Getting Started with scRNA-Seq Seminar Series

BTEP Team
BTEP/GAU/CCR/NCI/NIH - email ncibtep@mail.nih.gov
Bioinformatics Training and Education Program

Table of Contents

Getting Started with Seurat: QC to Clustering

Getting Started with Seurat: Differential Expression and Classification

The CCBR Single-cell RNA-seq Workflow on NIDAP

Additional Resources

Welcome to *Getting Started with scRNA-Seq*

This is a mini seminar series designed to help attendees learn more about single cell RNA-Seq, from applicable technologies to data analysis.

Seminar Schedule

April 3, 2024 - *The CCR Single Cell Analysis Facility (SCAF): An Overview* (Mike Kelly, SCAF) (Recording *(https://cbiit.webex.com/cbiit/ldr.php?RCID=18785288d22f105303a3d215300fd805)*)

April 10, 2024 - *Introduction to single cell RNA-Seq* (Charlie Seibert, Saeed Yadranji Aghdam, SCAF) (Recording *(https://cbiit.webex.com/cbiit/ldr.php?RCID=ee12bfaf912d5104b3fc3a837f99f096)*)

April 17, 2024 - *SCAF: Overview of Cell Ranger output files and single cell data analysis quality control* (Kimia Dadkhah, SCAF) (Recording *(https://cbiit.webex.com/cbiit/ldr.php?RCID=875af03d230855d7615079bd195002c7)*)

April 24, 2024 - *Introduction to scRNA-Seq with R (Seurat)* (Alex Emmons, BTEP) (Recording *(https://cbiit.webex.com/cbiit/ldr.php?RCID=80664d03945034fb9aedb4055be7114c)*)

May 1, 2024 - *Getting Started with Seurat: QC to Clustering* (Alex Emmons, BTEP) (Recording *(https://cbiit.webex.com/cbiit/ldr.php?RCID=2704d23912bf4821ee86a9f4a2c29f67)*)

May 22, 2024 (Rescheduled from May 8th) - *Differential Expression Analysis with Seurat* (Nathan Wong, CCBR) (Recording *(https://cbiit.webex.com/cbiit/ldr.php?RCID=8090bb2774d91198f92ff2ca333ce8de)*)

May 29, 2024 - *The CCBR Single-cell RNA-seq Workflow on NIDAP* (Josh Meyer, CCBR) (Recording *(https://cbiit.webex.com/cbiit/ldr.php?RCID=0f3bb1e17eeaba11e4d0a59aee7db055)*)

Questions

Please email us at ncibtep@nih.gov *(mailto:ncibtep@nih.gov)* with questions, comments, or concerns.

Recordings

All seminars will be recorded and made available on the BTEP Video Archive 24 to 48 hours following the event.

The CCR Single Cell Analysis Facility (SCAF): An Overview

The Single Cell Analysis Facility (SCAF) *(https://crtp.ccr.cancer.gov/labs/scaf/)* is a CCR facility dedicated to the application of single-cell technologies in cancer research. Based on the NIH Bethesda main campus, SCAF aims to provide the broadest range of project support from consultation on experimental design, sequencing, and data analysis. Learn more about SCAF and the single-cell genomics technologies available to CCR investigators in this overview presentation.

Introduction to single cell RNA-Seq

Single cell RNA sequencing (scRNA-Seq) is becoming increasingly more common in biomedical research, but what is scRNA-Seq? How does it differ from other transcriptomic approaches (e.g., bulk RNA-Seq), and what are the potential applications, technologies, and workflows? This presentation will introduce learners to scRNA-Seq, answering the above and touching on additional topics such as methodological challenges, concerns, and best practices.

SCAF: Overview of Cell Ranger output files and single cell data analysis quality control

Kimia Dadkhah, bioinformatics analyst (SCAF), will talk about Cell Ranger output and essential quality control metrics in single cell data analysis, how to interpret these and make informed decisions, and other considerations to keep in mind when assessing the quality of returned data from SCAF.

Introduction to scRNA-Seq with R (Seurat)

This lesson provides an introduction to R in the context of single cell RNA-Seq analysis with Seurat.

Learning Objectives

- Learn about options for analyzing your scRNA-Seq data.
- Learn about resources for learning R programming.
- Learn how to import your data for working with R.
- Learn about Seurat and the Seurat object including how to create the object and access and manipulate stored data.

Exploratory Data Analysis: scRNA-Seq

This tutorial assumes that all pre-processing steps (read demultiplexing, FASTQ QC, reference based alignment, error correction) have been completed. At this stage of the analysis, a gene-by-cell count matrix has been generated for each sample.

Analysis Ecosystems

Following Cell Ranger and/or other pre-processing tools, you will have a gene-by-cell counts table for each sample. The three most popular frameworks for analyzing these count matrices include:

1. R (Seurat *(https://satijalab.org/seurat/articles/get_started_v5_new)*).
    - Seurat, brought to you by the Satija lab, is a kind of one-stop shop for single cell transcriptomic analysis (scRNA-seq, multi-modal data, and spatial transcriptomics). It has a wide user base and is scalable, especially with Seurat v5. It is a great place to get started analyzing your data.
2. R (Bioconductor *(https://bioconductor.org/books/release/OSCA/)*)
    - If you are familiar with Bioconductor, a project and package repository for analyzing biological data, the Bioconductor ecosystem is a fantastic choice. This workflow will use many different, but highly interoperable packages. *Orchestrating Single-Cell Analysis with Bioconductor (https://bioconductor.org/books/release/OSCA/)* is a fantastic resource for basic to more complicated types of analyses.

3. Python (scverse *(https://scverse.org/learn/)*)
    - `scverse`, which is a multi-package ecosystem in python (core packages: `scanpy`, `muon`, `scvi-tools`, `scirpy`, and `squidpy`), is the obvious choice for python users, and boasts advantages such as "scalability, extendability, and strong interoperability." *(https://www.sc-best-practices.org/introduction/analysis_tools.html)*

These options require some knowledge of either R or Python programming languages. Which one you choose will most likely depend on your preference, your PI's preference, and your goals. **Here we will focus on the Seurat ecosystem.**

Note

You may benefit by working with tools from all three of these ecosystems. This can be done by converting object types using a variety of packages (e.g. `Seurat`, `sceasy`, `zellkonverter`). However, there may be some hurdles; for example, the Seurat function `as.SingleCellExperiment()` does not seem to work with Seurat v5 layers. Therefore, hopping between systems may require some degree of web searching or creative workarounds.

Point-and-click alternatives

If you lack programming skills and learning to code seems daunting, there are GUI based alternatives including, but not limited to the following:

- Partek Flow (License available to NCI researchers *(https://bioinformatics.ccr.cancer.gov/btep/partek-flow-bulk-and-single-cell-rna-seq-data-analysis/)*)
- Relevant trainings:
    - Single cell RNA sequencing analysis with Partek Flow *(https://cbiit.webex.com/cbiit/ldr.php?RCID=ce7fe9d4b9f3eb3419434ae731a42a2f)*
    - Basic single cell RNA sequencing analysis *(https://cbiit.webex.com/cbiit/ldr.php?RCID=3af6ad96ae7b0da3753611ed85ceacbc)*
    - Advanced single cell RNA sequencing analysis: cell type classification and comparison of gene expression among treatment groups, CITE-seq, and spatial transcriptomics *(https://cbiit.webex.com/cbiit/ldr.php?RCID=a9e4ea60bf8674f5298e640270d35757)*
- Qlucore Omics Explorer (License available to NCI-CCR researchers *(https://bioinformatics.ccr.cancer.gov/btep/qlucore-bioinformatics-software-for-next-gen-seq-analysis/)*)
- NIH Integrated Data Analysis Platform
    - Learn about the capabilities of the NIDAP platform for transcriptomics here *(https://cbiit.webex.com/cbiit/ldr.php?RCID=14ad1204eaf2612ee650e4d6fdf8bc53)* (with accompanying slides *(https://bioinformatics.ccr.cancer.gov/btep/wp-content/uploads/sites/2/NIDAP_Overview_20231214.pdf)*).
    - Training videos available here. *(https://bioinformatics.ccr.cancer.gov/ccbr/education-training/nidap-training/)*

What is R?

R is both a computational language and environment for statistical computing and graphics. It is open-source and widely used by scientists, not just bioinformaticians. Base packages of R

are built into your initial installation, but R functionality is greatly improved by installing other packages.

R is a particularly great resource for statistical analyses, plotting, and report generating. The fact that it is widely used means that users do not need to reinvent the wheel. There is a package available for most types of analyses, and if users need help, it is only a Google search away. As of now, CRAN houses +20,000 available packages. There are also many field specific packages, including those useful in the -omics (genomics, transcriptomics, metabolomics, etc.). For example, the latest version of Bioconductor (v 3.18) includes 2,266 software packages, 429 experiment data packages, 920 annotation packages, 30 workflows, and 4 books.

What is RStudio?

RStudio is an integrated development environment for R, and now python. RStudio includes a console, editor, and tools for plotting, history, debugging, and work space management. It provides a graphic user interface for working with R, thereby making R more user friendly. RStudio is open-source and can be installed locally or used through a browser (RStudio Server or Posit Cloud).

What is Seurat?

> Seurat is an R package designed for QC, analysis, and exploration of single-cell RNA-seq data. Seurat aims to enable users to identify and interpret sources of heterogeneity from single-cell transcriptomic measurements, and to integrate diverse types of single-cell data. ---https://satijalab.org/seurat/ *(https://satijalab.org/seurat/)*

Advantages:

- beginning-to-end workflows
- Nice vignettes
- Regularly updated
- Somewhat extensible
- Scalable

Disadvantages:

- updates break some functionality
- maintained by a single group
- Many functions are poorly documented

Updates with Seurat v5

Seurat v5 introduced the following new features:

- Integrative multi-modal analysis with bridge integration

- 'Sketch'-based analysis of large data sets
- methods for spatial transcriptomics
- assay layers

You can read about major changes between Seurat v5 and v4 here *(https://satijalab.org/seurat/articles/announcements.html)*.

How much R programming do I need to know to use Seurat?

The answer to this question will largely depend on the user. While some will be able to learn R on the go, others will need to know some amount of R programming prior to beginning their analysis. If needed, check out the quick R refresher under **Things to know about R**. There are also several R courses available through BTEP with self-contained, asynchronous content that can be used to learn independently of live sessions.

The most important thing to know is how to find help!

How to find help?

ALWAYS, ALWAYS, ALWAYS read the documentation.

Use the help pane in the lower right of RStudio or the functions, `help()` and `help.search()` or `?` and `??`.

Check out package vignettes (`vignette()`).

Check the Github site if one is available for help with a specific package. There may also be known issues listed under the Github Issues tab (Seurat issues *(https://github.com/satijalab/seurat/issues)*).

Google for help!

Use Google to troubleshoot error message or simply find help performing a specific task. But, make sure you are precise and informative in your search. Stack Overflow is a particularly great resource. A good Google search includes 3 elements:

1. The specific action (e.g., how to rename a column).
2. The programming language (e.g., R programming).
3. The specific style/technique for coding (e.g., dplyr or tidyverse package) Example: "How to rename a column in R with dplyr/tidyverse". --- https://crd230.github.io/tips.html *(https://crd230.github.io/tips.html)*.

If this fails, feel free to email us at BTEP *(mailto:ncibtep@nih.gov)*.

Things to know about R

R can be accessed from the command line using `R`, which opens the R console, or it can be accessed via and Integrated development environment (IDE) (e.g., RStudio, VSCode, etc.). R commands can be submitted together in a script or interactively in a console.

You can install and use R locally or via an HPC such as the NIH HPC Biowulf *(https://hpc.nih.gov/apps/R.html)*.

R is case sensitive, so avoid typos, and space agnostic.

How to navigate directories

- `setwd()` Set working directory (equivalent to `cd`)
- `getwd()` Get working directory (equivalent to `pwd`)

Getting help

- `help()` and `?` "provide access to the documentation pages for R functions, data sets, and other objects".
- `help.search()` "allows for searching the help system for documentation matching a given character string in the (file) name, alias, title, concept or keyword entries (or any combination thereof)"; equivalent to `?? pattern`

See more on getting help here *(https://www.r-project.org/help.html)*.

Installing and loading packages

To take full advantage of R, you need to install R packages. R packages are loadable extensions that contain code, data, documentation, and tests in a standardized shareable format that can easily be installed by R users. The primary repository for R packages is the Comprehensive R Archive Network (CRAN) *(https://cran.r-project.org/)*. CRAN is a global network of servers that store identical versions of R code, packages, documentation, etc (cran.r-project.org).

> An R library is, effectively, a directory of installed R packages which can be loaded and used within an R session. ---renv *(https://rstudio.github.io/renv/articles/renv.html)*

- `install.packages()` install packages from CRAN
- `library()` load packages in R session
- `BiocManager::install()` install packages from Bioconductor
- `devtools::install_github()` install an R package from Github.

