Single-cell RNA-seq workshop ed. 2024

Table of contents

Welcome **6**
Overview . . . 6
Format . . . 6
Learning outcomes . . . 7
Instructor . . . 7

Program **8**
Day 1 . . . 8
Day 2 . . . 8
Day 3 . . . 8
Day 4 . . . 9
Day 5 . . . 9

Docker image **10**

RStudio Server **11**

Course material **12**

Session info **13**

I Day 1 **16**

1 Demonstration: From fastq to count matrix **17**
1.1 1. Download sequencing reads in fastq format . . . 17
1.2 2. Prepare genome for alignment . . . 17
1.3 3. Map reads onto 10X-formatted genome with cellranger . . . 18
1.4 4. Check output files . . . 19

2 Exercises: From bcl to count matrix **20**
2.1 0. Introduction to `shell` terminal . . . 20
2.2 1. Prepare a place in your computer where you will follow the workshop . . . 22
2.2.1 Create a directory for the workshop . . . 22
2.2.2 Clone github directory in the workshop directory . . . 22

2.3 2. Process raw files into fastq files . . . 23
2.3.1 Getting input toy dataset . . . 23
2.3.2 Running `cellranger mkfastq` . . . 24
2.4 3. Generate gene count matrices with cellranger count . . . 26
2.4.1 Download genome index for the toy dataset . . . 26
2.4.2 Running `cellranger count` . . . 27
2.4.3 Checking `count` output files . . . 27
2.5 3 [Alternative] Generate gene count matrices with STARsolo . . . 28
2.6 4. Obtain single-cell RNA-seq datasets . . . 29
2.6.1 Raw fastq reads from GEO . . . 29
2.6.2 [BONUS] Pre-processed count matrices . . . 30

II Day 2 **32**

3 Demonstration: leveraging R/Bioconductor for single-cell analyses **33**
3.1 1. Installing packages in `R` . . . 33
3.2 2. Basic `R` and `Bioconductor` classes . . . 34
3.2.1 Important R concepts: . . . 34
3.2.2 `tibble` tables: . . . 34
3.2.3 Reading text files into `tibbles` . . . 35
3.2.4 Handling of `tibbles`: . . . 35
3.2.5 `%>%` pipe: . . . 36
3.2.6 Important Bioconductor concepts: . . . 36
3.2.7 `SummarizedExperiment` class: . . . 36
3.2.8 `GenomicRanges` class (a.k.a. `GRanges`): . . . 38
3.3 3. CRAN & Bioconductor approaches to scRNAseq . . . 39
3.3.1 scRNAseq in Bioconductor . . . 39

4 Exercises: scRNAseq analysis with R/Bioconductor (1/3) **42**
4.1 1. Import single-cell RNA-seq data in R . . . 42
4.1.1 Import data from cellranger workflow . . . 42
4.1.2 Use "pre-compiled" datasets . . . 44
4.1.3 UMI number / cell . . . 45
4.2 2. Filter emtpy and doublet droplets . . . 46
4.2.1 Removing empty droplets . . . 46
4.2.2 Flagging cell doublets . . . 48
4.3 3. (Optional) Exclude non-relevant genes from analysis . . . 48
4.4 4. Normalize data . . . 50
4.4.1 Log-normalization . . . 50
4.4.2 [BONUS] VST normalization . . . 52

III Day 3 **56**

5 Demonstration: Dimensional reduction visualization and clustering **57**
5.1 1. Dimensionality reduction . . . 57
5.2 2. Cell clustering . . . 60
5.3 3. Gene expression visualization . . . 62

6 Exercises: scRNAseq analysis with R/Bioconductor (2/3) **64**
6.1 1. Pre-processing PBMC dataset . . . 64
6.1.1 Preparing dataset . . . 64
6.1.2 Remove doublets and filter non-relevant genes . . . 65
6.1.3 Normalize counts using `sctransform` . . . 66
6.2 2. Dimensionality reduction . . . 67
6.2.1 Selection of hyper-variable genes (HVGs) . . . 67
6.2.2 Embedding in a lower dimensional linear space . . . 68
6.3 3. Clustering . . . 70
6.3.1 Clustering algorithms . . . 70
6.3.2 Dimensional reduction for clustering visualization . . . 71

IV Day 4 **73**

7 Demonstration: Cell type annotation and dataset integration **74**
7.1 1. Integrating two replicates together . . . 74
7.2 2. Reading CcnoKO dataset in R . . . 77
7.3 3. Annotating CcnoKO dataset with WT MCCs dataset using `scmap` . . . 77
7.4 4. Mapping CcnoKO onto WT MCCs cells . . . 78

8 Exercises: scRNAseq analysis with R/Bioconductor (3/3) **81**
8.1 0. Pre-processing PBMC dataset . . . 81
8.2 1. Differential expression analysis and marker genes . . . 83
8.3 2. Automated cell annotation . . . 85
8.4 3. Subclustering of T cells . . . 89

V Day 5 **93**

9 Demonstration: Trajectory inference and pseudotime **94**
9.1 1. Trajectories in WT MCCs . . . 94
9.2 2. Infer a single trajectory for WT + CcnoKO MCCs . . . 95
9.3 3. Perform DE expression analysis on a portion of the trajectory . . . 97
9.3.1 Filter cells based on 99% max of pseudotime_slingshot for CcnoKO . . . 97
9.3.2 Running tradeSeq using new slingshot pseudotimes . . . 97

10 Exercises: Trajectory inference and RNA velocity **99**
10.1 0. Prepare data from scratch . . . 99
10.2 1. Process testis data in R . . . 103
10.3 2. Trajectory inference (TI) in scRNAseq . . . 107
10.3.1 Slingshot . . . 109
10.3.2 Pseudotime inference and expression modelling . . . 112
10.4 3. Ordering trajectory with RNA velocity . . . 113

Appendices **114**

A Useful resources **114**
A.1 General bioinformatics . . . 114
A.2 R/Bioconductor . . . 114
A.3 Scientific readings . . . 114

Welcome

Package: scRNAseqWorkshop **Authors:** First Last [aut, cre] **Compiled:** 2024-01-16 **Package version:** 0.98.0 **R version:** R Under development (unstable) (2024-01-09 r85796) **BioC version:** 3.19 **License:** MIT + file LICENSE

This is the landing page of the workshop entitled *Single-cell RNA-seq analysis with R/Bioconductor*.

Overview

This course will introduce biologists and bioinformaticians to the field of single-cell RNA sequencing. We will cover a range of software and analysis workflows that extend over the spectrum from the best practices in the filtering scRNA-seq data
to the downstream analysis of cell clusters and temporal ordering. This course will help the attendees gain accurate insights in pre-processing, analysis and interpretation of scRNA-seq data.

We will start by introducing general concepts about single-cell RNA-sequencing. From there, we will then continue to describe the main analysis steps to go from raw sequencing data to processed and usable data. We will present classical analysis workflows, their output and the possible paths to investigate for downstream analysis.

Throughout this workshop, we will put an emphasis on `R/Bioconductor` ecosystem and the different packages which will be used to analyse datasets and learn new approaches.

Format

The course is structured in modules over five days.

During the first 1/3 of the day, formal lectures will cover the key concepts required to understand the principles of scRNA-seq analysis (~2h).

Following these lectures, practical examples will be shown to illustrate how to translate the acquired knowledge into functional `R` code (~1h). At this stage, trainees will get acquainted with state-of-the-art packages for scRNAseq analysis as well as the best practices in bioinformatics.

During the second half of the day (3h), trainees will work by themselves, following guided exercises to improve their understanding of scRNAseq analysis workflow. Hints and solution are provided for each exercise. The exercises will mainly focus on specific concepts introducted earlier that day. However, analytical steps studied throughout the previous days will also be integrated so that towards the end of the week, trainees are fully able to perform fundamental scRNAseq analyses from beginning to end.

Office hours will take place during the last hour of the exercises. An instructor will be available to answer individual questions related to daily exercises. A Slack channel will also be available so that Q&A are available for everybody.

Learning outcomes

At the end of this course, you should be able to:

- Understand the pros/cons of different single-cell RNA-seq methods
- Process and QC of scRNA-seq data
- Normalize scRNA-seq data
- Correct for batch effects
- Visualise the data and applying dimensionality reduction
- Perform cell clustering and annotation
- Perform differential gene expression analysis
- Infer cell trajectory and pseudotime, and perform temporal differential expression

Instructor

Dr. Jacques Serizay

Program

Day 1

- **[1h]** Lecture 1 - General introduction to single-cell RNA-seq experimental design [Pptx]
- **[1h]** Lecture 2 - scRNAseq: from raw sequencing files to count matrix [Pptx]
- **[1h]** Demonstration 1 - From fastq to count matrix [HTML | qmd (save to open in RStudio)]
- **[3h]** Homework - From bcl to count matrix [HTML | qmd (save to open in RStudio)]

Day 2

- **[1h]** Lecture 3 - Filtering cells in droplet-based scRNAseq data [Pptx]
- **[1h]** Lecture 4 - Normalizing scRNAseq data [Pptx]
- **[BONUS]** R/Bioconductor essentials: `GRanges` and `*Experiment` classes [Pptx]
- **[1h]** Demonstration 2 - Leveraging R/Bioconductor for single-cell analyses [HTML | qmd (save to open in RStudio)]
- **[3h]** Homework - scRNAseq analysis with R/Bioconductor (1/3) [HTML | qmd (save to open in RStudio)]

Day 3

- **[2h]** Lecture 5 - Clustering cells in scRNAseq [Pptx]
- **[1h]** Demonstration 3 - Dimensional reduction visualization and clustering [HTML | qmd (save to open in RStudio)]
- **[3h]** Homework - scRNAseq analysis with R/Bioconductor (2/3) [HTML | qmd (save to open in RStudio)]

Day 4

- **[1h]** Lecture 6 - Cell type annotations [Pptx]
- **[1h]** Lecture 7 - Batch correction [PDF]
- **[1h]** Demonstration 4 - Annotation transfer with `scmap` [HTML | qmd (save to open in RStudio)]
- **[3h]** Homework - scRNAseq analysis with R/Bioconductor (3/3) [HTML | qmd (save to open in RStudio)]

Day 5

- **[2h]** Lecture 8 - Trajectory inference and RNA velocity [Pptx]
- **[1h]** Demonstration 5 - Trajectory inference in multiciliated cells [HTML | qmd (save to open in RStudio)]
- **[3h]** Homework - Advanced scRNAseq topics: trajectory inference and RNA velocity [HTML | qmd (save to open in RStudio)]

Docker image

A `Docker` image built from this repository is available here:

ghcr.io/js2264/scrnaseqworkshop

Get started now

You can get access to all the packages used in this book in $<$ 1 minute, using this command in a terminal:

Listing 0.1 `bash`

```bash
docker run -it ghcr.io/js2264/scrnaseqworkshop:devel R
```

RStudio Server

An RStudio Server instance can be initiated from the `Docker` image as follows:

Listing 0.2 `bash`

```
docker run \
    --volume <local_folder>:<destination_folder> \
    -e PASSWORD=OHCA \
    -p 8787:8787 \
    ghcr.io/js2264/scrnaseqworkshop:devel
```

The initiated RStudio Server instance will be available at https://localhost:8787.

Course material

The workshop content is available at this adress: https://github.com/js2264/scRNAseq-workshop.

You can clone it locally with `git`:

```
git clone https://github.com/js2264/scRNAseq-workshop.git
```

To download it without the command-line tool `git`, go to the GitHub repo page, click on the green `Code` button, then `Download ZIP`. Beware, the download may take a significant time based on your internet connection (several hundreds MB).

Session info

Click to expand

Part I

Day 1

1 Demonstration: From fastq to count matrix

Goals:

- Introduce the `cellranger` toolkit

1.1 1. Download sequencing reads in fastq format

Here we will process a single-cell RNA-seq dataset provided by 10X Genomics, as an example.

Here is the link to the dataset: link

This is a single-cell RNA-seq sample from mouse embryonic (E18) heart cells. Let's first download the raw fastqs.

```
mkdir -p data/E18_Heart/fastq
curl https://cf.10xgenomics.com/samples/cell-exp/3.0.0/heart_1k_v3/heart_1k_v3_fastqs.tar
tar -xvf data/E18_Heart/fastq/heart_1k_v3_fastqs.tar
mv heart_1k_v3_fastqs/ data/E18_Heart/fastq/
ls --color -ltFh data/E18_Heart/fastq/heart_1k_v3_fastqs
zcat data/E18_Heart/fastq/heart_1k_v3_fastqs/heart_1k_v3_S1_L001_R1_001.fastq.gz | head
zcat data/E18_Heart/fastq/heart_1k_v3_fastqs/heart_1k_v3_S1_L001_R2_001.fastq.gz | head
zcat data/E18_Heart/fastq/heart_1k_v3_fastqs/heart_1k_v3_S1_L001_I1_001.fastq.gz | head
```

1.2 2. Prepare genome for alignment

Download GRCm38 genome reference and gene annotations from iGenomes to ensure that genome reference and gene annotations are uniformly processed.

```
# Download files
mkdir data/E18_Heart/GRCm38/ && cd data/E18_Heart/GRCm38
curl http://igenomes.illumina.com.s3-website-us-east-1.amazonaws.com/Mus_musculus/Ensembl/
```

```
tar -xzvf Mus_musculus_Ensembl_GRCm38.tar.gz
# Clean up gtf file to remove unscaffolded contigs
grep -vP "^CHR|^GL|^JH" Mus_musculus/Ensembl/GRCm38/Annotation/Genes/genes.gtf > Mus_muscu
cut -f1 Mus_musculus/Ensembl/GRCm38/Annotation/Genes/genes_filtered.gtf | uniq -c
# Build cellranger index
cellranger mkref \
    --genome=GRCm38 \
    --fasta=Mus_musculus/Ensembl/GRCm38/Sequence/WholeGenomeFasta/genome.fa \
    --genes=Mus_musculus/Ensembl/GRCm38/Annotation/Genes/genes_filtered.gtf \
    --nthreads=18 \
    --memgb=40
cd ../../../
ls --color -lthF data/E18_Heart/GRCm38/GRCm38
```

If you fail to download/build Cellranger index, you can always get another version from here:

```
wget https://cf.10xgenomics.com/supp/cell-exp/refdata-gex-mm10-2020-A.tar.gz
#md5sum: 886eeddde8731ffb58552d0bb81f533d
tar -xzvf refdata-gex-mm10-2020-A.tar.gz
ls --color -lthF data/E18_Heart/GRCm38/GRCm38
```

1.3 3. Map reads onto 10X-formatted genome with cellranger

```
mkdir -p data/E18_Heart/cellranger && cd data/E18_Heart/cellranger
cellranger count \
    --id=E18_Heart \
    --transcriptome=../../../data/E18_Heart/GRCm38/GRCm38 \
    --fastqs=../../../data/E18_Heart/fastq/heart_1k_v3_fastqs \
    --expect-cells=1000 \
    --localcores=18 \
    --localmem=40
cd ../../
```

Mapping takes a significant amount of time. On my machine, it takes nearly ~ 1h to finish.

This should appear at the end of the mapping/counting process:

```
## Outputs:
## - Run summary HTML:                         E18_Heart/web_summary.html
## - Run summary CSV:                          E18_Heart/metrics_summary.csv
```

```
## - BAM:                                        E18_Heart/possorted_genome_bam.bam
## - BAM index:                                  E18_Heart/possorted_genome_bam.bam.bai
## - Filtered feature-barcode matrices MEX:      E18_Heart/filtered_feature_bc_matrix
## - Filtered feature-barcode matrices HDF5:     E18_Heart/filtered_feature_bc_matrix.h5
## - Unfiltered feature-barcode matrices MEX:    E18_Heart/raw_feature_bc_matrix
## - Unfiltered feature-barcode matrices HDF5:   E18_Heart/raw_feature_bc_matrix.h5
## - Secondary analysis output CSV:              E18_Heart/analysis
## - Per-molecule read information:              E18_Heart/molecule_info.h5
## - Loupe Browser file:                         E18_Heart/cloupe.cloupe
## Waiting 6 seconds for UI to do final refresh.
## Pipestance completed successfully!
```

1.4 4. Check output files

A description of the different output files is available here.

```
ls --color -lthF data/E18_Heart/cellranger/E18_Heart/
ls --color -lthF data/E18_Heart/cellranger/E18_Heart/outs
```

We can check the `html` summary report in a web browser to get more insights about the results of the scRNAseq experiment.

```
data/E18_Heart/cellranger/E18_Heart/outs/web_summary.html
```

If samtools is installed, one can also check the bam file obtained using `cellranger count` workflow.

```
samtools view data/E18_Heart/cellranger/E18_Heart/outs/possorted_genome_bam.bam | head -n
samtools flagstat data/E18_Heart/cellranger/E18_Heart/outs/possorted_genome_bam.bam
```

The `analysis` folder contains relevant(ish) information obtained from after a rough post-alignment processing of the dataset by `cellranger`.

```
tree -L 2 data/E18_Heart/cellranger/E18_Heart/outs/analysis/
head data/E18_Heart/cellranger/E18_Heart/outs/analysis/clustering/graphclust/clusters.csv
cut -f 2 -d, data/E18_Heart/cellranger/E18_Heart/outs/analysis/clustering/graphclust/clust
```

2 Exercises: From bcl to count matrix

Goals:

- Understand the structure of raw sequencing files, fastq files, and output of `cellranger` workflow.
- Execute the `cellranger` pipeline (`mkfastq` + `count`) to see how things work!
- Learn more about public data access and recovery.

2.1 0. Introduction to `shell` terminal

`shell` (`sh`) is a software used to interpret commands typed in a terminal. It exists in both Mac and Linux environments.

The basic `sh` commands are useful to:

- Navigate within directories
- Manage files organization
- Launch command-line-based softwares (e.g. `cellranger`)

Here are some of the most important commands:

- Check your working directory

```
pwd
```

- Check history

```
history
```

- put history into a `history.txt` file

```
history > history.txt
```

- make a new folder called data

```
mkdir data
```

- Go to the new `data` directory

```
cd data
```

- move `history.txt` file into `data` directory

```
mv ../history.txt ./
```

- check manual page of `curl` command

```
man curl
```

- check specific help for `cellranger` command and subcommands

```
cellranger --help
cellranger count --help
```

- redirect `cellranger count` help output into a file called `cellranger-help.txt`

```
cellranger count --help > cellranger-help.txt
```

- Download a file from Internet with `curl`

```
curl https://cf.10xgenomics.com/supp/cell-exp/cellranger-tiny-bcl-1.2.0.tar.gz
```

- List all files in a folder

```
ls -l ~/
ls --color -Flh ~/
```

2.2 1. Prepare a place in your computer where you will follow the workshop

2.2.1 Create a directory for the workshop

Question

Open a terminal and navigate to your preferred location for the workshop.

