Part I: Pre-processing of 10X Single-Cell RNA Datasets

(Available on *usegalaxy.org*)

Analysis Strategy:

1. Producing a Count Matrix from FASTQ

- 10x Chemistries
- Performing the Demultiplexing and Quantification
- Inspecting the Output Files

2. Producing a Quality Count Matrix

10x Genomics has its own processing pipeline ***Cell Ranger*** , but this process requires much configuration to run and is significantly slower than other mappers.
We will use ***STARsolo*** in the first part of the tutorial instead, since it is a drop-in solution to the Cell Ranger pipeline.

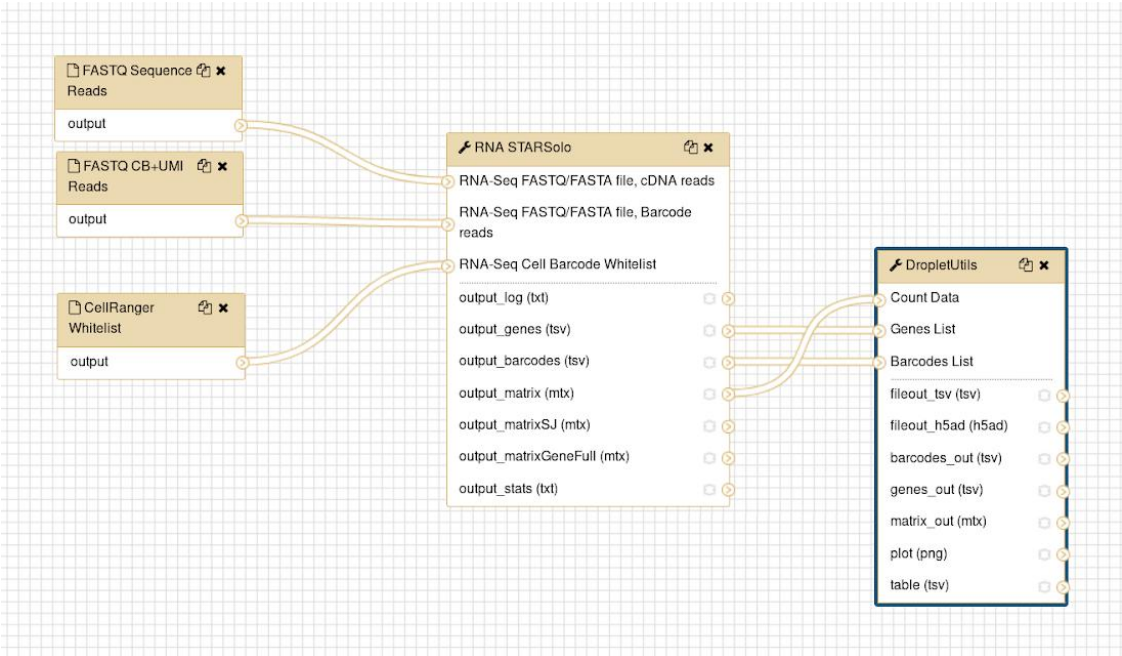


Figure 1: An overview of the workflow

1. Producing a Count Matrix from FASTQ

Here we will use Peripheral blood mononuclear cells (PBMCs) extracted from a healthy donor from 10x genomics, where PBMCs are primary cells with relatively small amounts of RNA. For this tutorial, we will use datasets sub-sampled from the source files to contain approximately 300 cells to save our time of analysis.

1.1 Data upload and organization

Requirement:

- sub-sampled source files
- a “whitelist” of known cell barcodes*
- a hg19 GTF file to annotate our reads

* A barcode whitelist is the list of all known barcode sequences that have been included in the assay kit and are available during library preparation.

✎ Hands-on:

1. Create a new history and rename it (e.g. scRNA-seq 10X dataset tutorial).

2. Import the sub-sampled FASTQ data from *Zenodo*:
   https://zenodo.org/record/3457880/files/subset_pbmc_1k_v3_S1_L001_R1_001.fastq.gz
   https://zenodo.org/record/3457880/files/subset_pbmc_1k_v3_S1_L001_R2_001.fastq.gz
   https://zenodo.org/record/3457880/files/subset_pbmc_1k_v3_S1_L002_R1_001.fastq.gz
   https://zenodo.org/record/3457880/files/subset_pbmc_1k_v3_S1_L002_R2_001.fastq.gz

3. Import the Gene Annotations and Cell Barcodes from *Zenodo*:
   https://zenodo.org/record/3457880/files/Homo_sapiens.GRCh37.75.gtf
   https://zenodo.org/record/3457880/files/3M-february-2018.txt.gz

1.2 10x Chemistries

There are two main reagent kits used during the library preparation. Below we can see the layout of the primers used in both chemistries.

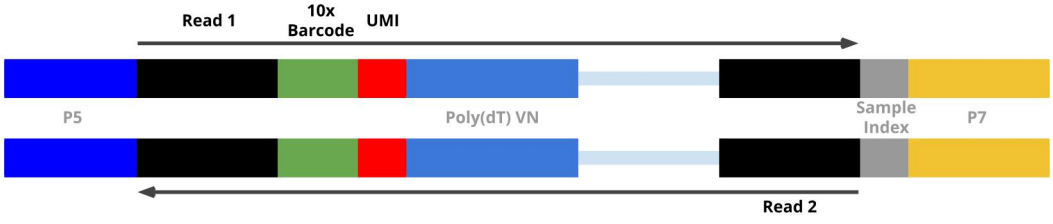


Figure 2: 10x Chromiumv2 and Chromiumv3 Chemistries

The primers of interest to us are the Cell Barcode (CB) and the Unique Molecular Identifiers (UMI) used in the Read 1 sequencing primer, as they describe to us how to demultiplex and deduplicate our reads.

Chemistry	Read 2	Read 1 (CB + UMI)	Insert (Read 2 + Read 1)
v2	98	26 (16 + 10)	124
v3	91	28 (16 + 12)	119

The table above gives a summary of the primers used in the image and the number of basepairs occupied by each. Note that for the Chromium v2 and v3 chemistries, the barcode information is purely within the R1 forward strand, so the strandedness of our generated library is forward.

To perform the demultiplexing, we need to tell ***RNA STARsolo*** where to look in the R1 FASTQ to find the cell barcodes. We can do this by simply counting the number of basepairs in any read of the R1 files.

Question:
Peek at one of the R1 FASTQ files using **the eye symbol** below the dataset name:
1. How many basepairs are there in any given read?
2. Which library preparation chemistry version was this read generated from?

Solution:
1. There are 28 basepairs.
2. The v2 has 26 basepairs, but the v3 has 28 basepairs. Therefore the reads we have here use the Chromium v3 chemistry.

1.3 Performing the Demultiplexing and Quantification

We will now proceed to demultiplex, map, and quantify both sets of reads using the correct chemistry discovered in the previous sub-section.

✎ Hands-on:

RNA STARsolo tool (*due to unknown bug, please use the **2.7.7a** version*):
- *"Custom or built-in reference genome"*: Use a built-in index
  - *"Reference genome with or without an annotation"*: use genome reference without builtin gene-model
    - *"Select reference genome"*: Human (Homo Sapiens): hg19 chr X*

    * only map to chr X in order to save time
    - *"Gene model (gff3,gtf) file for splice junctions"*: Homo_sapiens.GRCh37.75.gtf

        - *"Length of genomic sequence around annotated junctions"*: 100
- *"Type of single-cell RNA-seq"*: Drop-seq or 10X Chromium
    - *"Input Type"*: Separate barcode and cDNA reads
    - *"RNA-Seq FASTQ/FASTA file, Barcode reads"*: Multi-select L001_R1_001 and L002_R1_001 using the Ctrl key.
    - *"RNA-Seq FASTQ/FASTA file, cDNA reads"*: Multi-select L001_R2_001 and L002_R2_001 using the Ctrl key.
    - *"RNA-Seq Cell Barcode Whitelist"*: 3M-february-2018.txt.gz
    - *"Configure Chemistry Options"*: Cell Ranger v3
    - *"UMI deduplication (collapsing) algorithm"*: All
    - *"Matching the Cell Barcodes to the WhiteList"*: Multiple matches (CellRanger 3)
- Under *"Advanced Settings"*:
    - *"Strandedness of Library"*: Forward
    - *"Collect UMI counts for these genomic features"*: Gene: Count reads matching the Gene Transcript
    - *"Type of UMI filtering"*: Remove lower-count UMIs that map to more than one gene ((introduced in CellRanger 3.x.x)
    - *"Cell filter type and parameters"*: Do not filter*

* The in-built Cell filtering is a relatively new feature that emulates the CellRanger pipeline. Here, we set the filtering options to not filter because we will use our own methods to better

1.4 Inspecting the Output Files

At this stage ***RNA STARsolo*** has output 6 files:

1. Log *(program log)*
2. Feature Statistic Summaries *(mapping quality file)*
3. Alignments *(BAM file of alignments)*
4. Matrix Gene Counts *(count matrix in matrixmarket format)*
5. Barcodes *(list of barcodes)*
6. Genes *(list of genes)*

Mapping Quality

Question:
What percentage of reads are uniquely mapped?