`.libPaths()` reports the directory where your installed R packages are located.

Commenting

You can annotate your code by starting annotations with #. Comments to the right of # will be ignored by R.

Use `# ----` to create navigable code sections.

For report generation, use R Markdown *(https://rmarkdown.rstudio.com/)* or Quarto *(https://quarto.org/)*.

Assignment operators

Anything that you want assigned to memory must be assigned to an R object.

`<-` the primary assignment operator, assigning values on the right to objects on the left.
= can also be used to assign values to objects, but is usually reserved for other purposes (e.g., function arguments)

Use `ls()` to list objects created in R. `rm()` can be used to remove an object from memory.

For R objects names,
avoid spaces or special characters, excluding "_" and ".".
do not begin with numbers or underscores.
* do not use names with special meanings (`?Reserved`).

Object data types

The base data type (e.g., numeric, character, logical, etc.) and the class (dataframe, matrix, etc.) will be important for what you can do with an object. Learn more about an object with the following:

- `class()` returns the class of an object or base data type
- `str()` returns the structure of an object.

dplyr::glimpse()

Similar to `str()` but with much more succinct output.

Coercion is when converting from one type to another, which may throw various warning messages. Always make sure output matches expectations.

Importing and exporting data

Use the `read` functions to import data (e.g., `read.csv`, `read.delim`, etc.). Use `write` functions to export data (e.g., `write.table`).

There are specific functions for unique data. For example, we will learn how to specifically import scRNA-seq data using Seurat.

Using functions

R functions perform specific tasks. R has a ton of built-in functions and functions available through additional packages. You can also create your own functions.

The general syntax for a function is the name followed by parentheses, `function_name()` (e.g., `round()`).

To create a function:

```
function_name <- function(arg_1, arg_2, ...) {
  Function body
}
```

Vectors

> A vector is a collection of values that are all of the same type (numbers, characters, etc.) --- datacarpentry.org *(https://datacarpentry.org/genomics-r-intro/02-r-basics/index.html)*

- `c()` - used to combine elements of a vector

When you combine elements of different types in the same vector, they are forced into the same type via "coercion" (logical < numeric < character).
* `length()` - returns the number of elements in a vector

Use brackets to extract elements of a vector:

```
a <- 1:10
a[2]
```

Lists

Unlike vectors, lists can hold values of different types.

```
list(1, "apple", 3)
```

Data frames

Data frames hold tabular data comprised of rows and columns; they can be created using `data.frame()`.

To understand more about the structure of an object and data frame, consider the following functions:

- `str()` displays the structure of an object, not just data frames
- `dplyr::glimpse()` similar to `str` but applies to data frames and produces cleaner output
- `summary()` produces result summaries of the results of various model fitting functions
- `ncol()` returns number of columns in data frame
- `nrow()` returns number of rows of data frame
- `dim()` returns row and column numbers
- `unique()` returns a vector of with duplicates removed; also see `dplyr::distinct()`

We can subset data frames using bracket notation `df[row,column]`:

```
df<- data.frame(Counts=seq(1,5), animals=c("racoon","squirrel","bird","dog","cat"))
#to return just the animals column
df[,"animals"]
```

We can also use functions from `dplyr` such as `filter()` for subsetting by row and `select()` for subsetting by column.

Plotting

There are 3 primary plotting systems with R: base R, `ggplot2`, and `lattice`. Data visualization functions from Seurat primarily use `ggplot2` and can easily be customized by adding additional `ggplot2` layers.

Check out the R Graph Gallery *(https://r-graph-gallery.com/)* for data visualization examples and code.

Conditionals and Looping

See the attached resources on

- for loops *(https://datacarpentry.org/semester-biology/materials/for-loops-R/)*
- apply functions *(https://bookdown.org/rdpeng/rprogdatascience/loop-functions.html#vectorizing-a-function)*

- purr::map *(https://r4ds.had.co.nz/iteration.html#the-map-functions)*
- conditionals *(https://datacarpentry.org/semester-biology/materials/conditionals-R/)*.

Getting info on R Session

`sessionInfo()` Print version information about R, the OS and attached or loaded packages. This is useful for reporting methods for publication. Consider using the package `renv` *(https://rstudio.github.io/renv/articles/renv.html)* to track and share exact versions of packages used for any given R script / project.

Resources for learning R

- Base R cheat sheet *(https://iqss.github.io/dss-workshops/R/Rintro/base-r-cheat-sheet.pdf)*
- Other cheat sheets can be found here *(https://posit.co/resources/cheatsheets/)*.
- There is also a nice review here *(https://cosima.nceas.ucsb.edu/r-self-assessment/#section-r-overview)*.
- There are a ton of free tutorials. There are at least 230 Git repositories that focus on R training related to bioinformatics. Check out Glittr *(https://glittr.org/?per_page=25&sort_by=stargazers&sort_direction=desc&tags=61)*.
- DataQuest and Coursera licenses are available to CCR and NIH researchers respectively. See here *(https://bioinformatics.ccr.cancer.gov/btep/self-learning/)* for more information.

BTEP lessons / courses

- R Introductory Series *(https://bioinformatics.ccr.cancer.gov/docs/rintro/index.html)*
- Data Wrangling with R *(https://bioinformatics.ccr.cancer.gov/docs/data-wrangle-with-r/)*
- Data Visualization with R *(https://bioinformatics.ccr.cancer.gov/docs/data-visualization-with-r/index.html)*
- Toward Reproducibility with R on Biowulf *(https://bioinformatics.ccr.cancer.gov/docs/reproducible-r-on-biowulf/)*
- A Beginner's Guide to Troubleshooting R Code *(https://bioinformatics.ccr.cancer.gov/docs/btep-coding-club/CC2023/Troubleshooting_Rprog/)*

Seurat Computational Requirements

> Regarding computational requirements, rule of thumb is to keep it at 8 samples or less when running on a laptop. You may be able to stretch it to 10 samples, but any more than that strains the computer, especially when running batch correction or imaging. The imaging can be somewhat mitigated by printing the image to a file with the `png` or `pdf` commands, but it will definitely hog as much memory as it can get. --- CCBR Analyst

Note

Using the package `BPCells` to manage SeuratObjects can make your analysis more computationally efficient. We will create a merged seurat object in this tutorial, which will be ~ 2.1 GB. Using `BPCells`, the same object is ~63.9 MB.

See this vignette *(https://satijalab.org/seurat/articles/seurat5_bpcells_interaction_vignette)* for more information.

Working with R on Biowulf

Due to limits on computational resources, you may be interested in running your analysis on an HPC. Biowulf is the NIH high performance compute cluster. It has greater than 90k processors, and can easily perform large numbers of simultaneous jobs. Biowulf also includes greater than 600 pre-installed scientific software and databases.

You will need to know the basics of working on the command line to use Biowulf. Minimally, you should be able to navigate a directory tree using basic Unix commands. If the command line is foreign to you, check out this introduction *(https://bioinformatics.ccr.cancer.gov/docs/unix-on-biowulf-2024/)*.

Warning

The default version of R on Biowulf is currently 4.3.2 (`module -r avail '^R$'`), and the version of Seurat that complements this installation is v5.0.1.

Need a Biowulf Account?

If you do not already have a Biowulf account, you can obtain one by following the instructions here *(https://hpc.nih.gov/docs/accounts.html)*. NIH HPC accounts are available to all NIH employees and contractors listed in the NIH Enterprise Directory. Obtaining an account requires PI approval and a nominal fee of $35 per month. Accounts are renewed annually contingent upon PI approval.

When you apply for a Biowulf account you will be issued two primary storage spaces:
`/home/$USER` (16 GB)
`/data/$USER` (100 GB)

You may request more space in `/data/$USER` by filing an online storage request (https://hpcnihapps.cit.nih.gov/auth/dashboard/storage_request.php).

Accessing RStudio on Biowulf

If you plan to use RStudio to interactively analyze your data, RStudio Server, a browser-based interface very similar to the standard RStudio desktop environment, *(https://hpc.nih.gov/apps/rstudio-server.html)* is the best option to avoid issues with lag.

Each RStudio session with RStudio Server will start fresh, so be sure to save your environment (`save.image()`) or save important objects (`saveRDS()`) before you exit the session.

Interactive Sessions

To work interactively with RStudio on Biowulf, you will need to request an interactive session using `sinteractive` *(https://hpc.nih.gov/docs/userguide.html#int)*.

When using R, you will need to include a few more options while obtaining your interactive session. For example, you will likely need space for temporary files (`lscratch`) and more memory (`mem`). You may also need additional CPUs (`--cpus-per-task`) if your workflow uses multi-threading. A sinteractive tunnel (`--tunnel`) will be required for RStudio Server on Biowulf.

Example interactive session request:

```
sinteractive --cpus-per-task=8 --gres=lscratch:50 --mem=50g --tunnel
```

NIH HPC Documentation

If you need help with Biowulf, the NIH HPC systems are well-documented at hpc.nih.gov *(https://hpc.nih.gov/)*. The User guides *(https://hpc.nih.gov/docs/user_guides.html)*, Training documentation *(https://hpc.nih.gov/training/)*, and How To *(https://hpc.nih.gov/docs/how_to.html)* docs are also fantastic resources for getting help with most HPC tasks.

For all other questions, consider emailing the HPC team directly at staff@hpc.nih.gov. *(mailto:staff@hpc.nih.gov)*

Getting Started with Seurat

The Data

In this tutorial, we are using data from Nanduri et al. 2022, *Epigenetic regulation of white adipose tissue plasticity and energy metabolism by nucleosome binding HMGN proteins*, published in *Nature Communications (https://www.nature.com/articles/s41467-022-34964-5)*. The raw count matrices are available in GEO here *(https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE193462)*, and the raw fastq files are available in the SRA.

An R project directory containing these data and associated files can be downloaded here.

This study examined "the role of HMGNs in white adipocyte browning by comparing wild-type (WT) mice and cells to genetically derived mice and cells lacking the two major members of the HMGN protein family, HMGN1 and HMGN2 (DKO mice)." *(https://www.nature.com/articles/s41467-022-34964-5)* HMGN proteins are known chromatin binding proteins that have been shown to play a role in cell identity stabilization via epigenetic regulation.

Sample	Species	Condition	Time Point	Replicate	Status
W10	Mouse	WT	0	1	Preadipocytes
W20	Mouse	WT	0	2	Preadipocytes
D10	Mouse	DKO	0	1	Preadipocytes
D20	Mouse	DKO	0	2	Preadipocytes
W16	Mouse	WT	6	1	Adipocytes
W26	Mouse	WT	6	2	Adipocytes
D16	Mouse	DKO	6	1	Adipocytes
D26	Mouse	DKO	6	2	Adipocytes

Experimental Design

The general experimental design was as follows:

WT vs DKO mice at two time points, 0 and 6 days. At day 0, cells were in a preadipocyte state, while at day 6 they had differentiated into adipocytes. Each time point had two replicates of each condition (WT vs DKO), for a total of 8 samples.

Installation

Seurat can be installed directly from CRAN.

```
install.packages("Seurat")
```

If you would like to install the development version or previous versions, see the installation instructions available here *(https://satijalab.org/seurat/articles/install)*.

Load the package from your R library. We will also go ahead and load packages from the tidyverse.

```
library(Seurat)
library(tidyverse)
```

Importing Data

Now that we have loaded the package, we can import our data.

Generally, in R programming, functions that involve data import begin with "read / Read". Seurat includes a number of read functions for different data / platform types.

For 10X scRNA-Seq data, the following functions will be most relevant:

- `Read10X()` - primary argument is a directory from CellRanger containing the matrix.mtx, genes.tsv (or features.tsv), and barcodes.tsv files.
- `Read10X_h5()` - reads the hdf5 file from 10X CellRanger (scATAC-Seq or scRNA-Seq).
- `ReadMtx()` - read from local or remote (arguments for the .mtx file, cells/barcodes file, and features/genes file). **This option is not just for 10X data, but is useful for most platforms and pipelines**.

The function you choose will depend on the format of your data.

Note

There are a number of other read and load functions available for different platforms and data types (e.g., Akoya CODEX, Nanostring SMI, SlideSeq, 10X Visium, 10X Xenium, Vizgen MERFISH). You can find relevant functions here *(https://satijalab.org/seurat/reference/)*.

Importing data in a .tsv or .csv

If you are trying to import publicly available data, and the available count data was saved as a `.txt`, `.tsv`, or `.csv` file, you can import these data using regular base R import functions (e.g., `read.table()`, `read.csv()`). The data does not need to be in a sparse format to create a Seurat object. However, if you would like to save memory upon import by loading as a sparse data matrix, see `readSparseCounts()` *(https://rdrr.io/bioc/scuttle/man/readSparseCounts.html)* from the Bioconductor package `scuttle` *(https://www.bioconductor.org/packages/release/bioc/html/scuttle.html)*.

