Single-cell RNAseq analysis with R/Bioconductor

Contents

Welcome

This is the landing page for the **"Single-cell RNA-seq analysis with R/Bioconductor"** workshop, ed. 2023.

Authors: Jacques Serizay [aut, cre], Orr Ashenberg [aut, cre] **Version:** 1.0.0 **Modified:** 2023-05-28 **Compiled:** 2023-06-05 **Environment:** R version 4.3.0 (2023-04-21), Bioconductor 3.17 **License:** MIT + file LICENSE

What

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 downstream of this.

Throughout the workshop, `bash` tools and `R/Bioconductor` packages will be used to analyse datasets and learn new approaches.

When

From June 5 to June 9, 2023.

Where

This course will be held online.

How

The course is structured in modules over five days. Each day will include formal lectures covering the key concepts required to understand scRNA-seq analysis. The remainder of each day will consist in practical hands-on sessions focusing on analysis of scRNA-seq data. These sessions will involve a combination of both mirroring exercises with the instructor to demonstrate a skill, as well as applying these skills on your own to complete individual exercises.
During and after each exercise, interpretation of results will be discussed as a group.

Who

The course will be mostly beneficial to those who have, or will shortly have, scRNA-seq data ready to analyse.

The material is suitable both for experimentalists who want to learn more about data-analysis as well as computational biologists who want to learn about scRNA-seq methods.

Examples demonstrated in this course can be applied to any experimental protocol or biological system.

The requirements for this course are:

1. Working knowledge of Unix / command line interface (managing files, running programs, reading manuals!). Basic bash commands (`cd`, `ls`, `...`) and CLI usage will *not* be covered in this course. We advice attendees to not register if they lack fundamental experience in CLI.
2. Programming experience in `R` (writing a function, basic I/O operations, variable types, using packages). `Bioconductor` experience is a plus.
3. Familiarity with next-generation sequencing data and its analyses (using alignment and quantification tools for bulk sequencing data)

Why

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

Throughout the course, we will also have a focus on reproducible research, documented content and interactive reports.

Instructors

Jacques Serizay

Orr Ashenberg

Program

Classes are from:

- 2 to 8 pm Paris time.
- 1 to 7 pm London time.
- 8 am to 2 pm NY time.
- 5 am to 11 am SF time.

Monday – Classes from 14:00 to 20:00 (Paris time)

Lecture 1 – Introduction to scRNA-Seq analysis [Orr]

- General introduction: cell atlas overviews
- Comparison of bulk and single cell RNA-Seq
- Overview of available scRNA-seq technologies (10x) and experimental protocols

Lecture 2 - From sequencing reads to expression matrices [Jacques]

- scRNA-Seq processing workflow starting with choice of sequencer (NextSeq, HiSeq, MiSeq) / barcode swapping and bcl files
- Overview of Popular tools and algorithms
- Common single-cell analyses and interpretation
- Sequencing data: alignment and quality control
- IGV: Looking at cool things in alignment like where reads are, mutations, splicing

Lab 1 – Familiarizing yourself with the course AWS instance [Jacques]

- Using RStudio
- Logging in AWS
- Shell and Unix commands to navigate directories, create folders, open files
- Raw file formats
- Get data from 10x website, single cell portal, from GEO (fastqs, counts)

Lab 2 – From sequencing reads to expression matrices [Orr]

- Mapping sequencing data with Cellranger
- Quality Control reports (CellRanger, dropEst, fastqc)

Tuesday – Classes from 14:00 to 20:00 (Paris time)

Lecture 3 - Quality control for scRNA-Seq data [Orr]

- What CellRanger does for quality filtering
- Normalisation methods
- Doublets, empty droplets, DropletUtils
- Barcode swapping
- Regression with technical covariates

Lab 3 - Introduction to R/Bioconductor [Jacques]

- Installing packages with CRAN and Bioconductor
- Data types, data manipulation, slicing
- I/O for scRNAseq analysis in R

Lab 4 – scRNA-Seq data wrangling [Orr]

- Data structure
- Data filtering
- Exploratory data analysis

Flash talks [Everybody]

Wednesday – Classes from 14:00 to 20:00 (Paris time)

Lecture 4 - Identifying cell populations [Jacques]

- Feature selection
- Dimensionality reduction
- Graph-based clustering and other cluster methods
- Assigning cluster identity
- Differential expression tests

Lab 5 – Identifying Cell Populations: dimensionality reduction, clustering and annotation [Jacques]

- Feature selection
- Dimensional reduction
- Graph-based clustering
- Marker gene detection
- Cell type annotation
- Data visualization

Lecture 5 - Data integration and batch effect correction [Orr]

- Batch correction methods (regress out batch, scaling within batch, Seurat v3, MNN, Liger, Harmony, scvi, scgen)
- Evaluation methods for batch correction (ARI, average silhouette width, kBET…)

Lab 6 - Data integration and batch effect correction [Orr]

- Comparison of batch correction methods
- Choosing the optimal batch correction approach

Thursday – Classes from 14:00 to 20:00 (Paris time)

Lecture 6 - Trajectories and pseudotimes [Jacques]

- Trajectory inference
- Popular tools and packages for trajectory analysis (https://github.com/dynverse/dynmethods#list-of-included-methods)
- Pseudotime inference
- RNA velocity
- Differential expression through pseudotime

Lab 7 - Inferring differentiation trajectories and pseudotime [Jacques]

- Infering trajectory in sperm cell lineage
- Orientating a trajectory with RNA veloctiy
- DE analysis along a trajectory

Lecture 7 - Advances in single-cell genomics: the epigenome [Orr]

Lab 8 - Single-cell ATAC-Seq analysis [Jacques]

Friday – Classes from 14:00 to 20:00 (Paris time)

Lecture 8 - Advances in single-cell genomics: spatial transcriptomics [Orr]

Friday will then be divided in two parts:

- Morning & afternoon (1h + 1h30): Group projects: analysing scRNA-seq data by yourself, from A to Z
- Afternoon (1h): Group presentations (10' each group, max 5 slides: what/why/where/when/hov conclusions)

Happy hour time!!

RStudio

Rstudio offers a graphical interface to facilitate the interaction between a user and an underlying programming language (this is sometimes called IDE, or integrated development environment). It can be very useful when a user is not necessarily proficent with command line-based computing. However, such graphical interfaces are not always able to connect to services such as AWS.

Since most of the preliminary analysis we do is on AWS, we'd like to be able to use Rtudio directly from there. To do that, simply go to the following address:

`https://34.216.135.45:8787`

Don't forget, you can run a `bash` terminal from within RStudio! This may come handy if you want to process some data with `cellranger`, for example. To do this, simply click on the `terminal` button next on the bottom left panel. You should now be in your own `${home}` directory (`~`).

Prerequisites

The course is intended for those who have basic familiarity with Unix and the R scripting language.

We will also assume that you are familiar with mapping and analysing bulk RNA-seq data as well as with the commonly available computational tools.

- If a refresher is needed for Unix command line (hopefully not), please go over this tutorial and its companion cheatsheet.
- Getting down to basics: an introduction to the fundamentals of R (courtesy of Mark Ravinet).
- Gentle introduction to `R/Biocondutor`: here
- For a full in-depth guide of `Bioconductor` ecosystem: read the comprehensive `R/Bioconductor` book from Kasper D. Hansen available under the CC BY-NC-SA 4.0 license [PDF]

Local configuration

- Ideally (though not strictly required), a configured SSH client (it should be already installed on Linux/Mac machines, `PuTTY` can be set up for Windows).
- Ideally (though not strictly required), a SSH ftp client (`Forklift` is excellent for Mac, although not free beyond the trial version; `cyberduck` can be used for Windows; `FileZilla` can be used for both Mac, Windows and Linux).
- Computer with high-speed internet access (no specific configuration required - everything will be performed on a remote AWS machine).
- Zoom visioconference software

Remote configuration

The AWS machine is running with `Ubuntu` and has been set up as follows:

```
## --- Clean up previous R installs
sudo apt purge r-base* r-recommended r-cran-*
sudo apt autoremove
```

```
sudo apt update
sudo apt upgrade

## --- Libraries
sudo apt update
sudo apt install libc6 libicu60 -y
sudo apt install -y \
    gcc g++ perl python3 python3-pip python-dev \
    automake make cmake less vim nano fort77 \
    wget git curl bsdtar bzip2 gfortran unzip ftp \
    libpng-dev libjpeg-dev \
    texlive-latex-base default-jre build-essential \
    libbz2-dev liblzma-dev libtool \
    libxml2 libxml2-dev zlib1g-dev \
    libdb-dev libglu1-mesa-dev zlib1g-dev  \
    libncurses5-dev libghc-zlib-dev libncurses-dev \
    libpcre3-dev libxml2-dev \
    libblas-dev libzmq3-dev libreadline-dev libssl-dev \
    libcurl4-openssl-dev libx11-dev libxt-dev \
    x11-common libcairo2-dev \
    libreadline6-dev libgsl0-dev \
    libeigen3-dev libboost-all-dev \
    libgtk2.0-dev xvfb xauth xfonts-base \
    apt-transport-https libhdf5-serial-dev \
    libudunits2-dev libgdal-dev libgeos-dev libproj-dev \
    libv8-dev \
    libmagick++-dev \
    libharfbuzz-dev libfribidi-dev \
    fftw3

## --- R base install
sudo add-apt-repository 'deb https://cloud.r-project.org/bin/linux/ubuntu bionic-
add-apt-repository "deb https://cloud.r-project.org/bin/linux/ubuntu `lsb_release
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys E298A3A825C0D65DFD5
sudo apt update
sudo apt install r-base r-recommended r-base-core r-base-dev
```

The following packages have been installed (along with their many dependencies, of course!):

```
## --- Install important R packages for single-cell RNA-seq projects
## pak
```

```
sudo Rscript -e 'install.packages("pak", repos = sprintf("https://r-lib.github.io/p/pak/sta

## CRAN packages
sudo Rscript -e 'pak::pkg_install(c("tidyverse", "devtools", "umap", "corrplot", "gam", "gg

## Bioconductor Packages
sudo Rscript -e 'pak::pkg_install(c("SingleCellExperiment", "scran", "scater", "batchelor",

## --- Install other softwares (fastQC, samtools, cellranger and cellranger indexes, ffq)

# fastqc samtools
sudo apt install fastqc samtools python3.8

# cellranger
cd /opt/
sudo wget -O cellranger-7.1.0.tar.gz "https://cf.10xgenomics.com/releases/cell-exp/cellrang
sudo tar -xzvf cellranger-7.1.0.tar.gz
sudo ln -s /opt/cellranger-7.1.0/cellranger /usr/local/bin/cellranger
sudo wget https://cf.10xgenomics.com/supp/cell-exp/refdata-gex-mm10-2020-A.tar.gz
sudo tar -xzvf refdata-gex-mm10-2020-A.tar.gz

# ffq
sudo apt install python3-distutils
sudo pip install ffq

# seqkit
cd /opt/
sudo wget -O seqkit_linux_amd64.tar.gz https://github.com/shenwei356/seqkit/releases/downlo
sudo tar -xzvf seqkit_linux_amd64.tar.gz
sudo ln -s /opt/seqkit /usr/local/bin/seqkit

# bcl2fastq
sudo apt install alien
cd /opt/
wget http://support.illumina.com/content/dam/illumina-support/documents/downloads/software/
sudo alien bcl2fastq2-v2.17.1.14-Linux-x86_64.rpm
sudo dpkg -i bcl2fastq2_0v2.17.1.14-2_amd64.deb
```

Part I.

Day 1

1. Lecture 1 - Introduction to scRNAseq analysis

Slides here

2. Lab 1: Familiarizing yourself with the course AWS instance

2.1. Connect to RStudio Server

Most of single-cell RNA-seq analysis takes place either in `python` or in `R`. Here, we focus on how to leverage `R` to investigate scRNAseq data. `RStudio` is an `IDE` (Integrated Development Environment, in other words: a nice graphical interface to run `R`-related commands).

For this workshop, we have installed `R` and `RStudio` on AWS. We can directly use `RStudio` (actually, `RStudio-server` since it is installed on an AWS remote server). Simply open a browser and copy-paste the following address:

```
https://34.216.135.45:8787
```

An `RStudio` log in page will appear; to log in, use your `user` ID for both ID and password.

Notice how when you log in `Rstudio`, there are multiple panels. Familiarize yourself with the different panels.

The interactive `R` console is generally found in the bottom left corner of `RStudio`(though it may be in another corner sometimes). All the rest (history panel, environment panel, directory explorer panel, editor panel) are extra features provided by `RStudio`.

> Some useful commands in `R`:
>
> Within the `R` console, you can safely use `R`-dedicated commands. Do you know the most common ones? The semantics are a different from the `terminal` commands you may be used to...
> **R**

```
getwd() # equivalent of `pwd` in terminal
dir.create('~/data/') # equivalent of `mkdir ~/data/` in terminal
setwd("~/data/") # equivalent of `cd ~/data/` in terminal
list.files("~/data/") # equivalent of `ls` in terminal
download.file("...") # equivalent of `wget ...` in terminal
```

2.2. Use a AWS terminal within RStudio

A general issue with bioinformatic analyses stems from the fact that nobody works in the same environment:

- Are you working on Mac? Linux? Windows?
- Do you have a lot of computational power? Perhaps a GPU card?
- Are you connected to the Internet? With a fast connection? Are you working behind a proxy?

To ensure that we are all working in the same environment, we rely on AWS (Amazon Web Services) EC2 (Elastic Cloud 2) instances. EC2 instances are "virtual" computers to which you can connect remotely, from a local computer.

The instance is common for everybody. We are thus all sharing the same "computer"; this means:

- **Shared resources**
- **Same access to shared files**
- **Same access to system-wide softwares and conda environments**

The easiest way for us to launch **`bash` commands from a terminal** in AWS is to do it through `RStudio`: You can open up a `terminal` directly from within `RStudio` as follow: go to `Tools` > `Terminal` > `New terminal`. This should open up a new tab in the bottom left corner (next to the `R` console).

> ⚠ `R console` versus `terminal`:
>
> From here onwards, be sure you completely understand the difference between **"`R console`"** and **"`terminal (or shell)`"**. They are entirely different things, and can be both accessed within `RStudio`. It is crucial you understand the difference between the two to not get confused for the rest of the course.

