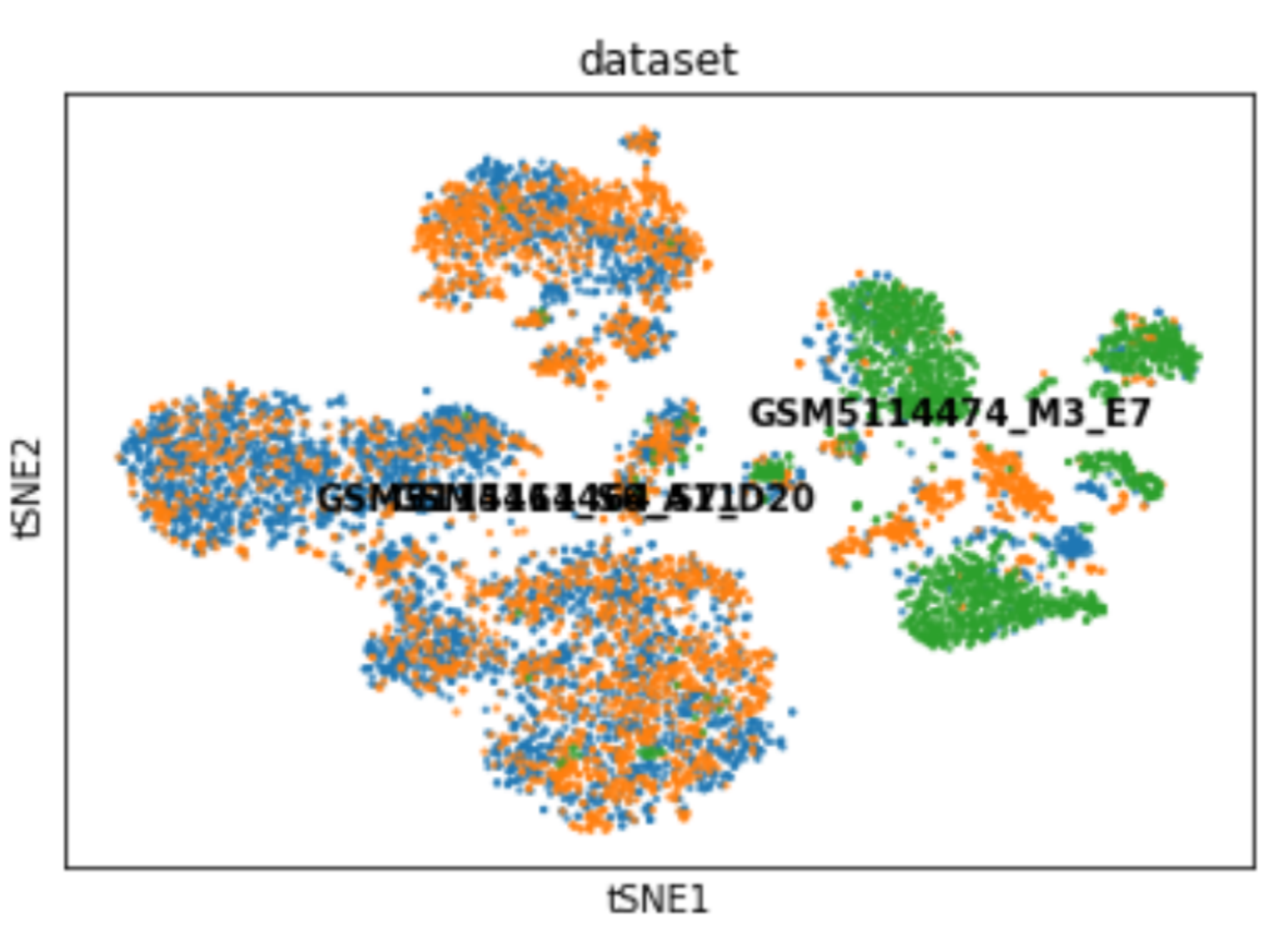
In [ ]:
```
# UMAP by dataset
#sc.pl.umap(adata_var, color = 'dataset') #color cells by the dataset
```