Should I use the filtered for unfiltered data from Cell Ranger?

The raw / unfiltered matrix includes all valid cell barcodes from GEMs (Gel Bead-In EMulsions). This includes GEMs with free-floating mRNA from lysed or dead cells *(https://kb.10xgenomics.com/hc/en-us/articles/360001892491-What-is-the-difference-between-the-filtered-and-raw-gene-barcode-matrix)*. Cell Ranger uses a barcode vs UMI count rank plot (knee plot) to eliminate such background noise.

While using the raw files will give you greater control over filtering, these files are also much larger. You can visualize the web summary output from Cell Ranger to see what was filtered. If you do not see a clear inflection point, you may want to use the raw files and apply other techniques for filtering.

Loading hdf5 files.

To load a single file, we use

```
W10 <- Read10X_h5("../data/W10_filtered_feature_bc_matrix.h5")
```

The only required argument is the path to the 10X h5 file.

To read multiple files, you can use a for loop or a function(s) from the apply family. For example,

```
#List files
files <- list.files(path="../data/",recursive=T,pattern="*.h5")

#Create a list of count matrices
h5_read <- lapply(paste0("../data/",
                         files), Read10X_h5)

#Automated way to assign names; modify for your purposes
#names(h5_read)<- sapply(files,
#                        function(x){str_split_1(x,"_")[1]},
#                        USE.NAMES = FALSE)

#Assign names manually
names(h5_read)<-c("D10","D16","D20","D26","W10","W16","W20","W26")
```

Let's take a look at the `W10`.

```
W10[1:5,1:5]
```

```
5 x 5 sparse Matrix of class "dgCMatrix"
       AAACCCACAGTGACCC-1 AAACCCACATGGCTGC-1 AAACCCAGTGATATAG-1
Xkr4                    .                  .                  .
Gm1992                  .                  .                  .
Gm19938                 .                  .                  1
Gm37381                 .                  .                  .
Rp1                     .                  .                  .
       AAACCCAGTGGAAGTC-1 AAACCCATCACGTCCT-1
Xkr4                    .                  .
Gm1992                  .                  .
Gm19938                 .                  .
Gm37381                 .                  .
Rp1                     .                  .
```

The data is stored as a sparse matrix to save CPU, time, and memory. Each `.` is actually a 0.

Creating a Seurat Object

Once we have read in the matrices, the next step is to create a Seurat object. The Seurat object will be used to store the raw count matrices, sample information, and processed data (normalized counts, plots, etc.).

`CreateSeuratObject()` is used to create the object. This requires the matrix of counts for the first argument (`counts=W10`). We can also include a project name (e.g., the sample name). Other useful arguments include `min.cells` and `min.features`, which allow some initial filtering. For example, `min.cells = 3` will filter genes / features that are not present across a minimum of 3 cells, while `min.feature=200` will filter cells that do not contain a minimum of 200 genes / features.

```
# Single sample
W10 <- CreateSeuratObject(counts=W10, project="W10", min.cells = 3,
                          min.features = 200)
# View W10
W10
```

```
An object of class Seurat
19059 features across 7382 samples within 1 assay
Active assay: RNA (19059 features, 0 variable features)
 1 layer present: counts
```

For all samples, we can use `mapply` to loop through the `h5_read` list and the list names (`names(h5_read)`).

```
# All samples
adp <- mapply(CreateSeuratObject,counts=h5_read,
              project=names(h5_read),
              MoreArgs = list(min.cells = 3, min.features = 200))
# View adp
adp
```

```
$D10
An object of class Seurat
19084 features across 7623 samples within 1 assay
Active assay: RNA (19084 features, 0 variable features)
 1 layer present: counts

$D16
An object of class Seurat
16943 features across 5990 samples within 1 assay
Active assay: RNA (16943 features, 0 variable features)
 1 layer present: counts

$D20
An object of class Seurat
```

```
18242 features across 5093 samples within 1 assay
Active assay: RNA (18242 features, 0 variable features)
 1 layer present: counts

$D26
An object of class Seurat
17751 features across 7436 samples within 1 assay
Active assay: RNA (17751 features, 0 variable features)
 1 layer present: counts

$W10
An object of class Seurat
19059 features across 7382 samples within 1 assay
Active assay: RNA (19059 features, 0 variable features)
 1 layer present: counts

$W16
An object of class Seurat
17024 features across 5465 samples within 1 assay
Active assay: RNA (17024 features, 0 variable features)
 1 layer present: counts

$W20
An object of class Seurat
18354 features across 5427 samples within 1 assay
Active assay: RNA (18354 features, 0 variable features)
 1 layer present: counts

$W26
An object of class Seurat
17373 features across 6013 samples within 1 assay
Active assay: RNA (17373 features, 0 variable features)
 1 layer present: counts
```

```
#remove the original sparse matrices
rm(h5_read)
```

We can merge our seurat objects, which are now in a list in order to view the QC metrics and compare across samples. If samples are fairly different, **QC thresholds should be applied on a per sample basis**.

```
adp <- merge(adp[[1]], y = adp[2:length(adp)],
                  add.cell.ids = names(adp),project="Adipose")
```

The cell IDs can overlap between samples, so here we can apend the sample ID as a prefix to each cell ID using `add.cell.ids`. If this isn't done "and any cell names are duplicated, cell names will be appended with _X, where X is the numeric index of the object in c(x, y)" (See `merge.Seurat()`).

Warning

We could have created a merged object using `h5_read`, as a list can be provided for the argument `counts`:

```
adp2<-CreateSeuratObject(counts=h5_read, min.cells = 3,
                         min.features = 200)
```

However, this seems to take longer and does not give us control over the cell IDs. The documentation for using a list with `CreateSeuratObject()` is currently lacking. It is unclear how to use the `metadata` option if cell ids are not unique across samples.

Use a for loop to load the data and create an object for each file

If you want to apply QC metrics independently for each sample, you can use this for loop to create an individual object for each sample.

```
# Create a Seurat object for each sample
for (file in files){
        seurat_data <- Read10X_h5(paste0("../data/",
                        file))
        seurat_obj <- CreateSeuratObject(counts = seurat_data,
                                         min.features = 200,
                                         min.cells = 3,
                                         project = file)
        assign(file, seurat_obj)
}
```

See here *(https://github.com/hbctraining/scRNA-seq_online/blob/master/lessons/03_SC_quality_control-setup.md)* for an explanation of the code.

Inspect and work with a Seurat object

The Seurat Object is a data container for single cell RNA-Seq and related data. It is an S4 object, which is a type of data structure that stores complex information (e.g., scRNA-Seq count matrix, associated sample information, and data /results generated from downstream analyses) in one or more slots. It is extensible and can interact with many different methods, including those from other developers.

You can see the primary slots using `glimpse()`.

```
glimpse(W10)
```

```
Formal class 'Seurat' [package "SeuratObject"] with 13 slots
  ..@ assays      :List of 1
  .. ..$ RNA:Formal class 'Assay5' [package "SeuratObject"] with 8 s
  ..@ meta.data   :'data.frame':    7382 obs. of  3 variables:
  .. ..$ orig.ident  : Factor w/ 1 level "W10": 1 1 1 1 1 1 1 1 1 1
  .. ..$ nCount_RNA  : num [1:7382] 23082 35379 17308 27507 2858 ...
  .. ..$ nFeature_RNA: int [1:7382] 4760 6122 4697 5526 995 4261 408
  ..@ active.assay: chr "RNA"
  ..@ active.ident: Factor w/ 1 level "W10": 1 1 1 1 1 1 1 1 1 1 ...
  .. ..- attr(*, "names")= chr [1:7382] "AAACCCACAGTGACCC-1" "AAACCCA
  ..@ graphs      : list()
  ..@ neighbors   : list()
  ..@ reductions  : list()
  ..@ images      : list()
  ..@ project.name: chr "W10"
  ..@ misc        : list()
  ..@ version     :Classes 'package_version', 'numeric_version'  hidd
  .. ..$ : int [1:3] 5 0 1
  ..@ commands    : list()
  ..@ tools       : list()
```

`W10` has 13 slots: assays, meta.data, active.assay, active.ident, graphs, neighbors, reductions, images, misc, version, commands, and tools. See a description of the slots here. *(https://satijalab.github.io/seurat-object/reference/Seurat-class.html#see-also-1)* For guidance on accessing data stored in the SeuratObject, see this vignette *(https://satijalab.org/seurat/articles/seurat5_essential_commands)*.

Unlike previous iterations of Seurat, Seurat v5 contains assays in layers to accommodate different assays and data types. "For example, these layers can store: raw counts (layer='counts'), normalized data (layer='data'), or z-scored/variance-stabilized data (layer='scale.data')." *(https://satijalab.org/seurat/articles/announcements.html)*

To return the names of the layers:

```
Layers(W10)
```

```
[1] "counts"
```

To access a specific count layer use:

```
W10[["RNA"]]$counts |> head()
# or
W10@assays$RNA$counts  |> head()
```

```
# or
LayerData(W10, assay="RNA", layer='counts') |> head()
```

Because we have just created this object, most of the slots are empty. We will save results back to the object and fill the slots as we work.

You can always check the version of Seurat that was used to build the object with

```
W10@version
```

```
[1] '5.0.1'
```

And check commands and parameters used to generate results with:

```
W10@commands
```

```
list()
```

We haven't worked on the object, so there is nothing here. Compare with `pbmc_small` a built in data set in the `SeuratObject` package.

```
head(pbmc_small@commands, 1)
```

```
$NormalizeData.RNA
Command: NormalizeData(object = pbmc_small)
Time: 2018-08-27 16:32:17.136699
assay : RNA
normalization.method : LogNormalize
scale.factor : 10000
verbose : TRUE
```

This information is useful for documenting your analysis.

Examine metadata

The metadata in the Seurat object is located in `adp@metadata` and contains the information associated with each cell.

We can access the metadata using:

```
head(adp@meta.data) #using head to return only the first 6 rows
```

```
                      orig.ident nCount_RNA nFeature_RNA
D10_AAACCCAAGATGCTTC-1        D10      11188         3383
D10_AAACCCAAGCTCTTCC-1        D10      32832         5823
D10_AAACCCAAGTCATCGT-1        D10      28515         5002
D10_AAACCCAGTAGCGTCC-1        D10      18954         4305
D10_AAACCCAGTGCACGCT-1        D10      27181         4586
D10_AAACCCAGTGTAAATG-1        D10       3090         1030
```

or

```
head(adp[[]])
```

```
                      orig.ident nCount_RNA nFeature_RNA
D10_AAACCCAAGATGCTTC-1        D10      11188         3383
D10_AAACCCAAGCTCTTCC-1        D10      32832         5823
D10_AAACCCAAGTCATCGT-1        D10      28515         5002
D10_AAACCCAGTAGCGTCC-1        D10      18954         4305
D10_AAACCCAGTGCACGCT-1        D10      27181         4586
D10_AAACCCAGTGTAAATG-1        D10       3090         1030
```

Accessing a column or columns:

```
#Access a single column.
head(adp$orig.ident)
```

```
D10_AAACCCAAGATGCTTC-1 D10_AAACCCAAGCTCTTCC-1 D10_AAACCCAAGTCATCGT-1
                  "D10"                  "D10"                  "D10"
D10_AAACCCAGTAGCGTCC-1 D10_AAACCCAGTGCACGCT-1 D10_AAACCCAGTGTAAATG-1
                  "D10"                  "D10"                  "D10"
```

```
head(adp[["orig.ident"]])
```

```
                      orig.ident
D10_AAACCCAAGATGCTTC-1        D10
D10_AAACCCAAGCTCTTCC-1        D10
D10_AAACCCAAGTCATCGT-1        D10
```

```
D10_AAACCCAGTAGCGTCC-1        D10
D10_AAACCCAGTGCACGCT-1        D10
D10_AAACCCAGTGTAAATG-1        D10
```

```
#Access multiple columns, rows.
head(adp[[c("orig.ident", "nCount_RNA")]])[1:3,]
```

```
                      orig.ident nCount_RNA
D10_AAACCCAAGATGCTTC-1        D10      11188
D10_AAACCCAAGCTCTTCC-1        D10      32832
D10_AAACCCAAGTCATCGT-1        D10      28515
```

As we can see, there are different ways to access and work with the metadata. In addition, we can see that we begin with built-in information such as the `orig.ident`, `nCount_RNA`, and `nFeature_RNA`.

`orig.ident` - defaults to the project labels
`nCount_RNA` - number of UMIs per cell
`nFeature_RNA` - number of genes / features per cell

This information, along with other metrics, will be used for quality filtering.

Seurat Idents?