2.3. Basic terminal commands

The same `bash` commands are available in AWS `terminal`, regardless of whether you access the terminal from `RStudio` or through `ssh`.

One can list files, download files, check help pages, ..., just like in `R`.

- Check the your present directory

bash

```
pwd
```

- Check history

bash

```
history
```

- put history into a `history.txt` file

bash

```
history > history.txt
```

- make a new folder called data

bash

```
mkdir data
```

- Go to the new `data` directory

bash

```
cd data
```

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

bash

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

- check manual page of `wget` command (hit `q` to exit)

bash

```
man wget
```

- check specific help for `cellranger` command and subcommands

bash

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

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

bash

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

- Download a file from Internet with `wget`

bash

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

- List all files in a folder

bash

```
ls -l ~/Share/
```

Tip

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

```
git clone https://github.com/js2264/scRNAseq_Physalia_2023.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), 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).

2.4. Single-cell RNA-seq datasets

> "This is a course about single-cell RNA-seq analysis, after all, 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.

2.4.1. 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.

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

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

- **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`...).

- **Try to install the `ffq` tool from the Patcher lab.**

> Installing `conda` packages
>
> `conda`-based environments allow easy installs of packages such as `ffq`. Your `(base)` conda environment should be active by default, and you will only have to type:
>
> ```
> pip install ffq
> ```

- **Check `ffq` help and try fetching metadata for the GSE ID `GSE148963`.**

bash

```
ffq --help
ffq -t GSE GSE148963 > GSE148963_search.txt
head -n 30 GSE148963_search.txt
```

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

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

bash

```
grep 'ftp://' GSE148963_search.txt
```

And with a bit of `sed` magick...

bash

```
grep 'ftp://' GSE148963_search.txt | sed 's,.*ftp:,ftp:,' | sed 's,".*,,' | grep
## `ffq` looks through GEO repository to fing metadata associated with the `GSE14
## grep 'ftp://' recovers the text lines that contain downloading links
## the `sed` commands clean up the text lines
## the `wget` command downloads locally the links listed in the generated file (G
# wget -i GSE148963_fastqlist.txt ## Do not run, it would take too long...
```

- **Check the content of the reads**

A subset of the reads has been downloaded and put in the `~/Share/` folder. Have a look at it!

bash

```
zcat ~/Share/SRR11575369_1.fastq.gz | head -n 12
```

Try and understand the structure of the `fastq.gz` file. What is the meaning of each line?

How many reads are there in the `fastq.gz` file? And how long are they? Can you get a summary of what is in this file? All of these questions can be quickly answered using the `seqkit` tool:

bash

```
seqkit --help
seqkit stats --help
seqkit stat ~/Share/SRR11575369_1.fq.gz
```

2.4.2. 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. GEO also hosts processed files.

- **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.**

2.5. Bonus

For those of you who are already familiar with the basics, you can fast-forward through this lab and start working on scRNAseq data directly. The script in `bin/prepare_Ernst.R` is a template to process a publicly available scRNAseq dataset. You can start exploring it to see if you understsand the different chunks of code and their importance. All the content from this template will eventually be covered in the next labs.

3. Lecture 2 - From sequencing reads to expression matrices

Slides here

4. Lab 2: From .bcl to count matrix

4.1. Demultiplexing sequencing data with `cellranger mkfastq`

Navigate to your terminal in `RStudio` on AWS.

Go to the cellranger mkfastq page and read the **Overview**.

Go to the Terminal tab in your `RStudio` and take a look at the 10x samplesheet file:

bash

```
cat ~/Share/data_wrangling/cellranger-tiny-bcl-simple-1.2.0.csv
```

Next, explore the contents of the sequencing directory:

bash

```
ls -l Share/data_wrangling/cellranger-tiny-bcl-1.2.0
```

Now we can demultiplex our bcl files by running the following command in the terminal:

bash

```
cellranger mkfastq --id tiny-bcl --run ~/Share/data_wrangling/cellranger-tiny-bcl-1.2.0 --c
```

The output folders can be viewed by running the `ls` command:

bash

```
ls -l tiny-bcl/outs/fastq_path/H35KCBCXY/test_sample
```

Look at the index read (`I1`), read 1 (`R1`), and read (`R2`) files using the command `less fastq_file_name.gz`. You can type `q` in the terminal to leave this view.

Open the html file `tiny-bcl/outs/fastq_path/Reports/html/index.html` by navigating to the file in RStudio, using the Files Tab. When you click on the file, select the option to View in Web Browser. Take some time to explore the demultiplexed outputs.

4.2. Generating gene count matrices with `cellranger count`

Go to the cellranger count algorithm overview and read the section on **Alignment** (Read Trimming, Genome Alignment, MAPQ adjustment, Transcriptome Alignment, UMI Counting).

In the terminal run the command:

bash

```
cellranger count --id counts --transcriptome ~/Share/refdata-gex-mm10-2020-A/ --f
```

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

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

bash

```
ls counts/outs
```

Can you locate the feature-barcode matrices? What is the difference between the `raw_feature_bc_matrix` and `filtered_feature_bc_matrix` data types?

Now open the html file `counts/outs/web_summary.html` by navigating to the file in RStudio, using the Files Tab. When you click on the file, select the option to View in Web Browser. Take some time to explore the gene expression matrix outputs.

Part II.

Day 2

5. Lab 3: Introduction to R/Bioconductor

5.1. Installing packages in `R`

> "Hey, I've heard so many good things about this piece of software, it's called 'slingshot'? Heard of it? I really want to try it out on my dataset!!"

Or, 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 that survived until 2023").
- `Bioconductor`, another package provider, with a primary focus on genomic-related packages: Bioconductor.
- Other unofficial sources, such as `GitHub`.

Let's start by going over package installation.

- **Install `mgcv`, `HCAData` and `revelio` packages**

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

R

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

- **Check package help pages**

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/

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?

5.2. 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.

5.2.1. Important R concepts:

5.2.1.1. 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
```

- **Import a text file into tibbles**

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

R

```
genes <- read_tsv('~/Share/GSM4486714_AXH009_genes.tsv', col_names = c('ID', 'Sym
genes
```

5.2.1.2. 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(genes, 1:4)
filter(genes, Symbol == 'CCDC67')
filter(genes, grepl('^CCDC.*', Symbol))
filter(genes, grepl('^CCDC.*', Symbol), grepl('.*5$', Symbol))
select(genes, 1)
select(genes, ID)
select(genes, matches('Sym.*'))
```

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

R

```
mutate(genes, chr = sample(1:22, n(), replace = TRUE))
```

5.2.1.3. |> pipe:

Actions on tibbles can be piped as a chain with |>, 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

```
read_tsv('~/Share/GSM4486714_AXH009_genes.tsv', col_names = c('ID', 'Symbol')) |>
    mutate(chr = sample(1:22, n(), replace = TRUE)) |>
    filter(chr == 2, grepl('^CCDC.*', Symbol)) |>
    select(ID) |>
    slice_head(n = 3)
```

5.2.2. Important Bioconductor concepts:

5.2.2.1. 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 that most of the underlying data (organized in precisely structured `"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 dimensions. Since the objects are `matrix`-like, `dim()`, `dimnames()`, and 2-dimensional `[`, `[<-` methods are available.
- `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`…)

R

```
library(SummarizedExperiment)
library(airway)
data(airway)
airway
```

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

R

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

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

R

```
untreated_LRG <- airway[grepl('^LRG_', rownames(airway)), airway$dex == 'untrt']
untreated_LRG
```

5.2.2.2. 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()`.

R

```
library(GenomicRanges)
gr <- read_tsv('~/Share/GSM4486714_AXH009_genes.tsv', col_names = c('ID', 'Symbol')) |>
    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.

R

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

- **Can you find a way to easily read common input files such as `bed` files into `GRanges`?**

R

```
library(rtracklayer)
genes2 <- import('~/Share/GRCm39_genes.bed')
genes2
```

- **How would you have proceeded without `rtracklayer`? Check the `start` coordinates: what do you see? Comment on the interest of using `Bioconductor`.**

R

```
library(rtracklayer)
genes2_manual <- read_tsv('~/Share/GRCm39_genes.bed', col_names = FALSE) |>
    drop_na() |>
    purrr::set_names(c('chr', 'start', 'stop', 'id', 'score', 'strand')) |>
    makeGRangesFromDataFrame(keep.extra.columns = TRUE)
genes2_manual
head(start(genes2))
head(start(genes2_manual))
```

5.3. CRAN & Bioconductor approaches to scRNAseq

5.3.1. 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.

R

```
source('~/Share/bin/prepare_Nestorowa.R') # Adapted from Nestorowa et al., Blood 2016 (doi:
sce
class(sce)
```

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

R

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

Quantitative metrics for scRNAseq studies can also be stored in `assays`:

R

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

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

R

```
colData(sce)
lapply(colData(sce), class)
head(colData(sce)[[1]])
head(colData(sce)[['FACS']])
head(sce$sizeFactor)
```

- **Are there any reduced dimensionality representation of the data stored in the `sce` object? How can we run a `PCA` using normalized counts?**

R

```
reducedDims(sce)
pca <- prcomp(t(logcounts(sce)))
names(pca)
dim(pca$x)
head(pca$x[, 1:50])
```

- **Now, let's compute a `UMAP` embedding from this `PCA` and compare it to the `PCA` embedding.**

R

```
umap <- uwot::umap(pca$x)
colnames(umap) <- c('UMAP1', 'UMAP2')
plot(pca$x[,1], pca$x[,2])
plot(umap[,1], umap[,2])
```

We will see more advanced ways of reducing scRNAseq data dimensionality in the coming lectures and labs.

5.3.2. scRNAseq in R

`Seurat` is another very popular ecosystem to investigate scRNAseq data. It is primarily developed and maintained by the Sajita Lab. It originally begun as a single package aiming at encompassing "all" (most) aspects of scRNAseq analysis. However, it rapidly evolved in a much larger project, and now operates along with other "wrappers" and extensions. It also has a very extended support from the lab group. All in all, is provides a (somewhat) simple workflow to start investigating scRNAseq data.

It is important to chose one standard that you feel comfortable with yourself. Which standard provides the most intuitive approach for you? Do you prefer an "all-in-one, plug-n-play" workflow (`Seurat`-style), or a modular approach (`Bioconductor`-style)? Which documentation is easier to read for you, a central full-featured website with extensive examples (`Seurat`-style), or "programmatic"-style vignettes (`Bioconductor`-style)?

This course will mostly rely on `Bioconductor`-based methods, but sometimes use `Seurat`-based methods.s In the absence of coordination of data structures, the next best solution is to write functions to coerce an object from a certain class to another class (i.e. `Seurat` to `SingleCellExperiment`, or vice-versa). Luckily, this is quite straightforward in `R` for these 2 data classes:

R

```
sce_seurat <- Seurat::as.Seurat(sce)
str(sce)
sce
sce_seurat
sce2 <- Seurat::as.SingleCellExperiment(sce_seurat)
```

- **Do you see any change between `sce` and the corresponding, "back-converted", `sce2` objects? Explain these differences.**

R

```
sce
sce2
#
colData(sce)
colData(sce2)
```

- **Try and access the underlying raw or normalized data from the `sce_seurat` object. How does it compare to data access from an `SingleCellExperiment` object?**

R

```
colnames(sce_seurat)
ncol(sce_seurat)
nrow(sce_seurat)
# cells and features access
head(Seurat::Cells(sce_seurat))
head(rownames(sce_seurat))
# cell data access
head(sce_seurat[[]])
head(sce_seurat$label)
# Counts access
Seurat::GetAssayData(object = sce_seurat, slot = "counts")[1:10, 1:10]
Seurat::GetAssayData(object = sce_seurat, slot = "data")[1:10, 1:10]
# Embeddings
head(Seurat::Embeddings(object = sce_seurat, reduction = "diffusion"))
```

5.4. Reading scRNAseq data

- **Try to load the raw `10X` single-cell RNA-seq data downloaded yesterday (from `Lier et al.`) into a `SingleCellExperiment` object using**

`DropletUtils` package.

R

```
library(SingleCellExperiment)
sce <- DropletUtils::read10xCounts('~/Share/data_wrangling/counts/outs/filtered_f
sce
colData(sce)
rowData(sce)
```

Public single-cell RNA-seq data can be retrieved from within R directly, thanks to several data packages, for instance `scRNAseq` or `HCAData`.

- **Check out the `He et al., Genome Biol. 2020` paper. Can you find a way to load the scRNAseq data from this paper without having to leave the `R` console?**

R

```
organs <- scRNAseq::HeOrganAtlasData(ensembl = TRUE)
organs
```

The interest of this approach is that one can recover a full-fledged **SingleCellExperiment** (often) provided by the authors of the corresponding study. This means that lots of information, such as batch ID, clustering, cell annotation, etc., may be readily available.

- **Check the data available for cells/features in the dataset from `He et al.`.**

R

```
colData(organs)
table(organs$Tissue)
table(organs$reclustered.fine, organs$Tissue)
```

5.5. Bonus

To compare the two different approaches, try preparing both a `SingleCellExperiment` or a `Seurat` object from scratch, using the matrix files generated in the previous lab. Read the documentation of the two related packages to understand how to do this. This will be extensively covered in the next lab for everybody.

5.6. Session info

6. Lecture 3 - Quality control for scRNA-Seq data

Slides here

7. Lab 4 - Single-cell RNA-seq data wrangling

Aims

- To give you experience with the analysis of single cell RNA sequencing (scRNA-seq) including performing quality control and identifying cell type subsets.
- To introduce you to scRNA-seq analysis using Bioconductor packages.

7.1. Introduction

Data produced in a single cell RNA-seq experiment has several interesting characteristics that make it distinct from data produced in a bulk population RNA-seq experiment. Two characteristics that are important to keep in mind when working with scRNA-Seq are drop-out (the excessive amount of zeros due to limiting mRNA) and the potential for quality control (QC) metrics to be confounded with biology. This combined with the ability to measure heterogeneity from cells in samples has shifted the field away from the typical analysis in population-based RNA-Seq. Here we demonstrate some approaches to quality control, followed by identifying and analyzing cell subsets.

7.1.1. Load necessary packages

When loading libraries, we are asking R to load code for us written by someone else. It is a convenient way to leverage and reproduce methodology developed by others.