Answer

```
# Create a directory for the workshop
cd ${HOME}
mkdir scRNAseq_Jan24
cd ${HOME}/scRNAseq_Jan24/
```

From now on, everything you do should take place in this folder! Be sure you have enough storage space in the filesystem you are using, as you will need lots of it!

2.2.2 Clone github directory in the workshop directory

Question

Download the `git` repository for this course from `GitHub`

Answer

```
cd ${HOME}/scRNAseq_Jan24/
git clone https://github.com/js2264/scRNAseq-workshop.git
```

This downloads the repository for this course to your home folder on the AWS machine.
To get it on your local computer (to save the lectures and exercises), you can also go to the GitHub repo page, click on the green `Code` button, then `Download ZIP`. Beware, the download may take a significant time based on your internet connection (several hundreds MB).

2.3 2. Process raw files into fastq files

NOTE: This is a step typically performed internally by sequencing platform, which delivers `.fastq` files rather than `.bcl` files.

First, familiarize yourself with `cellranger mkfastq` documentation: go to cellranger mkfastq webpage and read the **Overview**.

> Question
>
> What is the command you are going to use? What are the required and optional arguments for this command?
>
> > Answer
> >
> > An alternative to the web-based documentation is to use the command-line help:
> >
> > ```
> > cellranger mkfastq --help
> > ```

2.3.1 Getting input toy dataset

Let's download a toy dataset to process into fastq files. A `bcl` tiny file is available and provided by 10X Genomics at the following adress: https://cf.10xgenomics.com/supp/cell-exp/cellranger-tiny-bcl-1.2.0.tar.gz.

> Question
>
> Download the indicated `bcl` files and unzip it in a subdirectory called `data/bcl2fastq/`.
>
> > Answer
> >
> > ```
> > cd ${HOME}/scRNAseq_Jan24/
> > mkdir -p data/bcl2fastq/
> > curl https://cf.10xgenomics.com/supp/cell-exp/cellranger-tiny-bcl-1.2.0.tar.gz -o data
> > tar -xzvf data/bcl2fastq/cellranger-tiny-bcl-1.2.0.tar.gz && mv cellranger-tiny-bcl-1.
> > ```

Question

Explore the contents of the sequencing directory. What does each file correspond to? Can you locate the actual "sequencing" files?

Answer

```
ls --color -ltFh data/bcl2fastq/cellranger-tiny-bcl-1.2.0
```

Question

Alternatively, you can use the `tree` command (if available in your system!) to list the content of the `cellranger-tiny-bcl-1.2.0` directory:

Answer

```
tree -L 4 data/bcl2fastq/cellranger-tiny-bcl-1.2.0/
```

2.3.2 Running `cellranger mkfastq`

Question

Do we have all the required files to run the `cellranger mkfastq` workflow? What about a samplesheet?

Answer

Normally, when sequencing a library, a samplesheet is provided to the Illumina sequencing machine. In our case, we don't have direct access to this sample sheet. Regardless, we can create one manually. Here are the info for the different samples which were sequenced in this toy dataset

```
echo "Lane,Sample,Index
1,test_sample1,SI-GA-E3
1,test_sample2,SI-GA-F3
1,test_sample3,SI-GA-G3
1,test_sample4,SI-GA-H3
" > data/bcl2fastq/cellranger-tiny-bcl-samplesheet.csv
```

Question

What does each column corresponds to? How is this going to be used when generating fastq files?

Question

Now that we have a samplesheet ready, let's launch the `cellranger mkfastq` workflow.

Answer

```
cd ${HOME}/scRNAseq_Jan24/data/bcl2fastq/
cellranger mkfastq \
    --id=tiny-bcl \
    --run=cellranger-tiny-bcl-1.2.0/ \
    --csv=cellranger-tiny-bcl-samplesheet.csv
cd ${HOME}/scRNAseq_Jan24/
```

Watch out the memory usage! For `mkfastq` command with human genome, at least 32 Gb of RAM are required!

Question

What are the different files generated by this workflow?

Answer

Once the conversion is achieved, the output folders can be viewed by running the `ls` command:

```
ls --color -ltFv data/bcl2fastq/tiny-bcl/
ls --color -ltFv data/bcl2fastq/tiny-bcl/outs/fastq_path/H35KCBCXY/test_sample1/
### Or ...
tree -L 3 data/bcl2fastq/tiny-bcl/
```

Question

- How many fastq files have been generated? What does each one correspond to?
- Look at the index read (I1), read 1 (R1), and read (R2) files using the command `zcat <FASTQ_FILE_NAME>.gz | head`. What does each file contain?
- Open the html file `tiny-bcl/outs/fastq_path/Reports/html/index.html`. Take some time to explore the demultiplexed outputs.

2.4 3. Generate gene count matrices with cellranger count

Familiarize yourself with the `cellranger count` documentation available here: cellranger count algorithm overview. Notably, read the section on **Alignment** (`Read Trimming`, `Genome Alignment`, `MAPQ adjustment`, `Transcriptome Alignment`, `UMI Counting`).

Question

Which files are required for this step? Do we have all we need? Where is the index genome located?

2.4.1 Download genome index for the toy dataset

`mm10` pre-processed cellranger-formatted genome reference index is available here.

Question

Download it in a subdirectory named `${HOME}/scRNAseq_Jan24/`

Answer

```
curl https://cf.10xgenomics.com/supp/cell-exp/refdata-gex-mm10-2020-A.tar.gz -o refdat
tar -xzvf refdata-gex-mm10-2020-A.tar.gz && mv refdata-gex-mm10-2020-A/ data/bcl2fastq
```

```
ls --color -ltFh data/bcl2fastq/refdata-gex-mm10-2020-A/*
```

2.4.2 Running `cellranger count`

Question

In the terminal, run the `count` command.

Answer

```
cd ${HOME}/scRNAseq_Jan24/data/bcl2fastq/
cellranger count \
    --id=counts \
    --transcriptome=refdata-gex-mm10-2020-A \
    --fastqs=tiny-bcl/outs/fastq_path/ \
    --sample=test_sample1
```

While the count command is running, read about the format of the feature-barcode matrices.

2.4.3 Checking `count` output files

Once the count command is finished running, the pipeline outputs can be viewed as follows:

```
ls --color -ltFh counts/
ls --color -ltFh counts/outs/
### Or ...
tree -L 4 counts/
```

Question

- Can you locate the feature-barcode matrices? What is the difference between the `raw_feature_bc_matrix` and `filtered_feature_bc_matrix` data types? In term of storage size?
- Open the html file `counts/outs/web_summary.html`. Take some time to explore the gene expression matrix outputs.
- How many clusters seem to be found? What are the main markers associated with each cluster?
- Can you speculate what the main difference(s) is between the clusters?
- Do the different metrics suggest that this sample contains good-quality data?

2.5 3 [Alternative] Generate gene count matrices with STARsolo

```
# Install STAR
conda install -c bioconda star

# Build STAR index
curl https://cf.10xgenomics.com/supp/cell-exp/refdata-gex-mm10-2020-A.tar.gz -o refdata-ge
tar -xzvf refdata-gex-mm10-2020-A.tar.gz && mv refdata-gex-mm10-2020-A/ data/bcl2fastq/
STAR --runMode genomeGenerate --runThreadN 16 --genomeDir data/bcl2fastq/ --genomeFastaFil
STAR_GENOME_DIR=data/bcl2fastq/refdata-gex-mm10-2020-A/star/

# Get barcode whitelist
curl https://raw.githubusercontent.com/10XGenomics/cellranger/master/lib/python/cellranger
BC_WHITELIST_FILE=data/bcl2fastq/737K-august-2016.txt

# Run STAR
STAR \
    --genomeDir "${STAR_GENOME_DIR}" \
    --soloType CB_UMI_Simple \
    --soloCBwhitelist "${BC_WHITELIST_FILE}" \
    --readFilesIn data/bcl2fastq/tiny-bcl/outs/fastq_path/Undetermined_S0_L001_R2_001.fast
```

2.6 4. Obtain single-cell RNA-seq datasets

"This is a course about single-cell RNA-seq analysis, right, so where is my data?"

Ok, **"your"** data is (most likely) yet to be sequenced! Or maybe you're interested in digging already existing databases! I mean, who isn't interested in this mind-blowing achievement from 10X Genomics??

Human Cell Atlas is probably a good place to start digging, if you are interested in mammal-related studies. For instance, let's say I am interested in epididymis differentiation. Boom: here is an entry from the HCA focusing on epididymis: link to HCA data portal.

2.6.1 Raw fastq reads from GEO

Here is the link to the actual paper studying epididymis:
An atlas of human proximal epididymis reveals cell-specific functions and distinct roles for CFTR.

> Question
>
> Find and check out the corresponding `GEO` entries for this study. What type of sequencing data is available?

Here is the link to the `GEO` page: link.

> Question
>
> Can you find links to download the raw data from this paper?

There are several ways to find this information, e.g. `ffq` command line tool, or using the web-based `sra-explorer` page (here). You generally will need the `GEO` corresponding ID or SRA project ID (e.g. `SRPxxxxxx`...).

> Question
>
> Try to install and use `ffq` tool from the Patcher lab.
>
> > Answer
> >
> > ```
> > conda install -c bioconda ffq
> > ffq --help
> > ffq -t GSE GSE148963
> > ```

Question

Can you find the links to raw data associated with the `GSE148963` GEO ID?

Answer

You should use a `grep` command: `grep` returns the lines which match a given pattern (e.g. a link…)!

```
ffq -t GSE GSE148963 | grep 'ftp://'
```

And with a bit of `sed` magick…

```
ffq -t GSE GSE148963 | grep 'ftp://' | sed 's,.*ftp:,ftp:,' | sed 's,".*,,' > GSE14896
# wget -i GSE148963_fastqlist.txt ## Do not run, it would take too long...
```

2.6.2 [BONUS] Pre-processed count matrices

Many times, researchers will provide a filtered count matrix when they publish scRNAseq experiments (along with mandatory raw `fastq` data, of course). It's way lighter than `fastq` reads, and you can go ahead with downstream analyses a lot quicker. So how do you get these matrices?

- Human Cell Atlas Consortium provides many processed datasets. For instance, in our case, the `Leir et al` study is available at the following link: https://data.humancellatlas.org/explore/projects/842605c7-375a-47c5-9e2c-a71c2c00fcad.
- GEO also hosts processed files.

Question

- Find GEO-hosted processed files for the `Leir et al` study.

You can download some of the processed files available in GEO from the following web-page. Scrolling down to the bottom of the page, there is a box labelled "Supplementary data". By clicking on "(custom)", a list of extra supplementary files will appear.

- Download and check the content of the count matrix, the genes and the barcodes files.
- What type of information does each file contain? How is it formatted? is it easily imported in R?

- How many cells were sequenced? How many genes were counted?
- Is it easy to interpret the count matrix? Why is it in such format?
- Comment on the file sizes between processed count matrix files and raw reads.

Part II

Day 2

3 Demonstration: leveraging R/Bioconductor for single-cell analyses

Goals:

- Refreshing your knowledge on R
- Introducing the `SingleCellExperiment` object and exploratory data analysis

3.1 1. Installing packages in `R`

"Hey, I've heard so many good things about this piece of software, it's called 'Seurat'? Heard of it? I wanna try it out soooo badly!"

In other words: "how do I install this or that brand new cutting-edge fancy `package`?"

`R` works with `packages`, available from different sources:

- `CRAN`, the `R` developer team and official package provider: CRAN (which can probably win the title of "the worst webpage ever designed in 1982").
- `Bioconductor`, another package provider, with a primary focus on genomic-related packages: Bioconductor.
- Other unofficial sources, such as `GitHub`.

> Question
>
> Install `mgcv`, `HCAData` and `revelio` packages

Answer

Each of these three packages is available from a different source.

```
install.packages('mgcv')
BiocManager::install('HCAData')
remotes::install_github('danielschw188/revelio')
```

Package help pages are available at different places, depending on their source. That being said, there is a place I like to go to easily find information related to most packages:

https://rdrr.io/

Question

For instance, check out `Revelio` package help pages.

- What is this package designed for?
- What are its main functions? What type of input does it require?

3.2 2. Basic `R` and `Bioconductor` classes

While `CRAN` is a repository of general-purpose packages, `Bioconductor` is the greatest source of analytical tools, data and workflows dedicated to genomic projects in `R`. Read more about `Bioconductor` to fully understand how it builds up on top of `R` general features, especially with the specific classes it introduces.

The two main concepts behind `Bioconductor`'s success are the **non-redundant** classes of objects it provides and their **inter-operability**. Huber et al., Nat. Methods 2015 summarizes it well.

3.2.1 Important R concepts:

3.2.2 `tibble` tables:

`tibbles` are built on the fundamental `data.frame` objects. They follow "tidy" concepts, all gathered in a common `tidyverse`. This set of key concepts help general data investigation and data visualization through a set of associated packages such as `ggplot2`.

```r
library(tidyverse)
dat <- tibble(
    x = 1:5,
    y = 1,
    z = x ^ 2 + y,
    class = c('a', 'a', 'b', 'b', 'c')
)
dat
```

3.2.3 Reading text files into `tibbles`

`tibbles` can be created from text files (or Excel files) using the `readr` package (part of `tidyverse`)

```r
dir.create('data/R_101/')
download.file('https://ftp.ncbi.nlm.nih.gov/geo/samples/GSM4486nnn/GSM4486714/suppl/GSM448
genes <- read_tsv('data/R_101/GSM4486714_AXH009_genes.tsv.gz', col_names = c('ID', 'Symbol
genes
```

3.2.4 Handling of `tibbles`:

`tibbles` can be readily "sliced" (i.e. selecting rows by number/name), "filtered" (i.e. selecting rows by condition) and columns can be "selected". All these operations are performed using verbs (most of them provided by the `dplyr` package, part of `tidyverse`).

```r
# `slice` extract certain *rows* by integer location
slice(genes, 1:4)
slice_head(genes)
slice_sample(genes, n = 10)
# `filter` subsets the tibble, retaining all rows that satisfy your condition(s)
filter(genes, Symbol == 'CCDC67')
filter(genes, grepl('^CCDC.*', Symbol))
filter(genes, grepl('^CCDC.*', Symbol), grepl('.*5$', Symbol))
# `select` extract `columns` by integer location, name, or pattern...
select(genes, 1)
select(genes, ID)
select(genes, matches('Sym.*'))
```

Columns can also be quickly added/modified using the `mutate` verb.

```r
# `mutate` adds a new column
mutate(genes, chr = sample(1:22, n(), replace = TRUE))
```

3.2.5 %>% pipe:

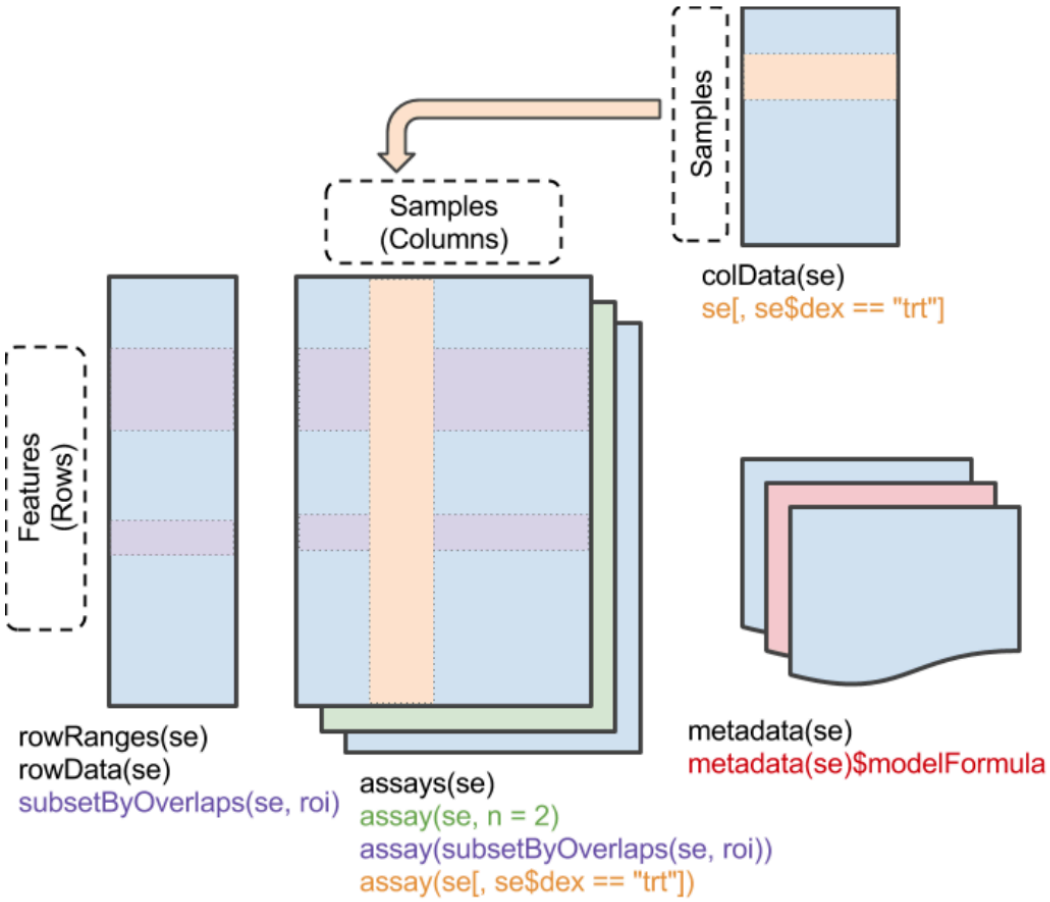
Actions on `tibbles` can be piped as a chain, just like `|` pipes `stdout` as the `stdin` of the next command in `bash`. In this case, the first argument is always the output of the previous function and is ommited. Because `tidyverse` functions generally return a modified version of the input, pipping works remarkably well in such context.

```r
genes %>%
    mutate(chr = sample(1:22, n(), replace = TRUE)) %>%
    filter(chr == 2, grepl('^CCDC.*', Symbol)) %>%
    select(ID) %>%
    slice_head(n = 3)
```

3.2.6 Important Bioconductor concepts:

3.2.7 `SummarizedExperiment` class:

The most fundamental class used to hold the content of large-scale quantitative analyses, such as counts of RNA-seq experiments, or high-throughput cytometry experiments or proteomics experiments.