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



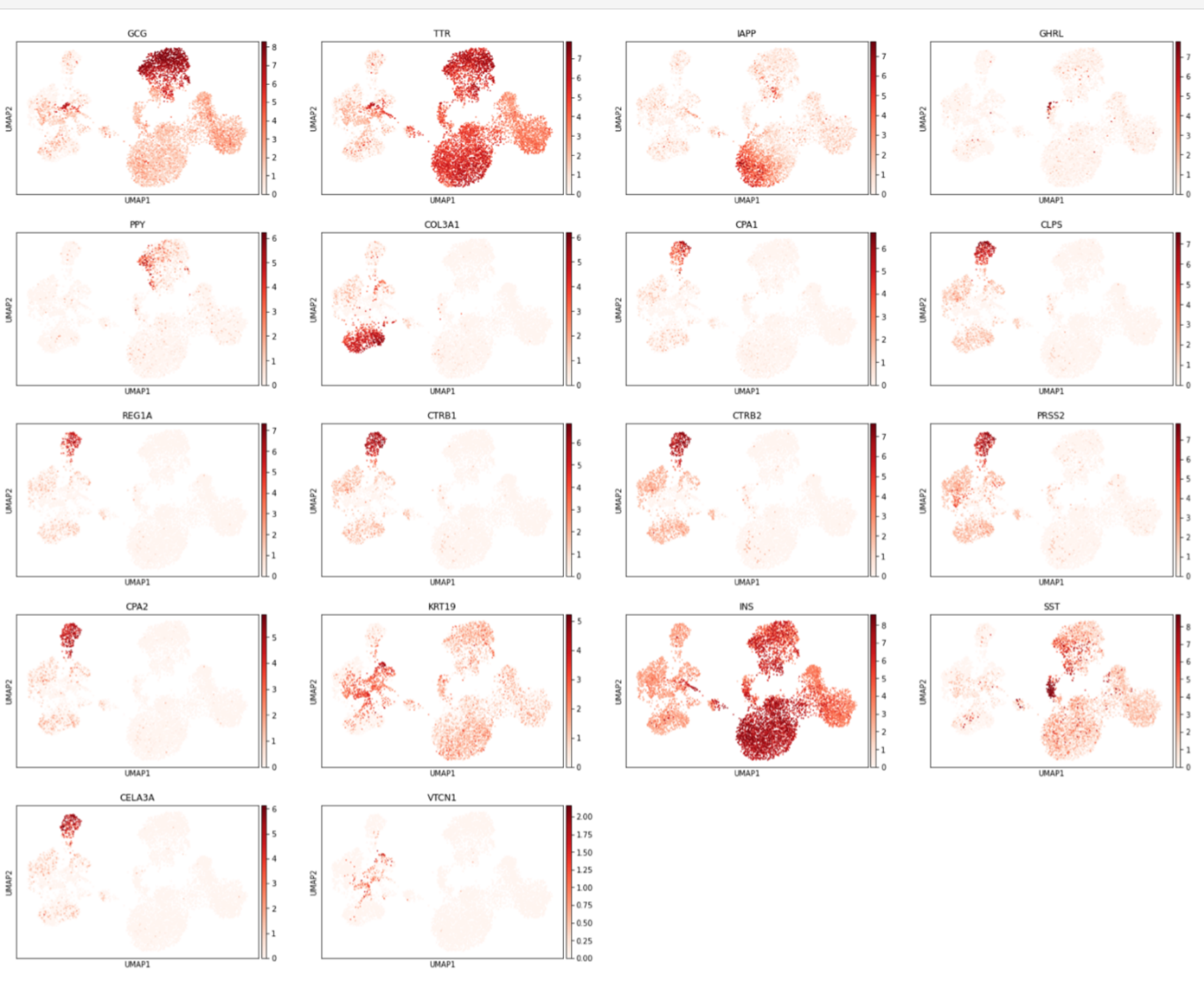
In [ ]:
```
# tSNE with dataset
#sc.pl.tsne(adata_var, color=['dataset'], legend_loc='on data', legend_fontsize=10, alpha=0.8, size=20)
```



Question 7:

In [ ]:
```
# Test mapping INS, GCG, SST
#sc.pl.umap(adata_var, color=["INS","GCG","SST"], color_map="Reds")
```

In [ ]:
```
# Map all Genes
#sc.pl.umap(adata_var, color=genes, color_map="Reds")
```



In [ ]:
```
# map by cluster
#sc.pl.heatmap(adata_var, genes, groupby='leiden', dendrogram=True)
```

In [ ]:
```
# map by dataset
#sc.pl.heatmap(adata_var, genes, groupby='dataset', dendrogram=True)
```