R

```
library(tidyverse)
library(SingleCellExperiment)
library(scater)
library(scran)
```

7.1.2. Read in Pancreas counts matrix.

For this tutorial, we will be analyzing a human pancreas scRNAseq dataset. It is freely available from GEO: link. We start by downloading the cell, features and counts matrix.

- **Get the downloadable links for each file**

R

```
download.file('https://ftp.ncbi.nlm.nih.gov/geo/series/GSE114nnn/GSE114802/suppl/
download.file('https://ftp.ncbi.nlm.nih.gov/geo/series/GSE114nnn/GSE114802/suppl/
download.file('https://ftp.ncbi.nlm.nih.gov/geo/series/GSE114nnn/GSE114802/suppl/
```

- **Import each table in `R`**

R

```
cells <- read_tsv('~/Share/GSE114802_org4_barcodes.tsv.gz', col_names = FALSE)
genes <- read_tsv('~/Share/GSE114802_org4_genes.tsv.gz', col_names = FALSE)
counts <- read_csv('~/Share/GSE114802_org4_counts.csv.gz', col_names = TRUE)
counts <- counts[, -1]
counts <- as(counts, 'matrix')
counts <- as(counts, 'dgCMatrix')
rownames(counts) <- genes$X1
```

- **Transform into a `SingleCellExperiment` object**

R

```
sce <- SingleCellExperiment(
    colData = cells,
    rowData = genes,
    assays = list('counts' = counts)
)
```

- **Examine the `SingleCellExperiment` object you've just created. Get an idea of the size of the dataset, the different data available, etc.**

R

```
colData(sce)
rowData(sce)
metadata(sce)
```

```
dim(sce)
assays(sce)
counts(sce)[1:10, 1:10]
reducedDims(sce)
```

- **How much memory does a sparse matrix take up relative to a dense matrix? (use `object.size()` to get the size of an object...)**

R

```
counts <- counts(sce)
object.size(counts) # size in bytes
object.size(as.matrix(counts)) # size in bytes
```

- **Compare it to the sparsity of the counts (the % of the counts equal to 0)**

R

```
sum(counts > 0) / (nrow(sce)*ncol(sce))
```

7.2. Basic QCs

You can learn a lot about your scRNA-seq data's quality with simple plotting.
Let's do some plotting to look at the number of reads per cell, reads per genes, expressed genes per cell (often called complexity), and rarity of genes (cells expressing genes).

- **Look at the summary counts for genes and cells**

R

```
counts_per_cell <- Matrix::colSums(counts)
counts_per_gene <- Matrix::rowSums(counts)
genes_per_cell <- Matrix::colSums(counts > 0) # count gene only if it has non-zero reads ma
hist(log10(counts_per_cell+1), main = '# of counts per cell', col = 'wheat')
hist(log10(genes_per_cell+1), main = '# of expressed genes per cell', col = 'wheat')
plot(counts_per_cell, genes_per_cell, log = 'xy', col = 'wheat')
title('Counts vs genes per cell')
```

- **Can you plot a histogram of counts per gene in log10 scale?**

R

```r
cells_per_gene <- Matrix::rowSums(counts > 0) # only count cells where the gene i
hist(log10(cells_per_gene+1), main = '# of cells expressing each gene', col = 'wh
```

- **Plot cells ranked by their number of detected genes**

To do that, first sort cells by their library complexity, ie the number of genes detected per cell.
This is a very useful plot as it shows the distribution of library complexity in the sequencing run.
One can use this plot to investigate observations (potential cells) that are actually failed libraries (lower end outliers) or observations that are cell doublets (higher end outliers).

R

```r
plot(sort(genes_per_cell), xlab = 'cell', log = 'y', main = '# of genes per cell
```

- Several QCs can be automatically computed using `quickPerCellQC()`. Try it out and check the results.
- **What are the `total` and `detected` columns?**

R

```r
sce <- scuttle::quickPerCellQC(sce)
colData(sce)
```

7.3. Access to stored informations

7.3.1. Assay slots

For typical scRNA-seq experiments, a `SingleCellExperiment` can have multiple assays, corresponding to different metrics. The most basic one is `counts`.
Different assays store different 'transformations' of the `counts`(e.g. 'logcounts).

- **Try to manually compute logcounts from counts and store it in a new slot**

R

```r
assay(sce, 'logcounts') <- log10(counts(sce) + 1)
```

7.3.2. Embeddings

Embeddings allow for a representation of large-scale data (N cells x M genes) into smaller dimensions (e.g. 2-50 dimensions). Typical embeddings can be PCA, t-SNE, UMAP, etc… Many embeddings can be computed using `run...()` functions from Bioconductor packages (e.g. `scran`, `scater`, …).

- **Compute PCA embedding of the dataset using `runPCA()` from `scater` package**

R

```
sce <- scater::runPCA(sce)
plotReducedDim(sce, "PCA")
```

- **Compute t-SNE embedding of the dataset using `runTSNE()` from `scater` package**

R

```
sce <- scater::runTSNE(sce)
plotReducedDim(sce, "TSNE")
```

- **Compute UMAP embedding of the dataset using `runUMAP()` from `scater` package**

R

```
sce <- scater::runUMAP(sce)
plotReducedDim(sce, "UMAP", colour_by = 'sum')
plotReducedDim(sce, "UMAP", colour_by = 'detected')
```

7.3.3. Multiple modalities

Alternative 'modalities' can be stored in the same `SingleCellExperiment` object (e.g. if you perform paired single-cell RNA-seq and ATAC-seq). This is done through `altExps` which can store summarized experiments.

- **Try to add an altExp (using `altExp<-` function)**

R

```
altExp(sce, "ATAC_counts") <- SummarizedExperiment(matrix(rpois(1000, 5), ncol =
swapAltExp(sce, "ATAC_counts", saved = "RNA_counts")
```

Note that features can be different between different `altExps`.

7.4. Filtering cells and features

7.4.1. Pre-filtering

- **Filter the SCE to only include (1) cells that have a complexity of 2000 genes or more and (2) genes that are are expressed in 10 or more cells.**

R

```
sce_filtered <- sce[
    Matrix::rowSums(counts(sce) > 0) > 10,
    Matrix::colSums(counts(sce) > 0) > 2000
 ]
```

Almost all our analysis will be on this single object, of class SingleCellExperiment. This object contains various "slots" that will store not only the raw count data, but also the results from various computations below. This has the advantage that we do not need to keep track of inidividual variables of interest - they can all be collapsed into a single object as long as these slots are pre-defined.

7.4.2. Filtering low-quality cells: mitochondrial counts

For each cell, we can calculate the percentage of counts mapping on mitochondrial genes and store it in a column `percent_mito` in our `colData()`.

- **Find mitochondrial genes, compute the % of total counts associated with these genes, and store it in `colData`**

R

```
rowData(sce_filtered)
mito_genes <- rownames(sce_filtered)[grep(pattern = "^MT-", x = rowData(sce_filte
mito_genes_counts <- counts(sce_filtered)[mito_genes, ]
percent_mito <- colSums(mito_genes_counts) / sce_filtered$total
hist(percent_mito*100, main = '% of total counts over mitochondrial genes', col =
colData(sce_filtered)$percent_mito <- percent_mito
```

- **Remove cells with a % of mitochondrial counts greater than 10%.**

R

```
sce_filtered <- sce_filtered[
    ,
    sce_filtered$percent_mito <= 0.10
]
```

7.4.3. Checking housekeeping genes

Another metric we use is the number of house keeping genes expressed in a cell. These genes reflect commomn processes active in a cell and hence are a good global quality measure. They are also abundant and are usually steadliy expressed in cells, thus less sensitive to the high dropout.

R

```
# Load the list of housekeeping genes
hkgenes <- read.table("Share/tirosh_house_keeping.txt", skip = 2)
hkgenes <- as.vector(hkgenes$V1)
hkgenes <- rownames(sce_filtered)[match(hkgenes, rowData(sce_filtered)$X2)]
hkgenes <- hkgenes[!is.na(hkgenes)]
```

- **Compute the number of detected HK genes for each cell and store it in `colData`**

R

```
colData(sce_filtered)$n_expressed_hkgenes <- Matrix::colSums(counts(sce_filtered)[hkgenes,
```

- **Plot (in a boxplot) the relationship between the # of detected housekeeping genes and the total UMI count (or # of detected genes) per cell. Comment**

R

```
colData(sce_filtered)$n_expressed_hkgenes <- Matrix::colSums(counts(sce_filtered)[hkgenes,
boxplot(colData(sce_filtered)$total ~ colData(sce_filtered)$n_expressed_hkgenes)
boxplot(colData(sce_filtered)$detected ~ colData(sce_filtered)$n_expressed_hkgenes)
```

- **Remove cells with a # of expressed housekeeping genes greater than 85**

R

```
sce_filtered <- sce_filtered[, sce_filtered$n_expressed_hkgenes <= 85]
```

7.4.4. Checking gene set expression

Sometimes we want to ask what is the expression of a gene / a set of a genes across cells. This set of genes may make up a gene expression program we are interested in. Another benefit at looking at gene sets is it reduces the effects of drop outs.

Let's look at genes involved in the stress signature upon cell dissociation. We calculate these genes average expression levels on the single cell level.

R

```
genes_dissoc <- c("ATF3", "BTG2", "CEBPB", "CEBPD", "CXCL3", "CXCL2", "CXCL1", "D
genes_dissoc <- rownames(sce_filtered)[match(genes_dissoc, rowData(sce_filtered)$
genes_dissoc <- unique(genes_dissoc[!is.na(genes_dissoc)])
```

- **Calculate the average gene set expression for each cell**

R

```
ave_expr_genes_dissoc <- colMeans(logcounts(sce_filtered[genes_dissoc, ]))
colData(sce_filtered)$ave_expr_genes_dissoc <- ave_expr_genes_dissoc
```

- **Plot an embedding of the dataset, using a color scale representing the average expression of genes involved in the stress signature upon cell dissociation. Comment.**

R

```
plotReducedDim(sce_filtered, dimred = 'PCA', colour_by = 'ave_expr_genes_dissoc')
```

7.5. Session info

```
- Session info -------------------------------------------------------------
 setting  value
 version  R version 4.3.0 (2023-04-21)
 os       macOS Monterey 12.5.1
 system   aarch64, darwin20
 ui       X11
```

```
 language (EN)
 collate  en_GB.UTF-8
 ctype    en_GB.UTF-8
 tz       Europe/Paris
 date     2023-06-05
 pandoc   2.19.2 @ /opt/homebrew/bin/ (via rmarkdown)

- Packages -------------------------------------------------------------------------------
 package     * version date (UTC) lib source
 cachem        1.0.7   2023-02-24 [1] CRAN (R 4.3.0)
 callr         3.7.3   2022-11-02 [1] CRAN (R 4.3.0)
 cli           3.6.0   2023-01-09 [1] CRAN (R 4.3.0)
 crayon        1.5.2   2022-09-29 [1] CRAN (R 4.3.0)
 devtools      2.4.5   2022-10-11 [1] CRAN (R 4.3.0)
 digest        0.6.31  2022-12-11 [1] CRAN (R 4.3.0)
 ellipsis      0.3.2   2021-04-29 [1] CRAN (R 4.3.0)
 evaluate      0.20    2023-01-17 [1] CRAN (R 4.3.0)
 fastmap       1.1.1   2023-02-24 [1] CRAN (R 4.3.0)
 fs            1.6.1   2023-02-06 [1] CRAN (R 4.3.0)
 glue          1.6.2   2022-02-24 [1] CRAN (R 4.3.0)
 htmltools     0.5.4   2022-12-07 [1] CRAN (R 4.3.0)
 htmlwidgets   1.6.1   2023-01-07 [1] CRAN (R 4.3.0)
 httpuv        1.6.9   2023-02-14 [1] CRAN (R 4.3.0)
 jsonlite      1.8.4   2022-12-06 [1] CRAN (R 4.3.0)
 knitr         1.42    2023-01-25 [1] CRAN (R 4.3.0)
 later         1.3.0   2021-08-18 [1] CRAN (R 4.3.0)
 lifecycle     1.0.3   2022-10-07 [1] CRAN (R 4.3.0)
 magrittr      2.0.3   2022-03-30 [1] CRAN (R 4.3.0)
 memoise       2.0.1   2021-11-26 [1] CRAN (R 4.3.0)
 mime          0.12    2021-09-28 [1] CRAN (R 4.3.0)
 miniUI        0.1.1.1 2018-05-18 [1] CRAN (R 4.3.0)
 pkgbuild      1.4.0   2022-11-27 [1] CRAN (R 4.3.0)
 pkgload       1.3.2   2022-11-16 [1] CRAN (R 4.3.0)
 prettyunits   1.1.1   2020-01-24 [1] CRAN (R 4.3.0)
 processx      3.8.0   2022-10-26 [1] CRAN (R 4.3.0)
 profvis       0.3.7   2020-11-02 [1] CRAN (R 4.3.0)
 promises      1.2.0.1 2021-02-11 [1] CRAN (R 4.3.0)
 ps            1.7.2   2022-10-26 [1] CRAN (R 4.3.0)
 purrr         1.0.1   2023-01-10 [1] CRAN (R 4.3.0)
 R6            2.5.1   2021-08-19 [1] CRAN (R 4.3.0)
 Rcpp          1.0.10  2023-01-22 [1] CRAN (R 4.3.0)
 remotes       2.4.2   2021-11-30 [1] CRAN (R 4.3.0)
 rlang         1.0.6   2022-09-24 [1] CRAN (R 4.3.0)
 rmarkdown     2.20    2023-01-19 [1] CRAN (R 4.3.0)
```

```
 sessioninfo   1.2.2   2021-12-06 [1] CRAN (R 4.3.0)
 shiny         1.7.4   2022-12-15 [1] CRAN (R 4.3.0)
 stringi       1.7.12  2023-01-11 [1] CRAN (R 4.3.0)
 stringr       1.5.0   2022-12-02 [1] CRAN (R 4.3.0)
 urlchecker    1.0.1   2021-11-30 [1] CRAN (R 4.3.0)
 usethis       2.1.6   2022-05-25 [1] CRAN (R 4.3.0)
 vctrs         0.5.2   2023-01-23 [1] CRAN (R 4.3.0)
 xfun          0.37    2023-01-31 [1] CRAN (R 4.3.0)
 xtable        1.8-4   2019-04-21 [1] CRAN (R 4.3.0)

 [1] /Users/jacques/Library/R/arm64/4.3/library
 [2] /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library

------------------------------------------------------------------------------
```

Part III.