Solution:
We can find it in the **Log** file:
18.33%. This is expected because we only chose the chr X for alignment.

Quantification Quality

Let us investigate the **STARsolo Feature summaries** file.

Barcodes:		
	nNoAdapter	0
	nNoUMI	0
	nNoCB	0
	nNinCB	0
	nNinUMI	358
	nUMIhomopolymer	707
	nTooMany	0
	nNoMatch	50037
	nMismatchesInMultCB	0
	nExactMatch	7530489
	nMismatchOneWL	19534
	nMismatchToMultWL	84800
Genes:		
	nUnmapped	5738754
	nNoFeature	1261478
	nAmbigFeature	13948
	nAmbigFeatureMultimap	7402
	nTooMany	2893
	nNoExactMatch	0
	nExactMatch	613268
	nMatch	617750
	nCellBarcodes	4049
	nUMIs	280778

The explanation of these parameters can be seen in the RNA STAR Manual under the STARsolo section.
The main information to gather at this stage is that the nCellBarcodes tell us how many cells were detected in our sample, where we see that there are **4049** cells.

2. Producing a Quality Count Matrix

Now we have the 10x analysis datasets, which compatible with any downstream single-cell RNA analysis pipeline, however the number of cells represented here are greatly over-represented, therefore the matrix represents any cells that were unambiguously detected in the sample.
To get a high quality count matrix we must apply the ***DropletUtils*** tool, which will produce a filtered dataset that is more representative of the *Cell Ranger* pipeline.

✎ Hands-on:

DropletUtils tool :

- *"Format for the input matrix":* Bundled (barcodes.tsv, genes.tsv, matrix.mtx)
  - *"Count Data":* Matrix Gene Counts *(output of **RNA STARsolo** tool)*
  - *"Genes List":* Genes *(output of **RNA STARsolo** tool)*
  - *"Barcodes List":* Barcodes *(output of **RNA STARsolo** tool)*
- *"Operation":* Filter for Barcodes
  - *"Method":* DefaultDrops
  - *"Format for output matrices":* Tabular

Question:

How many cells were detected?

Solution:

By clicking on the title of the output dataset in the history we can expand the box to see the output.

Although 4049 barcodes were detected in the previously analysis, only 256 of them were above an acceptable threshold of quality, based on the default upper quantile and lower proportion parameters given in the tool.

The tabular outputs here are easy to inspect. However, the datasets produced with 10x data are very large and very sparse, meaning there is much data redundancy due to the repetition of zeroes everywhere in the data.

Many analysis packages have attempted to solve this problem by inventing their own standard, which has led to the proliferation of many different “standards” in the scRNA-seq package ecosystem.

The ***AnnData*** format (hda5) is an extension of the *HDF5* format, which supports multidimensional datasets to be stored in a consistent and space-optimised way. This is the default output of ***Cell Ranger*** and so is also the default output of ***RNA STARsolo***.

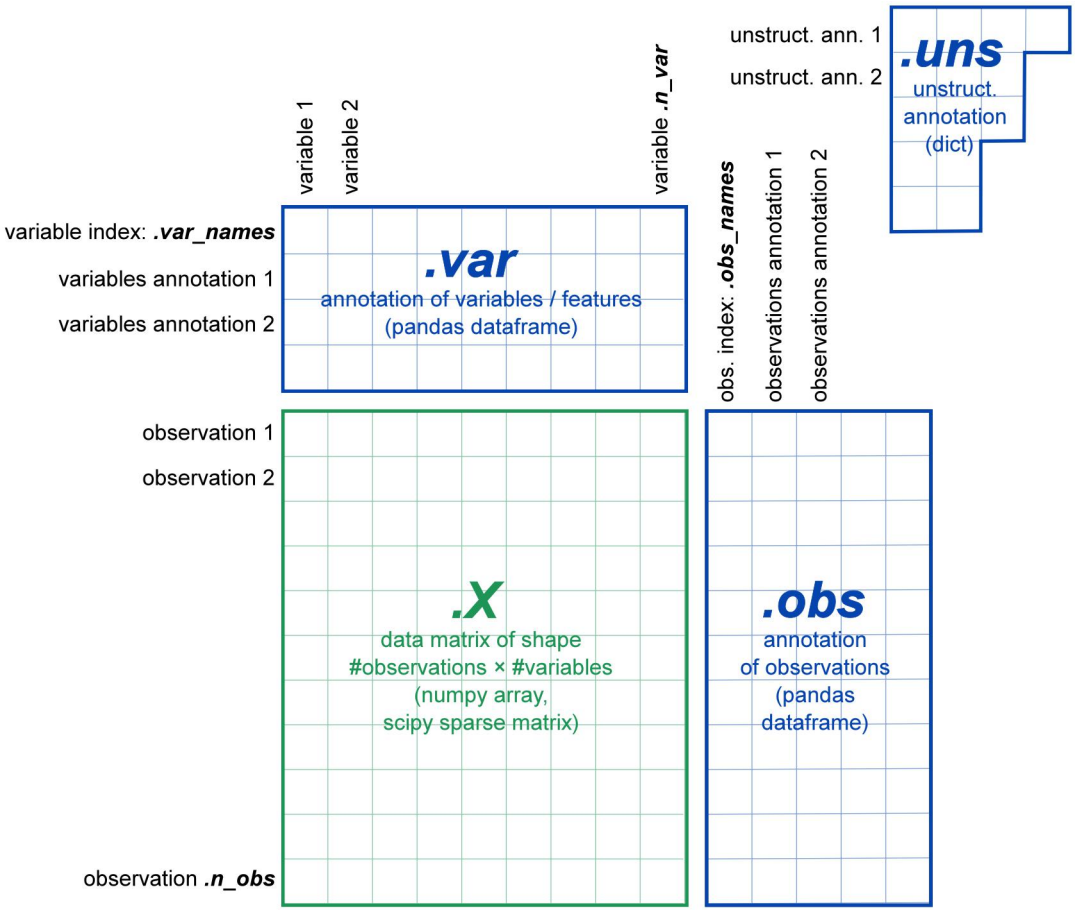


Figure 3: Contents of the AnnData format

Conclusion

In this workflow we have learned to quickly perform mapping and quantification of scRNA-seq FASTQ data in a single step via ***RNA STARsolo***, and have reproduced a Cell Ranger workflow using the ***DropletUtils*** suite. It produced a count matrix in a human readable tabular format which can be used in downstream analysis tools.

Part II: Clustering 3K PBMCs with Scanpy

(Available on *usegalaxy.org*)

Analysis Strategy:

1. **Data**
   - Data upload
   - AnnData
2. **Preprocessing**
   - Quality control
   - Normalization and scaling
   - Selection of features
   - Scaling the data
3. **Dimensionality reduction**
   - Principal Component Analysis
   - Visualization of PCA
   - Determination of the number of PCs to keep
4. **Clustering of the cells**
   - Computation of a neighborhood graph
   - Visualization of the neighborhood graph
   - Clustering of the neighborhood graph
5. **Finding marker genes**
   - Using Wilcoxon rank sum test
   - Visualization of expression of the marker genes
   - Comparison of the marker genes between clusters
6. **Cell type annotation**

1. Data

For this tutorial, we will continue to analyze a dataset of Peripheral Blood Mononuclear Cells (PBMC) extracted from a healthy donor. The raw sequences have been processed by the cellranger pipeline from 10X to extract a unique molecular identifier (UMI) count matrix, in a similar way to that as explained in the Part I tutorial.

Requirement:

- *genes.tsv*: a tabular file with information about the 32,738 genes in 2 columns (Ensembl gene id and the gene symbol)
- *barcodes.tsv*: a tabular file with the barcode for each of the 2,700 cells
- *matrix.mtx*: a condensed version of the count matrix

1.1 Data Upload

✎ Hands-on:

1. Create a new history for this tutorial.

2. Import the *genes.tsv*, *barcodes.tsv* and *matrix.mtx* from *Zenodo*:
   https://zenodo.org/record/3581213/files/genes.tsv
   https://zenodo.org/record/3581213/files/barcodes.tsv
   https://zenodo.org/record/3581213/files/matrix.mtx

3. Rename the datasets.
4. Inspect the *matrix* file.

Question:
How many non-zero values are in the matrix?

Solution:
There are 2,286,884 (2.6%) non-zero values for the 88,392,600 possible counts of the 32,738 genes (rows) and 2,700 cells (columns).