Make sure you understand the structure of objects from this class. A dedicated workshop that I would recommend quickly going over is available here. Generally speaking, a `SummarizedExperiment` object contains matrix-like objects (the `assays`), with rows representing features (e.g. genes, transcripts, …) and each column representing a sample. Information specific to genes and samples are stored in "parallel" data frames, for example to store gene locations, tissue of expression, biotypes (for genes) or batch, generation date, or machine ID (for samples). On top of that, metadata are also stored in the object (to store description of a project, …).

An important difference with S3 `list`-like objects usually used in `R` is most of the underlying data (organized in `"slots"`) is accessed using `getter` functions, rather than the familiar `$` or `[`. Here are some important `getters`:

```
- `assay()`, `assays()`: Extrant matrix-like or list of matrix-like objects of identical dime
- colData(): Annotations on each column (as a DataFrame): usually, description of each sample
- rowData(): Annotations on each row (as a DataFrame): usually, description of each gene
```

- metadata(): List of unstructured metadata describing the overall content of the object.

Let's dig into an example (you may need to install the airway package from Bioconductor…)

```
library(SummarizedExperiment)
#BiocManager::install('airway')
library(airway)
data(airway)
airway
```

Question

What are the dimensions of the dataset? What type of quantitative data is stored? Which features are assessed?

Answer

```
dim(airway)
rowData(airway)
colData(airway)
```

Question

Can you create a subset of the data corresponding to LRG genes in untreated samples?

Answer

```
untreated_LRG <- airway[grepl('^LRG_', rownames(airway)), airway$dex == 'untrt']
untreated_LRG
# Using tidyverse-like expression leveraging `tidySummarizedExperiment` pacakge:
library(tidySummarizedExperiment)
airway
filter(airway, dex == 'untrt', grepl('^LRG_',  feature))
```

3.2.8 GenomicRanges class (a.k.a. GRanges):

GenomicRanges are a type of IntervalRanges, they are useful to describe genomic intervals. Each entry in a GRanges object has a seqnames(), a start() and an end() coordinates, a

`strand()`, as well as associated metadata (`mcols()`). They can be built from scratch using `tibbles` converted with `makeGRangesFromDataFrame()`.

```
library(GenomicRanges)
gr <- genes %>%
    mutate(
        chr = sample(1:22, n(), replace = TRUE),
        start = sample(1:1000, n(), replace = TRUE),
        end = sample(10000:20000, n(), replace = TRUE),
        strand = sample(c('-', '+'), n(), replace = TRUE)
    ) %>%
    makeGRangesFromDataFrame(keep.extra.columns = TRUE)
gr
mcols(gr)
```

Just like `tidyverse` in R, `tidy` functions are provided for `GRanges` by the `plyranges` package.

```
library(plyranges)
gr %>%
    filter(start < 400, end > 12000, end < 15000) %>%
    seqnames() %>%
    table()
```

3.3 3. CRAN & Bioconductor approaches to scRNAseq

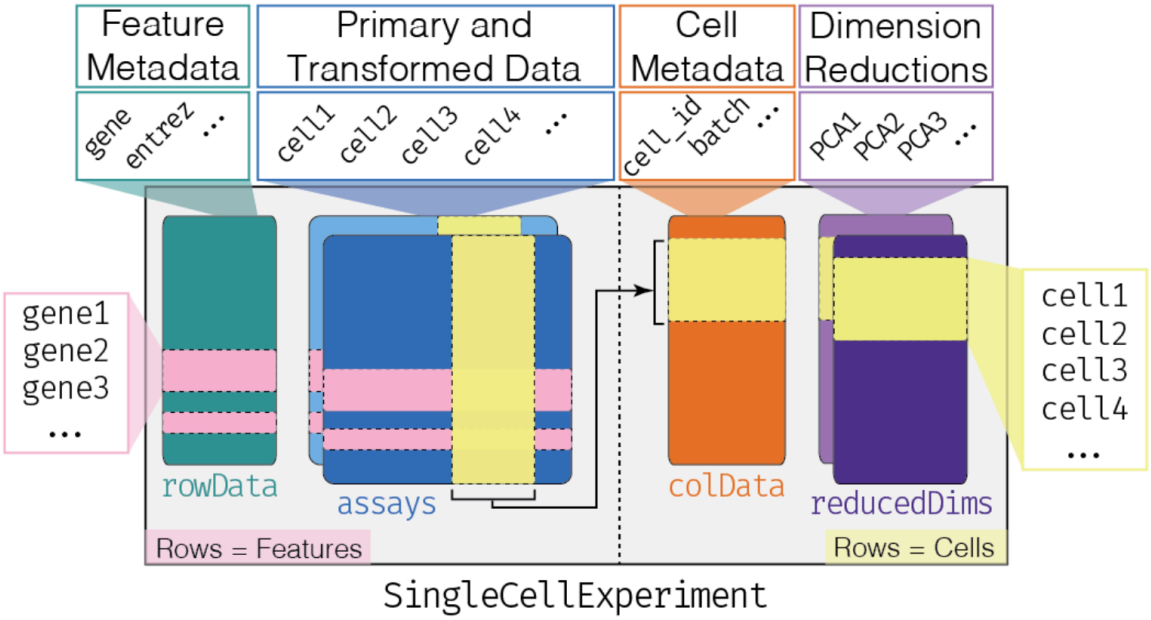
3.3.1 scRNAseq in Bioconductor

For single-cell RNA-seq projects, `Bioconductor` has been introducting new classes and standards very rapidly in the past few years. Notably, several packages are increasingly becoming central for single-cell analysis:

- **SingleCellExperiment**
- scater
- scran
- scuttle
- batchelor
- SingleR
- bluster
- DropletUtils
- slingshot

- tradeSeq
- ...

SingleCellExperiment is the fundamental class designed to contain single-cell (RNA-seq) data in `Bioconductor` ecosystem. It is a modified version of the `SummarizedExperiment` object, so most of the getters/setters are shared with this class.



Let's load a fully-fleged `SingleCellExperiment` object, so we can play around with it:

```
library(SingleCellExperiment)
library(tidySingleCellExperiment)
sce <- readRDS(url('https://github.com/js2264/scRNAseq-workshop/raw/main/pbmc3k.rds'), "rb
class(sce)
sce
```

Several slots can be accessed in a **SingleCellExperiment** object, just like the `SummarizedExperiment` object it's been adapted from:

```
colData(sce)
rowData(sce)
metadata(sce)
dim(sce)
assays(sce)
```

Important slots for scRNAseq studies can also be accessed:

```
counts(sce)[1:10, 1:10]
logcounts(sce)[1:10, 1:10]
```

Question

Check the `colData()` output of the `sce` object. What information is stored there? How can you access the different objects stored in `colData`?

Answer

```
colData(sce)
head(colData(sce)[[1]])
head(colData(sce)[['sizeFactor']])
head(colData(sce)$sizeFactor)
head(sce$sizeFactor)
```

Question

Are there any reduced dimensionality representation of the data stored in the `sce` object? How many dimensions does each embedding contain?

Answer

```
reducedDims(sce)
head(reducedDim(sce, 'PCA'))
head(reducedDim(sce, 'TSNE'))
head(reducedDim(sce, 'UMAP'))
```

Question

Now, plot the `UMAP` embedding and color cells by their cluster.

Answer

```
p <- scater::plotReducedDim(sce, 'UMAP', colour_by = 'cluster', text_by = 'cluster')
```

4 Exercises: scRNAseq analysis with R/Bioconductor (1/3)

Goals:

- Import single-cell experiments in R
- Identify empty droplets or doublets and filter them out
- Remove non-relevant genes from a `SingleCellExperiment` object
- Perform log-based normalization and variance stabilisation transformation (`vst`) and compare both normalization approaches

---

4.1 1. Import single-cell RNA-seq data in R

4.1.1 Import data from cellranger workflow

Importing 10X Genomics scRNAseq data in R can be done using `DropletUtils` package.

Question

- Read the documentation for DropletUtils utilities useful for 10X Genomics data import here.
- Download 1K mouse E18 heart scRNAseq filtered count matrix from 10X Genomics (in `HDF5` format available here).
- Import it into `R` using `read10xCounts()` function.

Answer

```
dir.create('data/E18_Heart/cellranger/E18_Heart/')
download.file(
    url = 'https://cf.10xgenomics.com/samples/cell-exp/3.0.0/heart_1k_v3/heart_1k_v3_f
    destfile = 'data/E18_Heart/cellranger/E18_Heart/heart_1k_v3_filtered_feature_bc_ma
    mode = 'wb'
)
heart <- DropletUtils::read10xCounts('data/E18_Heart/cellranger/E18_Heart/heart_1k_v3_
```

Question

- How many cells were sequenced in this dataset? How many genes are profiled?
- What is the distribution of genes being detected per cell, and of number of unique transcripts being detected per cell? And for each gene, what is the distribution of number of cells it is detected in? Use QC functions from the `scuttle` package to automatically compute these metrics.
- What is the sparsity of the data (in other words, how dense is the count matrix)?

Answer

```
# Experiment size
dim(heart)

# Genes / transcripts detected per cell
heart <- scuttle::addPerCellQCMetrics(heart)
heart <- scuttle::addPerFeatureQCMetrics(heart)
quantile(heart$sum, seq(0, 1, 0.1))
quantile(heart$detected, seq(0, 1, 0.1))
quantile(rowData(heart)$detected, seq(0, 1, 0.1))

# Count matrix density
sum(counts(heart) > 0) / {dim(heart)[1] * dim(heart)[2]}
```

4.1.2 Use "pre-compiled" datasets

The `scRNAseq` package (from Bioconductor) allows one to import public datasets directly in R.

Question

- Read the documentation vignette here.
- Import scRNAseq data from `Zeisel et al., Science 2015` (doi: 10.1126/science.aaa1934) in R using the `scRNAseq` package.

Answer

```
zeisel <- scRNAseq::ZeiselBrainData()
```

Question

- How many cells were sequenced in this dataset? How many genes are profiled?
- What is the distribution of genes being detected per cell, and of number of unique transcripts being detected per cell?
- What is the sparsity of the data (in other words, how dense is the count matrix)?
- Compare with 10X Genomics-provided data. Comment on the differences. What was the single-cell sequencing technique used in Zeisel et al.?

Answer

```
# Experiment size
dim(zeisel)

# Genes / transcripts detected per cell
zeisel <- scuttle::addPerCellQCMetrics(zeisel)
zeisel <- scuttle::addPerFeatureQCMetrics(zeisel)
quantile(zeisel$sum, seq(0, 1, 0.1))
quantile(zeisel$detected, seq(0, 1, 0.1))
quantile(rowData(zeisel)$detected, seq(0, 1, 0.1))

# Count matrix density
sum(counts(zeisel) > 0) / {dim(zeisel)[1] * dim(zeisel)[2]}
```

Question

- What are the different annotations available for the cells?
- What are the tissues used for cell profiling? Which type of cells come from which tissue?

Answer

```
# Check cell type annotations
colData(zeisel)
table(zeisel$level1class)
table(zeisel$level2class)
table(zeisel$level2class, zeisel$level1class)

# Check tissue of origin
table(zeisel$tissue)
table(zeisel$level1class, zeisel$tissue)
```

4.1.3 UMI number / cell

A useful diagnostic for scRNAseq data is the barcode rank plot, which shows the (log-)total UMI count for each barcode on the y-axis and the (log-)rank on the x-axis. This is effectively a transposed empirical cumulative density plot with log-transformed axes. It is useful as it allows users to examine the distribution of total counts across barcodes, focusing on those with the largest counts.

This diagnostic plot can be generated using `DropletUtils` package, notably the `barcodeRanks()` function.

Question

- Try to use `barcodeRanks()` function to compute and plot UMI # / cell across all the cells in either 10X Genomics-generated data or Zeisel's data.
- Comment the difference. Again, what are the important differences in term of single-cell approaches used here?

Answer

```
library(tidyverse)
heart_barcoderanks <- DropletUtils::barcodeRanks(heart)
zeisel_barcoderanks <- DropletUtils::barcodeRanks(zeisel)
p <- list(
    heart = as_tibble(heart_barcoderanks),
    zeisel = as_tibble(zeisel_barcoderanks)
) %>%
    bind_rows(.id = 'dataset') %>%
    ggplot(aes(x = rank, y = total, group = dataset, col = dataset)) +
    geom_line() +
    scale_y_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    theme_bw() +
    labs(x = 'Cells ranked by total UMI', y = 'Total UMI count / cell')
```

4.2 2. Filter emtpy and doublet droplets

An important step when getting your hands on droplet-based single-cell RNA-seq data is to be confident you are working with actual cell data. This means knowing how to deal with/remove (1) emtpy droplets and (2) droplets containing doublets.

4.2.1 Removing empty droplets

The ambient RNA "soup" sometimes makes it difficult to differentiate empty droplets from droplets containing cells with low amounts of RNA.

`emptyDrops()` function from the `DropletUtils` package provides a methodology to (1) estimate the ambient RNA contamination and then (2) compute a probability that each droplet contains a cell.

Since `emptyDrops()` assumes that most of the droplets in a matrix are empty, one needs to start the analysis from the **raw unfiltered** count matrix provided by 10X Genomics. Download the E18 mouse heart scRNAseq raw unfiltered count matrix from 10X Genomics here.

```
download.file(
    url = 'https://cf.10xgenomics.com/samples/cell-exp/3.0.0/heart_1k_v3/heart_1k_v3_raw_f
    destfile = 'data/E18_Heart/cellranger/E18_Heart/heart_1k_v3_raw_feature_bc_matrix.h5',
    mode = 'wb'
)
heart_raw <- DropletUtils::read10xCounts('data/E18_Heart/cellranger/E18_Heart/heart_1k_v3_
heart_raw
dim(heart_raw)
```

Question

Use `emptyDrops()` to differentiate empty droplets from cell-containing droplets.
Be aware, the empty droplet detection step is quite lenghty! After all, you are scanning several millions of droplets, most of them empty! To fasten the process, you can specify a number of cpus to use in parallel, with the `BiocParallel::MulticoreParam()` function.

Answer

```
# emptyDrops performs Monte Carlo simulations to compute p-values, so we need to set t
library(DropletUtils)
set.seed(100)
# Do not run if not on an HPC cluster:
# heart_droplets <- emptyDrops(counts(heart_raw), BPPARAM = BiocParallel::MulticorePar
heart_droplets
table(heart_droplets$FDR <= 0.001)
heart_filtered <- heart_raw[, which(heart_droplets$FDR <= 0.001)]
```

Even with multiple cpus, the computation can take up to several hours. So if you wish to skip this step for now, you can use the `cellranger`-automatically filtered matrix for now. It is not exactly equivalent, but does a fairly good job at finding non-empty droplets and I would recommend sticking to it for the beginning.

```
heart_filtered <- heart
```

4.2.2 Flagging cell doublets

Another artifact emerging from non-perfect experimental steps is the sequencing of two cells contained within a single droplet. This can occur when many cells are sequenced on a single 10X Genomics cassette (doublet increase of 1% per 1,000 cells sequenced). This can also occur when cells are not in a perfect single cell suspension.

A way to identify cell doublets is to artifically mix thousands of pairs of cells (columns) of a `SingleCellExperiment` object, then compare the resulting cells to each cell in the original dataset. Original cells which resemble a lot the artificial doublets are likely doublet themselves.

Question

- Read `scDblFinder` documentation here.
- Use `scDblFinder()` function to flag probable cell doublets in manually filtered heart dataset.

Answer

```
#BiocManager::install('scDblFinder')
library(scDblFinder)
heart_filtered <- scDblFinder(heart_filtered)
colData(heart_filtered)
table(heart_filtered$scDblFinder.class)
```

For now, we can keep these doublets. We will see in the future whether we remove them or not.

4.3 3. (Optional) Exclude non-relevant genes from analysis

The 10X Genomics-provided count matrix contains `31053` annotated genes. However, there are likely less than 20,000 of them which are genomic, protein-coding, expressed genes. We can filter genes based on the location, biotype and overall detection in the dataset.

Question

- Recover gene annotations as `gtf` from ensembl uing the `AnnotationHub`
- Filter to only get protein-coding, ENSEMBL+HAVANA-annotated genomic genes

Answer

```
library(plyranges)
ah <- AnnotationHub::AnnotationHub()
AnnotationHub::query(ah, c('gene annotation', 'ensembl', '102', 'mus_musculus', 'GRCm3
gtf <- AnnotationHub::query(ah, c('Mus_musculus.GRCm38.102.chr.gtf'))[[1]]
genes <- gtf %>%
    filter(type == 'gene') %>%
    filter(gene_biotype == 'protein_coding') %>%
    filter(gene_source == 'ensembl_havana')
```

Question

Filter genes from the SingleCellExperiment dataset to only protein-coding, ENSEMBL+HAVANA-annotated genomic genes

Answer

```
names(genes) <- genes$gene_id
table(rownames(heart_filtered) %in% names(genes))
heart_filtered <- heart_filtered[rownames(heart_filtered) %in% names(genes), ]
gr <- genes[rownames(heart_filtered)]
mcols(gr) <- cbind(mcols(gr), rowData(heart_filtered))[, c('gene_id', 'gene_name', 'me
rowRanges(heart_filtered) <- gr
```

Question

Filter remaining genes to only keep those detected in at least 10 cells

Answer

```
quantile(rowSums(counts(heart_filtered) > 0), seq(0, 1, 0.1))
table( rowSums(counts(heart_filtered) > 0) >= 10 )
heart_filtered <- heart_filtered[rowSums(counts(heart_filtered) > 0) >= 10, ]
```

4.4 4. Normalize data

Normalization can be done two ways:

- A crude sequencing depth normalization followed by log-transformation. This is usually referred to as "log normalizing".
- A more advanced (and probably more accurate) approach is the variance stabilizing transformation. This aims at removing the relationship between levels at which a gene is expressed and the variance of its expression.