Day 3

8. Lecture 4 - Identifying cell populations

Slides here

9. Lab 5: Dimension reduction, clustering and annotation

9.1. Dimensional reduction for clustering

9.1.1. Preparing dataset

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

- **Which package from `Bioconductor` gives streamlined access to PBMC scRNAseq dataset from 10X Genomics?**
- **What does the object contain (type of data, number of cells, batches, organism, ...)? Can you get the same data from somewhere else?**

R

```
library(tidyverse)
library(SingleCellExperiment)
sce <- TENxPBMCData::TENxPBMCData('pbmc4k')
rownames(sce) <- scuttle::uniquifyFeatureNames(rowData(sce)$ENSEMBL_ID, rowData(sce)$Symbol
sce
rowData(sce)
colData(sce)
table(sce$Library)
```

9.1.2. Normalize counts using `scran`

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 as the total sum of counts across all genes for each cell, the expected value of which is assumed to scale with any cell-specific biases. 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.

All of this can be done very simply using the combo `quickCluster() + computeSumFactors() + logNormCounts()` from `scran/scuttle` packages.

R

```
clusters <- scran::quickCluster(sce)
table(clusters)
sce <- scran::computeSumFactors(sce, cluster = clusters)
colData(sce)
sce <- scuttle::logNormCounts(sce)
assays(sce)
```

9.1.3. Feature selection

We often use scRNAseq data in exploratory analyses to characterize heterogeneity across cells. Procedures like clustering and 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 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.

R

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

## --- Visualizing the mean-variance fit
df <- tibble(
    mean = metadata(sce_filtered_variance)$mean,
    var = metadata(sce_filtered_variance)$var,
    trend = metadata(sce_filtered_variance)$trend(mean),
    HVG = rowData(sce)$isHVG
```

```
)
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')
```

9.1.4. PCA on filtered dataset

We now have normalized counts filtered for the top 500 genes varying with the greatest biological significance.
Still, that represents a 500 x nCells (~8,000) dataset (each row being a feature). 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.

- **Leverage `scater` package to compute a `PCA` embedding of the filtered data by taking into account the technical variability.**

R

```
sce <- scran::denoisePCA(
    sce,
    technical = sce_filtered_variance,
    subset.row = HVGs,
    min.rank = 15
)
p <- scater::plotReducedDim(sce, 'PCA', colour_by = 'sizeFactor') + ggtitle('denoised PCA')
p
```

9.2. Clustering

Clustering is an unsupervised learning procedure that is 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.

9.2.1. 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.

- **Compute graph-based clustering of the PBMC dataset.**

R

```
graph <- scran::buildSNNGraph(
    sce,
    k = 5,
    use.dimred = 'PCA'
)
sce_clust <- igraph::cluster_louvain(graph)$membership
table(sce_clust)
sce$clusters_graph <- factor(sce_clust)
```

- **What are the main parameters to choose? How do they impact the clustering?**

R

```
graph2 <- scran::buildSNNGraph(
    sce,
    k = 50,
    use.dimred = 'PCA'
)
sce_clust2 <- igraph::cluster_louvain(graph2)$membership
table(sce_clust, sce_clust2)
```

9.2.2. 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)
```

- **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. `?runTSNE`)

R

```
reducedDims(sce)
sce <- scater::runTSNE(sce, subset_row = HVGs)
sce <- scater::runUMAP(sce, subset_row = HVGs)
reducedDims(sce)
reducedDim(sce, 'UMAP')[1:10, ]
```

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

R

```
library(patchwork)
p<- scater::plotReducedDim(sce, 'PCA', colour_by = 'clusters_graph') + ggtitle('denoised PC
    scater::plotReducedDim(sce, 'TSNE', colour_by = 'clusters_graph') + ggtitle('tSNE') +
    scater::plotReducedDim(sce, 'UMAP', colour_by = 'clusters_graph') + ggtitle('UMAP')
```

9.2.3. For the pros of clustering... Compare different clustering approaches

Leveraging the `bluster` package, different clustering approaches can be performed using a uniformed syntax, to compare their output.

- **Using `clusterSweep()`, compare the effect of different `k` neighbor values when performing graph-based clustering.**

R

```
clusters <- bluster::clusterSweep(
    reducedDim(sce, 'PCA'),
    BLUSPARAM = bluster::SNNGraphParam(),
    k = c(5L, 15L, 25L, 50L),
    cluster.fun = c("louvain")
)
colnames(clusters$clusters)
head(clusters$clusters)
clusters$parameters
library(ggraph)
p <- cowplot::plot_grid(
    clustree::clustree(
        clusters$clusters %>% setNames(1:ncol(.)) %>% as.data.frame(),
        prefix = 'X',
        edge_arrow=FALSE
    ),
    cowplot::plot_grid(
        scater::plotReducedDim(sce, 'TSNE', colour_by = I(clusters$clusters[, 'k.
        scater::plotReducedDim(sce, 'TSNE', colour_by = I(clusters$clusters[, 'k.
        scater::plotReducedDim(sce, 'TSNE', colour_by = I(clusters$clusters[, 'k.
        scater::plotReducedDim(sce, 'TSNE', colour_by = I(clusters$clusters[, 'k.
    ),
    nrow = 2,
    rel_heights = c(0.3, 0.7)
)
table(clusters$clusters[, 'k.5_cluster.fun.louvain'])
```

9.3. Cell annotation

9.3.1. Find 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.

R

```
markers <- scran::findMarkers(sce, groups = sce$clusters_graph)
```

- **Find markers strongly overexpressed in each cluster. Check `?scran::findMarkers` to find the right options to use.**

R

```
markers <- scran::findMarkers(
    sce,
    groups = sce$clusters_graph,
    direction = "up",
    lfc = 1
)
head(markers[[1]])
markers <- lapply(markers, function(df) {
    rownames(df[df$Top <= 5,])
})
```

- **Plot average expression of the first marker of the first cluster in UMAP**

R

```
p <- scater::plotReducedDim(sce, 'TSNE', colour_by = markers[[2]][[1]])
```

9.3.2. Automated cell annotation

Many cell type reference databases are available over the Internet. Today, we will use a reference constructed from `Blueprint` and `ENCODE` data (`Martens and Stunnenberg 2013`; `The ENCODE Project Consortium 2012`). This reference is available as a `SummarizedExperiment` containing log-normalized gene expression for manually annotated samples.

R

```
ref <- celldex::BlueprintEncodeData()
prediction_types <- SingleR::SingleR(
    test = sce,
    ref = ref,
```

```
    labels = ref$label.main
)
sce$annotation <- prediction_types$labels
table(sce$annotation)
table(sce$annotation, sce$clusters_graph)
```

- **Using `scater` and `SingleR` utilities, visually compare the annotation scores for cells in each cluster.**
- **Did the automated annotation work robuslty? How does it compare to our clustering? Is automated annotation as sensitive as graph-based clustering?**

R

```
p <- SingleR::plotScoreHeatmap(prediction_types)
p <- scater::plotReducedDim(sce, 'TSNE', colour_by = 'annotation') + ggtitle('Aut
p <- pheatmap::pheatmap(
    log2(table(Annotation = sce$annotation, Cluster = sce$clusters_graph)+10),
    color = colorRampPalette(c("white", "darkred"))(101)
)
```

9.4. Bonus

Try to fill in the analysis template in `bin/prepare_Ernst.R` to execute the different processing/analysis steps we covered in the previous exercises and this one. If you prefer using `Seurat`, don't hesitate to modify the base template!

9.5. Acknowledgements

This exercise was adapted from Chapts. 7-12 of Orchestrating Single-Cell Analysis with Bioconductor.

9.6. Session info

```
- Session info -------------------------------------------------------------
 setting  value
 version  R version 4.3.0 (2023-04-21)
 os       macOS Monterey 12.5.1
 system   aarch64, darwin20
```

```
 ui       X11
 language (EN)
 collate  en_GB.UTF-8
 ctype    en_GB.UTF-8
 tz       Europe/Paris
 date     2023-06-05
 pandoc   2.19.2 @ /opt/homebrew/bin/ (via rmarkdown)

- Packages -----------------------------------------------------------------------------------
 package              * version   date (UTC) lib source
 AnnotationDbi        * 1.61.0    2022-11-01 [1] Bioconductor
 AnnotationHub        * 3.7.3     2023-03-01 [1] Bioconductor
 Biobase              * 2.59.0    2022-11-01 [1] Bioconductor
 BiocFileCache        * 2.7.2     2023-02-17 [1] Bioconductor
 BiocGenerics         * 0.45.0    2022-11-01 [1] Bioconductor
 BiocManager          * 1.30.20   2023-02-24 [1] CRAN (R 4.3.0)
 BiocVersion            3.17.1    2022-12-20 [1] Bioconductor
 Biostrings             2.67.0    2022-11-01 [1] Bioconductor
 bit                    4.0.5     2022-11-15 [1] CRAN (R 4.3.0)
 bit64                  4.0.5     2020-08-30 [1] CRAN (R 4.3.0)
 bitops                 1.0-7     2021-04-24 [1] CRAN (R 4.3.0)
 blob                   1.2.3     2022-04-10 [1] CRAN (R 4.3.0)
 cachem                 1.0.7     2023-02-24 [1] CRAN (R 4.3.0)
 callr                  3.7.3     2022-11-02 [1] CRAN (R 4.3.0)
 cli                    3.6.0     2023-01-09 [1] CRAN (R 4.3.0)
 colorspace             2.1-0     2023-01-23 [1] CRAN (R 4.3.0)
 cowplot              * 1.1.1     2020-12-30 [1] CRAN (R 4.3.0)
 crayon                 1.5.2     2022-09-29 [1] CRAN (R 4.3.0)
 curl                   5.0.0     2023-01-12 [1] CRAN (R 4.3.0)
 DBI                    1.1.3     2022-06-18 [1] CRAN (R 4.3.0)
 dbplyr               * 2.3.1     2023-02-24 [1] CRAN (R 4.3.0)
 devtools               2.4.5     2022-10-11 [1] CRAN (R 4.3.0)
 digest                 0.6.31    2022-12-11 [1] CRAN (R 4.3.0)
 dplyr                * 1.1.0     2023-01-29 [1] CRAN (R 4.3.0)
 ellipsis               0.3.2     2021-04-29 [1] CRAN (R 4.3.0)
 evaluate               0.20      2023-01-17 [1] CRAN (R 4.3.0)
 fansi                  1.0.4     2023-01-22 [1] CRAN (R 4.3.0)
 fastmap                1.1.1     2023-02-24 [1] CRAN (R 4.3.0)
 filelock               1.0.2     2018-10-05 [1] CRAN (R 4.3.0)
 forcats              * 1.0.0     2023-01-29 [1] CRAN (R 4.3.0)
 fs                     1.6.1     2023-02-06 [1] CRAN (R 4.3.0)
 generics               0.1.3     2022-07-05 [1] CRAN (R 4.3.0)
 GenomeInfoDb           1.35.15   2023-02-03 [1] Bioconductor
 GenomeInfoDbData       1.2.9     2022-11-04 [1] Bioconductor
```

```
ggplot2                 * 3.4.1     2023-02-10 [1] CRAN (R 4.3.0)
glue                      1.6.2     2022-02-24 [1] CRAN (R 4.3.0)
gtable                    0.3.1     2022-09-01 [1] CRAN (R 4.3.0)
hms                       1.1.2     2022-08-19 [1] CRAN (R 4.3.0)
htmltools                 0.5.4     2022-12-07 [1] CRAN (R 4.3.0)
htmlwidgets               1.6.1     2023-01-07 [1] CRAN (R 4.3.0)
httpuv                    1.6.9     2023-02-14 [1] CRAN (R 4.3.0)
httr                      1.4.5     2023-02-24 [1] CRAN (R 4.3.0)
interactiveDisplayBase    1.37.0    2022-11-01 [1] Bioconductor
IRanges                 * 2.33.0    2022-11-01 [1] Bioconductor
jsonlite                  1.8.4     2022-12-06 [1] CRAN (R 4.3.0)
KEGGREST                  1.39.0    2022-11-01 [1] Bioconductor
knitr                     1.42      2023-01-25 [1] CRAN (R 4.3.0)
later                     1.3.0     2021-08-18 [1] CRAN (R 4.3.0)
lifecycle                 1.0.3     2022-10-07 [1] CRAN (R 4.3.0)
lubridate               * 1.9.2     2023-02-10 [1] CRAN (R 4.3.0)
magrittr                  2.0.3     2022-03-30 [1] CRAN (R 4.3.0)
memoise                   2.0.1     2021-11-26 [1] CRAN (R 4.3.0)
mime                      0.12      2021-09-28 [1] CRAN (R 4.3.0)
miniUI                    0.1.1.1   2018-05-18 [1] CRAN (R 4.3.0)
munsell                   0.5.0     2018-06-12 [1] CRAN (R 4.3.0)
pillar                    1.8.1     2022-08-19 [1] CRAN (R 4.3.0)
pkgbuild                  1.4.0     2022-11-27 [1] CRAN (R 4.3.0)
pkgconfig                 2.0.3     2019-09-22 [1] CRAN (R 4.3.0)
pkgload                   1.3.2     2022-11-16 [1] CRAN (R 4.3.0)
png                       0.1-8     2022-11-29 [1] CRAN (R 4.3.0)
prettyunits               1.1.1     2020-01-24 [1] CRAN (R 4.3.0)
processx                  3.8.0     2022-10-26 [1] CRAN (R 4.3.0)
profvis                   0.3.7     2020-11-02 [1] CRAN (R 4.3.0)
promises                  1.2.0.1   2021-02-11 [1] CRAN (R 4.3.0)
ps                        1.7.2     2022-10-26 [1] CRAN (R 4.3.0)
purrr                   * 1.0.1     2023-01-10 [1] CRAN (R 4.3.0)
R6                        2.5.1     2021-08-19 [1] CRAN (R 4.3.0)
rappdirs                  0.3.3     2021-01-31 [1] CRAN (R 4.3.0)
Rcpp                      1.0.10    2023-01-22 [1] CRAN (R 4.3.0)
RCurl                     1.98-1.10 2023-01-27 [1] CRAN (R 4.3.0)
readr                   * 2.1.4     2023-02-10 [1] CRAN (R 4.3.0)
remotes                   2.4.2     2021-11-30 [1] CRAN (R 4.3.0)
rlang                     1.0.6     2022-09-24 [1] CRAN (R 4.3.0)
rmarkdown                 2.20      2023-01-19 [1] CRAN (R 4.3.0)
RSQLite                   2.3.0     2023-02-17 [1] CRAN (R 4.3.0)
S4Vectors               * 0.37.4    2023-02-26 [1] Bioconductor
scales                    1.2.1     2022-08-20 [1] CRAN (R 4.3.0)
sessioninfo               1.2.2     2021-12-06 [1] CRAN (R 4.3.0)
```

```
shiny                 1.7.4    2022-12-15 [1] CRAN (R 4.3.0)
stringi               1.7.12   2023-01-11 [1] CRAN (R 4.3.0)
stringr             * 1.5.0    2022-12-02 [1] CRAN (R 4.3.0)
tibble              * 3.1.8    2022-07-22 [1] CRAN (R 4.3.0)
tidyr               * 1.3.0    2023-01-24 [1] CRAN (R 4.3.0)
tidyselect            1.2.0    2022-10-10 [1] CRAN (R 4.3.0)
tidyverse           * 2.0.0    2023-02-22 [1] CRAN (R 4.3.0)
timechange            0.2.0    2023-01-11 [1] CRAN (R 4.3.0)
tzdb                  0.3.0    2022-03-28 [1] CRAN (R 4.3.0)
urlchecker            1.0.1    2021-11-30 [1] CRAN (R 4.3.0)
usethis               2.1.6    2022-05-25 [1] CRAN (R 4.3.0)
utf8                  1.2.3    2023-01-31 [1] CRAN (R 4.3.0)
vctrs                 0.5.2    2023-01-23 [1] CRAN (R 4.3.0)
withr                 2.5.0    2022-03-03 [1] CRAN (R 4.3.0)
xfun                  0.37     2023-01-31 [1] CRAN (R 4.3.0)
xtable                1.8-4    2019-04-21 [1] CRAN (R 4.3.0)
XVector               0.39.0   2022-12-20 [1] Bioconductor
yaml                  2.3.7    2023-01-23 [1] CRAN (R 4.3.0)
zlibbioc              1.45.0   2022-12-20 [1] Bioconductor

[1] /Users/jacques/Library/R/arm64/4.3/library
[2] /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library
```