1.2 AnnData

The most common format, called ***AnnData***, stores the matrix as well as gene and cell annotations in a concise, compressed and extremely readable manner. So we need first to import the matrix and annotations of genes and cells into an *AnnData* object.

✎ Hands-on:

1. **Import Anndata and loom** Tool:
- *"hd5 format to be created"*: Anndata file
- *"Format for the annotated data matrix"*: Matrix Market (mtx), from Cell ranger or not
  - *"Matrix"*: matrix.mtx
  - *"Use 10x Genomics formatted mtx"*: Output from Cell Ranger v2 or earlier versions
    - *"Genes"*: genes.tsv
    - *"Barcodes"*: barcodes.tsv
    - *"Variables index"*: gene_symbols
    - *"Make the variable index unique by appending '-1', '-2'?"*: Yes
2. Rename the generated file to Input 3k PBMC
3. Check that the format is h5ad

Because the ***AnnData*** format is an extension of the *HDF5* format, i.e. a binary format, an AnnData object can not be inspected directly in Galaxy by clicking on the eye icon.

We can use the **Insepect AnnData** tool to inspect different indexes that stored in the hdf5 file:
General information: count of observations (cells), and variables (genes)
Full data matrix: a count matrix of all the cells and genes
Key-indexed (obs): annotations of the observations, i.e. the cells
Key-indexed (var): annotations of the variables, i.e. the genes

2. Processing

Before perform the clustering, we need to:
1. **Selection and filtration of cells and genes based on quality metrics**
2. **Data normalization and scaling**
3. **Selection of features**

2.1 Quality Control

Remove genes found in less than 3 cells

> ✎ **Hands-on:**
>
> 1. **Filter with scanpy** Tool:
> - *"Annotated data matrix"*: Input 3k PBMC
> - *"Method used for filtering"*: Filter genes based on number of cells or counts, using 'pp.filter_genes'
>   - *"Filter"*: Minimum number of cells expressed
>   - *"Minimum number of cells expressed required for a gene to pass filtering"*: 3
> 2. Rename the generated file 3k PBMC
> 3. Inspect the dataset.

There are now 13,714 genes. So 19,024 (32,738 - 13,714) genes have been removed.

Computation of QC metrics

To identify low-quality cells, the simplest approach is to apply thresholds on the QC metrics.
The first 2 QC metrics (cell size and number of expressed genes) can be easily estimated from the count table. For the third metric (proportion of reads mapped to genes in the mitochondrial genome), we need a new annotation for var with ***True*** for mitochondrial gene or ***False*** for non mitochondrial genes.

1. Import the Mitochondrial annotation.csv from Local file, and ensure that the datatype is changed to tabular.

2. **Manipulate Anndata** Tool:
    - *"Annotated data matrix"*: 3k PBMC
    - *"Function to manipulate the object"*: Add new annotation(s) for observations or variables
        - *"What to annotate?"*: Variables (var)
        - *"Table with new annotations"*: Mitochondrial annotation.csv
3. Rename the generated file 3k PBMC with mito annotation

We can now compute QC metrics on the ***AnnData*** object.

✎ Hands-on:

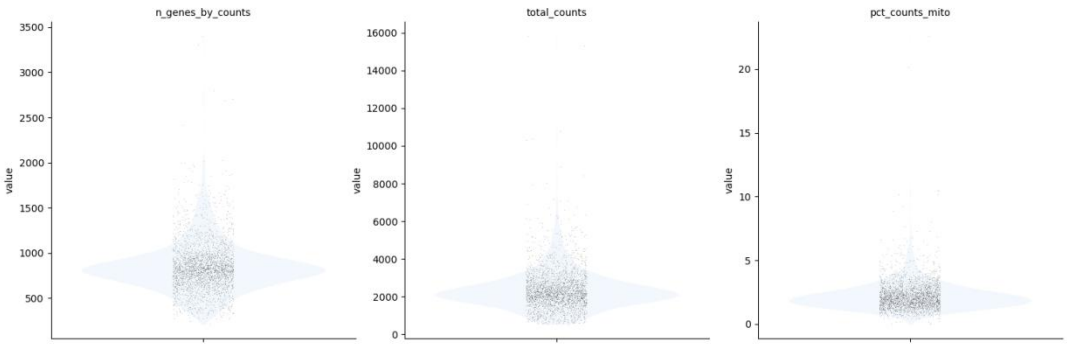
1. **Inspect and Manipulate with scanpy** Tool:
    - *"Annotated data matrix"*: 3k PBMC with mito annotation
    - *"Method used for inspecting"*: Calculate quality control metrics, using 'pp.calculate_qc_metrics'
        - *"Name of kind of values in X"*: counts
        - *"The kind of thing the variables are"*: genes
        - *"Keys for boolean columns of .var which identify variables you could want to control for"*: mito
2. Rename the generated file 3k PBMC with mito annotation and qc metrics.
3. Inspect the dataset.

We can see that the tool computed several QC metrics at both cell and gene levels. These metrics are added to the annotation tables. What we need are:
the cell size, i.e. total_counts
the number of expressed genes, i.e. n_genes_by_counts
the proportion of reads mapped to mitochondrial genes, i.e. pct_counts_mito

Now we would like to visualize these 3 informative QC metrics:

✎ Hands-on:

1. **Plot with scanpy** Tool:
- *"Annotated data matrix"*: 3k PBMC with mito annotation and qc metrics
- *"Method used for plotting"*: Generic: Violin plot, using 'pl.violin'
  - *"Keys for accessing variables"*: Subset of variables in 'adata.var_names' or fields of '.obs'
    - *"Keys for accessing variables"*: n_genes_by_counts, total_counts, pct_counts_mito
  - In *"Violin plot attributes"*:
    - *"Add a stripplot on top of the violin plot"*: Yes
      - *"Add a jitter to the stripplot"*: Yes
        - *"Size of the jitter points"*: 0.4
  - *"Display keys in multiple panels"*: Yes
2. Inspect the generated file



Question:
How do the distributions of the 3 QC metrics look?

Solution:
For the cell size, i.e. total_counts, most of the values are between 1,000 reads and 4,000 reads, with some extremely high values skewing the distribution.
The numbers of expressed genes, i.e. n_genes_by_counts, are mostly between 500 genes and 1,200 genes, with also some extremely high values skewing the distribution.
The distribution of the proportions of reads mapped to mitochondrial genes, i.e. pct_counts_mito, is even more narrow with some cells having no counts from mitochondrial genes but also having some really extreme values (above 5%).

Based on the previous plot, we would like to remove cells that have:

1. A number of expressed genes below 200 or above 2,500
2. A percentage of reads mapped to mitochondrial genes above 5%

✎ Hands-on:

1. **Filter with scanpy** Tool:
    - *"Annotated data matrix"*: 3k PBMC with mito annotation and qc metrics
    - *"Method used for filtering"*: Filter cell outliers based on counts and numbers of genes expressed, using 'pp.filter_cells'
        - *"Filter"*: Minimum number of genes expressed
            - *"Minimum number of genes expressed required for a cell to pass filtering"*: 200

2. **Filter with scanpy** Tool:
    - *"Annotated data matrix"*: output of **Filter** tool
    - *"Method used for filtering"*: Filter cell outliers based on counts and numbers of genes expressed, using 'pp.filter_cells'
        - *"Filter"*: Maximum number of genes expressed
            - *"Maximum number of genes expressed required for a cell to pass filtering"*: 2500

3. **Manipulate Anndata** Tool:
    - *"Annotated data matrix"*: output of **Filter** tool
    - *"Function to manipulate the object"*: Filter observations or variables
        - *"What to filter?"*: Observations (obs)
        - *"Type of filtering?"*: By key (column) values
            - *"Key to filter"*: pct_counts_mito
            - *"Type of value to filter"*: Number
                - *"Filter"*: less than
                - *"Value"*: 5.0
4. Rename the generated file 3k PBMC after QC filtering

There are now 2,638 cells after QC filtering.

2.2 Normalization and scaling

Normalization of the counts removes differences to avoid that they interfere with comparisons of the expression profiles between cells. Any observed heterogeneity or differential expression within the cell population after normalization are then driven by biology and not technical biases.
The simplest strategy for scaling normalization is the cell size normalization. The "cell size factor" for each cell is computed directly for its cell size and transformed such that the mean size factor across all cells is equal to 1.
Here we would to normalize our count table such that each cell have 10,000 reads.