4.4.1 Log-normalization

Just like in bulk high-throughput sequencing experiments, scRNAseq counts have to be normalized to the sequencing depth for each cell. We can define the library size (a.k.a. `size factor`)as the total sum of counts across all genes for each cell. However, this relies on the assumption that within the entire dataset, most genes are non-differentially expressed and expressed roughly within the same range. Depending on the set up of the scRNAseq experiment, this can be entirely false. To avoid relying on this hypothesis, we can (1) quickly pre-cluster cells, then (2) normalize cells using their library size factor separately in each cluster, then (3) rescaling size factors so that they are comparable across clusters.

Question

Read documentation for `scran` functions `quickCluster()` and `computeSumFactors()`. Compute size factors for each cell in the manually filtered E18 mouse heart scRNAseq dataset

Answer

```
clusters <- scran::quickCluster(heart_filtered)
table(clusters)
heart_filtered <- scran::computeSumFactors(heart_filtered, cluster = clusters)
colData(heart_filtered)
head(heart_filtered$sizeFactor)
quantile(heart_filtered$sizeFactor, seq(0, 1, 0.1))
```

Question

Compare the size factor to the total count of UMI / cell. Comment.

Answer

```
heart_filtered <- scuttle::addPerCellQCMetrics(heart_filtered)
heart_filtered <- scuttle::addPerFeatureQCMetrics(heart_filtered)
p <- tibble(
    cell = heart_filtered$Barcode,
    totUMIs = heart_filtered$total,
    sizeFactor = heart_filtered$sizeFactor
) %>%
    ggplot(aes(x = totUMIs, y = sizeFactor)) +
    geom_point() +
    scale_y_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    theme_bw() +
    labs(x = 'Total UMI', y = 'Size factors')
```

Question

Using the computed size factors, perform log-normalization of the data. Read `scuttle::logNormCounts()` documentation if needed.

Answer

```
heart_filtered <- scuttle::logNormCounts(heart_filtered)
assays(sce)
logcounts(heart_filtered)[1:10, 1:10]
p <- tibble(
    count = c(counts(heart_filtered)),
    logcount = c(logcounts(heart_filtered))
) %>%
    filter(count > 0) %>%
    ggplot(aes(x = count, y = logcount)) +
    ggrastr::geom_point_rast() +
    scale_y_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    theme_bw() +
    labs(x = 'Raw counts', y = 'log-normalized counts')
```

4.4.2 [BONUS] VST normalization

Question

- Quickly read the extensive vignette about scRNAseq normalization using variance stabilizing transformation here.
- First, check the relationship between (1) mean gene expression and gene expression variance and (2) mean gene expression and gene detection rate, in the manually filtered E18 mouse heart scRNAseq count matrix.

Answer

```
cnts <- as(SingleCellExperiment::counts(heart_filtered), 'dgCMatrix')
colnames(cnts) <- heart_filtered$Barcode
rownames(cnts) <- rownames(heart_filtered)
#
df <- tibble(
    gene = rownames(cnts),
    detection_rate = rowMeans(cnts > 0),
    mean = rowMeans(cnts),
    variance = apply(cnts, 1, var)
)
p1 <- ggplot(df, aes(x = mean, y = variance)) +
    geom_point(alpha = 0.3) +
    geom_density_2d(size = 0.3) +
    geom_abline(intercept = 0, slope = 1, color = "red") +
    scale_y_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    labs(x = 'Gene expression mean', y = 'Gene expression variance') +
    theme_bw()
p2 <- ggplot(df, aes(x = mean, y = detection_rate)) +
    geom_point(alpha = 0.3) +
    geom_density_2d(size = 0.3) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    labs(x = 'Gene expression mean', y = 'Gene detection rate') +
    theme_bw()
p <- cowplot::plot_grid(p1, p2, nrow = 1)
```

Question

Apply `sctransform::vst()` function on raw counts from manually filtered E18 mouse heart scRNAseq count matrix.

Answer

```
heart_vst <- sctransform::vst(cnts, return_cell_attr = TRUE)
```

Question

- Using variance-stabilized residuals, correct the raw counts in the heart scRNAseq count matrix. You will need `sctransform::correct()` function to do this
- Store the corrected counts in an assay named `corrected_counts`
- Log-transform the `corrected_counts` using `log1p()` function and store the transformed counts in an assay named `logcounts_vst`

Answer

```
corrected_cnts <- sctransform::correct(heart_vst)
heart_filtered <- heart_filtered[rownames(corrected_cnts),]
assay(heart_filtered, 'corrected_counts', withDimnames = FALSE) <- corrected_cnts
assay(heart_filtered, 'logcounts_vst', withDimnames = FALSE) <- log1p(corrected_cnts)
```

Question

- Once this is done, check again the relationship between mean gene expression and gene expression variance.
- Check how the count variance now varies with increasing mean gene counts. Comment.
- Check how the detection rate now varies with increasing mean gene counts. Comment.

Answer

```
df <- rbind(
    tibble(
        gene = rownames(cnts),
        detection_rate = rowMeans(cnts > 0),
        mean = rowMeans(cnts),
        variance = apply(cnts, 1, var),
        normalization = 'raw'
    ),
    tibble(
        gene = rownames(corrected_cnts),
        detection_rate = rowMeans(corrected_cnts > 0),
        mean = rowMeans(corrected_cnts),
        variance = apply(corrected_cnts, 1, var),
        normalization = 'vst_corrected'
    )
)
p1 <- ggplot(df, aes(x = mean, y = variance)) +
    geom_point(alpha = 0.3) +
    geom_density_2d(size = 0.3) +
    geom_abline(intercept = 0, slope = 1, color = "red") +
    scale_y_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    labs(x = 'Gene expression mean', y = 'Gene expression variance') +
    theme_bw() +
    facet_grid(~normalization)
p2 <- ggplot(df, aes(x = mean, y = detection_rate)) +
    geom_point(alpha = 0.3) +
    geom_density_2d(size = 0.3) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    labs(x = 'Gene expression mean', y = 'Gene detection rate') +
    theme_bw() +
    facet_grid(~normalization)
p <- cowplot::plot_grid(p1, p2, nrow = 2)
```

Part III

Day 3

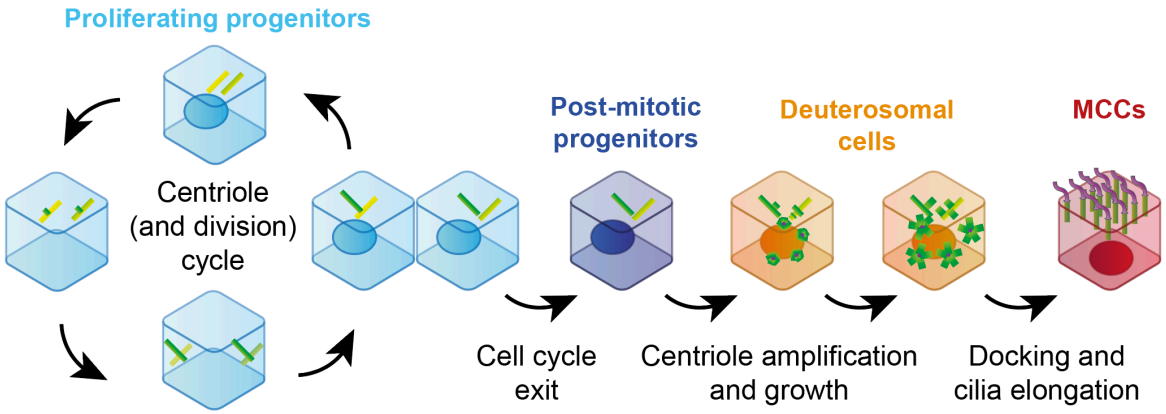
5 Demonstration: Dimensional reduction visualization and clustering

Goals:

- Perform different dimensionality reductions
- Try out different cell clustering approaches
- Visualize gene expression in each cluster

5.1 1. Dimensionality reduction

I am currently working on the differentiation of neural progenitor cells into multiciliated cells:



Let's load the data I have provided and inspect it:

```
library(SingleCellExperiment)
library(tidyverse)
#
MCCs <- readRDS('data/MCCs/MCCs.rds')
MCCs
dim(MCCs)
```

```
colData(MCCs)
rowData(MCCs)
assays(MCCs)
reducedDims(MCCs)
```

To perform PCA embedding, we need normalized data! Is this data normalized?

```
cnts <- as(assay(MCCs, 'counts'), 'dgCMatrix')
colnames(cnts) <- MCCs$Barcode
rownames(cnts) <- rownames(MCCs)
df <- tibble(
    gene = rownames(cnts),
    detection_rate = rowMeans(cnts > 0),
    mean = rowMeans(cnts),
    variance = apply(cnts, 1, var)
)
p <- ggplot(df, aes(x = mean, y = variance)) +
    geom_point(alpha = 0.3) +
    geom_density_2d(size = 0.3) +
    geom_abline(intercept = 0, slope = 1, color = "red") +
    scale_y_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    scale_x_log10(
        breaks = scales::trans_breaks("log10", function(x) 10^x),
        labels = scales::trans_format("log10", scales::math_format(10^.x))
    ) +
    labs(x = 'Gene expression mean', y = 'Gene expression variance') +
    theme_bw()
ggsave('data/MCCs/variance~mean.pdf')
```

The gene expression variance is more or less equals to the gene average expression, as it should be assuming counts follow a Poisson distribution. This suggests that the data is probably already normalized!

We can now flag HVGs and use them for dimensionality reduction.

```
assay(MCCs, 'logcounts') <- log1p(counts(MCCs))
MCCs_variance <- scran::modelGeneVar(MCCs)
MCCs_variance
quantile(MCCs_variance$bio, seq(0, 1, 0.1))
```

```
quantile(MCCs_variance$tech, seq(0, 1, 0.1))

HVGs <- scran::getTopHVGs(MCCs_variance, prop = 0.1)
rowData(MCCs)$isHVG <- rownames(MCCs) %in% HVGs
head(rowData(MCCs))
table(rowData(MCCs)$isHVG)

# Visualizing the mean-variance fit, coloring HVGs
df <- tibble(
    mean = metadata(MCCs_variance)$mean,
    var = metadata(MCCs_variance)$var,
    trend = metadata(MCCs_variance)$trend(mean),
    HVG = rowData(MCCs)$isHVG
)
p <- ggplot(df) +
    geom_point(aes(x = mean, y = var, col = HVG), alpha = 0.4) +
    geom_line(aes(x = mean, y = trend), col = 'darkred') +
    theme_minimal() +
    labs(x = 'Gene mean exp. (log1p)', y = 'Gene exp. variance')
ggsave('data/MCCs/variance~mean_HVGs.pdf')
```

Is the default gene variance fitting approach good? Can we try out another approach?

```
MCCs_CV <- scran::modelGeneCV2(MCCs)
MCCs_CV
quantile(MCCs_CV$bio, seq(0, 1, 0.1))
quantile(MCCs_CV$tech, seq(0, 1, 0.1))

HVGs_2 <- scran::getTopHVGs(MCCs_CV, prop = 0.1, var.field="ratio")
rowData(MCCs)$isHVG_2 <- rownames(MCCs) %in% HVGs_2
head(rowData(MCCs))
table(rowData(MCCs)$isHVG_2)

# Visualizing the mean-CV2 fit, coloring HVGs
df <- tibble(
    mean = metadata(MCCs_CV)$mean,
    CV2 = metadata(MCCs_CV)$cv2,
    trend = metadata(MCCs_CV)$trend(mean),
    HVG = rowData(MCCs)$isHVG_2
)
p <- ggplot(df) +
```

```
    geom_point(aes(x = mean, y = CV2, col = HVG), alpha = 0.4) +
    geom_line(aes(x = mean, y = trend), col = 'darkred') +
    scale_x_log10() + scale_y_log10() +
    theme_minimal() +
    labs(x = 'Gene mean exp.', y = 'Gene exp. CV2')
ggsave('data/MCCs/cv2~mean_HVGs.pdf')
```

Fitting gene expression CV2 ~ mean seems to be slightly more robust than gene expression variance ~ mean, here. We will use these HVGs for dimensionality reduction.

Now we can embed in PCA:

```
set.seed(1000)
MCCs <- scran::denoisePCA(
    MCCs,
    technical = MCCs_variance,
    subset.row = HVGs_2,
    min.rank = 15
)
dim(as.data.frame(reducedDim(MCCs)))
head(as.data.frame(reducedDim(MCCs)))
p <- cowplot::plot_grid(
    scater::plotReducedDim(MCCs, 'PCA', colour_by = 'detected'),
    scater::plotReducedDim(MCCs, 'PCA', colour_by = 'sum')
)
ggsave('data/MCCs/pca.pdf')
```

5.2 2. Cell clustering

We can cluster cells (embedded in PCA space) using different methods:

1. a hierarchical clustering approach
2. a k-means approach
3. a graph-based approach

```
set.seed(1000)
pca <- reducedDim(MCCs, 'PCA')
rownames(pca) <- MCCs$Barcode

# Hierarchical clustering
```

```
dists <- dist(pca)
hclusts <- hclust(dists)
MCCs$cluster_hclust <- factor(cutree(hclusts, k = 5))

# k-means clustering
kmeans <- kmeans(pca, centers = 5)
MCCs$cluster_kmeans <- factor(kmeans$cluster)

# Shared k-Nearest Neighbors graph clustering
graph <- bluster::makeSNNGraph(pca)
communities <- igraph::cluster_louvain(graph)
MCCs$cluster_SNN <- factor(communities$membership)
```

```
p <- cowplot::plot_grid(
    scater::plotReducedDim(MCCs, 'PCA', colour_by = 'cluster_hclust', text_by = 'cluster_h
    scater::plotReducedDim(MCCs, 'PCA', colour_by = 'cluster_kmeans', text_by = 'cluster_k
    scater::plotReducedDim(MCCs, 'PCA', colour_by = 'cluster_SNN', text_by = 'cluster_SNN'
)
ggsave('data/MCCs/clusters_PCA.pdf')
```

We can embed MCCs in UMAP space to better visualize the clusters

```
set.seed(1000)
MCCs <- scater::runUMAP(MCCs)
p <- cowplot::plot_grid(
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'cluster_hclust', text_by = 'cluster_
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'cluster_kmeans', text_by = 'cluster_
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'cluster_SNN', text_by = 'cluster_SNN
)
ggsave('data/MCCs/clusters_UMAP.pdf')
```

A new approach to check cell clustering is to leverage **bluster** package:

```
pca <- reducedDim(MCCs, 'PCA')
rownames(pca) <- MCCs$Barcode
evalClusters <- bluster::bootstrapStability(pca, clusters = MCCs$cluster_SNN, mode = "rati
pheatmap::pheatmap(evalClusters, breaks = seq(-1, 1, length=101), cluster_rows = FALSE, cl
```

5.3 3. Gene expression visualization

First we can order the clusters with a sensible order.

```
labels <- c(
    '1' = 'H',
    '2' = 'G',
    '3' = 'F',
    '4' = 'D',
    '5' = 'E',
    '6' = 'C',
    '7' = 'B',
    '8' = "B'",
    '9' = "B''",
    '10' = 'A'
)
MCCs$label <- labels[MCCs$cluster_SNN]
```

We can check expression of known markers of MCC progenitors, or differentiating progenitors, or terminally differentiated progenitors.

```
genes <- c(
    'Mki67',
    'Cdk1',
    'Ube2c',
    'Id3', 'Id4',
    'Ccno', 'Mcidas', 'Cdc20b',
    'Tmem212', 'Nnat'
)

# Plot clusters and gene expr. for a single gene
gene <- genes[[1]]
df <- tibble(
    UMAP1 = reducedDim(MCCs, 'UMAP')[, 1],
    UMAP2 = reducedDim(MCCs, 'UMAP')[, 2],
    annot = MCCs$label,
    expr = as.vector(logcounts(MCCs[gene,])),
)
p <- cowplot::plot_grid(
    ggplot(df, aes(x = UMAP1, y = UMAP2, col = annot)) +
        ggrastr::geom_point_rast() +
```

```
        ggtitle(gene) +
        theme_bw(),
    ggplot(df, aes(x = UMAP1, y = UMAP2, col = expr)) +
        ggrastr::geom_point_rast() +
        scale_color_distiller(palette = 'YlOrRd', direction = 1) +
        theme_bw(),
    ggplot(df, aes(x = annot, y = expr, fill = annot)) +
        geom_violin(scale = 'width') +
        theme_bw(),
    nrow = 1, align = 'vh', axis = 'trbl'
)
ggsave('data/MCCs/Mki67_UMAP.pdf', width = 10, height = 3)

# Multiple genes in a `lapply` function
p <- lapply(genes, function(gene) {
    message(gene)
    df <- tibble(
        UMAP1 = reducedDim(MCCs, 'UMAP')[, 1],
        UMAP2 = reducedDim(MCCs, 'UMAP')[, 2],
        annot = MCCs$label,
        expr = as.vector(logcounts(MCCs[gene,])),
    )
    cowplot::plot_grid(
        ggplot(df, aes(x = UMAP1, y = UMAP2, col = annot)) +
            ggrastr::geom_point_rast() +
            ggtitle(gene) +
            theme_bw(),
        ggplot(df, aes(x = UMAP1, y = UMAP2, col = expr)) +
            ggrastr::geom_point_rast() +
            scale_color_distiller(palette = 'YlOrRd', direction = 1) +
            theme_bw(),
        ggplot(df, aes(x = annot, y = expr, fill = annot)) +
            geom_violin(scale = 'width') +
            theme_bw(),
        nrow = 1, align = 'vh', axis = 'trbl'
    )
}) %>% cowplot::plot_grid(plotlist = ., ncol = 1)
ggsave('data/MCCs/gene-expr_UMAP.pdf', w = 15, h = 15)
```

6 Exercises: scRNAseq analysis with R/Bioconductor (2/3)

Goals:

- Pre-process a 4K PBMC scRNAseq dataset
- Select hyper-variable genes and perform dimensionality reduction
- Cluster cells into groups

6.1 1. Pre-processing PBMC dataset

We will prepare scRNAseq data from a PBMC run, provided by 10X and hosted by `Bioconductor` as a package.