Idents are the metadata used by Seurat to classify each of the datapoints. Each datapoint can have multiple metadata identities (e.g., sample, condition, cluster, etc.). The active.ident is the default Ident used at even given time by a particular Seurat function. Often it can be adjusted using function parameters; however, at other times, you will need to actively change the active.ident to another metadata column using `Idents()`. See `?Idents()` and this vignette *(https://satijalab.org/seurat/articles/essential_commands#identity-class-labels)* for examples.

Subsetting the object

We can subset by features or cells by `adp[1:10,]` or `adp[,1:10]` respectively. To subset using metadata, use `subset()`.

For example, if we want to subset to the sample W10, we can use.

```
subset(x = adp, idents = "W10")
```

```
An object of class Seurat
19059 features across 7382 samples within 1 assay
Active assay: RNA (19059 features, 0 variable features)
1 layer present: counts.W10
```

We will look at subsetting more when filtering cells based on quality.

Add to metadata.

We can add information to our metadata by accessing and assigning to metadata columns or using `?AddMetaData()`.

Add condition to metadata (either wildtype of double knockout).

```
adp$condition <- ifelse(str_detect(adp@meta.data$orig.ident, "^W"),
                        "WT","DKO")
```

Add time information to the metadata.

```
adp$time_point <- ifelse(str_detect(adp@meta.data$orig.ident, "0"),
                         "Day 0","Day 6")
```

Add Condition + Time to the metadata.

```
adp$cond_tp <- paste(adp$condition, adp$time_point)
```

Other Useful information from the Seurat object

The Seurat v5 object doesn't require all assays have the same cells. In this case, `Cells()` can be used to return the cell names of the default assay while `colnames()` can be used to return the cell names of the entire object.

```
head(Cells(adp,layer="counts.W10"))
```

```
[1] "W10_AAACCCACAGTGACCC-1" "W10_AAACCCACATGGCTGC-1" "W10_AAACCCAGT(
[4] "W10_AAACCCAGTGGAAGTC-1" "W10_AAACCCATCACGTCCT-1" "W10_AAACGAACA(
```

```
head(colnames(adp))
```

```
[1] "D10_AAACCCAAGATGCTTC-1" "D10_AAACCCAAGCTCTTCC-1" "D10_AAACCCAAG
[4] "D10_AAACCCAGTAGCGTCC-1" "D10_AAACCCAGTGCACGCT-1" "D10_AAACCCAGT(
```

To return feature/gene names:

```
head(Features(adp))
```

```
[1] "Xkr4"    "Gm19938" "Sox17"   "Mrpl15"  "Lypla1"  "Tcea1"
```

```
head(rownames(adp))
```

```
[1] "Xkr4"    "Gm19938" "Sox17"   "Mrpl15"  "Lypla1"  "Tcea1"
```

For multimodal data, list the assays using

```
Assays(adp)
```

```
[1] "RNA"
```

To return the number of cells or features use:

```
#return number of cells across all layers
num_cells <- ncol(adp)
#return number of features across all layers
num_features <- nrow(adp)
#to return row by column information
dim(adp)
```

```
[1] 20654 50429
```

Note

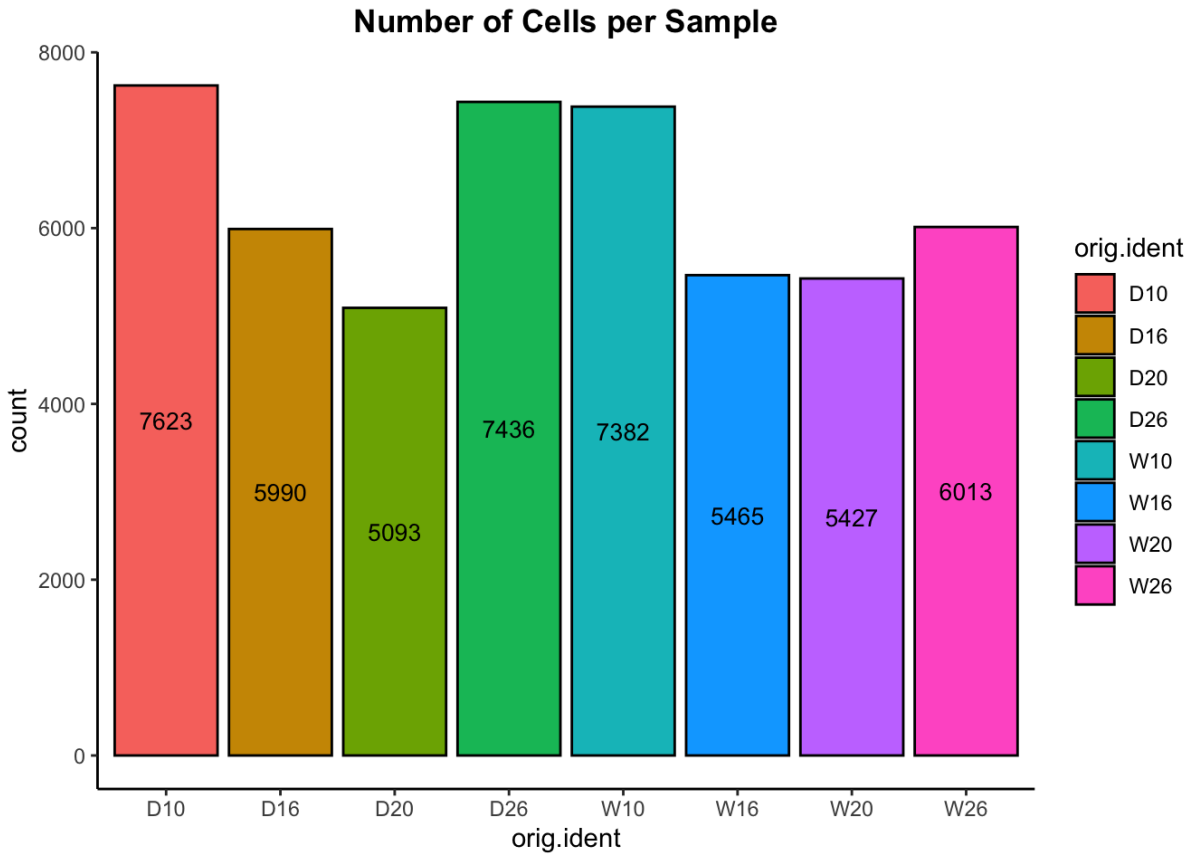
`ncol(adp)` will return the number of cells across all layers. If you want a specific layer, use `ncol(adp[["RNA"]]$counts.W10)`. This information is also quickly accessible via the Global Environment when using RStudio or by using `str()`.

We can also quickly plot the number of cells from the metadata:

```
# Visualize the number of cell counts per sample
adp@meta.data %>%
  ggplot(aes(x=orig.ident, fill=orig.ident)) +
  geom_bar(color="black") +
  stat_count(geom = "text", colour = "black", size = 3.5,
             aes(label = ..count..),
```

```
            position=position_stack(vjust=0.5))+
  theme_classic() +
  theme(plot.title = element_text(hjust=0.5, face="bold")) +
  ggtitle("Number of Cells per Sample")
```

```
Warning: The dot-dot notation (`..count..`) was deprecated in ggplot2 3.4.0.
ℹ Please use `after_stat(count)` instead.
```



Save the object

At this point, our Seurat object is fairly large (~2.1 GB). It is wise to save this for downstream applications using `saveRDS()`.

```
saveRDS(adp,"../outputs/merged_Seurat_adp.rds")
```

References

The following resources were instrumental in designing this lesson:

- https://www.sc-best-practices.org/introduction/analysis_tools.html#single-cell-analysis-frameworks-and-consortia *(https://www.sc-best-practices.org/introduction/analysis_tools.html#single-cell-analysis-frameworks-and-consortia)*
- https://satijalab.org/seurat/articles/essential_commands *(https://satijalab.org/seurat/articles/essential_commands)*
- https://github.com/hbctraining/scRNA-seq_online/blob/master/lessons/03_SC_quality_control-setup.md *(https://github.com/hbctraining/scRNA-seq_online/blob/master/lessons/03_SC_quality_control-setup.md)*

Getting Started with Seurat: QC to Clustering

Learning Objectives

This tutorial was designed to demonstrate common secondary analysis steps in a scRNA-Seq workflow. We will start with a merged Seurat Object with multiple data layers representing multiple samples.

Throughout this tutorial we will

1. Apply quality control parameters to retain only high quality cells
2. Normalize and scale the data
3. Apply dimensional reduction
4. Perform Clustering

This tutorial largely follows the standard unsupervised clustering workflow by Seurat *(https://satijalab.org/seurat/articles/pbmc3k_tutorial)* with slight deviations and a different data set.

> **Warning**
>
> This document in no way represents a complete analysis. Here, we are demonstrating many of the standard steps in Seurat. Results here would be considered preliminary, requiring greater care and consideration of biology. We are not reproducing published results.

The data

In this tutorial, we will continue to use data from Nanduri et al. 2022, *Epigenetic regulation of white adipose tissue plasticity and energy metabolism by nucleosome binding HMGN proteins*, published in *Nature Communications (https://www.nature.com/articles/s41467-022-34964-5)*.

As a reminder, this study examined "the role of HMGNs in white adipocyte browning by comparing wild-type (WT) mice and cells to genetically derived mice and cells lacking the two major members of the HMGN protein family, HMGN1 and HMGN2 (DKO mice)." *(https://www.nature.com/articles/s41467-022-34964-5)*. The experimental design included WT vs DKO mice, 2 replicates each, at two time points, 0 and 6 days. At day 0, cells were in a preadipocyte state, while at day 6 they had differentiated into adipocytes. Each time point had two replicates of each condition (WT vs DKO), for a total of 8 samples.

An R project directory containing these data and associated files can be downloaded here.

Consider the biology in your analysis

The biology of your experiment will inform your analysis. While scRNA-Seq can be exploratory, you should have some general idea of the cell types that you expect. Knowing some characteristics of expected cells, for example, cell size or whether cells are complex or more energetically active is helpful for data processing.

For example, in this experiment, we expect cells transitioning from preadipocytes to adipocytes and then beige adipocytes.

Load the packages

```
library(tidyverse) # dplyr and ggplot2
library(Seurat) # Seurat toolkit
library(hdf5r) # for data import
library(patchwork) # for plotting
library(presto) # for differential expression
library(glmGamPoi) # for sctransform
```

```
Warning: package 'glmGamPoi' was built under R version 4.3.3
```

Load the Seurat Object

Here, we will start with the data stored in a Seurat object. For instructions on data import and creating the object, see an *Introduction to scRNA-Seq with R (Seurat).*

```
adp <- readRDS("../outputs/merged_Seurat_adp.rds")
```

Examine the object

```
glimpse(adp)
```

```
Formal class 'Seurat' [package "SeuratObject"] with 13 slots
  ..@ assays      :List of 1
  .. ..$ RNA:Formal class 'Assay5' [package "SeuratObject"] with 8 s
  ..@ meta.data   :'data.frame':    50429 obs. of  6 variables:
  .. ..$ orig.ident  : chr [1:50429] "D10" "D10" "D10" "D10" ...
  .. ..$ nCount_RNA  : num [1:50429] 11188 32832 28515 18954 27181 .
  .. ..$ nFeature_RNA: int [1:50429] 3383 5823 5002 4305 4586 1030 5(
  .. ..$ condition   : chr [1:50429] "DKO" "DKO" "DKO" "DKO" ...
  .. ..$ time_point  : chr [1:50429] "Day 0" "Day 0" "Day 0" "Day 0"
  .. ..$ cond_tp     : chr [1:50429] "DKO Day 0" "DKO Day 0" "DKO Day
```

```
..@ active.assay: chr "RNA"
..@ active.ident: Factor w/ 8 levels "D10","D16","D20",..: 1 1 1 1
.. ..- attr(*, "names")= chr [1:50429] "D10_AAACCCAAGATGCTTC-1" "D1
..@ graphs      : list()
..@ neighbors   : list()
..@ reductions  : list()
..@ images      : list()
..@ project.name: chr "Adipose"
..@ misc        : list()
..@ version     :Classes 'package_version', 'numeric_version'  hid
.. ..$ : int [1:3] 5 0 1
..@ commands    : list()
..@ tools       : list()
```

Quality Control

The goal of quality control is to keep only high quality cells (i.e., remove low quality cells (dead or dying cells), cell-free RNA, or doublets). Low quality cells will impact downstream analyses.

Take care when filtering. Heavy filtering may result in the loss of rare cell populations, while minimal filtering, may impact our ability to annotate cell populations downstream. Best practices suggest being more permissive, filtering less cells. *(https://www.sc-best-practices.org/preprocessing_visualization/quality_control.html)*

Should quality control be assessed per sample?

Quality control should be applied on a sample per sample basis. Here, we will look at quality across all samples and then decide if quality thresholds need to be applied independently. If samples display the same distributions in quality metrics, it is likely okay to apply the same thresholds across all samples. If they do not follow the same distributions, you may lose valuable information by processing the samples together with the same thresholds.