---

10. Lecture 5 - Data integration and batch effect correction

Slides here

11. Lab 6 - Batch correction

In this lab, we will look at different single cell RNA-seq datasets collected from pancreatic islets. We will look at how different batch correction methods affect our data analysis.

11.1. Load settings and packages

R

```
library(Seurat)
library(Matrix)
library(fossil)
library(dplyr)
library(plyr)
library(rliger)

# Set folder location for saving output files. This is also the same location as input data
mydir <- "~/Share/batch_correction/"

# Objects to save.
writedir <- "~/"
Rda.sparse.path <- paste0(writedir, "pancreas_subsample.Rda")
Rda.path <- paste0(writedir, "pancreas_nobatchcorrect.Rda")
Rda.Seurat3.path <- paste0(writedir, "pancreas_Seurat3.Rda")
Rda.liger.path <- paste0(writedir, "pancreas_liger.Rda")
```

11.2. Read in pancreas expression matrices

R

```
# Read in all four input expression matrices
celseq.data <- read.table(paste0(mydir, "pancreas_multi_celseq_expression_matrix.txt.gz"))
celseq2.data <- read.table(paste0(mydir, "pancreas_multi_celseq2_expression_matrix.txt.gz")
fluidigmc1.data <- read.table(paste0(mydir, "pancreas_multi_fluidigmc1_expression_matrix.tx
```

```
smartseq2.data <- read.table(paste0(mydir, "pancreas_multi_smartseq2_expression_m

# Convert to sparse matrices for efficiency
celseq.data <- as(as.matrix(celseq.data), "dgCMatrix")
celseq2.data <- as(as.matrix(celseq2.data), "dgCMatrix")
fluidigmc1.data <- as(as.matrix(fluidigmc1.data), "dgCMatrix")
smartseq2.data <- as(as.matrix(smartseq2.data), "dgCMatrix")
```

11.3. Preparing the individual Seurat objects for each pancreas dataset without batch correction

R

```
# What is the size of each single cell RNA-seq dataset?
# Briefly describe the technology used to collect each dataset.
# Which datasets do you expect to be different and which do you expect to be simi
dim(celseq.data)
dim(celseq2.data)
dim(fluidigmc1.data)
dim(smartseq2.data)

# Create and setup Seurat objects for each dataset with the following 6 steps.
# 1. CreateSeuratObject
# 2. subset
# 3. NormalizeData
# 4. FindVariableGenes
# 5. ScaleData
# 6. Update @meta.data slot in Seurat object with tech column (celseq, celseq2, f
# Look at the distributions of number of genes per cell before and after FilterCe

# CEL-Seq (https://www.cell.com/cell-reports/fulltext/S2211-1247(12)00228-8)
# In subset, use low.thresholds = 1750
celseq <- CreateSeuratObject(counts = celseq.data)
VlnPlot(celseq, "nFeature_RNA")
celseq <- subset(celseq, subset = nFeature_RNA > 1750)
VlnPlot(celseq, "nFeature_RNA")
celseq <- NormalizeData(celseq, normalization.method = "LogNormalize", scale.fact
celseq <- FindVariableFeatures(celseq, selection.method = "vst", nfeatures = 2000
celseq <- ScaleData(celseq)
celseq[["tech"]] <- "celseq"
```

```
# CEL-Seq2 https://www.cell.com/molecular-cell/fulltext/S1097-2765(09)00641-8
# In subset, use low.thresholds = 2500.
celseq2 <- CreateSeuratObject(counts = celseq2.data)
VlnPlot(celseq2, "nFeature_RNA")
celseq2 <- subset(celseq2, subset = nFeature_RNA > 2500)
VlnPlot(celseq2, "nFeature_RNA")
celseq2 <- NormalizeData(celseq2, normalization.method = "LogNormalize", scale.factor = 100
celseq2 <- FindVariableFeatures(celseq2, selection.method = "vst", nfeatures = 2000)
celseq2 <- ScaleData(celseq2)
celseq2[["tech"]] <- "celseq2"

# Fluidigm C1
# Omit subset function because cells are already high quality.
fluidigmc1 <- CreateSeuratObject(counts = fluidigmc1.data)
VlnPlot(fluidigmc1, "nFeature_RNA")
fluidigmc1 <- NormalizeData(fluidigmc1, normalization.method = "LogNormalize", scale.factor
fluidigmc1 <- FindVariableFeatures(fluidigmc1, selection.method = "vst", nfeatures = 2000)
fluidigmc1 <- ScaleData(fluidigmc1)
fluidigmc1[["tech"]] <- "fluidigmc1"

# SMART-Seq2
# In subset, use low.thresholds = 2500.
smartseq2 <- CreateSeuratObject(counts = smartseq2.data)
VlnPlot(smartseq2, "nFeature_RNA")
smartseq2 <- subset(smartseq2, subset = nFeature_RNA > 2500)
VlnPlot(smartseq2, "nFeature_RNA")
smartseq2 <- NormalizeData(smartseq2, normalization.method = "LogNormalize", scale.factor =
smartseq2 <- FindVariableFeatures(smartseq2, selection.method = "vst", nfeatures = 2000)
smartseq2 <- ScaleData(smartseq2)
smartseq2[["tech"]] <- "smartseq2"

# This code sub-samples the data in order to speed up calculations and not use too much mem
Idents(celseq) <- "tech"
celseq <- subset(celseq, downsample = 500, seed = 1)
Idents(celseq2) <- "tech"
celseq2 <- subset(celseq2, downsample = 500, seed = 1)
Idents(fluidigmc1) <- "tech"
fluidigmc1 <- subset(fluidigmc1, downsample = 500, seed = 1)
Idents(smartseq2) <- "tech"
smartseq2 <- subset(smartseq2, downsample = 500, seed = 1)

# Save the sub-sampled Seurat objects
```

```
save(celseq, celseq2, fluidigmc1, smartseq2, file = Rda.sparse.path)
```

11.4. Cluster pancreatic datasets without batch correction

Let us cluster all the pancreatic islet datasets together and see whether there is a batch effect.

R

```
load(Rda.sparse.path)

# Merge Seurat objects. Original sample identities are stored in gcdata[["tech"]]
# Cell names will now have the format tech_cellID (smartseq2_cell1...)
add.cell.ids <- c("celseq", "celseq2", "fluidigmc1", "smartseq2")
gcdata <- merge(x = celseq, y = list(celseq2, fluidigmc1, smartseq2), add.cell.id
Idents(gcdata) <- "tech"  # use identity based on sample identity

# Look at how the number of genes per cell varies across the different technologi
VlnPlot(gcdata, "nFeature_RNA", group.by = "tech")

# The merged data must be normalized and scaled (but you only need to scale the v
# Let us also find the variable genes again this time using all the pancreas data
gcdata <- NormalizeData(gcdata, normalization.method = "LogNormalize", scale.fact
var.genes <- SelectIntegrationFeatures(SplitObject(gcdata, split.by = "tech"), nf
VariableFeatures(gcdata) <- var.genes
gcdata <- ScaleData(gcdata, features = VariableFeatures(gcdata))

# Do PCA on data including only the variable genes.
gcdata <- RunPCA(gcdata, features = VariableFeatures(gcdata), npcs = 40, ndims.pr

# Color the PC biplot by the scRNA-seq technology. Hint: use DimPlot
# Which technologies look similar to one another?
DimPlot(gcdata, reduction = "pca", dims = c(1, 2), group.by = "tech")

# Cluster the cells using the first twenty principal components.
gcdata <- FindNeighbors(gcdata, reduction = "pca", dims = 1:20, k.param = 20)
gcdata <- FindClusters(gcdata, resolution = 0.8, algorithm = 1, random.seed = 100

# Create a UMAP visualization.
gcdata <- RunUMAP(gcdata, dims = 1:20, reduction = "pca", n.neighbors = 15, min.d
```

```
# Visualize the Leiden clustering and the batches on the UMAP.
# Remember, the clustering is stored in @meta.data in column seurat_clusters and the techno
# stored in the column tech. Remember you can also use DimPlot
DimPlot(gcdata, reduction = "umap", group.by = "seurat_clusters")
DimPlot(gcdata, reduction = "umap", group.by = "tech")

# Are you surprised by the results? Compare to your expectations from the PC biplot of PC1
# What explains these results?

# Adjusted rand index test for overlap between technology and cluster labelings.
# This goes between 0 (completely dissimilar clustering) to 1 (identical clustering).
# The adjustment corrects for chance grouping between cluster elements.
# https://davetang.org/muse/2017/09/21/adjusted-rand-index/
ari <- dplyr::select(gcdata[[]], tech, seurat_clusters)
ari$tech <- plyr::mapvalues(ari$tech, from = c("celseq", "celseq2", "fluidigmc1", "smartseq
adj.rand.index(as.numeric(ari$tech), as.numeric(ari$seurat_clusters))

# Save current progress.
save(gcdata, file = Rda.path)
# To load the data, run the following command.
# load(Rda.path)
```

11.4.1. Batch correction: canonical correlation analysis (CCA)+ mutual nearest neighbors (MNN) using Seurat v3

Here we use Seurat v3 to see to what extent it can remove potential batch effects.

R

```
load(Rda.sparse.path)

# The first piece of code will identify variable genes that are highly variable in at least
# Why would we implement such a requirement?
ob.list <- list(celseq, celseq2, fluidigmc1, smartseq2)

# Identify anchors on the 4 pancreatic islet datasets, commonly shared variable genes acros
# and integrate samples.
gcdata.anchors <- FindIntegrationAnchors(object.list = ob.list, anchor.features = 2000, dim
gcdata <- IntegrateData(anchorset = gcdata.anchors, dims = 1:30)
DefaultAssay(gcdata) <- "integrated"

# Run the standard workflow for visualization and clustering.
```

```
# The integrated data object only stores the commonly shared variable genes.
gcdata <- ScaleData(gcdata, do.center = T, do.scale = F)
gcdata <- FindVariableFeatures(gcdata)
gcdata <- RunPCA(gcdata, npcs = 40, ndims.print = 1:5, nfeatures.print = 5)
DimPlot(gcdata, dims = c(1, 2), reduction = "pca", split.by = "tech")

# Clustering. Choose the dimensional reduction type to use and the number of alig
# canonical correlation vectors to use.
gcdata <- FindNeighbors(gcdata, reduction = "pca", dims = 1:20, k.param = 20)
gcdata <- FindClusters(gcdata, resolution = 0.8, algorithm = 1, random.seed = 100

# UMAP. Choose the dimensional reduction type to use and the number of aligned
# canonical correlation vectors to use.
gcdata <- RunUMAP(gcdata, dims = 1:30, reduction = "pca", n.neighbors = 15, min.d

# After data integration, use the original expression data in all visualization a
# The integrated data must not be used in DE tests as it violates assumptions of
DefaultAssay(gcdata) <- "RNA"

# Visualize the Louvain clustering and the batches on the UMAP.
# Remember, the clustering is stored in @meta.data in column seurat_clusters
# and the technology is stored in the column tech. Remember you can also use DimP
p1 <- DimPlot(gcdata, reduction = "umap", group.by = "seurat_clusters")
p2 <- DimPlot(gcdata, reduction = "umap", group.by = "tech")
p1 + p2

# Let's look to see how the adjusted rand index changed compared to using no batc
ari <- dplyr::select(gcdata[[]], tech, seurat_clusters)
ari$tech <- plyr::mapvalues(ari$tech, from = c("celseq", "celseq2", "fluidigmc1",
adj.rand.index(as.numeric(ari$tech), as.numeric(ari$seurat_clusters))

# We can also identify conserved marker genes across the batches. Differential ge
# done across each batch, and the p-values are combined. (requires metap packge i
markers <- FindConservedMarkers(gcdata, ident.1 = 0, grouping.var = "tech", assay
head(markers)

# Visualize the expression of the first 5 marker genes on UMAP across the differe
gcdata <- ScaleData(gcdata, features = rownames(gcdata), do.center = T, do.scale
DoHeatmap(gcdata, features = rownames(markers)[1:5], group.by = "tech", disp.max

# Markers for pancreatic cells from "A Single-Cell Transcriptome Atlas of the
# Human Pancreas".https://www.cell.com/cell-systems/pdfExtended/S2405-4712(16)302
```

```
genes <- c("GCG", "INS", "SST", "PPY", "PRSS1", "KRT19", "PECAM1", "COL1A1")
FeaturePlot(gcdata, genes, ncol = 4)

# Save current progress.
save(gcdata, file = Rda.Seurat3.path)
# To load the data, run the following command.
# load(Rda.Seurat3.path)
```

11.4.2. Batch correction: integrative non-negative matrix factorization (NMF) using LIGER

Here we use integrative non-negative matrix factorization to see to what extent it can remove potential batch effects.