✎ Hands-on:

1. **Normalize with scanpy** Tool:
- *"Annotated data matrix"*: 3k PBMC after QC filtering
- *"Method used for normalization"*: Normalize counts per cell, using 'pp.normalize_total'
  - *"Target sum"*: 10000.0
  - *"Exclude (very) highly expressed genes for the computation of the normalization factor (size factor) for each cell"*: No
  - *"Name of the field in 'adata.obs' where the normalization factor is stored"*: norm
  - *"List of layers to normalize"*: all
  - *"How to normalize layers?"*: After: for each layer in layers each cell has a total count equal to target_sum.

The normalized counts should be log-transformed afterwards to adjust for the mean-variance relationship.

2. **Inspect and Manipulate** Tool:
   - *"Annotated data matrix"*: output of **Normalize** tool
   - *"Method used for inspecting"*: Logarithmize the data matrix, using 'pp.log1p'

We will freeze the current state of the AnnData object, i.e. the logarithmized raw gene expression, in the a raw attribute. This information will be used later in differential testing and visualizations of gene expression.

3. **Manipulate Anndata** Tool:
   - *"Annotated data matrix"*: output of Inspect and manipulate tool
   - *"Function to manipulate the object"*: Freeze the current state into the 'raw' attribute
4. Rename the generated output 3k PBMC after QC filtering and normalization

2.3 Selection of features

ScRNA-Seq data is often used in exploratory analyses to characterize heterogeneity across cells, which are compared based on their gene expression profiles. We need then to remove genes with random noise while keeping genes containing useful information about the biology of the system.
Selecting the most variable genes based on their expression across the cells (i.e. genes highly expressed in some cells, and lowly expressed in others) is the simplest approach for feature selection.

Here, we will define the set of highly variable genes as those which, after normalization, have a normalized dispersion amount higher than 0.5.

✎ Hands-on:

1. **Filter with scanpy** Tool:
    - *"Annotated data matrix"*: 3k PBMC after QC filtering and normalization
    - *"Method used for filtering"*: Annotate (and filter) highly variable genes, using 'pp.highly_variable_genes'
        - *"Flavor for computing normalized dispersion"*: seurat
            - *"Minimal mean cutoff"*: 0.0125
            - *"Maximal mean cutoff"*: 3
            - *"Minimal normalized dispersion cutoff"*: 0.5
        - *"Inplace subset to highly-variable genes?"*: No

Both highly variable genes and other genes are still in the AnnData object. We can now actually keep only the highly variable genes.

2. **Manipulate Anndata** Tool:
    - *"Annotated data matrix"*: output of the last **Filter** tool
    - *"Function to manipulate the object"*: Filter observations or variables
        - *"What to filter?"*: Variables (var)
            - *"Type of filtering?"*: By key (column) values
                - *"Key to filter"*: highly_variable
                - *"Type of value to filter"*: Boolean
                - *"Value to keep"*: Yes
3. Rename the generated output 3k PBMC with only HVG

Only 1,838 (over the 13,714) genes are kept.

2.4 Scaling the data

Prior to any downstream analysis like dimensional reduction, we need to apply a linear transformation or scaling to:

1. Regress out unwanted sources of variation in the total counts per cell and the percentage of mitochondrial genes expressed.

2. Scale data to unit variance and zero mean, i.e. the variance across cells is 1 and the mean expression is 0, in order to give equal weight in downstream analyses and ensure that highly-expressed genes do not dominate.

✎ Hands-on

1. **Remove confounders with scanpy** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG
   - *"Method used for plotting"*: Regress out unwanted sources of variation, using 'pp.regress_out'
     - *"Keys for observation annotation on which to regress on"*: total_counts, pct_counts_mito
2. **Inspect and Manipulate with scanpy** Tool:
   - *"Annotated data matrix"*: output of **Remove confounders** tool
   - *"Method used for inspecting"*: Scale data to unit variance and zero mean, using 'pp.scale'
     - *"Zero center?"*: Yes
     - *"Maximum value"*: 10.0 *

* It clips values exceeding a standard deviation of 10.

3. Rename the generated output 3k PBMC with only HVG, after scaling

3. Dimensionality reduction

Dimensionality reduction aims then to reduce the number of separate dimensions in the data and then:

- reduces the computational work in downstream analyses to only a few dimensions.
- reduces noise by averaging across multiple genes to obtain a more precise representation of the patterns in the data.
- enables effective plotting of the data.

3.1 Principal Component Analysis

Principal Component Analysis (PCA) is a dimensionality reduction technique consisting in the identification of axes in high-dimensional space that capture the largest amount of variation.
By applying PCA to scRNA-Seq, we assume that multiple genes are affected by the same biological processes in a coordinated way and random technical or biological noise affects each gene independently.
As more variation can be captured by considering the correlated behaviour of many genes, the top PCs are likely to represent the biological signal and the noise are concentrated into the later PCs.

Here we perform the PCA on the log-normalized expression values and compute the first 50 PCs.

✎ Hands-on

1. **Cluster, infer trajectories and embed with scanpy** Tool:
    - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling
    - *"Method used for plotting"*: Computes PCA (principal component analysis) coordinates, loadings and variance decomposition, using 'tl.pca'
        - *"Number of principal components to compute"*: 50
        - *"Type of PCA?"*: Full PCA
            - *"Compute standard PCA from covariance matrix?"*: Yes
            - *"SVD solver to use"*: ARPACK wrapper in SciPy
2. Rename the generated output 3k PBMC with only HVG, after scaling and PCA

Inspect the AnnData, we can see that 3 new objects (uns, obsm, varm) have been added to the AnnData object with information that seem related to PCA:

- uns is an unstructured annotation
- obsm multi-dimensional annotation of the observations (i.e. genes)
- varm multi-dimensional annotation of the variables (i.e. cells)

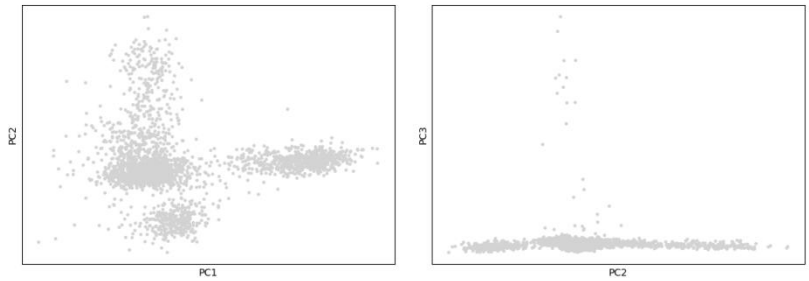
3.2 Visualization of PCA

Scanpy provides several useful ways of visualizing both cells and genes that define the PCA. The simplest approach for visualization is to plot the top 3 PCs on 2D plots (PC1 vs PC2 and PC2 vs PC3).

✎ Hands-on

Plot with scanpy Tool:

- *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling and PCA
- *"Method used for plotting"*: PCA: Plot PCA results, using 'pl.pca_overview'
    - In *"Plot attributes"*
        - In *"Component"*
            - Click on *"Insert Component"*
            - In *"1: Component"*
                - *"X-axis"*: 1
                - *"Y-axis"*: 2
            - Click on *"Insert Component"*
            - In *"1: Component"*
                - *"X-axis"*: 2
                - *"Y-axis"*: 3
        - *"Number of panels per row"*: 2

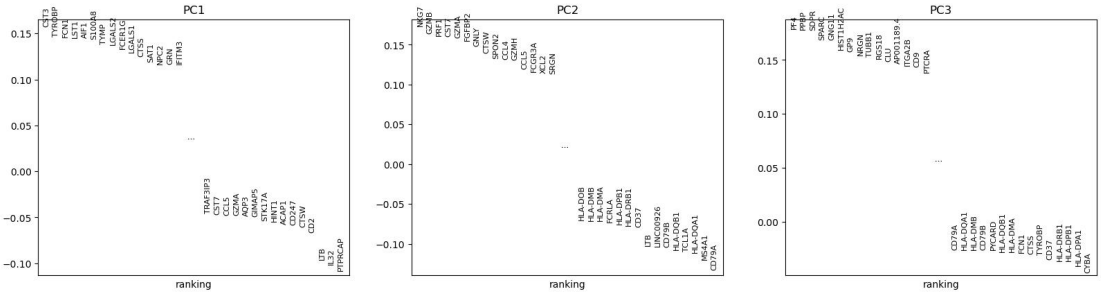


On these plots we see the different cells projected onto the first 3 PCs.
And now we will visualize the top genes associated with PCs.

✎ Hands-on

Plot with scanpy Tool:

- *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling and PCA
- *"Method used for plotting"*: PCA: Rank genes according to contributions to PCs, using 'pl.pca_loadings'
  - *"List of comma-separated components"*: 1,2,3



We see that CST3 is the gene the most associated with the 1st PC, NKG7 the one for the 2nd PC, and PF4 and PPBP for the 3rd PC (for consistency with the published *scanpy* and *Seurat* documentation, we will use PPBP).