QC metrics

There are several metrics that can be used to assess overall quality. The base workflow from Seurat suggests the following:

- `nCount_RNA` - the absolute number of RNA molecules (UMIs) per cell (i.e., count depth). Each unique RNA molecule (non-PCR duplicates) will have its own Unique Molecular Identifier (UMI).
    - high total count - potential doublets or multiplets
    - low total count - potential ambient mRNA (not real cells)
    - Cell Ranger threshold set at 500 UMIs

- `nFeature_RNA` - number of genes expressed (detected) per cell.
    - high number of detected genes - potential doublets or multiplets
    - low number of detected genes - potential ambient mRNA (not real cells)
- Percent mitochondrial(`percent.mt`) - the fraction of reads from mitochondrial genes per cell
    - high mtDNA - cellular degradation

In general, low quality cells and empty droplets will have few genes (low `n_feature_RNA`, `low nCount_RNA`), whereas doublets and multiplets will exhibit high `n_feature_RNA` and high `nCount_RNA`. Low quality or dying cells will also have high `percent_mt`.

BUT

There are also biological explanations for some observed patterns.

> Cells with a relatively high fraction of mitochondrial counts might for example be involved in respiratory processes and should not be filtered out. Whereas, cells with low or high counts might correspond to quiescent cell populations or cells larger in size. --- Single Cell Best Practices *(https://www.sc-best-practices.org/preprocessing_visualization/quality_control.html)*

Doublets and multiplets *(https://www.biostars.org/p/446554/)* are also not as easy to predict. There are dedicated tools for this purpose (e.g., DoubletFinder *(https://github.com/chris-mcginnis-ucsf/DoubletFinder)*, scDblFinder *(https://bioconductor.org/packages/release/bioc/html/scDblFinder.html)*,scrublet (python package) *(https://github.com/swolock/scrublet)*), which you may want to integrate into your workflow. Eliminating ambient RNA is also not as straight forward as it may seem. Ambient RNA can be included in GEMs along with intact cells. There are tools that can be used to predict and remove ambient RNA (e.g., SoupX *(https://github.com/constantAmateur/SoupX)* and DecontX *(https://www.bioconductor.org/packages/release/bioc/html/decontX.html)*).

Other QC Indicators

There are additional quality metrics that can be used for initial cell quality filtering (e.g., complexity /novelty, percent ribosomal).

QC metrics are stored as metadata

The quality metric information is stored in the metadata (`adp@meta.data`), so we really only need to work with the metadata file to get an idea of the thresholds we want to set for quality control. We will extract this information after we calculate the percentage of mitochondrial genes.

`nCount_RNA` and `nFeature_RNA` are already available to view, so we need to add the percentage of reads that mapped to the mitochondrial genome.

QC can be an iterative process

This entire workflow is exploratory in nature and can be highly iterative. You may need to adjust thresholds after performing downstream steps.

Add percent mitochondrial

To calculate `percent.mt`, we use `PercentageFeatureSet()`, which calculates the percentage of all the counts belonging to a subset of the possible features (e.g., mitochondrial genes, ribosomal genes) for each cell.

```
adp[["percent.mt"]] <- PercentageFeatureSet(adp, pattern = "^mt-")
```

Mitochondrial pattern changes based on species

The pattern used here is species dependent. `pattern = "^MT-"`, works for human genomes, whereas `pattern = "^mt-"` works for mouse.

```
mt_genes<-grep("^mt-", rownames(adp[["RNA"]]$counts.W10),value = T)
mt_genes
```

```
 [1] "mt-Nd1"  "mt-Nd2"  "mt-Co1"  "mt-Co2"  "mt-Atp8" "mt-Atp6" "mt-Co3"
 [8] "mt-Nd3"  "mt-Nd4l" "mt-Nd4"  "mt-Nd5"  "mt-Nd6"  "mt-Cytb"
```

Extract the metadata

```
metadata <- adp@meta.data
```

QC assessment

The quality of the cells should be assessed considering the above metrics jointly and not simply in isolation.

QC can be applied manually by subjectively choosing and applying thresholds, which can be quite arbitrary. Alternatively, QC can be applied automatically using adaptive thresholds like the mean absolute deviation (MAD) (outliers generally 3 or 5 MAD from the median). You can find more information on adaptive thresholds here. *(https://bioconductor.org/books/3.15/OSCA.basic/quality-control.html#quality-control-outlier)*

Here, we are going to start with applying thresholds based on a subjective visual assessment of quality metrics.

Step 1: Visualize the data

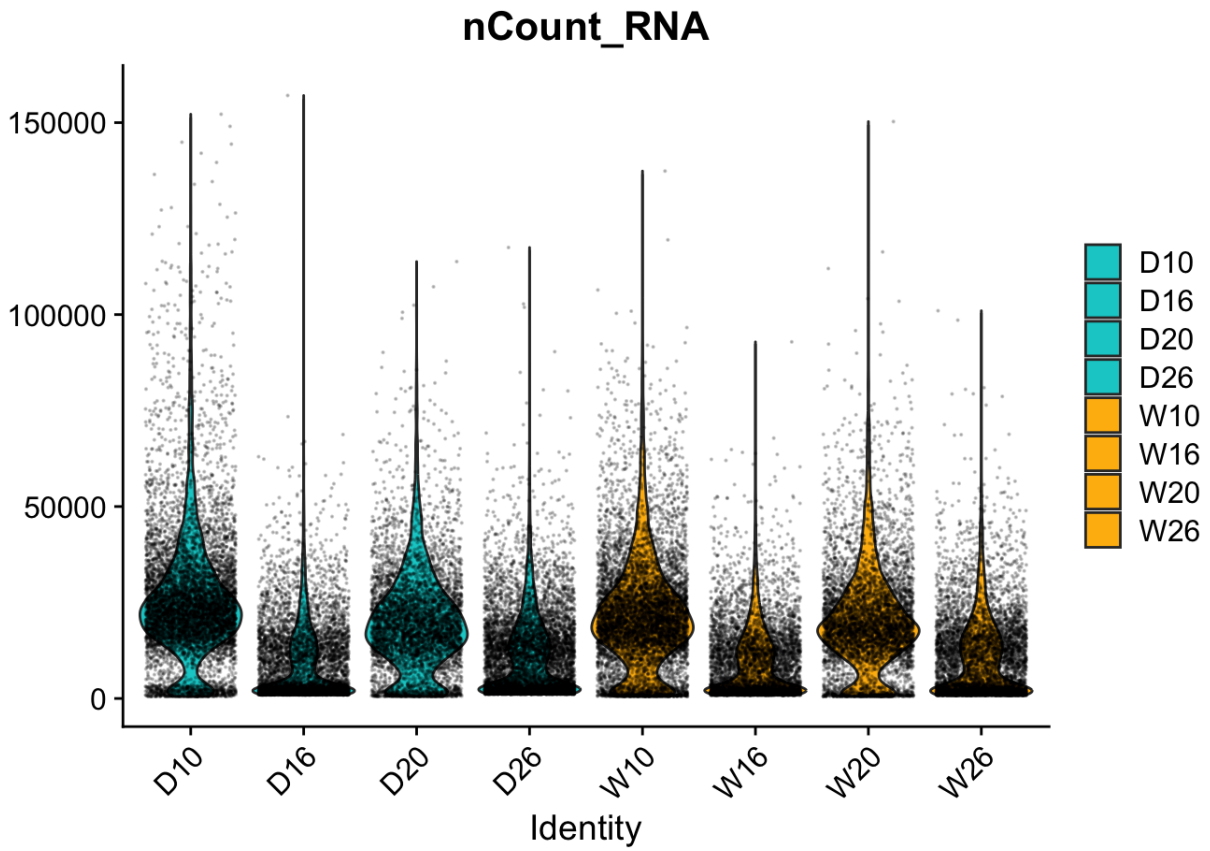
There are several built in functions for visualizing data with Seurat. We can use violin plots and scatter plots to check out the individual distributions and correlations between metrics.

nCount_RNA

Look at the distribution of `nCount_RNA` with a Violin plot:

```
# set colors
cnames<-setNames(rep(c("cyan3","darkgoldenrod1"),each=4),levels(facto

# plot
VlnPlot(adp, features = "nCount_RNA", layer="counts", group.by="orig
  scale_fill_manual(values=cnames)
```



A violin plot shows the distribution of a numeric variable across categorical variables of interest; it is a hybrid between a box plot and a density plot. The wider sections on the plot are associated with increased probabilities of data points being found at that location in the

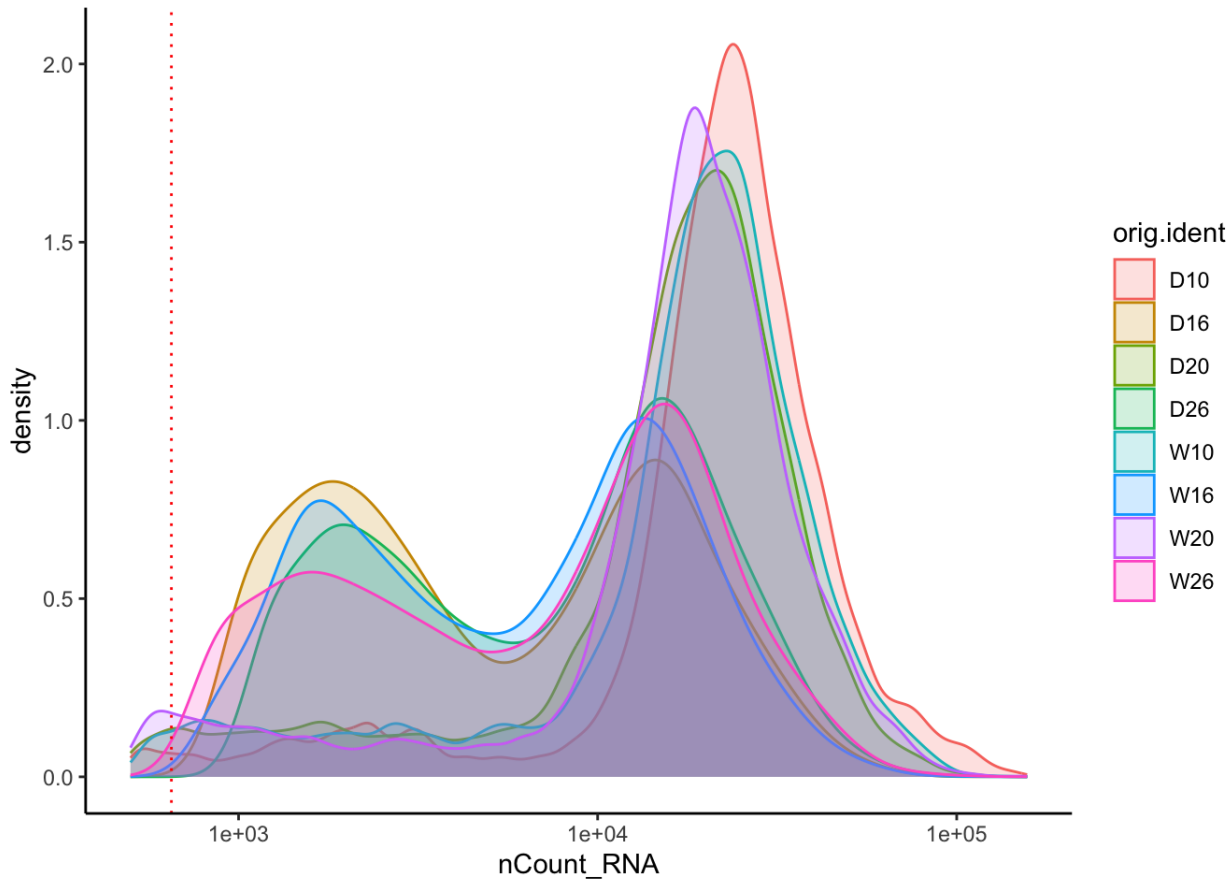
distribution. `VlnPlot`, a Seurat function that uses the Seurat object directly, overlays the individual data points to ease the interpretation of the figure.

Here, we can see that we have two bulges of increased density within the distribution, one closer to zero and another from 10 k to 30 k.

Notice that we can modify Seurat visualization using ggplot2 layers, or we can explore the data with greater flexibility by using `ggplot2`directly.