The important parameters in the batch correction are the number of factors (k), the penalty parameter (lambda), and the clustering resolution. The number of factors sets the number of factors (consisting of shared and dataset-specific factors) used in factorizing the matrix. The penalty parameter sets the balance between factors shared across the batches and factors specific to the individual batches. The default setting of lambda=5.0 is usually used by the Macosko lab. Resolution=1.0 is used in the Louvain clustering of the shared neighbor factors that have been quantile normalized.

R

```
load(Rda.sparse.path)

ob.list <- list("celseq" = celseq, "celseq2" = celseq2, "fluidigmc1" = fluidigmc1, "smartse

# Identify variable genes that are variable across most samples.
var.genes <- SelectIntegrationFeatures(ob.list, nfeatures = 2000, verbose = TRUE, fvf.nfeat

# Next we create a LIGER object with raw counts data from each batch.
data.liger <- createLiger(sapply(ob.list, function(data) data[['RNA']]@counts[, colnames(da

# Normalize gene expression for each batch.
data.liger <- rliger::normalize(data.liger)

# Use my method or Liger method for selecting variable genes (var.thresh changes number of
data.liger@var.genes <- var.genes
# data.liger <- selectGenes(data.liger, var.thresh = 0.1, do.plot = F)

# Print out the number of variable genes for LIGER analysis.
```

```
print(length(data.liger@var.genes))

# Scale the gene expression across the datasets.
# Why does LIGER not center the data? Hint, think about the use of
# non-negative matrix factorization and the constraints that this imposes.
data.liger <- scaleNotCenter(data.liger)

# These two steps take 10-20 min. Only run them if you finish with the rest of th
# Use the `suggestK` function to determine the appropriate number of factors to u
# Use the `suggestLambda` function to find the smallest lambda for which the alig
# k.suggest <- suggestK(data.liger, k.test = seq(5, 30, 5), plot.log2 = T)
# lambda.suggest <- suggestLambda(gcdata.liger, k.suggest)

# Use alternating least squares (ALS) to factorize the matrix.
# Next, quantile align the factor loadings across the datasets, and do clustering
k.suggest <- 20  # with this line, we do not use the suggested k by suggestK()
lambda.suggest <- 5  # with this line, we do not use the suggested lambda by sugg
set.seed(100)  # optimizeALS below is stochastic
data.liger <- optimizeALS(data.liger, k = k.suggest, lamda = lambda.suggest)

# What do matrices H, V, and W represent, and what are their dimensions?
dim(data.liger@H$celseq)
dim(data.liger@V$celseq)
dim(data.liger@W)

# Next, do clustering of cells in shared nearest factor space.
# Build SNF graph, do quantile normalization, cluster quantile normalized data
data.liger <- quantileAlignSNF(data.liger, resolution = 1)  # SNF clustering and

# What are the dimensions of H.norm. What does this represent?
dim(data.liger@H.norm)

# Let's see what the liger data looks like mapped onto a UMAP visualization.
data.liger <- runUMAP(data.liger, n_neighbors = 15, min_dist = 0.5)  # conda inst
p <- plotByDatasetAndCluster(data.liger, return.plots = T)
print(p[[1]])  # plot by dataset
plot_grid(p[[1]], p[[2]])

# Let's look to see how the adjusted rand index changed compared to using no batc
tech <- unlist(lapply(1:length(data.liger@H), function(x) {
  rep(names(data.liger@H)[x], nrow(data.liger@H[[x]]))}))
clusters <- data.liger@clusters
```

```
ari <- data.frame("tech" = tech, "clusters" = clusters)
ari$tech <- plyr::mapvalues(ari$tech, from = c("celseq", "celseq2", "fluidigmc1", "smartseq
adj.rand.index(as.numeric(ari$tech), as.numeric(ari$clusters))

# Look at genes that are specific to a dataset and shared across datasets.
# Use the plotWordClouds function and choose 2 datasets.
pdf(paste0(writedir, "word_clouds.pdf"))
plotWordClouds(data.liger, dataset1 = "celseq2", dataset2 = "smartseq2")
dev.off()

# Look at factor loadings for each cell using plotFactors.
pdf(paste0(writedir, "plot_factors.pdf"))
plotFactors(data.liger)
dev.off()

# Identify shared and batch-specific marker genes from liger factorization.
# Use the getFactorMarkers function and choose 2 datasets.
# Then plot some genes of interest using plotGene.
markers <- getFactorMarkers(data.liger, dataset1 = "celseq2", dataset2 = "smartseq2", num.g
plotGene(data.liger, gene = "INS", pt.size = 1)

# Save current progress.
save(data.liger, file = Rda.liger.path)
# To load the data, run the following command.
# load(Rda.liger.path)
```

11.5. Additional exploration

11.5.1. Regressing out unwanted covariates

Learn how to regress out different technical covariates (number of UMIs, number of genes, percent mitochondrial reads) by studying Seurat's PBMC tutorial and the ScaleData() function.

11.5.2. kBET

Within your RStudio session, install k-nearest neighbour batch effect test and learn how to use its functionality to quantify batch effects in the pancreatic data.

11.5.3. Seurat v4

Read how the new version of Seurat does data integration

11.6. Acknowledgements

This document builds off a tutorial from the Seurat website and a tutorial from the LIGER website.

Part IV.

Day 4

12. Lecture 6 - Trajectories and pseudotimes

Slides here

13. Lab 7: Pseudotime analyses

Aims

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

13.1. Process testis data in R

13.1.1. Import testis data from GSE112013 and pre-process it

The workflow we have covered in the previous days can be reused here.

- **Import testis dataset in R, remove doublets, filter genes, normalize counts, correct for batch effect and cluster cells.**

R

```
library(SingleCellExperiment)
library(tidyverse)

# Download raw file
dir.create('Guo_testis')
download.file(
    'https://ftp.ncbi.nlm.nih.gov/geo/series/GSE112nnn/GSE112013/suppl/GSE112013_Combined_U
    'Guo_testis/GSE112013_Combined_UMI_table.txt.gz'
)
system('gunzip Guo_testis/GSE112013_Combined_UMI_table.txt.gz')

# Create SingleCellExperiment object
x <- vroom::vroom('Guo_testis/GSE112013_Combined_UMI_table.txt')
counts <- as.matrix(x[, -1])
counts <- as(counts, 'dgCMatrix')
genes <- as.data.frame(x[, 1])
```

```
cells <- data.frame(cellid = colnames(x[, -1]))
testis <- SingleCellExperiment(
    assays = list(counts = counts),
    colData = cells,
    rowData = genes
)
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', '
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)], ]
rowRanges(testis) <- genes[match(rownames(testis), genes$gene_name)]
rowData(testis) <- rowData(testis)[, c('gene_name', 'gene_id')]
rownames(testis) <- scuttle::uniquifyFeatureNames(rowData(testis)$gene_id, rowDat

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

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

# Normalize counts using VST
```

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

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

# Embedding dataset in PCA space, correcting for batch effect
mergedBatches <- batchelor::fastMNN(
    testis,
    batch = testis$donor,
    subset.row = HVGs,
    BPPARAM = BiocParallel::MulticoreParam(workers = 12)
)
reducedDim(testis, 'corrected') <- reducedDim(mergedBatches, 'corrected')

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

# Cluster cells using Louvain community finding algorithm
testis_clust <- igraph::cluster_louvain(graph)$membership
table(testis_clust)
testis$cluster <- factor(testis_clust)

# Embedding dataset in TSNE space for visualization
set.seed(10)
testis <- scater::runTSNE(testis, dimred = 'corrected')

# Visualize embeddings
p <- cowplot::plot_grid(
    scater::plotReducedDim(testis, 'corrected', colour_by = 'donor'),
    scater::plotReducedDim(testis, 'corrected', colour_by = 'cluster'),
    scater::plotReducedDim(testis, 'TSNE', colour_by = 'donor'),
    scater::plotReducedDim(testis, 'TSNE', colour_by = 'cluster')
)
```

13.1.2. Annotate cells using HPA resources

- **Load `HPA` data from internet.**
- **Try to format it as a `SummarizedExperiment`.**
- **What celltypes are profiled?**

R

```
download.file(
    'https://www.proteinatlas.org/download/rna_single_cell_type.tsv.zip',
    'Guo_testis/rna_single_cell_type.tsv.zip'
)
system('unzip Guo_testis/rna_single_cell_type.tsv.zip')
system('mv rna_single_cell_type.tsv Guo_testis/')
HPA <- vroom::vroom('Guo_testis/rna_single_cell_type.tsv') |>
    pivot_wider(names_from = `Cell type`, values_from = 'nTPM') |>
    dplyr::select(-Gene) |>
    distinct(`Gene name`, .keep_all = TRUE) |>
    column_to_rownames('Gene name')
HPA_se <- SummarizedExperiment::SummarizedExperiment(HPA, colData = tibble(cell_t
```

- **Use these cell type profiles to annotate cell types in the testis dataset.**

R

```
# Transfer annotations to `testis`
predictions <- SingleR::SingleR(
    test = testis,
    ref = HPA_se,
    labels = colData(HPA_se)$cell_type,
    clusters = testis$cluster
)
testis$annotation <- predictions$labels[testis$cluster]
p <- cowplot::plot_grid(
    scater::plotReducedDim(testis, dimred = 'corrected', colour_by = 'cluster', t
    scater::plotReducedDim(testis, dimred = 'corrected', colour_by = 'annotation'
    scater::plotReducedDim(testis, dimred = 'TSNE', colour_by = 'cluster', text_b
    scater::plotReducedDim(testis, dimred = 'TSNE', colour_by = 'annotation', tex
)
```

13.1.3. Filter the testis dataset to only germinal cells.

R

```
germcells <- testis[
    ,
    testis$annotation %in% c("Spermatogonia", "Spermatocytes", "Early spermatids", "Late sp
]
```

13.2. 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):

13.2.1. Trajectory

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

- **Read Slingshot documentation to understand how to identify lineages in a scRNAseq dataset in R**
- **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?**
- **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).**

R

```
reducedDim(germcells, 'corrected_2') <- reducedDim(germcells, 'corrected')[, 1:49]
germcells_slingshot <- slingshot::slingshot(
    germcells,
    reducedDim = 'corrected_2',
    clusterLabels = germcells$annotation
)
slingshot::slingLineages(germcells_slingshot)
```

13.2.2. Pseudotime

- **Recover pseudotime values and principal curves from `slingshot` output**

R

```
germcells$pseudotime <- slingshot::slingPseudotime(germcells_slingshot)[, 'Lineag
pca_curve <- slingshot::slingCurves(germcells_slingshot, as.df = TRUE)
colnames(pca_curve) <- paste0('PC', 1:ncol(pca_curve))
tsne_curve <- slingshot::embedCurves(germcells_slingshot, 'TSNE', smoother = 'loe
tsne_curve <- tsne_curve[order(tsne_curve$Order), ]
colnames(tsne_curve) <- paste0('TSNE', 1:ncol(tsne_curve))
```

- **Plot PCA and tSNE embeddings, coloring cells by either their annotation or their pseudotime value. Overlay the principal curves to each embedding**

R

```
df <- tibble(
    PC1 = reducedDim(germcells, 'corrected')[,1],
    PC2 = reducedDim(germcells, 'corrected')[,2],
    TSNE1 = reducedDim(germcells, 'TSNE')[,1],
    TSNE2 = reducedDim(germcells, 'TSNE')[,2],
    annotation = germcells$annotation,
    pseudotime = germcells$pseudotime
)
p <- cowplot::plot_grid(
    df |>
        ggplot() +
        geom_point(aes(PC1, PC2, col = annotation)) +
        geom_path(data = pca_curve, aes(x = PC1, y = PC2)) +
        theme_bw() +
        coord_fixed(),
    df |>
        ggplot() +
        geom_point(aes(TSNE1, TSNE2, col = annotation)) +
        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()
)
```

- **Check the distribution of pseudotime values across the different cell clusters.**
- **What do you observe? Where you expecting this?**

R

```
p <- tibble(
    annotation = factor(germcells$annotation, c("Spermatogonia", "Spermatocytes", "Early sp
    pseudotime = germcells$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))
```

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

R

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

13.2.3. BONUS: Daunting snippet but that makes a cool figure for a paper: modeling pseudotime-dependent gene expression

Using pseudotime / cell, one can model gene expression along the differentiation process. This alleviates the need to study gene expression **per cell**, and allows one to focus on process-related effects (e.g. gene expression during a developmental trajectory).

- **Try to do so for few markers of spermatogonia, spermatocytes and spermatids.**

R