6.1.1 Preparing dataset

Question

- Which package from `Bioconductor` gives streamlined access to PBMC scRNAseq dataset from 10X Genomics?
- Import the 4K PBMCs dataset provided by 10X Genomics directly in R.
- What does the object contain (type of data, number of cells, batches, organism, …)?

Answer

```
pbmc <- TENxPBMCData::TENxPBMCData('pbmc4k')
rownames(pbmc) <- scuttle::uniquifyFeatureNames(rowData(pbmc)$ENSEMBL_ID, rowData(pbmc
pbmc
rowData(pbmc)
colData(pbmc)
table(pbmc$Library)
```

6.1.2 Remove doublets and filter non-relevant genes

Question

Use `scDblFinder` to flag and remove cell doublets

Answer

```
pbmc <- scDblFinder::scDblFinder(pbmc)
table(pbmc$scDblFinder.class)
pbmc <- pbmc[, pbmc$scDblFinder.class == 'singlet']
```

You will then need to import gene annotations (from the right organism!) in R, to then filter out irrelevant genes.

Question

- Get gene loci from Ensembl using `AnnotationHub`
- Filter to only get protein-coding, ENSEMBL+HAVANA-annotated genomic genes
- Further remove genes that are not expressed in at least 10 cells

Answer

```
# Annotate genes in pbmc dataset
library(plyranges)
ah <- AnnotationHub::AnnotationHub()
AnnotationHub::query(ah, c('gene annotation', 'ensembl', '102', 'homo_sapiens', 'GRCh3
gtf <- AnnotationHub::query(ah, c('Homo_sapiens.GRCh38.102.chr.gtf'))[[1]]
genes <- gtf %>%
    filter(type == 'gene') %>%
    filter(gene_biotype == 'protein_coding') %>%
    filter(gene_source == 'ensembl_havana')

pbmc <- pbmc[genes$gene_id[genes$gene_id %in% rownames(pbmc)], ]
rowRanges(pbmc) <- genes[match(rownames(pbmc), genes$gene_id)]
rowData(pbmc) <- rowData(pbmc)[, c('gene_name', 'gene_id')]
rownames(pbmc) <- scuttle::uniquifyFeatureNames(rowData(pbmc)$gene_id, rowData(pbmc)$g

# Genes / transcripts detected per cell
pbmc <- scuttle::addPerCellQCMetrics(pbmc)
pbmc <- scuttle::addPerFeatureQCMetrics(pbmc)

# Remove genes not expressed in at least 10 cells
pbmc <- pbmc[rowSums(counts(pbmc) > 0) >= 10, ]
```

6.1.3 Normalize counts using `sctransform`

```
cnts <- as(SingleCellExperiment::counts(pbmc), 'dgCMatrix')
colnames(cnts) <- pbmc$Barcode
rownames(cnts) <- rownames(pbmc)
pbmc_vst <- sctransform::vst(cnts, return_cell_attr = TRUE)
corrected_cnts <- sctransform::correct(pbmc_vst)
assay(pbmc, 'corrected_counts', withDimnames = FALSE) <- corrected_cnts
assay(pbmc, 'logcounts', withDimnames = FALSE) <- log1p(corrected_cnts)
```

6.2 2. Dimensionality reduction

6.2.1 Selection of hyper-variable genes (HVGs)

Dimensionality reduction compare cells based on their gene expression profiles. The choice of genes to include in this comparison may have a major impact on the performance of downstream methods. Ideally, one wants to only select genes that contain useful information about the biology of the system while removing genes that contain random noise. This aims to preserve interesting biological structure without the variance that obscures that structure.

The simplest approach to feature selection is to simply compute the variance of the log-normalized expression values, to select the most variable genes. Modelling of the mean-variance relationship can be achieved by the `modelGeneVar()` function from the `scran` package.

Question

- Read more about `scran::modelGeneVar()` online
- Model gene variance ~ gene average expression. What is the range of biological variance and technical variance?

Answer

```
# Fit the gene variance as a function of the gene mean expression
pbmc_variance <- scran::modelGeneVar(pbmc)
pbmc_variance
quantile(pbmc_variance$bio, seq(0, 1, 0.1))
quantile(pbmc_variance$tech, seq(0, 1, 0.1))

# Visualizing the mean-variance fit
require(tidyverse)
df <- tibble(
    mean = metadata(pbmc_variance)$mean,
    var = metadata(pbmc_variance)$var,
    trend = metadata(pbmc_variance)$trend(mean),
)
p <- ggplot(df) +
    geom_point(aes(x = mean, y = var), alpha = 0.4) +
    geom_line(aes(x = mean, y = trend), col = 'darkred') +
    theme_minimal() +
    labs(x = 'Gene mean exp. (norm.)', y = 'Gene exp. variance')
```

Question

- Extract the 20% genes with the highest biological variance.
- Plot gene variance ~ gene average expression, coloring genes which are flagged as HVGs.

Answer

```
HVGs <- scran::getTopHVGs(pbmc_variance, prop = 0.1)
rowData(pbmc)$isHVG <- rownames(pbmc) %in% HVGs
head(rowData(pbmc))
table(rowData(pbmc)$isHVG)

# Visualizing the mean-variance fit, coloring HVGs
df <- tibble(
    mean = metadata(pbmc_variance)$mean,
    var = metadata(pbmc_variance)$var,
    trend = metadata(pbmc_variance)$trend(mean),
    HVG = rowData(pbmc)$isHVG
)
p <- ggplot(df) +
    geom_point(aes(x = mean, y = var, col = HVG), alpha = 0.4) +
    geom_line(aes(x = mean, y = trend), col = 'darkred') +
    theme_minimal() +
    labs(x = 'Gene mean exp. (norm.)', y = 'Gene exp. variance')
```

6.2.2 Embedding in a lower dimensional linear space

We now have normalized counts filtered for the top 20% genes varying with the greatest biological significance.
Still, that represents a ~ 1,000 genes x ~4000 cells dataset. This is still too big to reliably use in standard clustering approaches. We can further compress the dataset. The most widely used approach is `PCA`: it computes a small number of "components" (typically 5-50) optimally summarizing the variability of the whole dataset, while retaining linearity of the underlying numerical data and being computationallt quite efficient.

Question

- Read `scater::denoisePCA()` documentation. What is the benefit of this function compared to `runPCA()`?
- Leverage `scater` package to compute `PCA` embedding of the filtered data, by taking into account the technical variability.

Answer

```
pbmc <- scran::denoisePCA(
    pbmc,
    technical = pbmc_variance,
    subset.row = HVGs,
    min.rank = 15
)
dim(as.data.frame(reducedDim(pbmc)))
head(as.data.frame(reducedDim(pbmc)))
p <- cowplot::plot_grid(
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'detected'),
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'sum')
)
```

Question

Check levels of gene expression for few genes (e.g. `CD8A`, `MS4A1`, ...) using PCA embedding for visualization. Comment

Answer

```
p <- cowplot::plot_grid(
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'CD8A'),
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'MS4A1'),
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'PPBP'),
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'FCER1A')
)
```

6.3 3. Clustering

Clustering is an unsupervised learning procedure used in scRNA-seq data analysis to empirically define groups of cells with similar expression profiles. Its primary purpose is to summarize the data in a digestible format for human interpretation.

After annotation based on marker genes, the clusters can be treated as proxies for more abstract biological concepts such as cell types or states. Clustering is thus a critical step for extracting biological insights from scRNA-seq data.

6.3.1 Clustering algorithms

Three main approaches can be used:

1. Hierarchical clustering
2. k-means clustering
3. Graph-based clustering

Today, we will focus on graph-based clustering, as it is becoming the standard for scRNAseq: it is a flexible and scalable technique for clustering even the largest scRNA-seq datasets. We first build a graph where each node is a cell that is connected by edges to its nearest neighbors in the high-dimensional space. Edges are weighted based on the similarity between the cells involved, with higher weight given to cells that are more closely related.

Question

Compute graph-based clustering of the PBMC dataset.

Answer

```
graph <- scran::buildSNNGraph(pbmc, use.dimred = 'PCA')
pbmc_clust <- igraph::cluster_louvain(graph)$membership
table(pbmc_clust)
pbmc$clusters_graph <- factor(pbmc_clust)
```

Question

- What are the main parameters to choose? How do they impact the clustering?
- Try a non-default value for `k` argument. What is the impact on the clustering?

Answer

```
# Re-compute a graph changing the `k` parameter, and identify resulting clusters
graph2 <- scran::buildSNNGraph(pbmc, k = 50, use.dimred = 'PCA')
pbmc_clust2 <- igraph::cluster_louvain(graph2)$membership
pbmc$clusters_graph_2 <- factor(pbmc_clust2)

# Compare original and new clusters
table(pbmc_clust, pbmc_clust2)

# Visually compare original and new clusters
p <- cowplot::plot_grid(
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'clusters_graph', text_by = 'clust
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'clusters_graph_2', text_by = 'clu
)
```

6.3.2 Dimensional reduction for clustering visualization

`PCA` is a powerful linear approach to compress large datasets into smaller dimensional spaces. However, it struggles at emphasizing the existence of clusters in complex datasets, when visualized in 2D.

`scater` provides a handy way to perform more complex data embeddings:

- `tSNE`
- `UMAP`
- `Diffusion Map`
- `Multi-Dimensional Scaling (MDS)`
- `Non-negative Matrix Factorization (NMF)`

Question

- Explore the different dimensional reduction algorithms, trying different hyperparameters combinations.
- When you run these commands, pay attention to how long each command takes to run!
- While this run, check the `Help` page for each function (e.g. `?scater::runTSNE`)

Answer

```
reducedDims(pbmc)
pbmc <- scater::runTSNE(pbmc)
pbmc <- scater::runUMAP(pbmc)
pbmc <- scater::runDiffusionMap(pbmc, dimred = 'PCA')
reducedDims(pbmc)
reducedDim(pbmc, 'DiffusionMap')[1:10, ]
```

Question

- Use the `scater::plotReducedDim()` function to plot cells in each embedding. Comment.

Answer

```
p<- cowplot::plot_grid(
    scater::plotReducedDim(pbmc, 'PCA', colour_by = 'clusters_graph') + ggtitle('denoi
    scater::plotReducedDim(pbmc, 'TSNE', colour_by = 'clusters_graph') + ggtitle('tSNE
    scater::plotReducedDim(pbmc, 'UMAP', colour_by = 'clusters_graph') + ggtitle('UMAP
    scater::plotReducedDim(pbmc, 'DiffusionMap', colour_by = 'clusters_graph') + ggtit
)
```

Part IV

Day 4

7 Demonstration: Cell type annotation and dataset integration

Goals:

- Integrate multiple datasets together
- Transfer annotations from one dataset to another
- Project one dataset onto another dataset's embedding

7.1 1. Integrating two replicates together

We have sequenced 2 replicates of WT cells differentiated into MCCs. Let's process both datasets independantly.

```
library(SingleCellExperiment)
library(tidyverse)
Bl6J_WT <- readRDS('data/MCCs/Bl6J_WT.rds')
Bl6N_WT <- readRDS('data/MCCs/Bl6N_WT.rds')
future::plan(strategy = "multicore", workers = 16)
set.seed(1000)

# Bl6J_WT
cnts <- as(assay(Bl6J_WT, 'counts'), 'dgCMatrix')
colnames(cnts) <- Bl6J_WT$Barcode
rownames(cnts) <- rownames(Bl6J_WT)
Bl6J_WT_vst <- sctransform::vst(cnts, return_cell_attr = TRUE)
Bl6J_WT <- Bl6J_WT[rownames(Bl6J_WT_vst$y), ]
assay(Bl6J_WT, 'corrected_counts', withDimnames = FALSE) <- sctransform::correct(Bl6J_WT_v
assay(Bl6J_WT, 'logcounts', withDimnames = FALSE) <- log1p(assay(Bl6J_WT, 'corrected_count
Bl6J_WT_variance <- scran::modelGeneVar(Bl6J_WT)
HVGs <- scran::getTopHVGs(Bl6J_WT_variance, prop = 0.1)
Bl6J_WT <- scran::denoisePCA(Bl6J_WT, technical = Bl6J_WT_variance, subset.row = HVGs, min
```

```
Bl6J_WT <- scater::runUMAP(Bl6J_WT)
Bl6J_WT$cluster <- factor(igraph::cluster_louvain(scran::buildSNNGraph(Bl6J_WT, use.dimred

# Bl6N_WT
cnts <- as(assay(Bl6N_WT, 'counts'), 'dgCMatrix')
colnames(cnts) <- Bl6N_WT$Barcode
rownames(cnts) <- rownames(Bl6N_WT)
Bl6N_WT_vst <- sctransform::vst(cnts, return_cell_attr = TRUE)
Bl6N_WT <- Bl6N_WT[rownames(Bl6N_WT_vst$y), ]
assay(Bl6N_WT, 'corrected_counts', withDimnames = FALSE) <- sctransform::correct(Bl6N_WT_v
assay(Bl6N_WT, 'logcounts', withDimnames = FALSE) <- log1p(assay(Bl6N_WT, 'corrected_count
Bl6N_WT_variance <- scran::modelGeneVar(Bl6N_WT)
HVGs <- scran::getTopHVGs(Bl6N_WT_variance, prop = 0.1)
Bl6N_WT <- scran::denoisePCA(Bl6N_WT, technical = Bl6N_WT_variance, subset.row = HVGs, min
Bl6N_WT <- scater::runUMAP(Bl6N_WT)
Bl6N_WT$cluster <- factor(igraph::cluster_louvain(scran::buildSNNGraph(Bl6N_WT, use.dimred

# Compare side-by-side
p <- cowplot::plot_grid(
    scater::plotReducedDim(Bl6N_WT, 'UMAP', colour_by = 'cluster', text_by = 'cluster') +
    scater::plotReducedDim(Bl6J_WT, 'UMAP', colour_by = 'cluster', text_by = 'cluster') +
)
ggsave('data/MCCs/WT-replicates_UMAP.pdf', w = 10, h = 5)
```

Process them together without genotype correction

```
Bl6J_WT <- readRDS('data/MCCs/Bl6J_WT.rds')
Bl6N_WT <- readRDS('data/MCCs/Bl6N_WT.rds')
# Merge two genotypes
MCCs <- cbind(Bl6J_WT, Bl6N_WT)
set.seed(1000)

# Normalize counts with VST
cnts <- as(assay(MCCs, 'counts'), 'dgCMatrix')
colnames(cnts) <- MCCs$Barcode
rownames(cnts) <- rownames(MCCs)
MCCs_vst <- sctransform::vst(cnts, return_cell_attr = TRUE)
MCCs <- MCCs[rownames(MCCs_vst$y), ]
assay(MCCs, 'corrected_counts', withDimnames = FALSE) <- sctransform::correct(MCCs_vst)
assay(MCCs, 'logcounts', withDimnames = FALSE) <- log1p(assay(MCCs, 'corrected_counts'))
# Flag HVGs
```

```
MCCs_variance <- scran::modelGeneVar(MCCs)
HVGs <- scran::getTopHVGs(MCCs_variance, prop = 0.1)
rowData(MCCs)$HVG <- rownames(MCCs) %in% HVGs
# Embed in PCA
MCCs <- scran::denoisePCA(MCCs, technical = MCCs_variance, subset.row = HVGs, min.rank = 1
MCCs <- scater::runUMAP(MCCs)
# Cluster cells
MCCs$cluster <- factor(igraph::cluster_louvain(scran::buildSNNGraph(MCCs, use.dimred = 'PC
p <- cowplot::plot_grid(
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'cluster', text_by = 'cluster') + ggt
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'batch', text_by = 'cluster') + ggtit
)
ggsave('data/MCCs/WT-replicates-merged_UMAP.pdf', w = 10, h = 5)
```

Now let's do it again, but correcting for genotype.

```
set.seed(1000)
mergedBatches <- batchelor::fastMNN(
    MCCs,
    batch = MCCs$batch,
    subset.row = HVGs,
    BPPARAM = BiocParallel::MulticoreParam(workers = 12)
)
mergedBatches
rowData(mergedBatches)
MCCs
reducedDims(mergedBatches)
reducedDim(MCCs, 'corrected_PCA') <- reducedDim(mergedBatches, 'corrected')
MCCs$corrected_cluster <- factor(igraph::cluster_louvain(scran::buildSNNGraph(MCCs, use.di
set.seed(1000)
reducedDim(MCCs, 'corrected_UMAP') <- scater::calculateUMAP(t(reducedDim(MCCs, 'corrected_
p <- cowplot::plot_grid(
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'cluster', text_by = 'cluster') + ggt
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'batch', text_by = 'batch') + ggtitle
    scater::plotReducedDim(MCCs, 'corrected_UMAP', colour_by = 'batch', text_by = 'batch')
    scater::plotReducedDim(MCCs, 'corrected_UMAP', colour_by = 'corrected_cluster', text_b
)
ggsave('data/MCCs/WT-replicates-corrected_UMAP.pdf', w = 10, h = 10)
```

7.2 2. Reading CcnoKO dataset in R

There is a scRNAseq dataset of Ccno KO cells trying to undergo in vitro differentiation.

```
CcnoKO <- readRDS('data/MCCs/CcnoKO.rds')
set.seed(1000)

# Normalize counts with VST
cnts <- as(assay(CcnoKO, 'counts'), 'dgCMatrix')
colnames(cnts) <- CcnoKO$Barcode
rownames(cnts) <- rownames(CcnoKO)
CcnoKO_vst <- sctransform::vst(cnts, return_cell_attr = TRUE)
CcnoKO <- CcnoKO[rownames(CcnoKO_vst$y), ]
assay(CcnoKO, 'corrected_counts', withDimnames = FALSE) <- sctransform::correct(CcnoKO_vst
assay(CcnoKO, 'logcounts', withDimnames = FALSE) <- log1p(assay(CcnoKO, 'corrected_counts'
# Flag HVGs
CcnoKO_variance <- scran::modelGeneVar(CcnoKO)
HVGs <- scran::getTopHVGs(CcnoKO_variance, prop = 0.1)
rowData(CcnoKO)$HVG <- rownames(CcnoKO) %in% HVGs
# Embed in PCA
CcnoKO <- scran::denoisePCA(CcnoKO, technical = CcnoKO_variance, subset.row = HVGs, min.ra
CcnoKO <- scater::runUMAP(CcnoKO)
# Cluster cells
CcnoKO$cluster <- factor(igraph::cluster_louvain(scran::buildSNNGraph(CcnoKO, use.dimred =
p <- cowplot::plot_grid(
    scater::plotReducedDim(CcnoKO, 'UMAP', colour_by = 'cluster', text_by = 'cluster') + g
    scater::plotReducedDim(CcnoKO, 'UMAP', colour_by = 'batch', text_by = 'cluster') + ggt
)
ggsave('data/MCCs/CcnoKO_UMAP.pdf', w = 10, h = 5)
```

7.3 3. Annotating CcnoKO dataset with WT MCCs dataset using `scmap`

We have high-quality annotations for MCCs dataset, but not for CcnoKO dataset. Can we transfer annotations from MCCs to CcnoKO?

```
# Prepare feature indices from MCCs
set.seed(1000)
rowData(MCCs)$feature_symbol <- rowData(MCCs)$Symbol
```

```
MCCs <- scmap::selectFeatures(MCCs, suppress_plot = TRUE)
MCCs <- scmap::indexCluster(MCCs, cluster_col = 'annotation')
metadata(MCCs)
head(metadata(MCCs)[['scmap_cluster_index']])

# Map clusters from MCCs onto CcnoKO
set.seed(1000)
rowData(CcnoKO)$feature_symbol <- rowData(CcnoKO)$Symbol
CcnoKO_scmap_clus <- scmap::scmapCluster(
    projection = CcnoKO,
    index_list = list(yan = metadata(MCCs)$scmap_cluster_index)
)

# Get transferred annotations
CcnoKO$annotation_projected <- factor(CcnoKO_scmap_clus$combined_labs, levels = levels(MCC

# Plot reduced dims
p <- cowplot::plot_grid(
    scater::plotReducedDim(CcnoKO, 'UMAP', colour_by = 'batch', text_by = 'batch') + ggtit
    scater::plotReducedDim(CcnoKO, 'UMAP', colour_by = 'cluster', text_by = 'cluster') + g
    scater::plotReducedDim(CcnoKO, 'UMAP', colour_by = 'annotation_projected', text_by = '
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'annotation', text_by = 'annotation')
    ncol = 2
)
ggsave('data/MCCs/CcnoKO-transferred-annotations_UMAP.pdf', w = 10, h = 10)
```

7.4 4. Mapping CcnoKO onto WT MCCs cells

Another way to **visualize** which WT cell types the CcnoKO cells spatially overlap with is to project CcnoKO cells onto MCC cells in UMAP embedding. This *could* be done manually, by using the rotation matrix obtained from MCCs embedding in PCA space to "learn" PCA embedding of the CcnoKO data, etc..., however this process is rather hazardous when `fastMNN()` is first used to correct for batch bias.