Use `ggplot2` directly and check the distributions:

```
metadata %>%
    ggplot(aes(color=orig.ident, x=nCount_RNA, fill= orig.ident)) +
    geom_density(alpha = 0.2) +
    theme_classic() +
    scale_x_log10() +
    geom_vline(xintercept = 650,color="red",linetype="dotted")
```



Using a density plot directly, we can see that the samples at time point 0 are different from those at time point 6. **We should always be thinking about whether these differences are biological or technical in nature.**

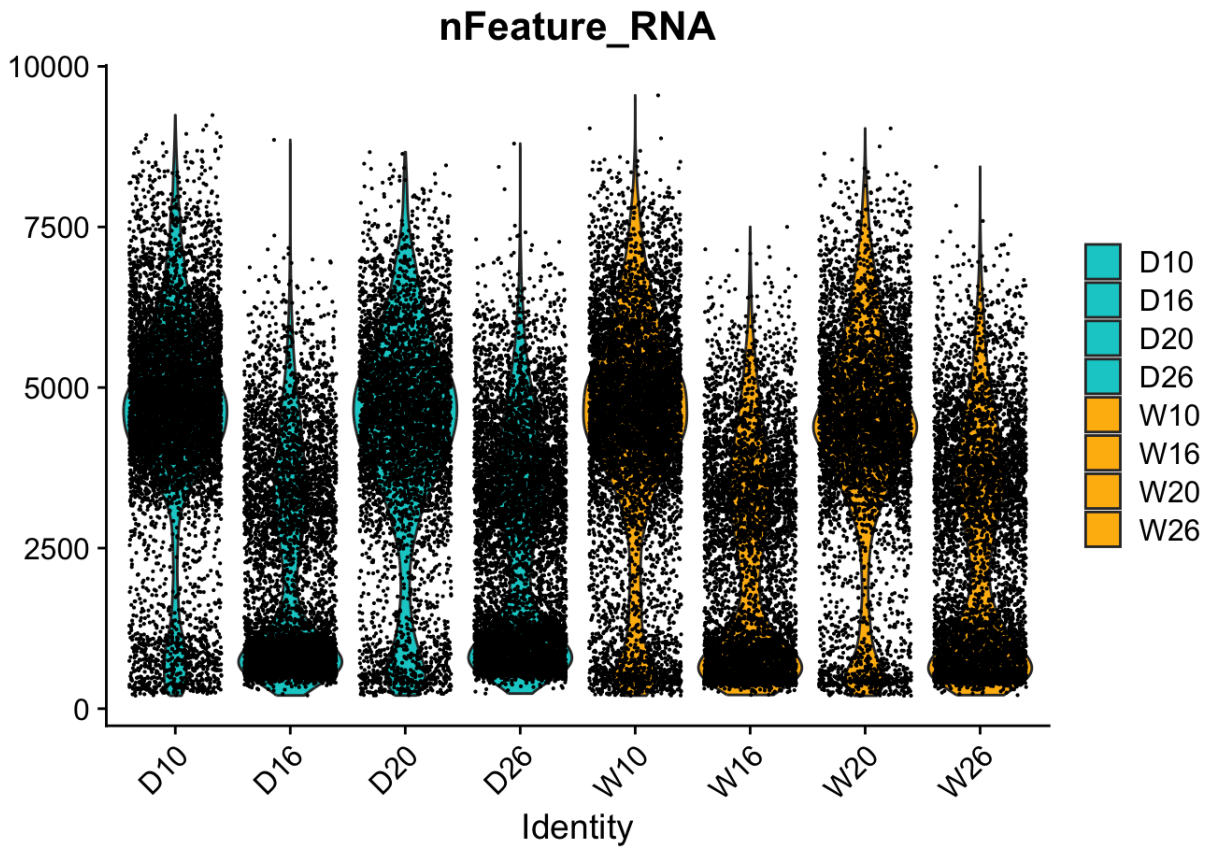
In this case, it is a bit of both. Samples from time point 6 exhibited diminished quality but also represent different biology.

Number of Genes

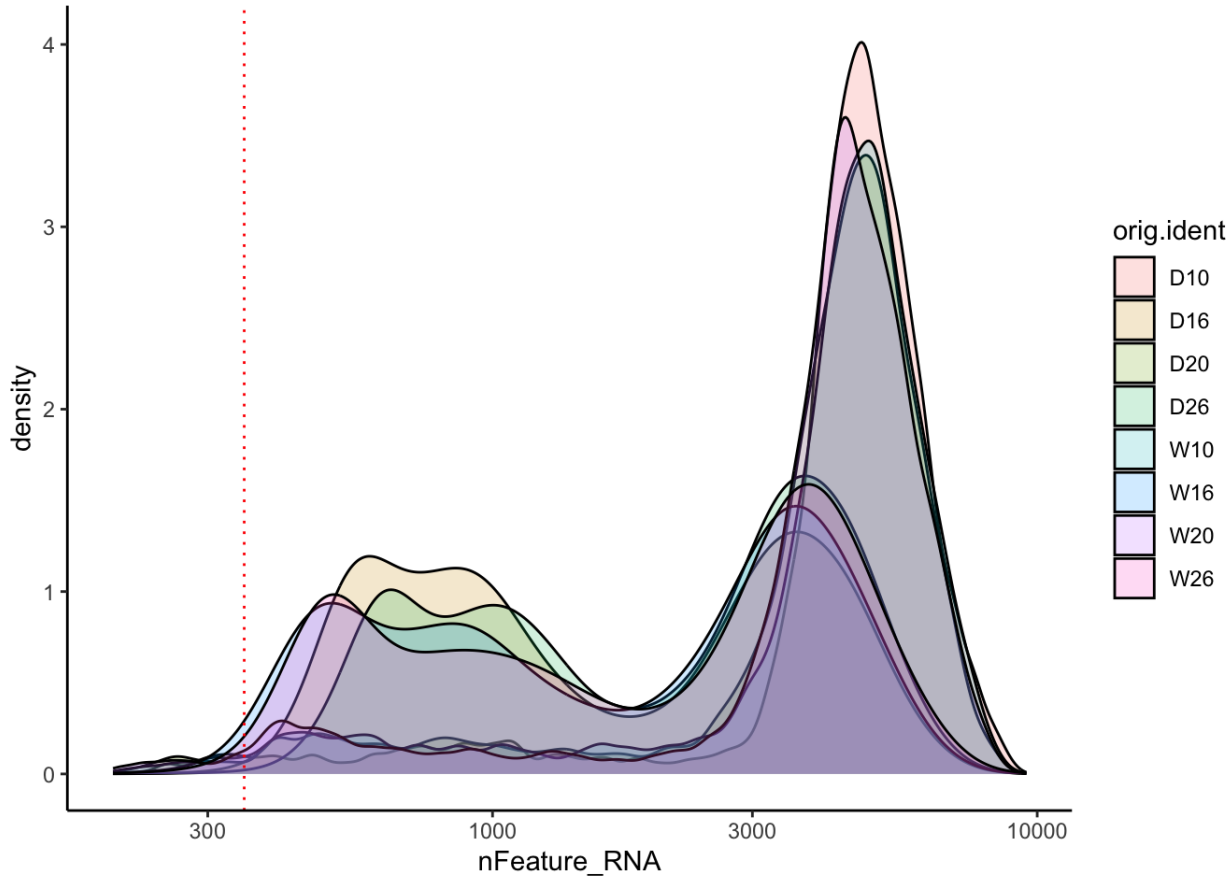
Now, let's take a look at the number of detected features.

```
VlnPlot(adp, features = "nFeature_RNA", group.by="orig.ident") +
  scale_fill_manual(values=cnames)
```

```
Warning: Default search for "data" layer in "RNA" assay yielded no re
utilizing "counts" layer instead.
```



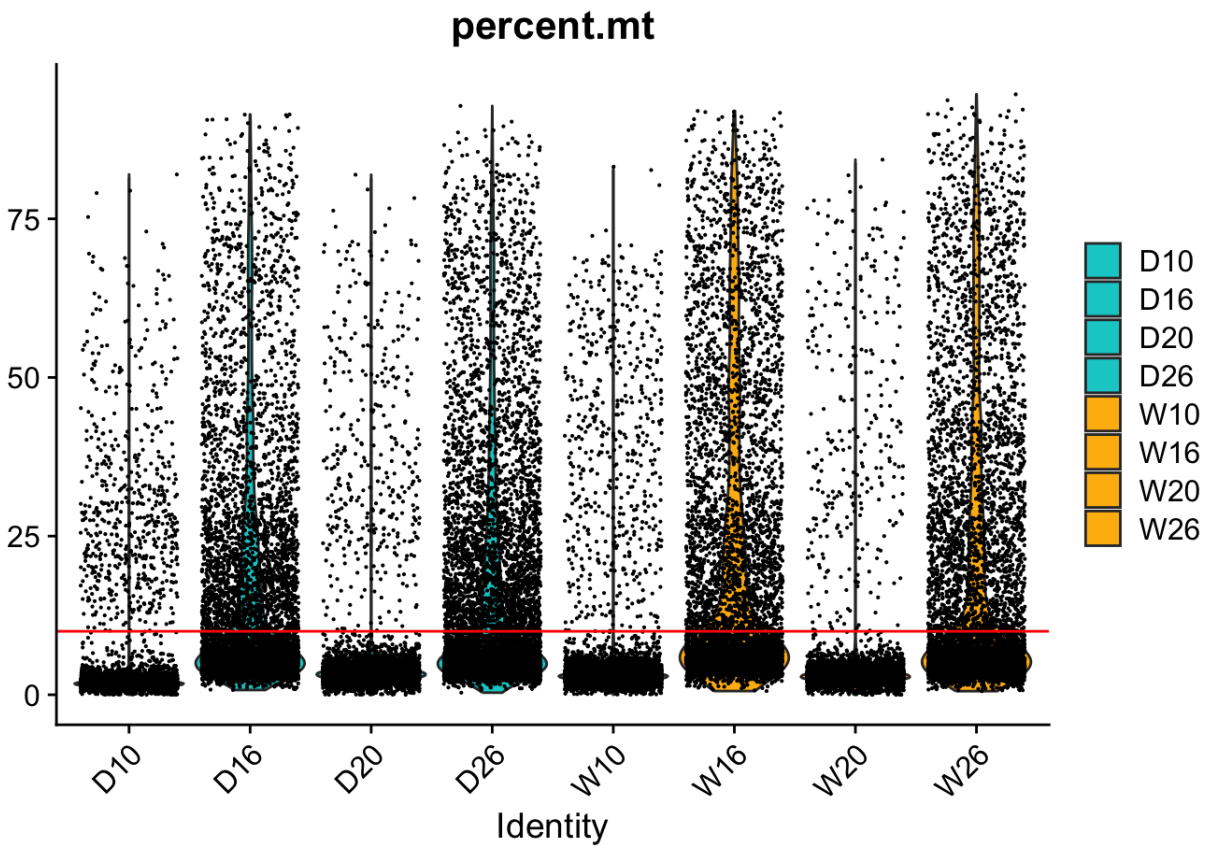

```
ggplot(metadata, aes(x=nFeature_RNA,fill=orig.ident)) +
   geom_density(alpha = 0.2) +
    theme_classic() +
    scale_x_log10() +
    geom_vline(xintercept = 350,color="red",linetype="dotted")
```



Percent mitochondrial

```
VlnPlot(adp, features = "percent.mt", group.by="orig.ident") +
  scale_fill_manual(values=cnames) +
  geom_hline(yintercept=10,color="red")
```