```
genes <- c('ID4', 'SYCP3', 'DMC1', 'ACR', 'PRM1', 'PGK2')
fitExprs <- logcounts(germcells[genes, ]) |> # ---------------------------------
    as.matrix() |>
    t() |>
    as_tibble() |>
    mutate(  # ---------------------------------------------------------------
        cellID = colnames(germcells),
        annotation = factor(germcells$annotation, c("Spermatogonia", "Spermatocyt
        pseudotime = germcells$pseudotime
    ) |>
    pivot_longer(contains(genes), names_to = 'gene', values_to = 'obs_expr') |> #
    mutate(gene = factor(gene, genes)) |>
    group_by(gene) |> # ------------------------------------------------------- G
    nest(.key = 'data') |> # -------------------------------------------------- F
    mutate(
        gamModel = map(data, ~mgcv::gam(obs_expr ~ s(pseudotime, bs = "cs"), data
        gamFitted_expr = map(gamModel, predict) # ------------------------------
    ) |>
    dplyr::select(-ends_with('Model')) |>
    unnest(c(data, ends_with('_expr'))) # ------------------------------------
p <- ggplot(fitExprs) +
    geom_point(aes(x = pseudotime, y = obs_expr, col = annotation), alpha = 0.5)
    geom_line(aes(x = pseudotime, y = gamFitted_expr), col = 'white', size = 2, a
    geom_line(aes(x = pseudotime, y = gamFitted_expr), col = '#af2d0c', size = 1)
    theme_bw() +
    facet_grid(gene~., scales = 'free') +
    labs(y = 'logcounts') +
    ggtitle('Fitted models of pseudotime-dependent gene expression')
```

13.3. Ordering trajectory with RNA velocity

As we saw earlier, TI does not necessarily know which direction is right for a given trajectory. This orientation can be sometimes refined using RNA velocity. Let's see whether RNA velocity helps orientating our spermatocyte differentiation lineage trajectory here.

- **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.**

R

```
## - Import entire GSE112013 dataset with spliced/unspliced counts
full_GSE112013_counts <- LoomExperiment::import('Share/Guo_testis/Guo_testis_full-counts.lo
## - Filter `germcells` genes and cells to only retain those present in `full_GSE112013_cou
germcells <- germcells[
    rowData(germcells)$gene_id %in% rowData(full_GSE112013_counts)$Accession,
    germcells$Barcode %in% full_GSE112013_counts$Barcode
]
## - Reorder rows of `full_GSE112013_counts_germcells` to match those of `germcells`
full_GSE112013_counts_germcells <- full_GSE112013_counts[match(rowData(germcells)$gene_id,
dim(germcells)
dim(full_GSE112013_counts_germcells)
## - Add spliced/unspliced counts to germcells
assay(germcells, 'spliced', withDimnames=FALSE) <- assay(full_GSE112013_counts_germcells, '
assay(germcells, 'unspliced', withDimnames=FALSE) <- assay(full_GSE112013_counts_germcells,
## - Run velociraptor
velo_out <- velociraptor::scvelo(
    germcells,
    assay.X = "counts",
    subset.row = rowData(germcells)$isHVG,
    use.dimred = "corrected"
)
velo_out
```

- **Embed the velocity field in tSNE scRNAseq embedding and plot the RNA velocity field on top of tSNE projection. Conclude.**

R

```
embedded_velo <- velociraptor::embedVelocity(reducedDim(germcells, "TSNE"), velo_out)
head(embedded_velo)
grid_df <- velociraptor::gridVectors(reducedDim(germcells, "TSNE"), embedded_velo, resoluti
head(grid_df)
p <- scater::plotReducedDim(germcells, 'TSNE', colour_by = "annotation", point_alpha = 0.5)
    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")
    )
```

13.4. Session info

```
- Session info ---------------------------------------------------------------------------
```

```
 setting  value
 version  R version 4.3.0 (2023-04-21)
 os       macOS Monterey 12.5.1
 system   aarch64, darwin20
 ui       X11
 language (EN)
 collate  en_GB.UTF-8
 ctype    en_GB.UTF-8
 tz       Europe/Paris
 date     2023-06-05
 pandoc   2.19.2 @ /opt/homebrew/bin/ (via rmarkdown)

- Packages -----------------------------------------------------------------------
 package               * version   date (UTC) lib source
 AnnotationDbi         * 1.61.0    2022-11-01 [1] Bioconductor
 AnnotationHub         * 3.7.3     2023-03-01 [1] Bioconductor
 Biobase               * 2.59.0    2022-11-01 [1] Bioconductor
 BiocFileCache         * 2.7.2     2023-02-17 [1] Bioconductor
 BiocGenerics          * 0.45.0    2022-11-01 [1] Bioconductor
 BiocManager           * 1.30.20   2023-02-24 [1] CRAN (R 4.3.0)
 BiocVersion             3.17.1    2022-12-20 [1] Bioconductor
 Biostrings              2.67.0    2022-11-01 [1] Bioconductor
 bit                     4.0.5     2022-11-15 [1] CRAN (R 4.3.0)
 bit64                   4.0.5     2020-08-30 [1] CRAN (R 4.3.0)
 bitops                  1.0-7     2021-04-24 [1] CRAN (R 4.3.0)
 blob                    1.2.3     2022-04-10 [1] CRAN (R 4.3.0)
 cachem                  1.0.7     2023-02-24 [1] CRAN (R 4.3.0)
 callr                   3.7.3     2022-11-02 [1] CRAN (R 4.3.0)
 cli                     3.6.0     2023-01-09 [1] CRAN (R 4.3.0)
 colorspace              2.1-0     2023-01-23 [1] CRAN (R 4.3.0)
 cowplot               * 1.1.1     2020-12-30 [1] CRAN (R 4.3.0)
 crayon                  1.5.2     2022-09-29 [1] CRAN (R 4.3.0)
 curl                    5.0.0     2023-01-12 [1] CRAN (R 4.3.0)
 DBI                     1.1.3     2022-06-18 [1] CRAN (R 4.3.0)
 dbplyr                * 2.3.1     2023-02-24 [1] CRAN (R 4.3.0)
 devtools                2.4.5     2022-10-11 [1] CRAN (R 4.3.0)
 digest                  0.6.31    2022-12-11 [1] CRAN (R 4.3.0)
 dplyr                 * 1.1.0     2023-01-29 [1] CRAN (R 4.3.0)
 ellipsis                0.3.2     2021-04-29 [1] CRAN (R 4.3.0)
 evaluate                0.20      2023-01-17 [1] CRAN (R 4.3.0)
 fansi                   1.0.4     2023-01-22 [1] CRAN (R 4.3.0)
 fastmap                 1.1.1     2023-02-24 [1] CRAN (R 4.3.0)
 filelock                1.0.2     2018-10-05 [1] CRAN (R 4.3.0)
 forcats               * 1.0.0     2023-01-29 [1] CRAN (R 4.3.0)
```

```
fs                      1.6.1     2023-02-06 [1] CRAN (R 4.3.0)
generics                0.1.3     2022-07-05 [1] CRAN (R 4.3.0)
GenomeInfoDb            1.35.15   2023-02-03 [1] Bioconductor
GenomeInfoDbData        1.2.9     2022-11-04 [1] Bioconductor
ggplot2               * 3.4.1     2023-02-10 [1] CRAN (R 4.3.0)
glue                    1.6.2     2022-02-24 [1] CRAN (R 4.3.0)
gtable                  0.3.1     2022-09-01 [1] CRAN (R 4.3.0)
hms                     1.1.2     2022-08-19 [1] CRAN (R 4.3.0)
htmltools               0.5.4     2022-12-07 [1] CRAN (R 4.3.0)
htmlwidgets             1.6.1     2023-01-07 [1] CRAN (R 4.3.0)
httpuv                  1.6.9     2023-02-14 [1] CRAN (R 4.3.0)
httr                    1.4.5     2023-02-24 [1] CRAN (R 4.3.0)
interactiveDisplayBase  1.37.0    2022-11-01 [1] Bioconductor
IRanges               * 2.33.0    2022-11-01 [1] Bioconductor
jsonlite                1.8.4     2022-12-06 [1] CRAN (R 4.3.0)
KEGGREST                1.39.0    2022-11-01 [1] Bioconductor
knitr                   1.42      2023-01-25 [1] CRAN (R 4.3.0)
later                   1.3.0     2021-08-18 [1] CRAN (R 4.3.0)
lattice                 0.21-8    2023-04-05 [2] CRAN (R 4.3.0)
lifecycle               1.0.3     2022-10-07 [1] CRAN (R 4.3.0)
lubridate             * 1.9.2     2023-02-10 [1] CRAN (R 4.3.0)
magrittr                2.0.3     2022-03-30 [1] CRAN (R 4.3.0)
Matrix                  1.5-4     2023-04-04 [2] CRAN (R 4.3.0)
memoise                 2.0.1     2021-11-26 [1] CRAN (R 4.3.0)
mgcv                  * 1.8-42    2023-03-02 [2] CRAN (R 4.3.0)
mime                    0.12      2021-09-28 [1] CRAN (R 4.3.0)
miniUI                  0.1.1.1   2018-05-18 [1] CRAN (R 4.3.0)
munsell                 0.5.0     2018-06-12 [1] CRAN (R 4.3.0)
nlme                  * 3.1-162   2023-01-31 [2] CRAN (R 4.3.0)
pillar                  1.8.1     2022-08-19 [1] CRAN (R 4.3.0)
pkgbuild                1.4.0     2022-11-27 [1] CRAN (R 4.3.0)
pkgconfig               2.0.3     2019-09-22 [1] CRAN (R 4.3.0)
pkgload                 1.3.2     2022-11-16 [1] CRAN (R 4.3.0)
png                     0.1-8     2022-11-29 [1] CRAN (R 4.3.0)
prettyunits             1.1.1     2020-01-24 [1] CRAN (R 4.3.0)
processx                3.8.0     2022-10-26 [1] CRAN (R 4.3.0)
profvis                 0.3.7     2020-11-02 [1] CRAN (R 4.3.0)
promises                1.2.0.1   2021-02-11 [1] CRAN (R 4.3.0)
ps                      1.7.2     2022-10-26 [1] CRAN (R 4.3.0)
purrr                 * 1.0.1     2023-01-10 [1] CRAN (R 4.3.0)
R6                      2.5.1     2021-08-19 [1] CRAN (R 4.3.0)
rappdirs                0.3.3     2021-01-31 [1] CRAN (R 4.3.0)
Rcpp                    1.0.10    2023-01-22 [1] CRAN (R 4.3.0)
RCurl                   1.98-1.10 2023-01-27 [1] CRAN (R 4.3.0)
```

```
 readr                * 2.1.4     2023-02-10 [1] CRAN (R 4.3.0)
 remotes                2.4.2     2021-11-30 [1] CRAN (R 4.3.0)
 rlang                  1.0.6     2022-09-24 [1] CRAN (R 4.3.0)
 rmarkdown              2.20      2023-01-19 [1] CRAN (R 4.3.0)
 RSQLite                2.3.0     2023-02-17 [1] CRAN (R 4.3.0)
 S4Vectors            * 0.37.4    2023-02-26 [1] Bioconductor
 scales                 1.2.1     2022-08-20 [1] CRAN (R 4.3.0)
 sessioninfo            1.2.2     2021-12-06 [1] CRAN (R 4.3.0)
 shiny                  1.7.4     2022-12-15 [1] CRAN (R 4.3.0)
 stringi                1.7.12    2023-01-11 [1] CRAN (R 4.3.0)
 stringr              * 1.5.0     2022-12-02 [1] CRAN (R 4.3.0)
 tibble               * 3.1.8     2022-07-22 [1] CRAN (R 4.3.0)
 tidyr                * 1.3.0     2023-01-24 [1] CRAN (R 4.3.0)
 tidyselect             1.2.0     2022-10-10 [1] CRAN (R 4.3.0)
 tidyverse            * 2.0.0     2023-02-22 [1] CRAN (R 4.3.0)
 timechange             0.2.0     2023-01-11 [1] CRAN (R 4.3.0)
 tzdb                   0.3.0     2022-03-28 [1] CRAN (R 4.3.0)
 urlchecker             1.0.1     2021-11-30 [1] CRAN (R 4.3.0)
 usethis                2.1.6     2022-05-25 [1] CRAN (R 4.3.0)
 utf8                   1.2.3     2023-01-31 [1] CRAN (R 4.3.0)
 vctrs                  0.5.2     2023-01-23 [1] CRAN (R 4.3.0)
 withr                  2.5.0     2022-03-03 [1] CRAN (R 4.3.0)
 xfun                   0.37      2023-01-31 [1] CRAN (R 4.3.0)
 xtable                 1.8-4     2019-04-21 [1] CRAN (R 4.3.0)
 XVector                0.39.0    2022-12-20 [1] Bioconductor
 yaml                   2.3.7     2023-01-23 [1] CRAN (R 4.3.0)
 zlibbioc               1.45.0    2022-12-20 [1] Bioconductor

 [1] /Users/jacques/Library/R/arm64/4.3/library
 [2] /Library/Frameworks/R.framework/Versions/4.3-arm64/Resources/library

------------------------------------------------------------------------------
```

14. Lecture 7 - Advances in single-cell genomics: the epigenome

Slides here

15. Lab 8 - Single-cell ATAC-seq analysis workflow

ATAC-seq data may be obtained in isolation using a single-cell ATAC-seq protocol (e.g. 10X scATACseq) or in combination with gene expression data using a single-cell multiome protocole (e.g. 10X multiome and SHARE-seq).

Several packages are currently avaialble to process scATAC-seq data in R. These include `[Signac](https://satijalab.org/signac/index.html)` and `[ArchR](https://www.archrproject.com/book` This lab will closely follow the processing steps outlined in `Signac`, which interfaces well with `Seurat` for single-cell analysis.

In this lab, we will process a PBMC single-cell ATAC-seq (scATAC-seq) dataset and perform preliminary analysis to assess quality of these data. The data for this lab comes from 10X Genomics. The dataset contains roughly ~ 10,000 cells.

Overarching goals

1. Import cells from human PBMC scATACseq dataset
2. Perform scATACseq quality controls and checks
3. Filter, normalize and plot PBMC scATACseq dataset
4. Run chromVAR to identify TF activity in each cell cluster
5. Compute gene activity scores and check known markers

15.1. Process human PBMC dataset

15.1.1. Download data

- **Download the files related to scATACseq of all human PBMC cells.**

Data comes from 10X Genomics.

Direct download links are provided below

R

```
dir.create('scATAC')
download.file("https://cf.10xgenomics.com/samples/cell-atac/1.0.1/atac_v1_pbmc_10
download.file("https://cf.10xgenomics.com/samples/cell-atac/1.0.1/atac_v1_pbmc_10
download.file("https://cf.10xgenomics.com/samples/cell-atac/1.0.1/atac_v1_pbmc_10
download.file("https://cf.10xgenomics.com/samples/cell-atac/1.0.1/atac_v1_pbmc_10
download.file("https://cf.10xgenomics.com/samples/cell-atac/1.0.1/atac_v1_pbmc_10
```

15.1.2. Import data

Notice how the count matrix is in a `.h5` format. We have already encountered this format in Lab3. Back then, we imported it with `DropletUtils::read10xCounts`.

- **Does this function work here?**

R