Luckily, this process is facilitated in Seurat, with the `MapQuery()` function. However, we do need to re-process most of the data in order to project CcnoKO cells onto WT MCC UMAP embedding.

```
# Re-process each dataset separately with Seurat
library(Seurat)
```

```
options(future.globals.maxSize= 891289600)
MCCs_seurat <- as.Seurat(MCCs) %>%
    NormalizeData() %>%
    FindVariableFeatures(selection.method = "vst", nfeatures = 2000) %>%
    ScaleData() %>%
    RunPCA() %>%
    RunUMAP(reduction = "pca", dims = 1:30, return.model = TRUE)
CcnoKO_seurat <- as.Seurat(CcnoKO) %>%
    NormalizeData() %>%
    FindVariableFeatures(selection.method = "vst", nfeatures = 2000) %>%
    ScaleData() %>%
    RunPCA() %>%
    RunUMAP(reduction = "pca", dims = 1:30, return.model = TRUE)

# Transfer anchors from MCCs to CcnoKO
anchors <- FindTransferAnchors(
    reference = MCCs_seurat,
    query = CcnoKO_seurat,
    reference.reduction = "pca"
)

# Project CcnoKO onto MCCs UMAP embedding
CcnoKO_seurat <- MapQuery(
    anchorset = anchors,
    reference = MCCs_seurat,
    query = CcnoKO_seurat,
    reference.reduction = "pca",
    reduction.model = "umap"
)

# Exporting back the learnt UMAP to CcnoKO
reducedDim(MCCs, 'learnt_UMAP') <- Embeddings(MCCs_seurat, reduction = "umap")
reducedDim(CcnoKO, 'learnt_UMAP') <- Embeddings(CcnoKO_seurat, reduction = "ref.umap")

# Plot new embeddings
p <- cowplot::plot_grid(
    scater::plotReducedDim(MCCs, 'UMAP', colour_by = 'annotation', text_by = 'annotation')
    scater::plotReducedDim(MCCs, 'learnt_UMAP', colour_by = 'annotation', text_by = 'annot
    scater::plotReducedDim(CcnoKO, 'UMAP', colour_by = 'annotation_projected', text_by = '
    scater::plotReducedDim(CcnoKO, 'learnt_UMAP', colour_by = 'annotation_projected', text
    ncol = 2
```

```
)
ggsave('data/MCCs/CcnoKO-projected_UMAP.pdf', w = 10, h = 10)
```

8 Exercises: scRNAseq analysis with R/Bioconductor (3/3)

Goals:

- Perform differential expression to suggest preliminary cell type annotations
- Perform automated cell annotation using public reference datasets
- Attempt scRNAseq sub-clustering to better resolve single cell heterogeneity

---

8.1 0. Pre-processing PBMC dataset

During the previous day, the homeworks focused on processing 4K PBMC dataset to obtain main cell clusters. Here are the main commands to process this dataset.

```
set.seed(1000)
# Importing 4K PBMC data from 10X Genomics in R
pbmc <- TENxPBMCData::TENxPBMCData('pbmc4k')
rownames(pbmc) <- scuttle::uniquifyFeatureNames(rowData(pbmc)$ENSEMBL_ID, rowData(pbmc)$Sy

# Remove doublets
pbmc <- scDblFinder::scDblFinder(pbmc)
pbmc <- pbmc[, pbmc$scDblFinder.class == 'singlet']

# Recover human genomic, protein-coding gene informations
library(plyranges)
ah <- AnnotationHub::AnnotationHub()
AnnotationHub::query(ah, c('gene annotation', 'ensembl', '102', 'homo_sapiens', 'GRCh38'))
gtf <- AnnotationHub::query(ah, c('Homo_sapiens.GRCh38.102.chr.gtf'))[[1]]
genes <- gtf %>%
    filter(type == 'gene') %>%
    filter(gene_biotype == 'protein_coding') %>%
    filter(gene_source == 'ensembl_havana')
```

```r
# Annotate genes in PBMC dataset and filter out non-relevant genes
pbmc <- pbmc[genes$gene_name[genes$gene_name %in% rownames(pbmc)], ]
rowRanges(pbmc) <- genes[match(rownames(pbmc), genes$gene_name)]
rowData(pbmc) <- rowData(pbmc)[, c('gene_name', 'gene_id')]
rownames(pbmc) <- scuttle::uniquifyFeatureNames(rowData(pbmc)$gene_id, rowData(pbmc)$gene_

# Get preliminary QCs per cell and per gene
pbmc <- scuttle::addPerCellQCMetrics(pbmc)
pbmc <- scuttle::addPerFeatureQCMetrics(pbmc)

# Filter out genes not expressed in at least 10 cells
pbmc <- pbmc[rowSums(counts(pbmc) > 0) >= 10, ]

# Normalize counts using VST
cnts <- as(SingleCellExperiment::counts(pbmc), 'dgCMatrix')
colnames(cnts) <- pbmc$Barcode
rownames(cnts) <- rownames(pbmc)
pbmc_vst <- sctransform::vst(cnts, return_cell_attr = TRUE)
corrected_cnts <- sctransform::correct(pbmc_vst)
assay(pbmc, 'corrected_counts', withDimnames = FALSE) <- corrected_cnts
assay(pbmc, 'logcounts', withDimnames = FALSE) <- log1p(corrected_cnts)

# Computing biological variance of each gene
pbmc_variance <- scran::modelGeneVar(pbmc)
HVGs <- scran::getTopHVGs(pbmc_variance, prop = 0.1)
rowData(pbmc)$isHVG <- rownames(pbmc) %in% HVGs

# Embedding dataset in PCA space and removing technical variance
pbmc <- scran::denoisePCA(
    pbmc,
    technical = pbmc_variance,
    subset.row = HVGs,
    min.rank = 15
)

# Embedding dataset in shared k-nearest neighbors graph for clustering
graph <- scran::buildSNNGraph(pbmc, use.dimred = 'PCA')

# Cluster cells using Louvain community finding algorithm
pbmc_clust <- igraph::cluster_louvain(graph)$membership
table(pbmc_clust)
```

```r
pbmc$clusters_graph <- factor(pbmc_clust)

# Embedding dataset in t-SNE space for visualization
pbmc <- scater::runTSNE(pbmc)
```

8.2 1. Differential expression analysis and marker genes

To interpret clustering results, one needs to identify the genes that drive separation between clusters. These marker genes allow to assign biological meaning to each cluster based on their functional annotation. In the most obvious case, the marker genes for each cluster are *a priori* associated with particular cell types, allowing us to treat the clustering as a *proxy* for cell type identity.

A general strategy is to perform DE tests between pairs of clusters and then combine results into a single ranking of marker genes for each cluster.

Question

- Read `scran::findMarkers()` documentation
- Run the function on the PBMC dataset to find all the markers associated with individual graph-based clusters.

Answer

```r
markers <- scran::findMarkers(pbmc, groups = pbmc$clusters_graph)
markers %>%
    as('list') %>%
    map(function(x){as_tibble(x, rownames = 'gene') %>% filter(Top <= 5)}) %>%
    bind_rows(.id = 'cluster')
```

Question

- Re-run `scran::findMarkers()` to only find markers strongly overexpressed in each cluster.

Answer

```
markers <- scran::findMarkers(
    pbmc,
    groups = pbmc$clusters_graph,
    direction = "up",
    lfc = 1
)
head(markers[[1]])
markers %>%
    as('list') %>%
    map(function(x){as_tibble(x, rownames = 'gene') %>% filter(Top <= 5)}) %>%
    bind_rows(.id = 'cluster')
```

Question

- Plot average expression of the first marker of the first cluster in tSNE

Answer

```
p <- scater::plotReducedDim(pbmc, 'TSNE', colour_by = rownames(markers[[1]])[1])
```

Question

- Check knwon PBMC markers in the Human Protein Atlas, which compiles a very nice overview of gene expression in different cell types, e.g. here.
- Looking at these PBMC markers in the dataset, speculate to propose a label for each cluster in this 4K PBMC dataset.

Answer

```
markers <- c(
    'FCER1A', # DC markers
    'GNLY', # NK markers
    'PPBP', # Platelets markers
    'MS4A7', # Monocytes markers
    'MS4A1', # B cell markers
    'IL7R', # CD4 T cell markers
    'CD8A', 'CD8B' # CD8 T cell markers
)
p <- lapply(markers, function(g) {
    scater::plotReducedDim(pbmc, 'TSNE', colour_by = g) + ggtitle(g) + theme(legend.po
}) %>% cowplot::plot_grid(plotlist = .)
```

8.3 2. Automated cell annotation

Many human cell type reference databases are available over the Internet, especially for blood tissue. Today, we will use a reference constructed from `Monaco et al., Cell Reports 2019` (doi: 10.1016/j.celrep.2019.01.041). This reference is available as a `SummarizedExperiment` containing log-normalized gene expression for manually annotated samples.

Question

- Import Monaco dataset in R. Inspect its content. The structure of the object should feel familiar: it's a `Summarized Experiment`!
- Check the publication report. How was each sample (column) obtained? What type of sequencing?
- What types of cell annotation are available? Can this reference be useful for the annotation of the PBMC dataset?

Answer

```
monaco <- celldex::MonacoImmuneData()
monaco
dim(monaco)
colData(monaco)
rowData(monaco)
table(monaco$label.main)
table(monaco$label.fine)
```

Question

- Read `SingleR` documentation. Can it be leveraged to transfer reference annotations to PBMC dataset?
- Use `SingleR` to transfer reference annotations to PBMC dataset.
- Check how transferred annotations recapitulate manual graph-based clustering.

Answer

```
predictions_main <- SingleR::SingleR(
    test = pbmc,
    ref = monaco,
    labels = monaco$label.main
)
predictions_fine <- SingleR::SingleR(
    test = pbmc,
    ref = monaco,
    labels = monaco$label.fine
)
pbmc$annotation_hierarchy_1 <- predictions_main$labels
pbmc$annotation_hierarchy_2 <- predictions_fine$labels
table(pbmc$annotation_hierarchy_1)
table(pbmc$annotation_hierarchy_2)
table(pbmc$annotation_hierarchy_1, pbmc$clusters_graph)
table(pbmc$annotation_hierarchy_2, pbmc$annotation_hierarchy_1)
p <- cowplot::plot_grid(
    scater::plotReducedDim(pbmc, dimred = 'TSNE', colour_by = 'clusters_graph', text_b
    scater::plotReducedDim(pbmc, dimred = 'TSNE', colour_by = 'annotation_hierarchy_1'
    scater::plotReducedDim(pbmc, dimred = 'TSNE', colour_by = 'annotation_hierarchy_2'
)
```

Question

- Using `scater` and `SingleR` utilities, check the annotation score for each cell in the scRNAseq. Did the automated annotation work robuslty?
- Is automated annotation as sensitive as graph-based clustering, in this context?

Answer

```
p <- SingleR::plotScoreHeatmap(predictions_fine)
p <- pheatmap::pheatmap(
    log2(table(Assigned = pbmc$annotation_hierarchy_2, Cluster = pbmc$clusters_graph)+
    color=colorRampPalette(c("white", "darkred"))(101)
)
```

Question

- Using main and fine annotations, label each cluster in 2 lists of hierarchical labels

Answer

```
hierarchy_1 <- c(
    '1' = 'DC',
    '2' = 'DC',
    '3' = 'B',
    '4' = 'NK',
    '5' = 'Mono',
    '6' = 'T',
    '7' = 'Mono',
    '8' = 'T',
    '9' = 'T',
    '10' = 'T',
    '11' = 'Mono'
)
hierarchy_2 <- c(
    '1' = 'Myel. DC',
    '2' = 'Plasma. DC',
    '3' = 'B',
    '4' = 'NK',
    '5' = 'Inter./non-classic Mono',
    '6' = 'Helper T',
    '7' = 'Classical Mono',
    '8' = 'Eff. CD8 T',
    '9' = 'Naive T',
    '10' = 'Naive T',
    '11' = 'Classical Mono'
)
pbmc$label_hierarchy_1 <- hierarchy_1[as.character(pbmc$clusters_graph)]
pbmc$label_hierarchy_2 <- hierarchy_2[as.character(pbmc$clusters_graph)]
p <- cowplot::plot_grid(
    scater::plotReducedDim(pbmc, dimred = 'TSNE', colour_by = 'label_hierarchy_1', tex
    scater::plotReducedDim(pbmc, dimred = 'TSNE', colour_by = 'label_hierarchy_2', tex
)
```

Note how cells from cluster 1 and 2 are both robustly identifed as DCs. Yet, they appear in tSNE as 2 well-separated clusters. This discrepancy most likely comes from the fact that at a

finer level, they seem to be 2 different types of DCs: plasmacytoid DCs and myeloid DCs.

Question

- Check genes preferentially enriched in plasma. DCs vs myeloid DCs

Answer

```
DCs <- pbmc[ , pbmc$label_hierarchy_1 == 'DC']
markers <- scran::findMarkers(
    DCs,
    groups = DCs$label_hierarchy_2,
    direction = "up",
    lfc = 1
)
markers[[2]] %>%
    as_tibble(rownames = 'gene') %>%
    dplyr::filter(summary.logFC > log2(2), FDR <= 0.01)
```

8.4 3. Subclustering of T cells

T cells are spatially separated in 2 or 3 broad groups. However, their complexity is much more important than this. Despite the fine annotations obtained from transfer of Monaco data, T cells heterogeneity are poorly resolved.

Question

- Subset the T cells and re-process them (variance modelling, PCA embedding, graph-based clustering and tSNE embedding)

Answer

```
Tcells <- pbmc[ , pbmc$label_hierarchy_1 == 'T']

# Computing biological variance of each gene
set.seed(1000)
Tcells_variance <- scran::modelGeneVar(Tcells)
HVGs <- scran::getTopHVGs(Tcells_variance, prop = 0.2)
rowData(Tcells)$isHVG <- rownames(Tcells) %in% HVGs

# Embedding dataset in PCA space and removing technical variance
set.seed(1000)
Tcells <- scran::denoisePCA(
    Tcells,
    technical = Tcells_variance,
    subset.row = HVGs,
    min.rank = 15
)

# Embedding dataset in shared k-nearest neighbors graph for clustering
graph <- scran::buildSNNGraph(Tcells, k = 5, use.dimred = 'PCA', type = 'jaccard')

# Cluster cells using Louvain community finding algorithm
Tcells_clust <- igraph::cluster_louvain(graph)$membership
table(Tcells_clust)
Tcells$subclusters_graph <- factor(Tcells_clust)
table(Tcells$subclusters_graph, Tcells$clusters_graph)

# Embedding dataset in t-SNE space for visualization
set.seed(1000)
Tcells <- scater::runTSNE(Tcells, name = 'subTSNE')

# Visualize earlier clustering and new clustering
p <- cowplot::plot_grid(
    scater::plotReducedDim(Tcells, dimred = 'TSNE', colour_by = 'clusters_graph', text
    scater::plotReducedDim(Tcells, dimred = 'subTSNE', colour_by = 'clusters_graph', t
    scater::plotReducedDim(Tcells, dimred = 'subTSNE', colour_by = 'subclusters_graph'
)
```

Question

- Re-tranfer annotations from Monaco et al. to only T cells
- Does re-transferring annotations on a subset of cells change the annotation obtained for each individual cell?

Answer

```
Tcells_predictions_main <- SingleR::SingleR(
    test = Tcells,
    ref = monaco,
    labels = monaco$label.main
)
Tcells_predictions_fine <- SingleR::SingleR(
    test = Tcells,
    ref = monaco,
    labels = monaco$label.fine
)
Tcells$subannotation_hierarchy_1 <- Tcells_predictions_main$labels
Tcells$subannotation_hierarchy_2 <- Tcells_predictions_fine$labels
table(Tcells$subannotation_hierarchy_1, Tcells$annotation_hierarchy_1)
table(Tcells$subannotation_hierarchy_2, Tcells$annotation_hierarchy_2)
```

Question

- Compare the subclusters to transferred annotations. Does subclustering help better representing the heterogeneity of T cells?
- Propose alternative(s) to better resolve single-cell transcriptomes of T cells.