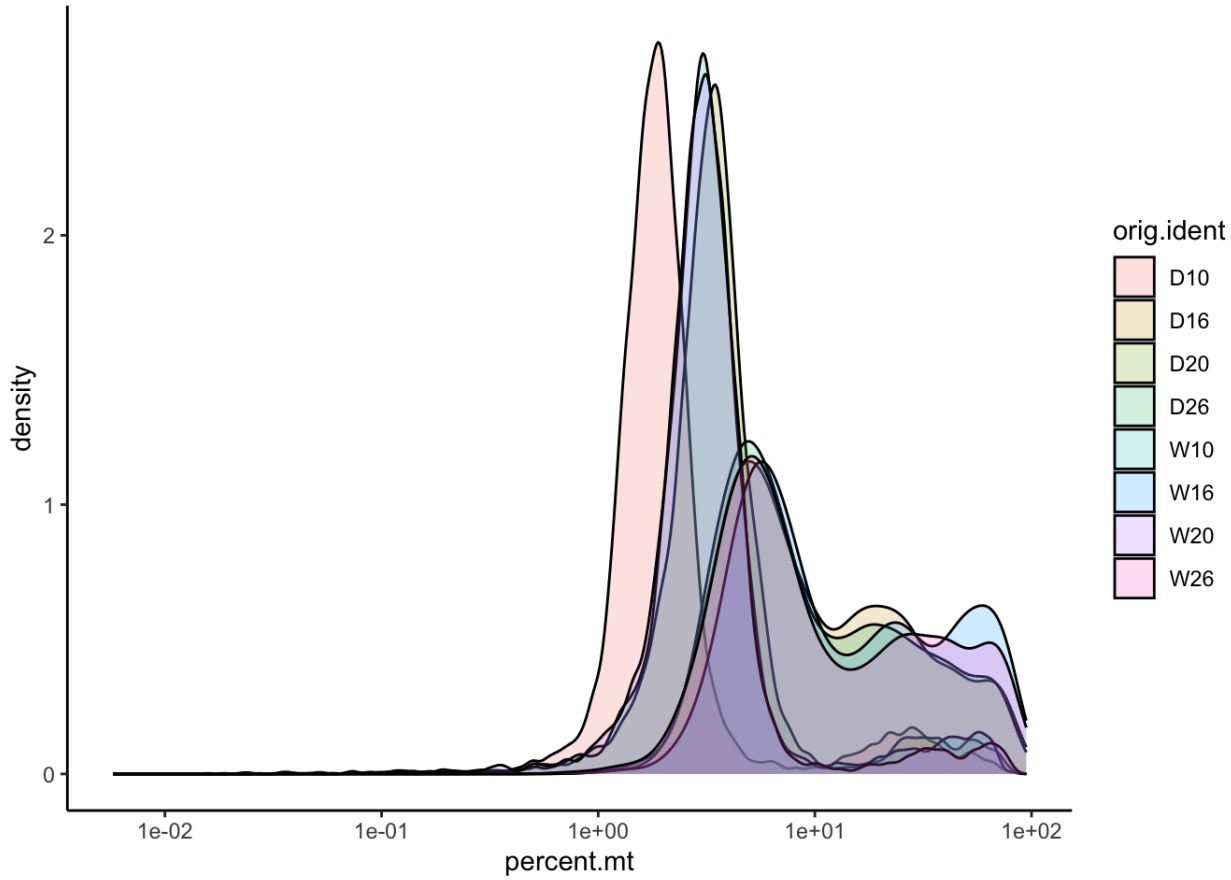
```
Warning: Default search for "data" layer in "RNA" assay yielded no re
utilizing "counts" layer instead.
```




```
ggplot(metadata, aes(x=percent.mt,fill=orig.ident)) +
   geom_density(alpha = 0.2) +
  scale_x_log10()+
    theme_classic()
```

```
Warning in scale_x_log10(): log-10 transformation introduced infinite
```

```
Warning: Removed 13 rows containing non-finite outside the scale rang
(`stat_density()`).
```

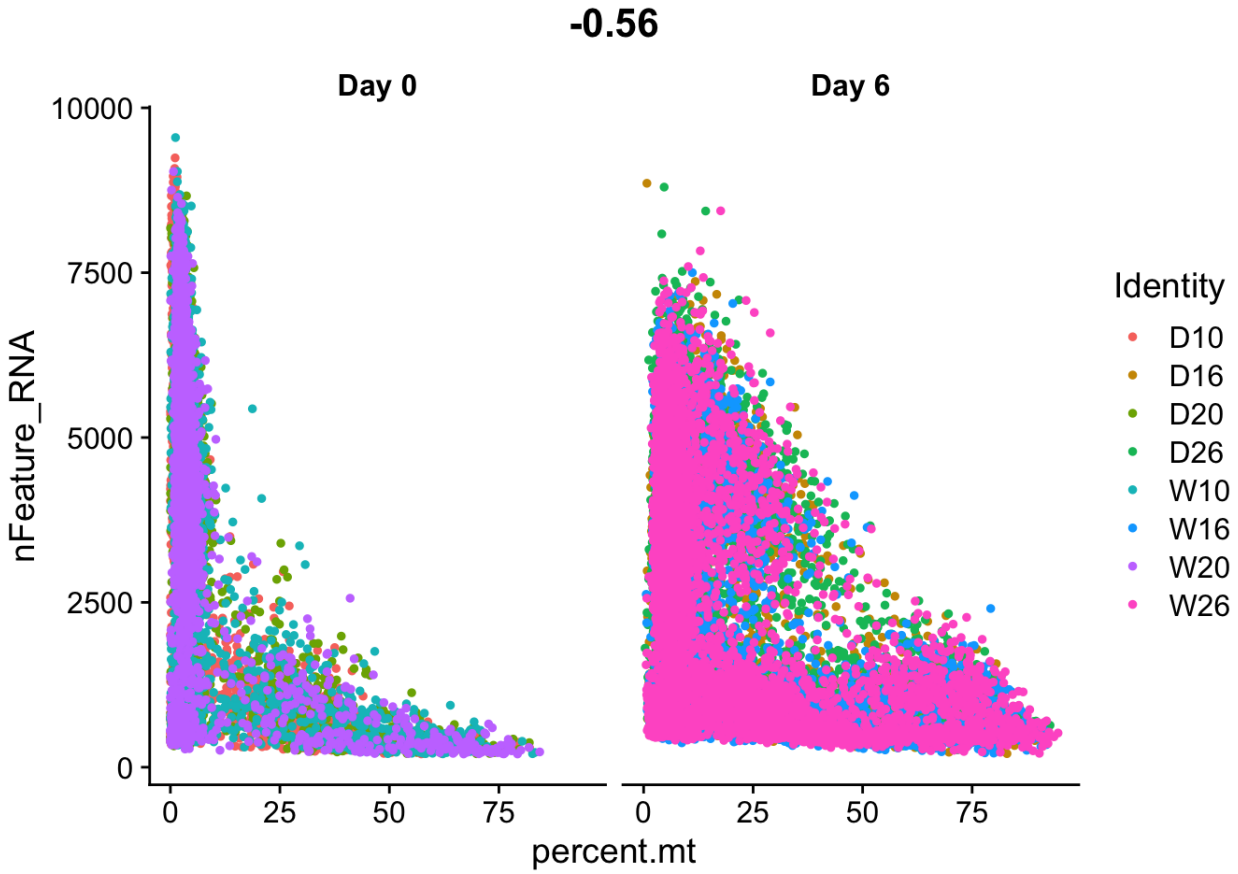


We can see that the mitochondrial content is higher for the second time point. This could be due to technical challenges and cellular degradation or biology, for example, in this case, white adipocyte browning. Beige adipocytes have a higher potential for mitochondrial respiration *(https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9589375/)*, so this is not unexpected.

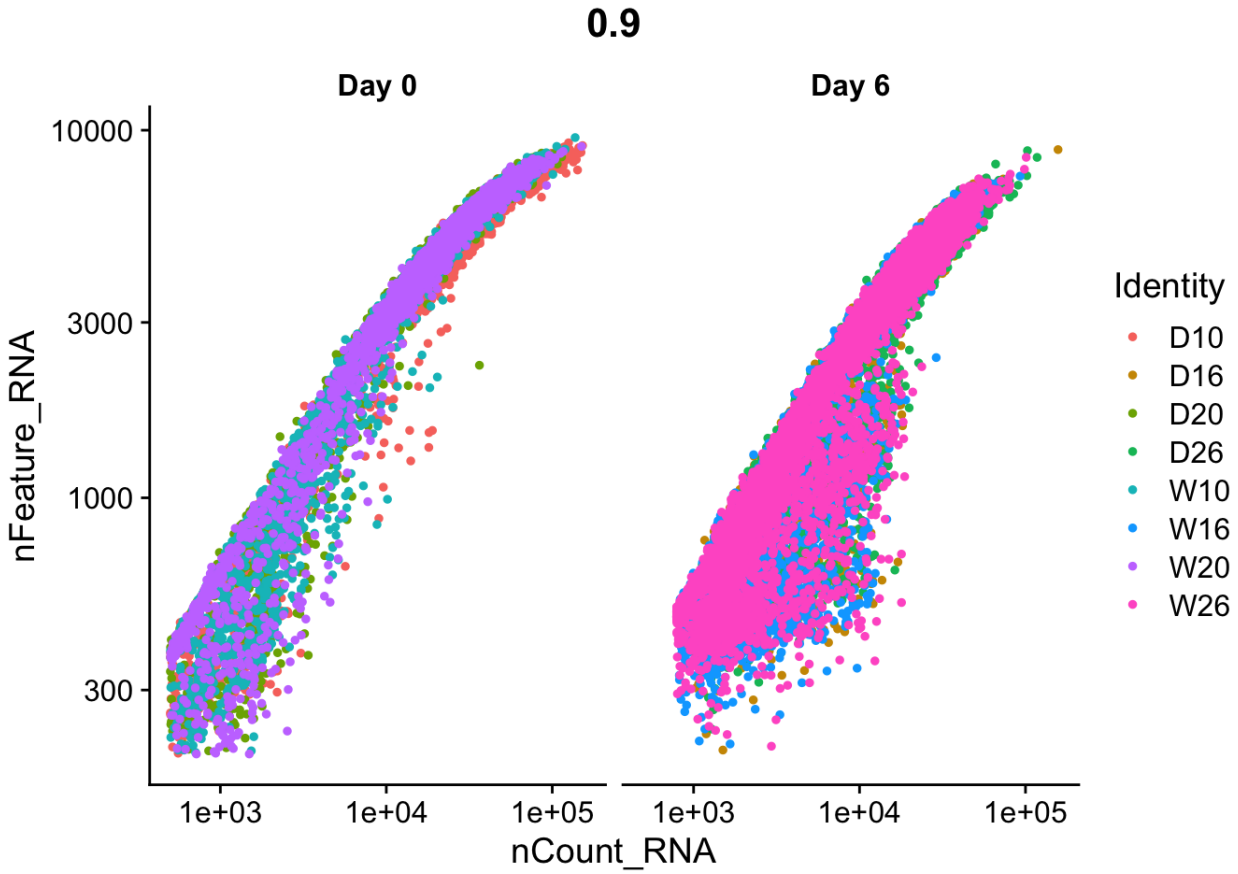
Examine these metrics together

To look at how these metrics correlate, we can use `FeatureScatter()`, which can be used to visualize feature-feature relationships and also be applied to other data stored in our Seurat object (e.g., metadata columns, PC scores).

```
FeatureScatter(adp, feature1 = "percent.mt", feature2 ="nFeature_RNA"
```

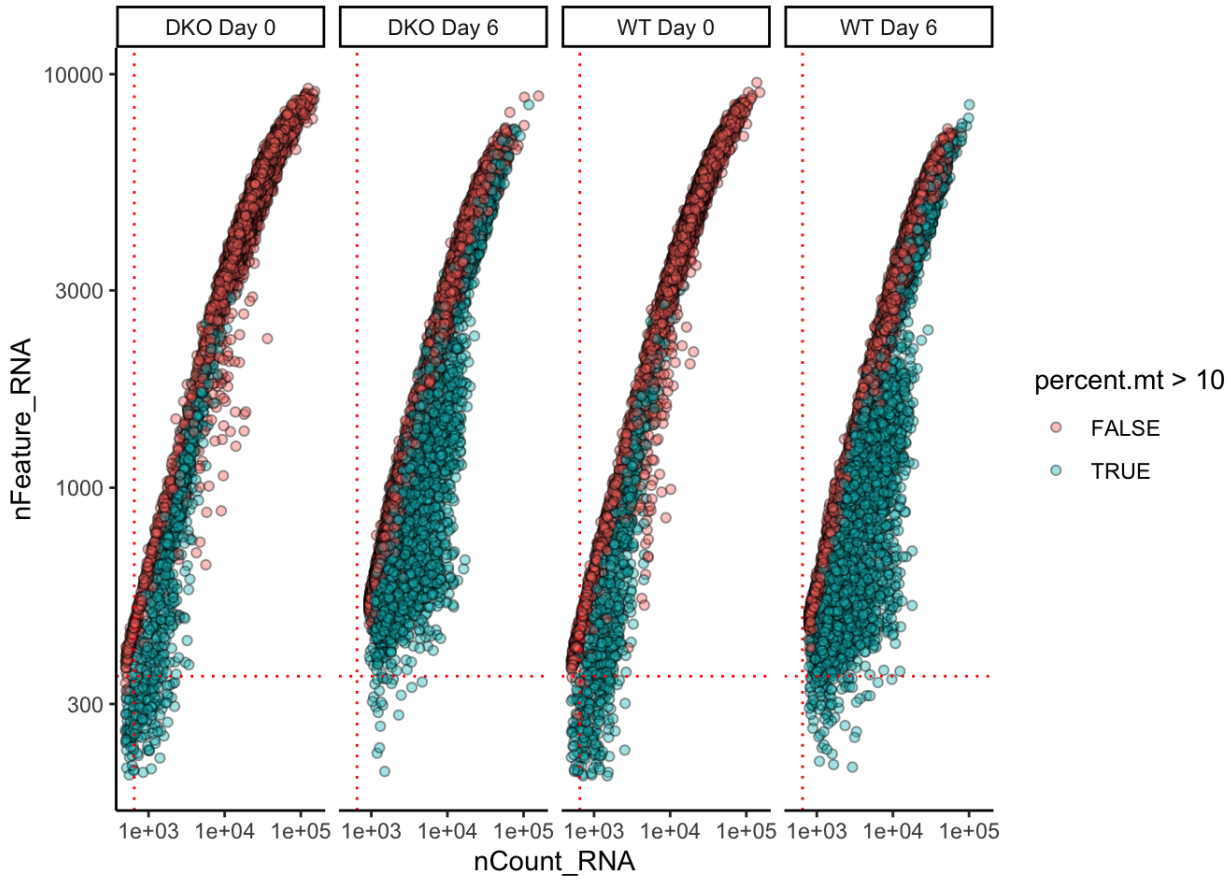



```
FeatureScatter(adp, feature1 = "nCount_RNA", feature2 = "nFeature_RNA
```