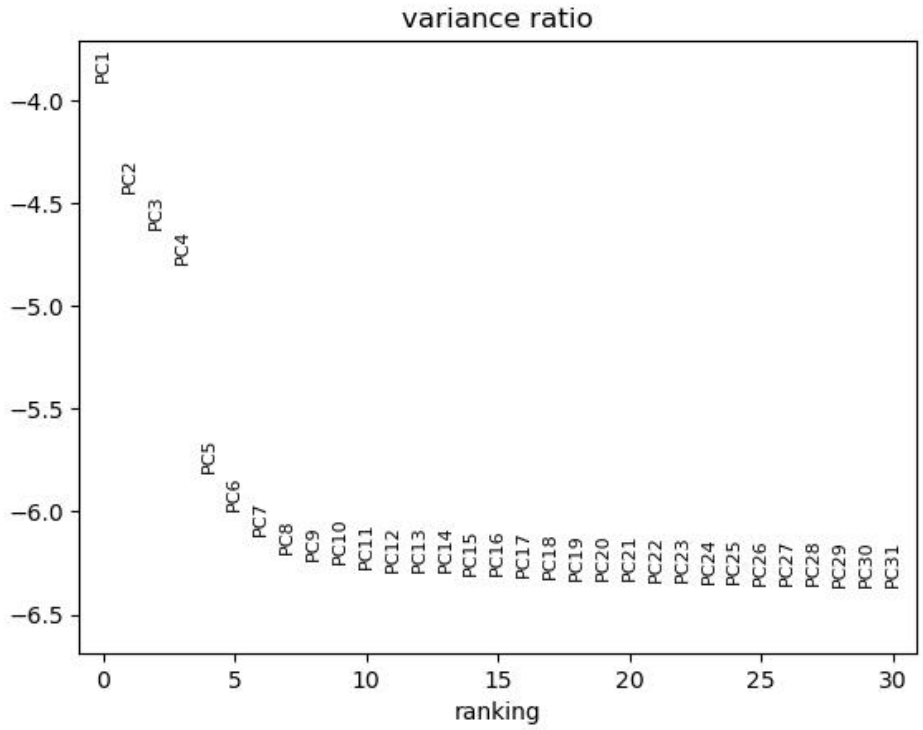
3.3 Determination of the number of PCs to keep

The choice of the number of PCs is a similar question to the choice of the number of highly variable genes to keep.
A simple heuristic for choosing the number of PCs generates an *"Elbow plot"*: a ranking of the PCs based on the percentage of variance they explain.

✎ Hands-on

Plot with scanpy Tool:

- *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling and PCA
- *"Method used for plotting"*: PCA: Scatter plot in PCA coordinates, using 'pl.pca_variance_ratio'
  - *"Use the log of the values?"*: Yes



To determine the elbow point, we assume that each of the PCs should explain much more variance than the remaining PCs. So after the last PCs we choose, the percentage of variance explained should not drop much.
Here we choose here to keep 10 PCs.

4. Clustering of the cells

Clustering summarizes the data and allows us to describe the population heterogeneity in terms of discrete and easily understandable labels.
The subpopulations can be afterwards treated as proxies for biological objects like cell types or states.

4.1 Computation of a neighborhood graph

Graph-based clustering has been popularized for clustering large scRNA-Seq datasets by its use in *Seurat*.
Here, to reproduce original results, we choose 10 neighbors for a **K-nearest neighbor (KNN)** graph, the **Euclidian distance metrics** and the **UMAP** method to compute the connectivities.

✎ Hands-on

1. **Inspect and Manipulate with scanpy** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling and PCA
   - *"Method used for inspecting"*: Compute a neighborhood graph of observations, using 'pp.neighbors'
     - *"The size of local neighborhood (in terms of number of neighboring data points) used for manifold approximation"*: 10
     - *"Number of PCs to use"*: 10
     - *"Use a hard threshold to restrict the number of neighbors to n_neighbors?"*: Yes
     - *"Method for computing connectivities"*: umap (McInnes et al, 2018)
     - *"Distance metric"*: euclidean
2. Rename the generated output 3k PBMC with only HVG, after scaling, PCA and KNN graph

4.2 Visualization of the neighborhood graph

To visualize and explore the neighborhood graph, we can apply an extra step of non-linear dimensional reduction techniques to learn the underlying manifold of the data in order to place similar cells together in low-dimensional space.
Two techniques are commonly used for the non-linear dimensional reduction: *t-SNE* (t-distributed stochastic neighbor embedding) and *UMAP*. *Scanpy* authors recommend to use here *UMAP* as it better preserves trajectories and easily accommodates new data.

✎ Hands-on

1. **Cluster, infer trajectories and embed with scanpy** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA and KNN graph
   - *"Method used for plotting"*: Embed the neighborhood graph using UMAP, using 'tl.umap'
2. Rename the generated output 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP

We can see that an extra object X_umap has been added to obsm with the 2 UMAP coordinates for each cell, as a table of 2 columns and 2,638 lines.

4.3 Clustering of the neighborhood graph

Currently, the ***Louvain graph-clustering method*** is the modularity optimization technique recommended. We need to define a value for the resolution parameter. High values lead to a greater number of clusters.
For single-cell datasets of around 3K cells, we recommend to use a value between 0.4 and 1.2. For larger datasets, the optimal resolution will be higher.

✎ Hands-on

1. **Cluster, infer trajectories and embed with scanpy** Tool:
    - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP
    - *"Method used for plotting"*: Cluster cells into subgroups, using 'tl.louvain'
        - *"Flavor for the clustering"*: vtraag (much more powerful)
            - *"Resolution"*: 0.45
2. Rename the generated output 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering

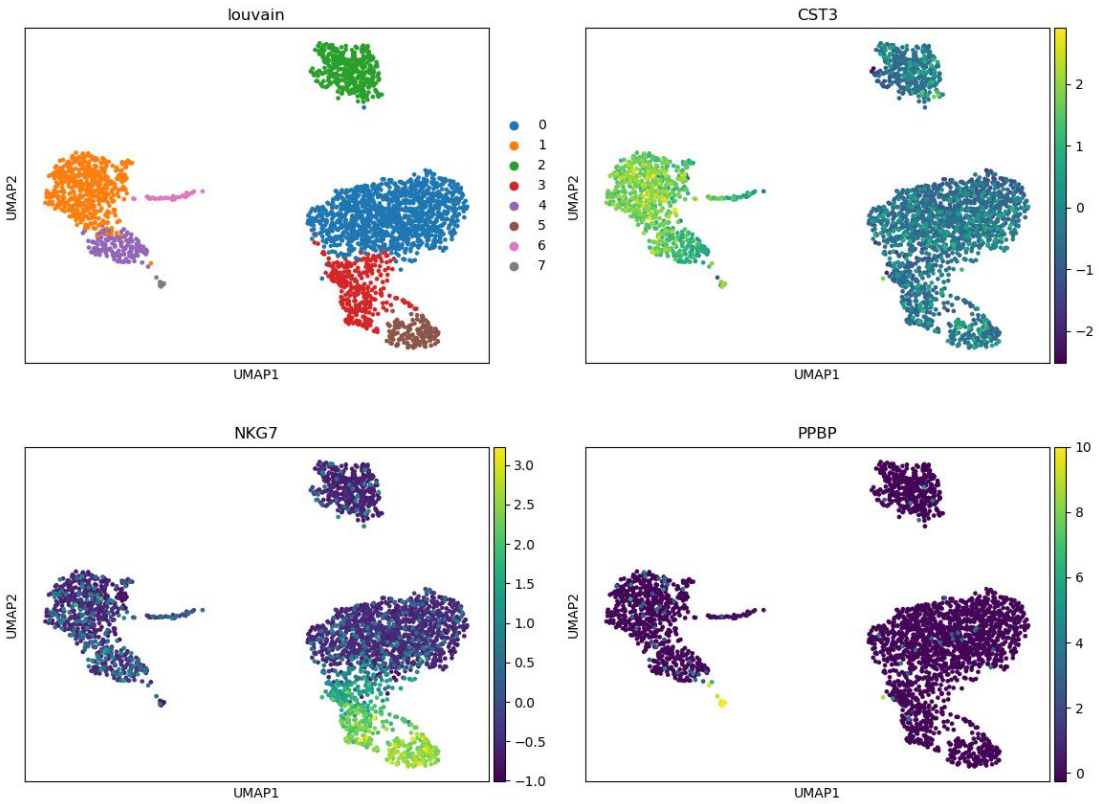
Now an extra column louvain has been added to the obs object with the cluster id for each cell. The cells in the same clusters should be co-localized in the UMAP coordinate plots.

We can plot the neighborhood graph and the clusters now.