Answer

```
# Visualize earlier clustering and new clustering
p <- cowplot::plot_grid(
    scater::plotReducedDim(Tcells, dimred = 'subTSNE', colour_by = 'subclusters_graph'
    scater::plotReducedDim(Tcells, dimred = 'subTSNE', colour_by = 'annotation_hierarc
)

# Compare T cells original clusters and new subclusters to transferred fine annotation
table(Tcells$annotation_hierarchy_2, Tcells$clusters_graph)
table(Tcells$subannotation_hierarchy_2, Tcells$subclusters_graph)
p <- pheatmap::pheatmap(
    log2(table(Assigned = Tcells$annotation_hierarchy_2, Cluster = Tcells$clusters_gra
    color=colorRampPalette(c("white", "darkred"))(101),
    cluster_rows = FALSE, cluster_cols = FALSE
)
p <- pheatmap::pheatmap(
    log2(table(Assigned = Tcells$annotation_hierarchy_2, Cluster = Tcells$subclusters_
    color=colorRampPalette(c("white", "darkred"))(101),
    cluster_rows = FALSE, cluster_cols = FALSE
)
```

Part V

Day 5

9 Demonstration: Trajectory inference and pseudotime

Goals:

- Infer trajectory in a single scRNAseq dataset
- Infer a trajectories for conditions in an integrated dataset
- Perform DE expression analysis on a portion of the trajectory

---

9.1 1. Trajectories in WT MCCs

```
# Load data
MCCs <- readRDS('data/MCCs/MCCs_processed.rds')
CcnoKO <- readRDS('data/MCCs/CcnoKO_processed.rds')
# Infer lineages
library(slingshot)
MCCs_slingshot <- slingshot(
    MCCs,
    reducedDim = 'PCA',
    clusterLabels = 'annotation',
    start.clus = 'CyclingProgenitors',
    end.clus = 'MCCs',
    approx_points = 60
)
slingLineages(MCCs_slingshot)

# Plot individual lineage tracing
MCCs_slingData <- embedCurves(MCCs_slingshot, "corrected_UMAP")
MCCs$slingshot1 <- colData(MCCs_slingshot)[, 'slingPseudotime_1']
MCCs$slingshot2 <- colData(MCCs_slingshot)[, 'slingPseudotime_2']
p <- cowplot::plot_grid(
    scater::plotReducedDim(MCCs, "corrected_UMAP", colour_by = 'slingshot1', text_by = 'an
```

```
        geom_path(
            data = setNames(as.data.frame(slingCurves(MCCs_slingData)[[1]])$s[slingCurves(M
            mapping = aes(x = X, y = Y),
            inherit.aes = FALSE, size = 2
        ),
    scater::plotReducedDim(MCCs, "corrected_UMAP", colour_by = 'slingshot2', text_by = 'an
        geom_path(
            data = setNames(as.data.frame(slingCurves(MCCs_slingData)[[2]])$s[slingCurves(M
            mapping = aes(x = X, y = Y),
            inherit.aes = FALSE, size = 2
        )
)
ggsave('data/MCCs/WTs-trajectories.pdf', w = 10, h = 5)
```

9.2 2. Infer a single trajectory for WT + CcnoKO MCCs

```
Bl6J_WT <- readRDS('data/MCCs/Bl6J_WT.rds')
Bl6N_WT <- readRDS('data/MCCs/Bl6N_WT.rds')
CcnoKO <- readRDS('data/MCCs/CcnoKO.rds')

# Merge all datasets together
Bl6J_WT$sample <- 'WT'
Bl6N_WT$sample <- 'WT'
CcnoKO$sample <- 'KO'
rowData(CcnoKO) <- rowData(CcnoKO)[, c(1:7)]
rowData(Bl6J_WT) <- rowData(CcnoKO)
rowData(Bl6N_WT) <- rowData(CcnoKO)
mergedSCEs <- cbind(Bl6J_WT, Bl6N_WT, CcnoKO)
mergedSCEs <- scuttle::logNormCounts(mergedSCEs)
reducedDim(mergedSCEs, 'PCA') <- scater::calculatePCA(mergedSCEs)

# Find markers in WT cells
markers <- scran::findMarkers(
    mergedSCEs,
    groups = factor(mergedSCEs$annotation),
    pval.type = "any"
) %>% lapply(function(x) {
        as.data.frame(x) %>%
        arrange(desc(summary.logFC)) %>%
```

```
        rownames_to_column('marker') %>%
        '['(, 1:5) %>%
        filter(summary.logFC > log2(2) & p.value < 0.05) %>%
        pull(marker)
}) %>% do.call(c, .) %>% unique()

# Correct samples assuming they are all the same
mergedSCEs_corrected <- batchelor::fastMNN(
    mergedSCEs,
    batch = paste0(mergedSCEs$sample, '_', mergedSCEs$batch),
    d = 50,
    k = 15,
    correct.all = TRUE,
    subset.row = which(rownames(mergedSCEs) %in% markers),
    BSPARAM = BiocSingular::RandomParam(deferred = TRUE)
)
reducedDim(mergedSCEs, 'PCA_corrected') <- reducedDim(mergedSCEs_corrected, 'corrected')
reducedDim(mergedSCEs, 'UMAP_corrected') <- scater::calculateUMAP(mergedSCEs, dimred = "PC
p <- cowplot::plot_grid(
    scater::plotReducedDim(mergedSCEs, "PCA_corrected", colour_by = 'annotation', text_by
    scater::plotReducedDim(mergedSCEs, "PCA_corrected", colour_by = 'sample', text_by = 's
    scater::plotReducedDim(mergedSCEs, "PCA_corrected", colour_by = 'batch', text_by = 'ba
    scater::plotReducedDim(mergedSCEs, "UMAP_corrected", colour_by = 'annotation', text_by
    scater::plotReducedDim(mergedSCEs, "UMAP_corrected", colour_by = 'sample', text_by = '
    scater::plotReducedDim(mergedSCEs, "UMAP_corrected", colour_by = 'batch', text_by = 'b
    ncol = 3
)

# Slingshot TI
mergedSCEs <- mergedSCEs[, !is.na(mergedSCEs$annotation)]
reducedDim(mergedSCEs, 'PCA_corrected_2') <- reducedDim(mergedSCEs, 'PCA_corrected')[, 1:4
mergedSCEs_slingshot <- slingshot(
    mergedSCEs,
    reducedDim = 'PCA_corrected_2',
    clusterLabels = 'annotation',
    start.clus = 'CyclingProgenitors',
    end.clus = 'MCCs',
    approx_points = 30
)
slingLineages(mergedSCEs_slingshot)
```

```
# Plot individual lineage tracing
mergedSCEs_slingData <- embedCurves(mergedSCEs_slingshot, "UMAP_corrected")
mergedSCEs$slingshot1 <- pathStats(mergedSCEs_slingData)[[1]][, 'Lineage1']
p <- scater::plotReducedDim(mergedSCEs, "UMAP_corrected", colour_by = 'slingshot1', text_b
    geom_path(
        data = setNames(as.data.frame(slingCurves(mergedSCEs_slingData)[[1]]$s[slingCurves
        mapping = aes(x = X, y = Y),
        inherit.aes = FALSE, size = 2
    )
ggsave('data/MCCs/joint-WT-CcnoKO-trajectory.pdf')
```

9.3 3. Perform DE expression analysis on a portion of the trajectory

9.3.1 Filter cells based on 99% max of pseudotime_slingshot for CcnoKO

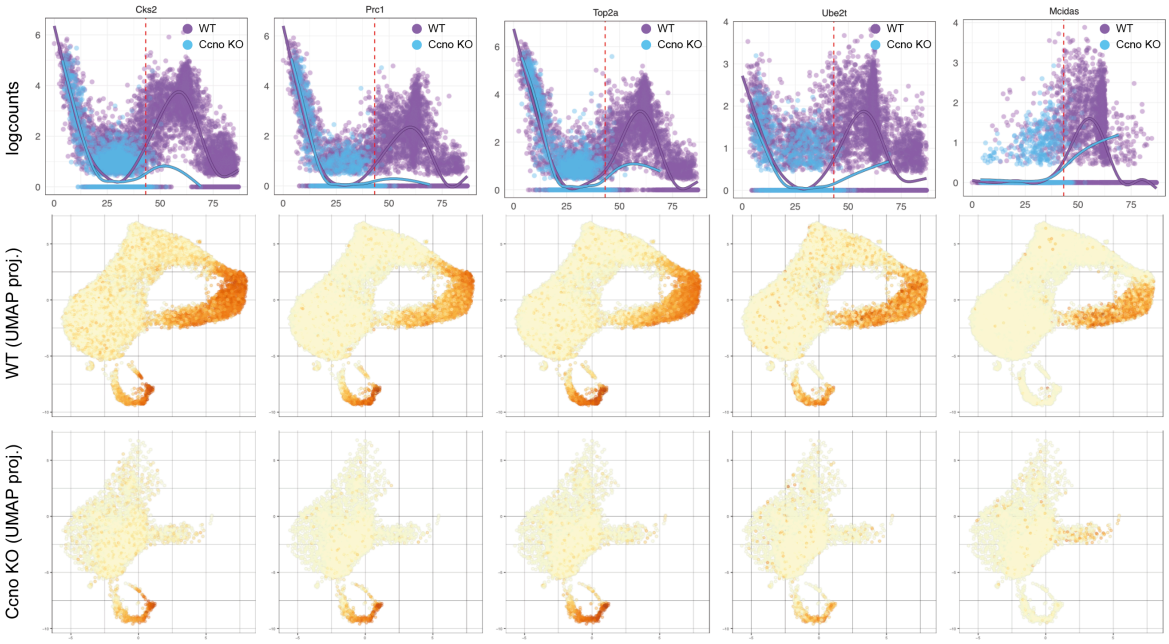
```
threshold <- quantile(mergedSCEs$slingshot1[mergedSCEs$sample == 'KO'], 0.99, na.rm = TRUE
mergedSCEs_subset <- mergedSCEs[, mergedSCEs$slingshot1 <= threshold & !is.na(mergedSCEs$s
p <- cowplot::plot_grid(
    scater::plotReducedDim(mergedSCEs_subset, "UMAP_corrected", colour_by = 'sample', text
    scater::plotReducedDim(mergedSCEs_subset, "UMAP_corrected", colour_by = 'batch', text_
    scater::plotReducedDim(mergedSCEs_subset, "UMAP_corrected", colour_by = 'annotation',
    ncol = 2
)
```

9.3.2 Running tradeSeq using new slingshot pseudotimes

Using counts, blocking on genotype, with WTs/CcnoKO as a condition.

```
counts <- as.matrix(assay(mergedSCEs_subset, 'counts'))
pseudotime <- data.frame("1" = mergedSCEs_subset$slingshot1)
cellWeights <- data.frame("1" = rep(1, ncol(mergedSCEs_subset)))
tradeSeq_res <- tradeSeq::fitGAM(
    counts = counts,
    pseudotime = pseudotime,
    cellWeights = cellWeights,
    U = as.matrix(data.frame(
        'genotype' = as.numeric(factor(mergedSCEs_subset$batch))
    )),
```

```
    conditions = factor(mergedSCEs_subset$sample, c('WT', 'KO')),
    parallel = FALSE
)
condRes <- tradeSeq::conditionTest(tradeSeq_res, l2fc = log2(1.5))
```



10 Exercises: Trajectory inference and RNA velocity

Goals:

- Understand the requirements for RNA velocity computation
- Process scRNAseq using 'spliced' counts
- Perform lineage inference
- Compute RNA velocity and use it to orientate lineages

10.1 0. Prepare data from scratch

Because RNA velocity reflects the balance between immature and mature transcript content in each cell, one need to count the reads overlapping both spliced regions and unspliced regions. These counts are still generally not available when using public datasets. A way to generate them is to:

1. Get a `cellranger`-generated bam file of a scRNAseq experiment
2. Get the corresponding gene annotation file as a `gtf` file
3. Run `velocyto` to count reads mapped to introns or to exons

Let's do this on a dataset published by Guo et al., Cell Res. 2018 (doi: 10.1038/s41422-018-0099-2). There are 6 `bam` files corresponding to human male testis single-cell RNA-seq profiling (GSE: GSE112013).

> Question
>
> - Download bam files from GEO

Answer

```
mkdir data/Guo_testis/
ffq -t GSE "GSE112013" | grep 'ftp://' | sed 's,.*ftp:,ftp:,' | sed 's,".*,,' > data/G
for FILE in `cat data/Guo_testis/GSE112013_bam-list.txt | sed '$d'`
do
    echo $FILE
    curl ${FILE} -o data/Guo_testis/`basename ${FILE}`
done
```

To run `velocyto`, one needs to know where introns and exons are located in the genome reference used to process reads. In our case, `GRCh38` genome reference was used.

Question

- Read `cellranger` instructions on how to generate a `gtf` file corresponding to GRCh38 genome reference here.
- Create `GRCh38` gene annotation gtf file following Cellranger recommendations

Answer

```
mkdir data/Guo_testis/genome
curl http://ftp.ebi.ac.uk/pub/databases/gencode/Gencode_human/release_32/gencode.v32.p
gunzip data/Guo_testis/genome/gencode.v32.primary_assembly.annotation.gtf.gz
ID="(ENS(MUS)?[GTE][0-9]+)\.([0-9]+)"
cat data/Guo_testis/genome/gencode.v32.primary_assembly.annotation.gtf \
    | sed -E 's/gene_id "'"$ID"'";/gene_id "\1"; gene_version "\3";/' \
    | sed -E 's/transcript_id "'"$ID"'";/transcript_id "\1"; transcript_version "\3";/
    | sed -E 's/exon_id "'"$ID"'";/exon_id "\1"; exon_version "\3";/' \
    > data/Guo_testis/genome/gencode.v32.primary_assembly.annotation_modified.gtf
cat data/Guo_testis/genome/gencode.v32.primary_assembly.annotation_modified.gtf \
    | awk '$3 == "transcript"' \
    | grep -E "$GENE_PATTERN" \
    | grep -E "$TX_PATTERN" \
    | grep -Ev "$READTHROUGH_PATTERN" \
    | grep -Ev "$PAR_PATTERN" \
    | sed -E 's/.*(gene_id "[^"]+").*/\1/' \
    | sort \
    | uniq \
    > data/Guo_testis/genome/gene_allowlist
grep -E "^#" data/Guo_testis/genome/gencode.v32.primary_assembly.annotation_modified.g
grep -Ff data/Guo_testis/genome/gene_allowlist data/Guo_testis/genome/gencode.v32.prim
    | sed -E 's,^chr,,' \
    | sed -E 's,^M\t,MT\t,' \
    >> data/Guo_testis/genome/gencode.v32.primary_assembly.annotation_filtered.gtf
```

To make the `velocyto` step faster, one can only use reads from bam files originating from cell-containing droplets. One way to do so is to extract cell barcodes from the already filtered scRNAseq dataset, and use them in the `-b` argument of `velocyto`.

Question

- (Optional) Get cell barcodes from cellranger (this requires pre-processed scRNAseq data, available from GEO)

Answer

```
library(tidyverse)
vroom::vroom('data/Guo_testis/GSE112013_Combined_UMI_table.txt') %>%
    colnames() %>%
    str_replace_all('Donor.-', '') %>%
    tail(-1) %>%
    unique() %>%
    writeLines('data/Guo_testis/testis_cell-barcodes.txt')
```

By now, we have obtained 3 different files:

1. BAM files of scRNAseq data mapped onto GRCh38
2. GTF file of GRCh38 gene annotations
3. A barcode file for BAM file pre-filtering

Question

- Run velocyto on each sample

Answer

```
mkdir data/Guo_testis/velocyto
for FILE in `cat data/Guo_testis/GSE112013_bam-list.txt`
do
    curl ${FILE} -o data/Guo_testis/`basename ${FILE}`
    velocyto run \
        -b data/Guo_testis/testis_cell-barcodes.txt \
        -o data/Guo_testis/velocyto/ \
        --samtools-threads 15 \
        -vv \
        data/Guo_testis/`basename ${FILE}` \
        data/Guo_testis/genome/gencode.v32.primary_assembly.annotation_filtered.gtf
    rm data/Guo_testis/`basename ${FILE}`
done
```

We have obtained six `loom` files, but ideally we want them merged as a single `SingleCellExperiment` object readable in R.

Question

- Read about `LoomExperiment` package. Is there an easy (and reliable) way to import loom files in R?
- Merge loom files directly in R and save the resulting object as a `rds` binary file.

Answer

```
library(tidyverse)
library(LoomExperiment)
looms <- list.files('data/Guo_testis/velocyto/', full.names = TRUE) %>%
    lapply(LoomExperiment::import) %>%
    do.call(cbind, .)
looms$sample <- str_replace_all(looms$CellID, ':.*', '')
looms$Barcode <- str_replace_all(looms$CellID, '.*:', '') %>% str_replace('x', '')
# Some additional tidying up...
# testis_2 <- testis[rownames(testis)[rownames(testis) %in% rowData(looms)$Gene], ]
# genes <- rownames(testis_2)
# bcs <- testis_2$Barcode
# looms <- looms[match(genes, rowData(looms)$Gene), ]
# looms <- scuttle::aggregateAcrossCells(looms, looms$Barcode, use.assay.type = c('spl
# looms <- looms[, match(bcs, looms$Barcode)]
#
saveRDS(looms, 'data/Guo_testis/testis_velocity-counts.rds')
```

10.2 1. Process testis data in R

The same workflow than previous days can be reused here.

Question

- Import testis dataset in R, filter cells and genes, normalize `counts`, embed data and cluster cells.