```
DropletUtils::read10xCounts("scATAC/atac_v1_pbmc_10k_filtered_peak_bc_matrix.h5")
```

This works because 10X Genomics make sure to distribute files in `.h5` format that are consistent accross single-cell sequencing mthodologies. However, the `SingleCellExperiment` obtained with this approach is not the most convenient, as it cannot natively leverage `fragments` file (see below).

Instead, we can create a `Signac` object, a flavour of `Seurat` objects.

- **Import counts matrix and feature annotations using an import function provided by `Seurat`.**

R

```
library(Seurat)
library(Signac)
library(rtracklayer)
library(stringr)
cnts <- Read10X_h5('scATAC/atac_v1_pbmc_10k_filtered_peak_bc_matrix.h5')
features <- import('scATAC/atac_v1_pbmc_10k_peaks.bed')
features$peak <- as.character(features) |> str_replace(':', '-')
metadata <- read.csv(
  file = "scATAC/atac_v1_pbmc_10k_singlecell.csv",
  header = TRUE,
  row.names = 1
)
```

- **How many accessible genomic segments were found in this dataset?**

R

```
features
length(features)
```

15.1.3. Create a `Seurat` object

The next step is to aggregate `counts` and `features` into a `ChromatinAssay`, a scATAC-seq flavour of `Seurat` standard `Assays`. The documentation for `?CreateChromatinAssay` indicates that the user can provide:

1. A `fragments` file, corresponding to the full list of all unique fragments mapped across all single cells.
2. Genomic annotations to the `ChromatinAssay`, corresponding to gene annotations, promoter positions, etc. Such annotations can be generated from Ensembl.

- **Generate human annotations from Ensembl using a parsing function from `Seurat`.**

R

```
## - Get human gene annotations (hg19/GRCh37) to feed it into the future `ChromatinAssay`
BiocManager::install('EnsDb.Hsapiens.v75')
annotations <- GetGRangesFromEnsDb(ensdb = EnsDb.Hsapiens.v75::EnsDb.Hsapiens.v75)
seqlevelsStyle(annotations) <- 'UCSC'
```

- **Create a `ChromatinAssay` using `counts`, `features`, `fragments` and `annotations`.**

R

```
## - Create Chromatin Assay
assay <- Signac::CreateChromatinAssay(
    counts = cnts,
    ranges = features,
    fragments = "scATAC/atac_v1_pbmc_10k_fragments.tsv.gz",
    annotation = annotations,
    genome = "hg19",
    min.cells = 10,
    min.features = 10
)
assay
```

- **What are the dimensions of this object? Are they comparable to the count matrix? Comment.**

R

```
dim(cnts)
dim(assay)
```

It's finally time to wrap the `ChromatinAssay` into a `Seurat` standard object. This is done using the `CreateSeuratObject`, as already covered in Lab6

- **Create a `PBMC Seurat` object.**

R

```
## - Create Seurat object
PBMC <- Seurat::CreateSeuratObject(
    counts = assay,
    assay = 'ATAC',
    meta.data = metadata[metadata$is__cell_barcode == 1, ]
)
PBMC

PBMC[['ATAC']]
granges(PBMC)
Annotation(PBMC)
```

15.1.4. Check QCs

15.1.4.1. Cell-based QCs

The fraction of reads in peaks (FRiP) is a good indicator of how well each cell was handled during scATACseq processing.

R

```
PBMC@meta.data$FRiP <- PBMC$peak_region_fragments / PBMC$passed_filters
PBMC@meta.data$nCount_ATAC <- colSums(GetAssayData(PBMC, slot = "counts"))
PBMC@meta.data$nFeature_ATAC <- colSums(GetAssayData(PBMC, slot = "counts") > 0)

quantile(PBMC$FRiP, seq(0, 1, 0.1))
quantile(PBMC$nCount_ATAC, seq(0, 1, 0.1))
quantile(PBMC$nFeature_ATAC, seq(0, 1, 0.1))
```

15.1.4.2. Peaks-based QCs

- **Which analysis are the `fragments` required for, exactly?**
- **Could we still perform normalization/clustering/annotation without them? And motif enrichment analysis?**

Since we do have the `fragments` file at hand, most of the QC steps are available (e.g. `TSSEnrichment`, `NucleosomeSignal` or fragment size distribution). Let's go through them one by one.

R

```
# compute nucleosome signal score per cell
PBMC <- NucleosomeSignal(object = PBMC)

# compute TSS enrichment score per cell
PBMC <- Signac::TSSEnrichment(object = PBMC, fast = FALSE)
```

The `TSSPlot` function from `Signac` can be used to plot the fragment count per peak ~ TSS enrichment.

R

```
PBMC$high.tss <- ifelse(PBMC$TSS.enrichment > 3.5, 'High', 'Low')
TSSPlot(PBMC, group.by = 'high.tss') + NoLegend()
PBMC$high.tss <- ifelse(PBMC$TSS.enrichment > 2.5, 'High', 'Low')
TSSPlot(PBMC, group.by = 'high.tss') + NoLegend()
```

The `FragmentHistogram` function from `Signac` can be used to plot the fragment size distribution in peaks with different nucleosome signals.

R

```
PBMC$nucleosome_group <- ifelse(PBMC$nucleosome_signal > 4, 'NS > 4', 'NS < 4')
FragmentHistogram(object = PBMC, group.by = 'nucleosome_group')
```

The new `DensityScatter` function from `Signac` can be used to plot the fragment count per peak ~ TSS enrichment, combining both metrics into a single plot.

R

```
p <- DensityScatter(PBMC, x = 'TSS.enrichment', y = 'nucleosome_signal', log_x = FALSE, qua
```

15.1.5. Filter cells and features

We do have other pre-computed metrics (e.g. `FRIP` and `depth`).

- **Check these metrics and filter the `Seurat` object (cells and features) as deemed appropriate.**

R

```
## - Filter data
PBMC <- subset(PBMC, subset = nCount_ATAC > 3000 & nCount_ATAC < 100000)
PBMC <- subset(PBMC, subset = nFeature_ATAC > 1000 & nFeature_ATAC < 20000)
PBMC <- subset(PBMC, subset = FRIP > 0.30)
## - Remove peaks with low coverage
PBMC <- PBMC[rowSums(GetAssayData(PBMC, slot = "counts")) > 10, ]
PBMC <- PBMC[rowSums(GetAssayData(PBMC, slot = "counts") > 0) > 10, ]
```

15.1.6. Dimensionality reduction and clustering

- **Now that the dataset is filtered, normalize (by using TF-IDF approach) then further reduce the dimensionality for visualization purposes.**

R

```
## - Normalize data
PBMC <- RunTFIDF(PBMC)

## - Reduce dimensionality
PBMC <- FindTopFeatures(PBMC, min.cutoff = 'q50')
PBMC <- RunSVD(PBMC)

## - Label clusters according to the original publication
PBMC <- RunUMAP(object = PBMC, reduction = 'lsi', dims = 2:30)
PBMC <- FindNeighbors(object = PBMC, reduction = 'lsi', dims = 2:30)
PBMC <- FindClusters(object = PBMC, verbose = FALSE, algorithm = 3)

## - Visualize data
p <- DimPlot(PBMC, label = TRUE) + NoLegend()
```

- **What can you observe in the `UMAP` projection of the dataset? Comment on the separation of some cell types into different spatially-resolved clusters.**

15.2. chromVAR analysis

15.2.1. Get a SummarizedExperiment of scATACseq counts over peaks

`chromVAR` works with raw counts stored as a `RangedSummarizedExperiment`. By now, you should be able to easily extract the raw counts from a specific `Seurat` assay. Store it in a `RangedSummarizedExperiment` (~ equivalent to `Seurat` assay but in `Bioconductor`).

- **What are the dimensions of the `RangedSummarizedExperiment` generated? What are the features and what are the columns?**

R

```
library(SummarizedExperiment)
sumExp <- SummarizedExperiment(
    assays = list('counts' = GetAssayData(PBMC, assay = 'ATAC', slot = 'counts')),
    rowRanges = GetAssayData(PBMC, assay = 'ATAC', slot = 'ranges'),
    colData = PBMC[[]]
)
dim(sumExp)
```

15.2.2. Add GC bias to peaks

This step is important to correct GC bias associated with each peak.

R

```
sumExp <- chromVAR::addGCBias(sumExp, genome = BSgenome.Hsapiens.UCSC.hg19::BSgenome.Hsapie
```

15.2.3. Map motifs over peaks

Using `motifmatchr` package, one can map a list of motifs (e.g. from JASPAR database) over a `RangedSummarizedExperiment` (or anything which can be coerced to a `GRanges`, actually).

- **First, you need to import motifs from a public database in `R`. This can be done (among other ways) with the `TFBSTools` pacakge.**

R

```
## - Import motifs from JASPAR database
motifs <- TFBSTools::getMatrixSet(
    JASPAR2020::JASPAR2020,
    list(species = 9606, all_versions = FALSE)
)
names(motifs) <- TFBSTools::name(motifs)
```

- **Now map the subset of motifs of interest to the filtered peaks, using `motifmatchr::matchMotifs()`.**
- **Read `matchMotifs()` documentation and run it on your `sumExp`.**
- **What is the class of the generated object? Its dimensions?**
- **What are the features and what are the columns?**

R

```
library(BiocParallel)
register(MulticoreParam(workers = 16, progressbar = TRUE))
motif_mappings <- motifmatchr::matchMotifs(motifs, sumExp, genome = BSgenome.Hsap
class(motif_mappingss)
dim(motif_mappings)
```

Because there are so many motifs and so many peaks, and in interest of time, we will perform `chromVAR` analysis on a subset of motifs only. You can manually filter the motifs list to motifs that may be relevant in our context (e.g. `EOMES`, or `GATA2`, …).

R

```
## - Filter to interesting motifs to keep analysis relatively quick
#       Feel free to select any TF of interest!
motif_mappings_sub <- motif_mappings[, c('EOMES', 'GATA2', 'IRF8', 'EBF1', 'SPI1'
```

15.2.4. Search for motifs with high deviation of mapping compared to background

`chromVAR`'s `computeDeviations()` function combines (1) the peak counts / cell (stored here in `sumExp`) and (2) the TF mapping over each peak (stored here in `motif_mappings`). to assess the mapping deviation for each TF over each cell compared to other cells.

- **Run `computeDeviations()` on the set of peaks and motifs.**
- **What is the class of the generated object? Its dimensions?**
- **What are the features and what are the columns?**

R

```
## - Find background signal
bg <- chromVAR::getBackgroundPeaks(object = sumExp)
## - For each motif, compute its mapping deviation over the filtered peaks
motif_deviations <- chromVAR::computeDeviations(object = sumExp, annotations = motif_mappin
```

15.2.5. Check TF motif enrichment in different cell types

The `motif_deviations` object can be added as a new assay to the `PBMC Seurat` object. This way, one can rely on `Seurat`-based plotting functions to plot cells in their preferred dimensional space, and color them using motif deviation scores computed with `chromVAR`.

R

```
PBMC[['MOTIF']] <- Seurat::CreateAssayObject(counts = chromVAR::deviationScores(motif_devia
DefaultAssay(PBMC) <- 'MOTIF'
list_p <- lapply(rownames(PBMC), function(motif) {
    FeaturePlot(PBMC, features = motif, reduction = "umap") +
        scale_colour_gradientn(
            colors = c('#190886', '#6F07F8', '#F954A5', '#FF9D66', '#edf118'),
            limits = c(-5, 5), oob = scales::squish
        ) +
        theme(aspect.ratio = 1)
})
list_p[[length(list_p) + 1]] <- DimPlot(PBMC, group.by = 'renamed_clusters', reduction = "u
p <- cowplot::plot_grid(plotlist = list_p)
```

15.3. Compute gene activity scores

Signac's `GeneActivity()` function require scATACseq fragment information. Since we have them, we can estimate a gene activity score for each gene in the `annotations`.

R

```
gene.activities <- GeneActivity(PBMC)
```

We can now save this new object as an `Assay` in the `PBMC` object and normalize it.

R

```
PBMC[['RNA']] <- CreateAssayObject(counts = gene.activities)

# - Normalize the new RNA assay, this time with `SCTransform`
PBMC <- SCTransform(
  object = PBMC,
  assay = 'RNA',
)

PBMC
DefaultAssay(PBMC) <- "SCT"
```

One can now perform "gene differential expression"-like analysis using the `SCT` assay!

R

```
## - Reduce dimensionality to visualize cells in 2D
PBMC <- RunPCA(PBMC) |> RunUMAP(reduction = 'pca', dims = 1:50)

## - Plot gene expression
p <- FeaturePlot(PBMC, features = 'MS4A1', reduction = "umap") # Or NCR1 (NKp46),
```

Now, we can leverage this to annotate clusters according to known PBMC markers

R

```
## - Reduce dimensionality to visualize cells in 2D
PBMC <- RunPCA(PBMC) |> RunUMAP(reduction = 'pca', dims = 1:50)

## - Plot gene expression
p <- FeaturePlot(PBMC, features = 'MS4A1', reduction = "umap") # Or NCR1 (NKp46),
```

15.4. Find differentially accessible peaks

R

```
# change back to working with peaks instead of gene activities
DefaultAssay(pbmc) <- 'peaks'

da_peaks <- FindMarkers(
  object = pbmc,
  ident.1 = "CD4 Naive",
  ident.2 = "CD14+ Monocytes",
```

```
  test.use = 'LR',
  latent.vars = 'nCount_peaks'
)



plot1 <- VlnPlot(
  object = pbmc,
  features = rownames(da_peaks)[1],
  pt.size = 0.1,
  idents = c("CD4 Naive","CD14+ Monocytes")
)
plot2 <- FeaturePlot(
  object = pbmc,
  features = rownames(da_peaks)[1],
  pt.size = 0.1
)

plot1 | plot2
```

Part V.

Day 5

16. Lecture 8 - Advances in single-cell genomics: spatial transcriptomics

Slides here

Extra 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.

General bioinformatics

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

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

Scientific readings

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