✎ Hands-on

Plot with scanpy Tool:

- *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering
- *"Method used for plotting"*: Embeddings: Scatter plot in UMAP basis, using 'pl.umap'
    - *"Keys for annotations of observations/cells or variables/genes"*: louvain, CST3, NKG7, PPBP
    - In *"Plot attributes"*
        - *"Number of panels per row"*: 2
        - *"Colors to use for plotting categorical annotation groups"*: rainbow



8 clusters are identified.

5. Finding marker genes

To give sense to the clusters, we need to identify the genes that drive separation between clusters. Marker genes are usually detected by their differential expression between clusters, as the more strongly DE genes are more likely to have caused separate clustering of cells.
The identification of the marker genes for each cluster is made not only on the highly variable genes, but on the whole set (currently stored in the raw attribute).

5.1 Using Wilcoxon rank sum test

Wilcoxon rank sum test is a widely used method for pairwise comparisons between groups of observations. This test directly assesses separation between the expression distributions of different clusters.

✎ Hands-on

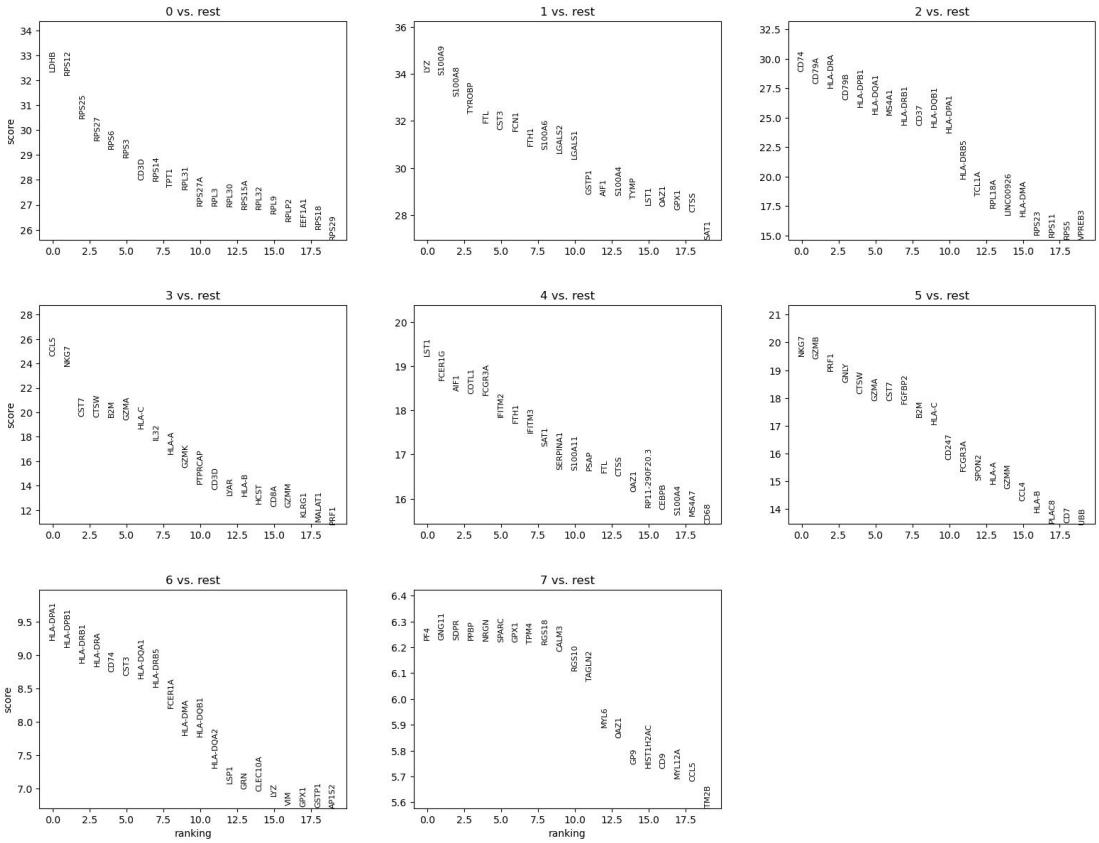
1. **Inspect and Manipulate with scanpy** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering
   - *"Method used for inspecting"*: Rank genes for characterizing groups, using 'tl.rank_genes_groups'
     - *"The key of the observations grouping to consider"*: louvain
     - *"Use 'raw' attribute of input if present"*: Yes
     - *"Comparison"*: Compare each group to the union of the rest of the group
     - *"The number of genes that appear in the returned tables"*: 100
     - *"Method"*: Wilcoxon-Rank-Sum
       - *"P-value correction method"*: Benjamini-Hochberg

2. Rename the generated output 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test

3. **Plot with scanpy** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test
   - *"Method used for plotting"*: Marker genes: Plot ranking of genes using dotplot plot, using 'pl.rank_genes_groups'
     - *"Number of genes to show"*: 20
     - *"Number of panels per row"*: 3
     - *"Should the y-axis of each panels be shared?"*: No

4. **Inspect AnnData** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test
   - *"What to inspect?"*: Unstructured annotation (uns)
     - *"What to inspect in uns?"*: Rank gene groups (rank_genes_groups)

5. Rename the Names for rank genes file to Ranked genes with Wilcoxon test
6. Inspect Ranked genes with Wilcoxon test file



0	1	2	3	4	5	6	7
LDHB	LYZ	CD74	CCL5	LST1	NKG7	HLA-DPA1	PF4
RPS12	S100A9	CD79A	NKG7	FCER1G	GZMB	HLA-DPB1	GNG11
RPS25	S100A8	HLA-DRA	CST7	AIF1	PRF1	HLA-DRB1	SDPR
RPS27	TYROBP	CD79B	CTSW	COTL1	GNLY	HLA-DRA	PPBP
RPS6	FTL	HLA-DPB1	B2M	FCGR3A	CTSW	CD74	NRGN
RPS3	CST3	HLA-DQA1	GZMA	IFITM2	GZMA	CST3	SPARC
CD3D	FCN1	MS4A1	HLA-C	FTH1	CST7	HLA-DQA1	GPX1
RPS14	FTH1	HLA-DRB1	IL32	IFITM3	FGFBP2	HLA-DRB5	TPM4
TPT1	S100A6	CD37	HLA-A	SAT1	B2M	FCER1A	RGS18
RPL31	LGALS2	HLA-DQB1	GZMK	SERPINA1	HLA-C	HLA-DMA	CALM3

We see that:

1. CST3 is a ranked genes for clusters 1, 4, 6.
2. NKG7 for clusters 3 and 5.
3. PPBP for cluster 7.

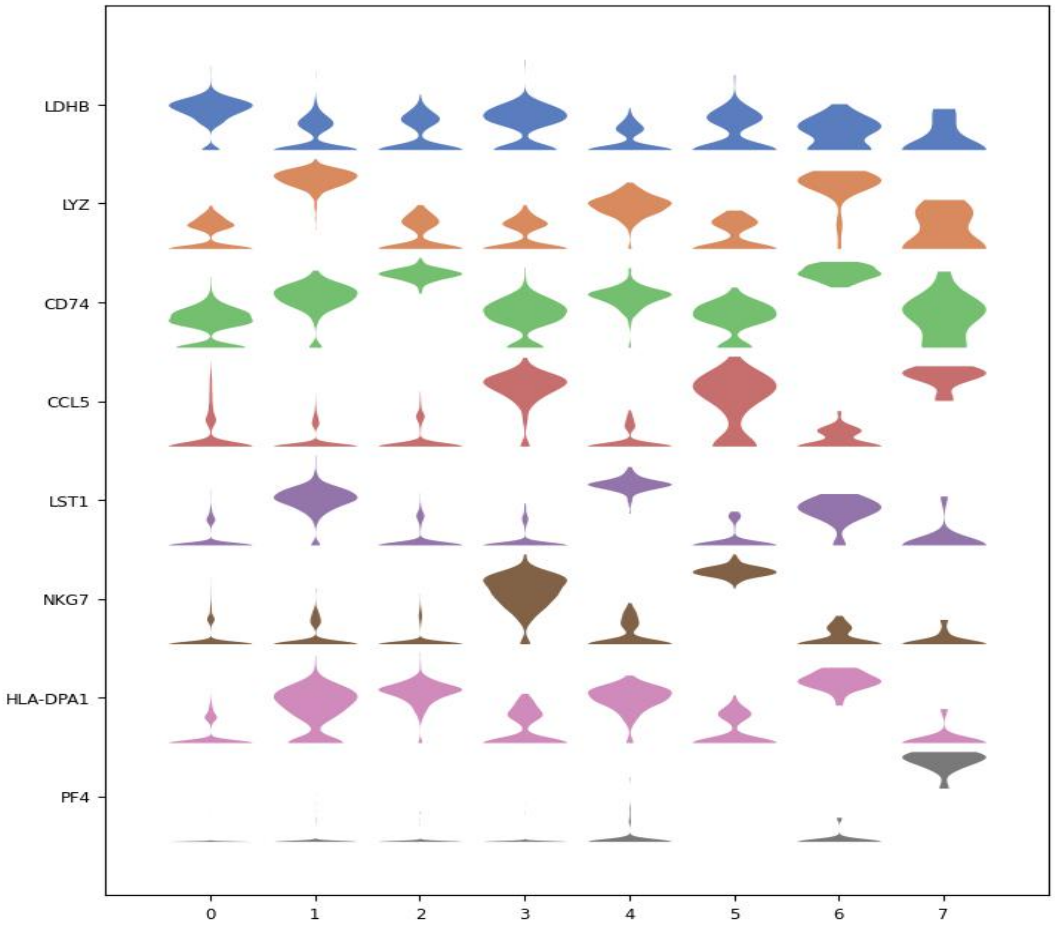
5.2 Expression of the top marker genes in each cluster

The first way to visualize the expression of the top marker genes is to look at the distribution of the expression of each marker gene in cells for each cluster.