Answer

```
library(SingleCellExperiment)
library(tidyverse)
download.file(
    'https://ftp.ncbi.nlm.nih.gov/geo/series/GSE112nnn/GSE112013/suppl/GSE112013_Combi
    'data/Guo_testis/GSE112013_Combined_UMI_table.txt.gz'
)
system('gunzip data/Guo_testis/GSE112013_Combined_UMI_table.txt.gz')
x <- vroom::vroom('data/Guo_testis/GSE112013_Combined_UMI_table.txt')
cnts <- as.matrix(x[, -1])
gData <- as.data.frame(x[, 1])
cData <- data.frame(cellid = colnames(x[, -1]))
testis <- SingleCellExperiment(
    assays = list(counts = cnts),
    colData = cData,
    rowData = gData
)
testis$Barcode <- str_replace(testis$cellid, 'Donor.-', '') %>% str_replace('-.', '')
testis <- testis[, !duplicated(testis$Barcode)]
testis$donor <- str_replace(testis$cellid, '-.*', '')
testis$replicate <- str_replace(testis$cellid, '.*-', '')
rownames(testis) <- rowData(testis)$Gene
set.seed(1000)

# Remove doublets
testis <- scDblFinder::scDblFinder(testis)
testis <- testis[, testis$scDblFinder.class == 'singlet']

# Recover human genomic, protein-coding gene informations
library(plyranges)
ah <- AnnotationHub::AnnotationHub()
AnnotationHub::query(ah, c('gene annotation', 'ensembl', '102', 'homo_sapiens', 'GRCh3
gtf <- AnnotationHub::query(ah, c('Homo_sapiens.GRCh38.102.chr.gtf'))[[1]]
genes <- gtf %>%
    filter(type == 'gene') %>%
    filter(gene_biotype == 'protein_coding') %>%
    filter(gene_source == 'ensembl_havana')
genes <- genes[!duplicated(genes$gene_name)]

# Annotate genes in testis dataset and filter out non-relevant genes
testis <- testis[genes$gene_name[genes$gene_name %in% rownames(testis)], ]
```

Question

- Load `HPA` data from internet. Try to format it as a `SummarizedExperiment`. What celltypes are profiled?
- Use these cell type profiles to annotate cell types in the testis dataset.
- How do the annotations look like? Can you find a reason why the label transfer worked so well?

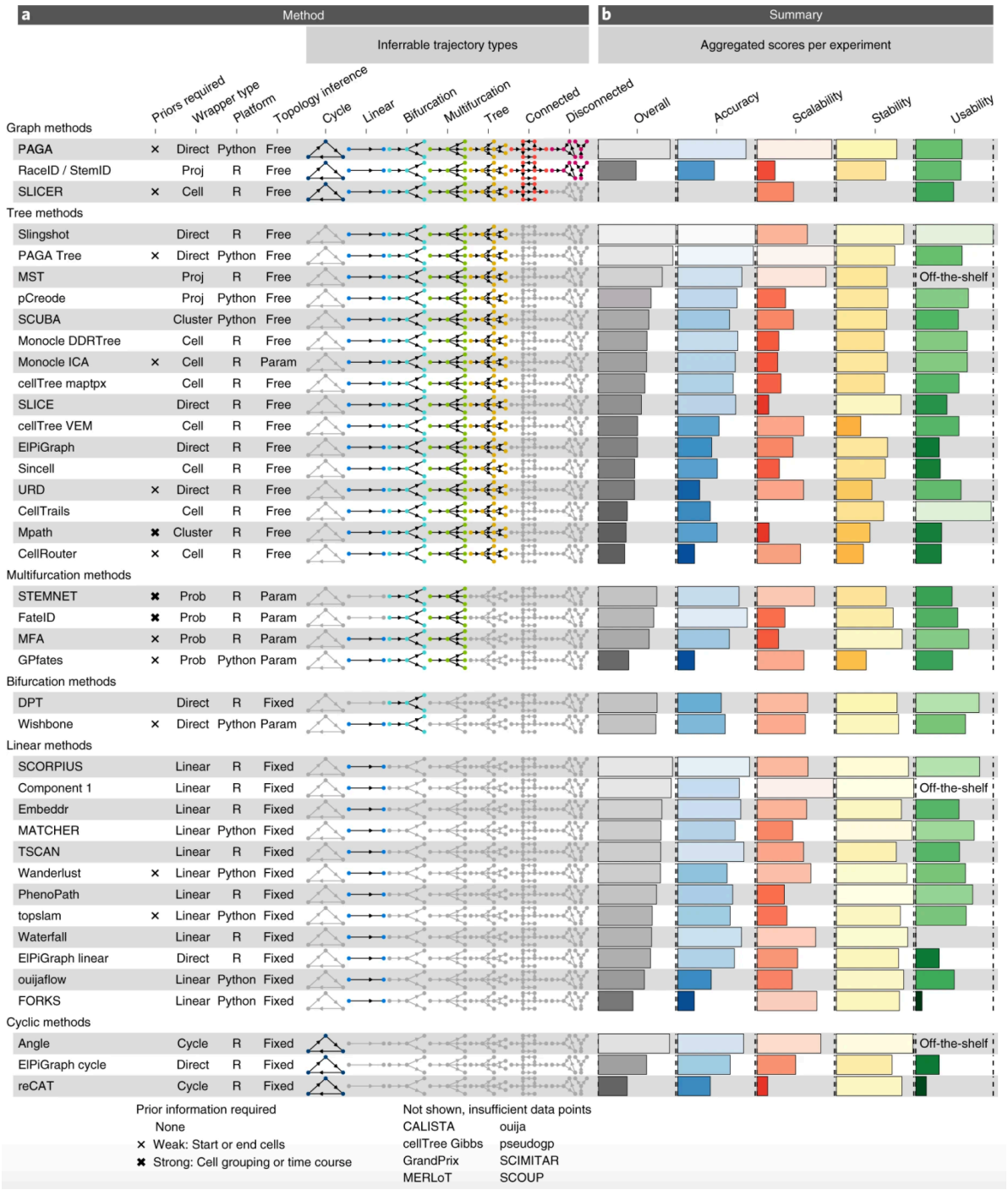
Answer

```
download.file(
    'https://www.proteinatlas.org/download/rna_single_cell_type.tsv.zip',
    'data/Guo_testis/rna_single_cell_type.tsv.zip'
)
system('unzip data/Guo_testis/rna_single_cell_type.tsv.zip')
system('mv rna_single_cell_type.tsv data/Guo_testis/')
HPA <- vroom::vroom('data/Guo_testis/rna_single_cell_type.tsv') %>%
    pivot_wider(names_from = `Cell type`, values_from = 'NX') %>%
    dplyr::select(-Gene) %>%
    distinct(`Gene name`, .keep_all = TRUE) %>%
    column_to_rownames('Gene name') %>%
    SummarizedExperiment::SummarizedExperiment(assays = list('logcounts' = .))

# Transfer annotations to `testis`
predictions <- SingleR::SingleR(
    test = testis,
    ref = HPA,
    labels = colnames(HPA)
)
table(predictions$labels, testis$cluster)
labels <- table(predictions$labels, testis$cluster) %>%
    data.matrix() %>%
    apply(2, which.max) %>%
    sort(unique(predictions$labels))[.]
names(labels) <- 1:length(labels)
testis$annotation <- labels[testis$cluster]
p <- cowplot::plot_grid(
    scater::plotReducedDim(testis, dimred = 'corrected', colour_by = 'cluster', text_b
    scater::plotReducedDim(testis, dimred = 'corrected', colour_by = 'annotation', tex
    scater::plotReducedDim(testis, dimred = 'TSNE', colour_by = 'cluster', text_by = '
    scater::plotReducedDim(testis, dimred = 'TSNE', colour_by = 'annotation', text_by
)
```

10.3 2. Trajectory inference (TI) in scRNAseq

An important question in scRNAseq field of research is: how to identify a cell trajectory from high-dimensional expression data and map individual cells onto it? A large number of methods have currently emerged, each one with their own specificities, assumptions, and strengths. A nice breakdown (from 2019, so already very outdated!) is available from `Saelens et al., Nat. Biotech. 2018` (doi: 10.1038/s41587-019-0071-9):


a
Method
Inferrable trajectory types
b
Summary
Aggregated scores per experiment
Priors required
Wrapper type
Platform
Topology inference
Cycle
Linear
Bifurcation
Multifurcation
Tree
Connected
Disconnected
Overall
Accuracy
Scalability
Stability
Usability
Graph methods
PAGA × Direct Python Free
RaceID / StemID Proj R Free
SLICER × Cell R Free
Tree methods
Slingshot Direct R Free
PAGA Tree × Direct Python Free
MST Proj R Free
pCreode Proj Python Free
SCUBA Cluster Python Free
Monocle DDRTree Cell R Free
Monocle ICA × Cell R Param
cellTree maptpx Cell R Free
SLICE Direct R Free
cellTree VEM Cell R Free
ElPiGraph Direct R Free
Sincell Cell R Free
URD × Direct R Free
CellTrails Cell R Free
Mpath ✖ Cluster R Free
CellRouter × Cell R Free
Multifurcation methods
STEMNET ✖ Prob R Param
FateID ✖ Prob R Param
MFA × Prob R Param
GPfates × Prob Python Param
Bifurcation methods
DPT Direct R Fixed
Wishbone × Direct Python Param
Linear methods
SCORPIUS Linear R Fixed
Component 1 Linear R Fixed
Embeddr Linear R Fixed
MATCHER Linear Python Fixed
TSCAN Linear R Fixed
Wanderlust × Linear Python Fixed
PhenoPath Linear R Fixed
topslam × Linear Python Fixed
Waterfall Linear R Fixed
ElPiGraph linear Direct R Fixed
ouijaflow Linear Python Fixed
FORKS Linear Python Fixed
Cyclic methods
Angle Cycle R Fixed
ElPiGraph cycle Direct R Fixed
reCAT Cycle R Fixed
Off-the-shelf
Prior information required
None
× Weak: Start or end cells
✖ Strong: Cell grouping or time course
Not shown, insufficient data points
CALISTA
cellTree Gibbs
GrandPrix
MERLoT
ouija
pseudogp
SCIMITAR
SCOUP

10.3.1 Slingshot

`Slingshot` is perhaps one of the most widely used algorithms for users who want to focus on `R`-based approaches.

Question

- Read Slingshot documentation to understand how to identify lineages in a scRNAseq dataset in R
- Infer lineages in the testis dataset
- Why is it recommended to infer lineages from PCA space rather than t-SNE or UMAP space, even though these spaces do "reveal" an obvious trajectory in 2D?

Answer

```
testis_slingshot <- slingshot::slingshot(testis, reducedDim = 'corrected')
testis_slingshot
slingshot::slingLineages(testis_slingshot)
```

Question

- Check the inferred trajectory(ies) in 2D projection. You can use the `embedCurves()` to embed the curves in any given dimensional space. Do they fit your expectations?

Answer

```
pca_curve <- slingCurves(testis_slingshot, as.df = TRUE)
colnames(pca_curve) <- paste0('PC', 1:ncol(pca_curve))
tsne_curve <- slingshot::embedCurves(testis_slingshot, 'TSNE', smoother = 'loess', spa
tsne_curve <- tsne_curve[order(tsne_curve$Order), ]
colnames(tsne_curve)[1:2] <- paste0('TSNE', 1:ncol(tsne_curve))
df <- tibble(
    PC1 = reducedDim(testis, 'corrected')[,1],
    PC2 = reducedDim(testis, 'corrected')[,2],
    TSNE1 = reducedDim(testis, 'TSNE')[,1],
    TSNE2 = reducedDim(testis, 'TSNE')[,2],
    cluster = testis$cluster
)
p <- cowplot::plot_grid(
    df %>%
        ggplot() +
        geom_point(aes(PC1, PC2, col = cluster)) +
        geom_path(data = pca_curve, aes(x = PC1, y = PC2)) +
        theme_bw() +
        coord_fixed(),
    df %>%
        ggplot() +
        geom_point(aes(TSNE1, TSNE2, col = cluster)) +
        geom_path(data = tsne_curve, aes(x = TSNE1, y = TSNE2)) +
        theme_bw() +
        coord_fixed()
)
```

Question

- Filter the testis dataset to only germinal cells.
- Re-infer lineages, using cluster annotations as information to build the MST. Note that you will first need to remove the 50th PCA dimension for `slingshot` to work (bug reported).
- What do you observe? Discuss.

Answer

```
germcells <- testis[, testis$annotation %in% c("Spermatogonia", "Spermatocytes", "Earl
reducedDim(germcells, 'corrected_2') <- reducedDim(germcells, 'corrected')[, 1:49]
germcells_slingshot <- slingshot::slingshot(
    germcells,
    reducedDim = 'corrected_2',
    clusterLabels = germcells$cluster
)
germcells$pseudotime <- slingshot::slingPseudotime(germcells_slingshot)[, 'Lineage1']

pca_curve <- slingCurves(germcells_slingshot, as.df = TRUE)
colnames(pca_curve) <- paste0('PC', 1:ncol(pca_curve))
tsne_curve <- slingshot::embedCurves(germcells_slingshot, 'TSNE', smoother = 'loess',
tsne_curve <- tsne_curve[order(tsne_curve$Order), ]
colnames(tsne_curve) <- paste0('TSNE', 1:ncol(tsne_curve))
df <- tibble(
    PC1 = reducedDim(germcells, 'corrected')[,1],
    PC2 = reducedDim(germcells, 'corrected')[,2],
    TSNE1 = reducedDim(germcells, 'TSNE')[,1],
    TSNE2 = reducedDim(germcells, 'TSNE')[,2],
    cluster = germcells$cluster,
    pseudotime = germcells$pseudotime
)
p <- cowplot::plot_grid(
    df %>%
        ggplot() +
        geom_point(aes(PC1, PC2, col = cluster)) +
        geom_path(data = pca_curve, aes(x = PC1, y = PC2)) +
        theme_bw() +
        coord_fixed(),
    df %>%
        ggplot() +
        geom_point(aes(TSNE1, TSNE2, col = cluster)) +
        geom_path(data = tsne_curve, aes(x = TSNE1, y = TSNE2)) +
        theme_bw() +
        coord_fixed(),
    df %>%
        ggplot() +
        geom_point(aes(PC1, PC2, col = pseudotime)) +
        geom_path(data = pca_curve, aes(x = PC1, y = PC2)) +
        theme_bw() +
        coord_fixed(),
    df %>%
        ggplot() +
        geom_point(aes(TSNE1, TSNE2, col = pseudotime)) +
        geom_path(data = tsne_curve, aes(x = TSNE1, y = TSNE2)) +
        theme_bw() +
        coord_fixed()
)
```

10.3.2 Pseudotime inference and expression modelling

The pseudotime is a metric describing the relative position of a cell in the trajectory, where cells with larger values are consider to be "after" their counterparts with smaller values. In trajectories describing time-dependent processes like differentiation, a cell's pseudotime value is generally used as a proxy for its relative age.

10.3.2.1 Pseudotime inference

Question

- Extract the pseudotime values automatically computed by `slingshot`.
- Check the distribution of pseudotime values across the different cell clusters. What do you observe? Where you expecting this?

Answer

```
p <- tibble(
    annotation = factor(germcells$annotation, c("Spermatogonia", "Spermatocytes", "Ear
    pseudotime = germcells$pseudotime
) %>%
    ggplot(aes(x = annotation, y = pseudotime, fill = annotation)) +
    geom_violin(scale = 'width') +
    geom_boxplot(outlier.shape = NULL, width = 0.1, fill = 'white') +
    theme_bw() +
    theme(axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1))
```

Question

- Correct pseudotime values as you would expect it to be.

Answer

```
germcells$pseudotime <- scales::rescale((-1 * slingshot::slingPseudotime(germcells_sli
```

10.4 3. Ordering trajectory with RNA velocity

As we saw earlier, TI does not necessarily know which direction is right for a given trajectory. This can be safely estimated using RNA velocity. For a given gene, a high ratio of unspliced to spliced transcripts indicates that that gene is being actively upregulated. Conversely, a low ratio indicates that the gene is being downregulated as the rate of production and processing of pre-mRNAs cannot compensate for the degradation of mature transcripts. Thus, we can infer that cells with high and low ratios are moving towards a high- and low-expression state, respectively, allowing us to assign directionality to trajectories or even individual cells.

Question

- Read `velociraptor` documentation. What do you need to compute RNA velocity scores in R?
- Import `spliced` and `unspliced` counts computed with `velocyto` in R.
- Try and compute RNA velocity (on germcells only). What do you see?

Answer

```
looms <- readRDS('data/Guo_testis/testis_velocity-counts.rds')
assays(looms)
rownames(looms) <- rowData(looms)$Gene
testis <- testis[rownames(looms), ]
assay(testis, 'spliced', withDimnames=FALSE) <- assay(looms, 'spliced')
assay(testis, 'unspliced', withDimnames=FALSE) <- assay(looms, 'unspliced')
germcells <- testis[, testis$annotation %in% c("Spermatogonia", "Spermatocytes", "Earl
velo_out <- velociraptor::scvelo(
    germcells,
    assay.X = "counts",
    use.dimred = "corrected",
    subset.row = scran::getTopHVGs(scran::modelGeneVar(germcells), prop = 0.1),
    mode = 'dynamical'
)
embedded_velo <- velociraptor::embedVelocity(reducedDim(germcells, "TSNE"), velo_out)
grid.df <- velociraptor::gridVectors(reducedDim(germcells, "TSNE"), embedded_velo, res
p <- scater::plotReducedDim(germcells, 'TSNE', colour_by = "annotation", point_alpha =
    geom_segment(
        data = grid.df,
        mapping = aes(x = start.1, y = start.2, xend = end.1, yend = end.2),
        arrow = arrow(length = unit(0.05, "inches"), type = "closed")
    )
```

A Useful resources

Analyzing NGS data can be a complex process, especially with the rise of multi-omics approaches.
Here is a list of resources we thought would be useful for people interested in going deeper in the analysis of NGS data.

A.1 General bioinformatics

- A comprehensive overview of the different types of bioinformatic analyses, divided in 4 fundamental modules: LINK

A.2 R/Bioconductor

- ***The*** excellent R guide for beginners, by Emmanuel Paradis: PDF
- ***The*** 150+ pages comprehensive book to learn everything about Bioconductor. This ebook has been published by Kasper D. Hansen and is freely available under the CC BY-NC-SA 4.0 license: PDF
- A digested PowerPoint summarizing two R/Bioconductor fundamental classes: `GRanges` and `*Experiment` classes

A.3 Scientific readings

- 2014 Nat. Methods paper from Bioconductor core team describing important object classes
- To see from how far Bioc comes from...