✎ Hands-on

Plot with scanpy Tool:

- *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test
- *"Method used for plotting"*: Generic: Stacked violin plot, using 'pl.stacked_violin'
  - *"Variables to plot (columns of the heatmaps)"*: Subset of variables in 'adata.var_names'
    - *"List of variables to plot"*: LDHB, LYZ, CD74, CCL5, LST1, NKG7, HLA-DPA1, PF4
  - *"The key of the observation grouping to consider"*: louvain
  - *"Number of categories"*: 8
  - *"Use 'raw' attribute of input if present"*: Yes
  - *"Custom figure size"*: Yes
  - *"Swap axes?"*: Yes
  - In *"Violin plot attributes"*:
    - "Add a stripplot on top of the violin plot": No



Question:

Are the top marker genes only expressed in the cluster for which they are markers?

Solution:

- LDHB, LYZ and CD74, even if they are top markers genes for the cluster 0, 1, 2 respectively, are also expressed in all other clusters (and also found in the top 100 marker genes for other clusters), but with higher level in the cluster for they are markers.
- CCL5, LST1, NKG7 and HLA-DPA1 are not expressed in all clusters but also not only in the one they are markers.
- PF4 is only expressed in cluster 7.

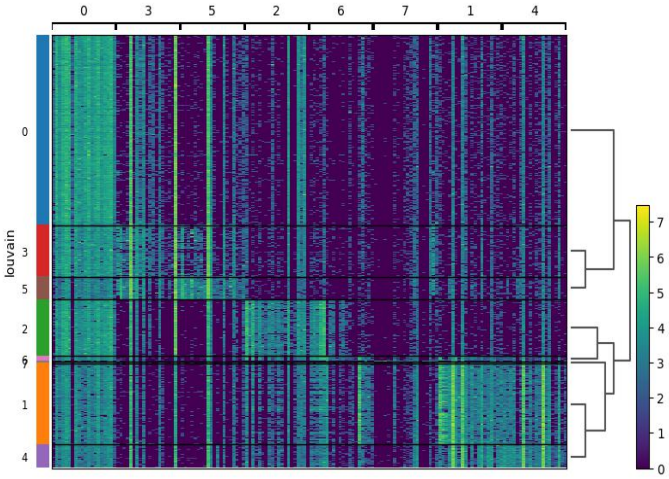
5.3 Expression of the top 20 markers genes in cells for each cluster

We would like now to have a look at the expression of the top 20 marker genes in the different cells for each cluster.

✎ Hands-on

Plot with scanpy Tool:

- *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test
- *"Method used for plotting"*: Marker genes: Plot ranking of genes as heatmap plot, using 'pl.rank_genes_groups_heatmap'
  - *"Number of genes to show"*: 20
  - *"Use 'raw' attribute of input if present"*: Yes
  - *"Compute and plot a dendrogram?"*: Yes



Question:
How are clusters close to each other in terms of expression of the top 20 marker genes?

Solution:
- Clusters 0, 3, and 5 are similar in term of expression. This was expected as they are physically close on the neighborhood graph.
- Clusters 1 and 4 are together and then 2 and 6 are together after 7. These observations are less expected given the neighborhood graph: 1 and 4 are physically close, but 2 is far from 7 and 6.

5.3 Comparison of the marker genes between clusters

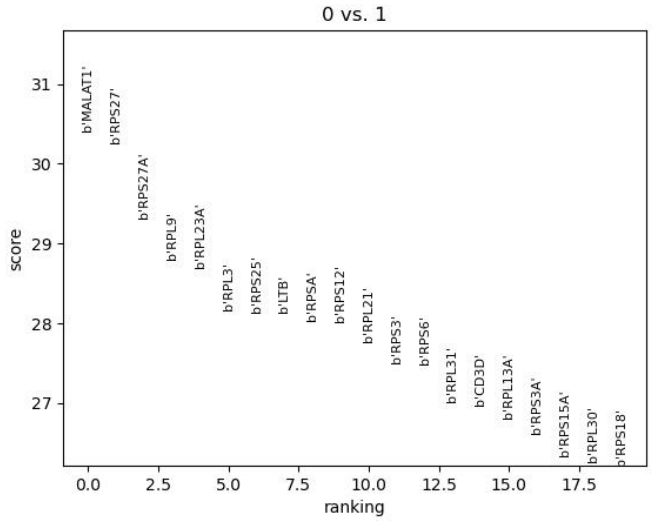
In some cases, it may also be interesting to find marker genes distinguishing one given cluster from one or several clusters. For example, the marker genes distinguishing cluster 0 from cluster 1.

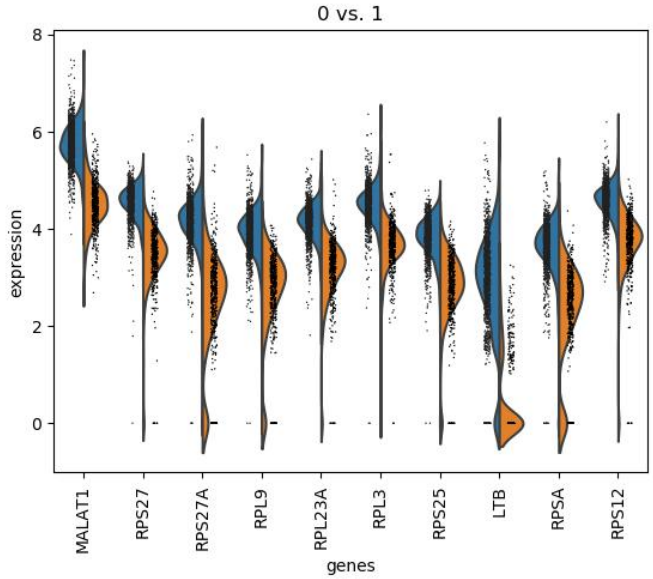
✎ Hands-on

1. **Inspect and Manipulate with scanpy** Tool:
    - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test
    - *"Method used for inspecting"*: Rank genes for characterizing groups, using 'tl.rank_genes_groups'
        - *"The key of the observations grouping to consider"*: louvain
        - *"Use 'raw' attribute of input if present"*: Yes
        - *"Subset of groups to which comparison shall be restricted"*: 0
        - *"Comparison"*: Compare with respect to a specific group
            - *"Group identifier with respect to which compare"*: 1
        - *"The number of genes that appear in the returned tables"*: 100
        - *"Method"*: Wilcoxon-Rank-Sum
            - *"P-value correction method"*: Benjamini-Hochberg
2. Rename the generated output 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes for 0 vs 1 with Wilcoxon test

3. **Plot with scanpy** Tool:
    - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes for 0 vs 1 with Wilcoxon test
    - *"Method used for plotting"*: Marker genes: Plot ranking of genes using dotplot plot, using 'pl.rank_genes_groups'
        - *"Number of genes to show"*: 20
        - *"Should the y-axis of each panels be shared?"*: No

4. **Plot with scanpy** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes for 0 vs 1 with Wilcoxon test
   - *"Method used for plotting"*: Marker genes: Plot ranking of genes as violin plot, using 'pl.rank_genes_groups_violin'
     - *"Which genes to plot?"*: A number of genes
       - *"Number of genes to show"*: 10
     - *"Use 'raw' attribute of input if present"*: Yes





It shows that MALAT1 is the most differentially expressed gene between cells in cluster 0 and cells in cluster 1, even though it was not in the set of top marker genes for cluster 0.

6. Cell type annotation

Determining what biological state is represented by each of those clusters is likely the most challenging task in scRNA-Seq data analysis. To do so, we need to bridge the gap between our current dataset and prior biological knowledge.
Canonical markers are usually found in the literature and are also aggregated into dedicated database like the PanglaoDB.
In the case of our dataset, we can use canonical markers to known cell types:

Cell type	Marker genes
CD4+ T cells	IL7R, CCR7
CD8+ T cells	CD8A
CD14+ Monocytes	CD14, LYZ
B cells	MS4A1, CD79A
Natural killer (NK) cells	GNLY, NKG7, KLRB1
Dendritic Cells	FCER1A, CST3
Megakaryocytes	PPBP
FCGR3A+ Monocytes	FCGR3A, MS4A7

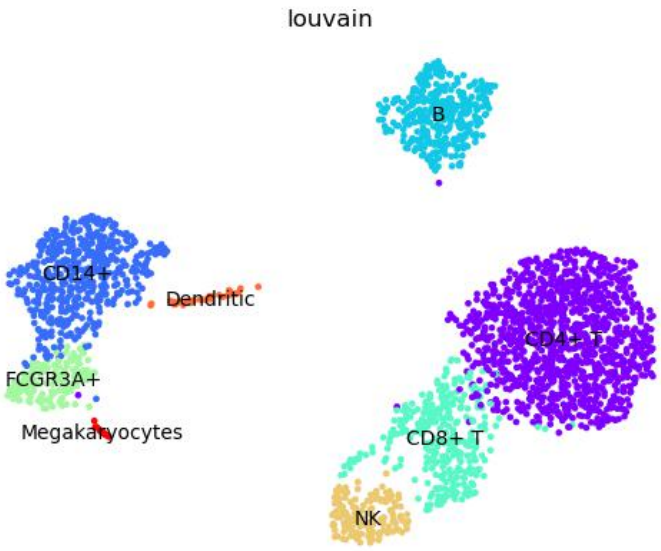
These canonical marker genes can be match the clusters to known cell types:

Cluster	Cell type
0	CD4+ T cells
1	CD14+ Monocytes
2	B cells
3	CD8+ T cells
4	FCGR3A+ Monocytes
5	Natural killer (NK) cells
6	Dendritic Cells
7	Megakaryocytes

✎ Hands-on

1. **Manipulate Anndata** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test
     Note: Take note that this is not the "0 vs 1 Wilcoxon" dataset!
   - *"Function to manipulate the object"*: Rename categories of annotation
     - *"Key for observations or variables annotation"*: louvain
     - *"Comma-separated list of new categories"*: CD4+ T, CD14+, B, CD8+ T, FCGR3A+, NK, Dendritic, Megakaryocytes
2. Rename the generated output 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test, annotation

3. **Plot with scanpy** Tool:
   - *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test, annotation
   - *"Method used for plotting"*: Embeddings: Scatter plot in UMAP basis, using 'pl.umap'
     - *"Keys for annotations of observations/cells or variables/genes"*: louvain
     - *"Use 'raw' attribute of input if present"*: Yes
     - In *"Plot attributes"*
       - *"Location of legend"*: on data
       - *"Draw a frame around the scatter plot?"*: No
       - *"Colors to use for plotting categorical annotation groups"*: rainbow



From the plot we can find that T cells (CD4+ and CD8+) are clustered together with NK cells. Monocytes cells (CD14+ and FCGR3A+) are close to each others, with Dendritic and Megakaryocytes. B cells are physically independent.

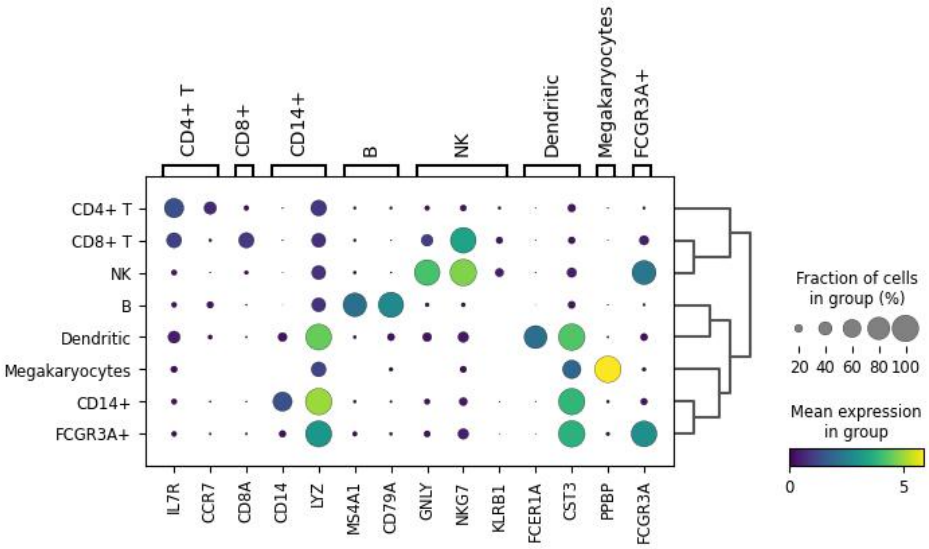
In addition, we can also plot expression of canonical marker genes for the annotated cell types.

✎ Hands-on

Plot with scanpy Tool:

- *"Annotated data matrix"*: 3k PBMC with only HVG, after scaling, PCA, KNN graph, UMAP, clustering, marker genes with Wilcoxon test, annotation
- *"Method used for plotting"*: Generic: Makes a dot plot of the expression values, using 'pl.dotplot'
  - *"Variables to plot (columns of the heatmaps)"*: Subset of variables in 'adata.var_names'
    - *"List of variables to plot"*: IL7R, CCR7, CD8A, CD14, LYZ, MS4A1, CD79A, GNLY, NKG7, KLRB1, FCER1A, CST3, PPBP, FCGR3A
  - *"The key of the observation grouping to consider"*: louvain
  - *"Number of categories"*: 8
  - *"Use 'raw' attribute of input if present"*: Yes
  - *"Compute and plot a dendrogram?"*: Yes
  - In *"Group of variables to highlight"*
    - Click on *"Group of variables to highlight"*
    - In *"1: Group of variables to highlight"*
      - *"Start"*: 0
      - *"End"*: 1
      - *"Label"*: CD4+ T
    - Click on *"Group of variables to highlight"*
    - In *"2: Group of variables to highlight"*
      - *"Start"*: 2
      - *"End"*: 2
      - *"Label"*: CD8+
    - Click on *"Group of variables to highlight"*
    - In *"3: Group of variables to highlight"*
      - *"Start"*: 3
      - *"End"*: 4
      - *"Label"*: CD14+
    - Click on *"Group of variables to highlight"*
    - In *"4: Group of variables to highlight"*
      - *"Start"*: 5
      - *"End"*: 6
      - *"Label"*: B

- Click on *"Group of variables to highlight"*
- In *"5: Group of variables to highlight"*
  - *"Start"*: 7
  - *"End"*: 9
  - *"Label"*: NK
- Click on *"Group of variables to highlight"*
- In *"6: Group of variables to highlight"*
  - *"Start"*: 10
  - *"End"*: 11
  - *"Label"*: Dendritic
- Click on *"Group of variables to highlight"*
- In *"7: Group of variables to highlight"*
  - *"Start"*: 12
  - *"End"*: 12
  - *"Label"*: Megakaryocytes
- Click on *"Group of variables to highlight"*
- In *"8: Group of variables to highlight"*
  - *"Start"*: 13
  - *"End"*: 13
  - *"Label"*: FCGR3A+



Conclusion

In this tutorial, we investigated clustering and annotation of single-cell data from 10x Genomics using ***Scanpy***.
This workflow used here was typical for scRNA-seq data analysis:

1.Preprocessing with:
   1. Selection and filtration of cells and genes based on quality metrics
   2. Data normalization and scaling
   3. Selection of features, i.e. marker genes
2.Reduction of the dimensionality via a Principal Component Analysis
3.Clustering of the cells by:
   1. Computation of a neighborhood graph
   2. Clustering of the neighborhood graph into 8 clusters of cells
4.Identification of marker genes for the clusters
5.Annotation of the clusters with cell types

References:

1. Mehmet Tekman, Hans-Rudolf Hotz, Daniel Blankenberg, Wendi Bacon, 2021 Pre-processing of 10X Single-Cell RNA Datasets (Galaxy Training Materials). /training-material/topics/transcriptomics/tutorials/scrna-preprocessing-tenx/tutorial.html Online; accessed Fri Apr 16 2021

2. Bérénice Batut, Hans-Rudolf Hotz, Mehmet Tekman, 2021 Clustering 3K PBMCs with Scanpy (Galaxy Training Materials). /training-material/topics/transcriptomics/tutorials/scrna-scanpy-pbmc3k/tutorial.html Online; accessed Thu Apr 15 2021