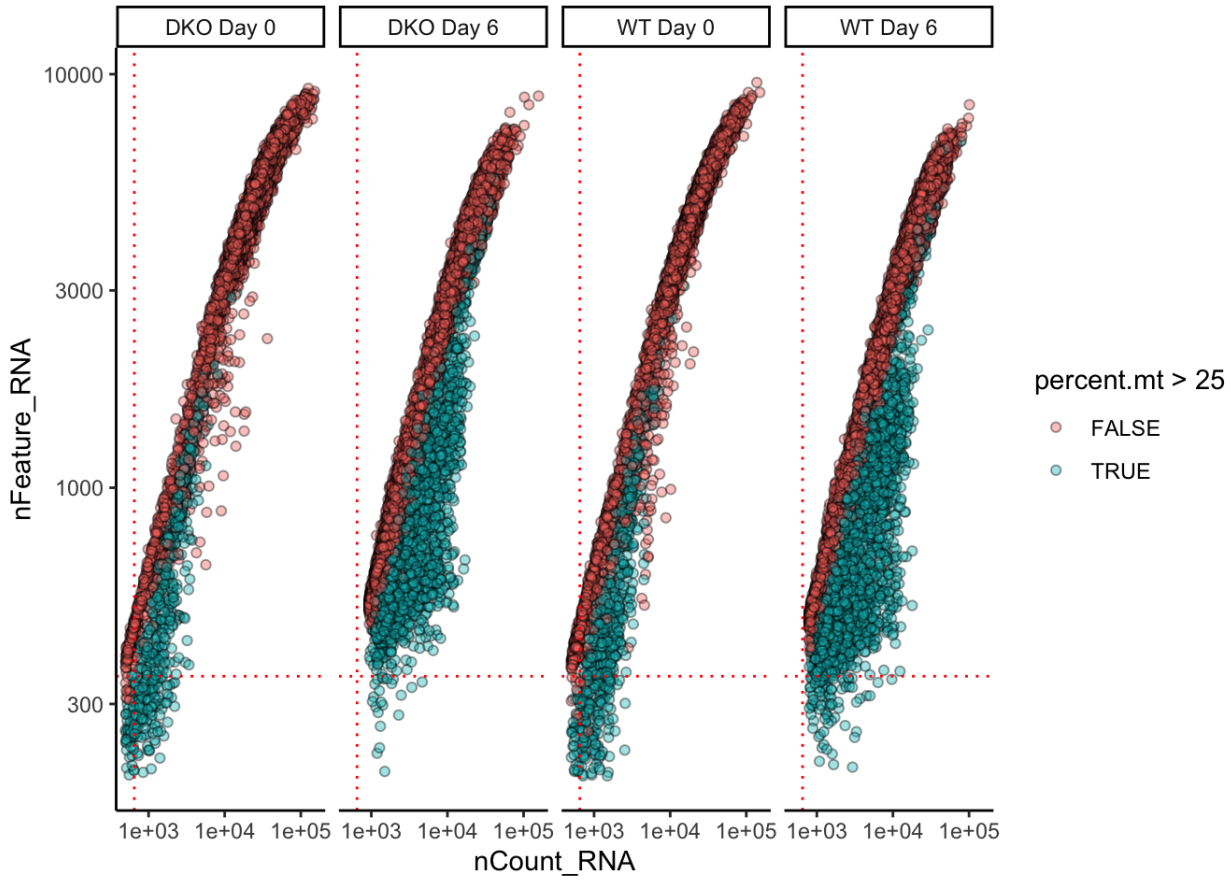


Again, we can also use `ggplot2` directly.

```
ggplot(metadata) +
  geom_point(aes(x=nCount_RNA,y=nFeature_RNA,fill=percent.mt > 10),sh
  theme_classic() +
  scale_x_log10()+
  scale_y_log10()+
  facet_grid(.~cond_tp) +
  geom_vline(xintercept = 650,color="red",linetype="dotted")+
  geom_hline(yintercept=350,color="red", linetype="dotted")
```




```
ggplot(metadata) +
  geom_point(aes(x=nCount_RNA,y=nFeature_RNA,fill=percent.mt > 25),sh
  theme_classic() +
  scale_x_log10()+
  scale_y_log10()+
  facet_grid(.~cond_tp) +
  geom_vline(xintercept = 650,color="red",linetype="dotted")+
  geom_hline(yintercept=350,color="red", linetype="dotted")
```



Warning

Again, quality filtering should be done at a per sample level. Here, there are different distributions by time point, which we may want to consider when filtering.

Is it possible to introduce bias when applying quality filtering on a per sample basis?

Yes, it's possible. There can be variation between the quality of different samples, as seen here. By not adjusting to the specific sample, you can just as easily introduce bias by keeping or removing different populations of cells for each of the samples. Best case scenario is all samples look similar and you can employ a standard filtering threshold across all samples.

Step 2: Decide on thresholds and apply filtering

Because of the different layers, you will need to break down the object to apply different thresholds by group or sample. The easiest way to do this is to work with the metadata and use the cell barcodes for filtering with `Seurat::subset()`.

```
# Set one set of parameters for Day 0 samples;
# keep the rownames (Cell barcodes)
t0<- metadata |> filter(time_point=="Day 0", nFeature_RNA > 350,
                        nCount_RNA >650, percent.mt <10 ) |>
  rownames_to_column("Cell") |> pull(Cell)
```

```
# Set an alternative set of thresholds for Day 6 samples;
# keep the rownames (Cell barcodes)
t6<- metadata |> filter(time_point=="Day 6", nFeature_RNA > 350,
                        nCount_RNA >650, percent.mt <25 ) |>
  rownames_to_column("Cell") |> pull(Cell)

keep<-c(t0,t6)
```

What is |>?

The |> is the native R pipe. It allows you to pipe input from the left to the first argument on the right. See here *(https://www.tidyverse.org/blog/2023/04/base-vs-magrittr-pipe/)*.

Step 3: Filter

Now, we can either use our filtering parameters directly with `subset()` or provide a `cells` argument.

```
# use different parameters; established above
adp_filt<-subset(adp, cells=keep)
```

You do not need to run this:

```
# OR use the same parameters across all samples
adp_filt<-subset(adp, nFeature_RNA > 350 &
                        nCount_RNA >650 & percent.mt <10 )
```

Note

This is not the only way to filter the Seurat object. There is a lot of redundancy built into R. Feel free to explore your options.

Once you have subset your object, it is recommended to run through these plots again. We are skipping this here.

Want more information about quality filtering?

Looking for more information on quality control filters? Here *(https://www.10xgenomics.com/analysis-guides/common-considerations-for-quality-control-filters-for-single-cell-rna-seq-data)* is a useful guide from 10X genomics.

Save the quality filtered object

I recommend saving the object post filtering to work with downstream. This way you can return to this point should you need to.

```
saveRDS(adp_filt,"../outputs/adp_merge_filt.rds")
```

Normalization, find variable features, and scale

Standard Workflow

Normalize

The total number of detected transcripts expressed in a cell is dependent on the amount of mRNA in a cell. Cells naturally vary in the total amount of mRNA expressed. However, the chemistry of the assay inserts technical noise that results in variation in the counts of mRNA even between identical cells. Sources of technical variation include differences in cell lysis, reverse transcription efficiency, and stochastic molecular sampling during sequencing *(https://genomebiology.biomedcentral.com/articles/10.1186/s13059-019-1874-1)*, among others. Because differences in observed expression result from a combination of biological and technical variation, we need to minimize or remove the impact of technical effects in order to compare expression levels across cells.

The standard normalization method in Seurat is a global-scaling normalization method (`?NormalizeData()`, default `LogNormalize`), which transforms the feature counts for each cell by dividing by the total counts of the cell and multiplying by a scale factor (default = 10000) and then applying a natural log with a pseudocount of 1. This should help reduce variation from sequencing depth and somewhat prevent high abundance genes from dominating downstream analyses. *(https://www.10xgenomics.com/analysis-guides/single-cell-rna-seq-data-normalization)*

Find Variable features (`FindVariableFeatures`)

Following normalization, the next step is to find variable features. In most scRNA-seq experiments only a small proportion of the genes will be informative and biologically variable. A subset of cells with high cell to cell variation can be selected using the variance stabilization transformation method of `FindVariableFeatures`. This uses a mean-variance relationship to return 2,000 variable features to prioritize for downstream analyses. You can read more about the method here *(https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6687398/)*.

ScaleData

It is next standard to scale and center the features in the data set prior to dimension reduction or visualization via heatmap. Scaling the data will keep highly expressed genes from dominating our analysis. This step is done using `ScaleData()`, and by default, this is only applied to the variable features found above.

Warning

Scaling only the top variable features could result in warnings later when attempting to visualize differentially expressed genes; this is also true of SCTransform. Note: it is also possible to apply `ScaleData` to the entire data set.

Scaling the data

> Shifts the expression of each gene, so that the mean expression across cells is 0. Scales the expression of each gene, so that the variance across cells is 1.--- Seurat Vignette *(https://satijalab.org/seurat/articles/pbmc3k_tutorial#scaling-the-data)*.

`ScaleData()` can also be used to remove unwanted sources of variation (e.g., cell cycle or percent mitochondrial) using the argument `vars.to.regress`. There is a great tutorial from the Harvard Chan Bioinformatics Core demonstrating how you can explore these unwanted sources prior to deciding whether they should be mitigated.

Standard protocol: NormalizeData, FindVariableFeatures, and ScaleData

```
#normalize
adp_filt <- NormalizeData(adp_filt,
                          normalization.method = "LogNormalize",
                          scale.factor = 10000)

# find variable features
adp_filt <- FindVariableFeatures(adp_filt, selection.method = "vst",
                                 nfeatures = 2000)

# Identify the 10 most highly variable genes
top10 <- head(VariableFeatures(adp_filt), 10)

# plot variable features with and without labels
plot1 <- VariableFeaturePlot(adp_filt)
LabelPoints(plot = plot1, points = top10, repel = TRUE)


adp_filt <- ScaleData(adp_filt, vars.to.regress = "percent.mt")
```

Normalized counts are stored in data layers; for example, we can access the log normalized counts of W10 using `adp_filt@assays$RNA$counts.W10`. `ScaleData()` are stored at `adp_filt[["RNA"]]$scale.data`.

SCTransform

Rather than relying on the above steps (`NormalizeData()`, `FindVariableFeatures()`, and `ScaleData()`), we are going to proceed with a newer method (`SCtransform`) instead. This method uses Pearson residuals for transformation, which better accounts for the overall distribution of expression. This method assigns greater weight to lowly expressed genes that may exhibit cell type specific expression rather than highly expressed genes with broad expression. Read more about the method here *(https://link.springer.com/article/10.1186/s13059-021-02584-9)*.

Things to know

- Transformed data will be available in the SCT assay, which is set as the default after running sctransform.
- During normalization, we can also remove confounding sources of variation, for example, mitochondrial mapping percentage
- The glmGamPoi package substantially improves speed and is used by default if installed, with instructions here --- SCTransform vignette *(https://satijalab.org/seurat/articles/sctransform_vignette)*

Also, by default, the scaled data layer (`adp_filt[["SCT"]]$scale.data`) only contains variable features (3,000). This saves substantial memory.

Let's run the function. Notice that we are regressing out the effect of percent mitochondria. Of note, if we think differences in mitochondrial gene expression are related to biology, we may not want to do this, as it could help with clustering. We can always repeat steps as needed. This analysis can be highly interactive.

```
adp_filt <- SCTransform(adp_filt, vars.to.regress = "percent.mt", ver
```

At this point, the active.assay for `adp_filt` is now "SCT".

Default Assays

It is important to know the arguments for each function used. You can switch between assays, for example, setting the default assay to "RNA", using:

```
#Check default assay
DefaultAssay(object = adp_filt)

#Set default assay
DefaultAssay(object = adp_filt) <- "RNA"
```

Confused about normalization?

I recommend this excellent high level overview *(https://www.youtube.com/watch?v=huxkc2GH4lk)* available by Florian Wagner on YouTube.

There is also a normalization guide by 10X genomics *(https://www.10xgenomics.com/analysis-guides/single-cell-rna-seq-data-normalization)*.

Linear dimension reduction

The role of cell clustering is to identify cells with similar transcriptomic profiles by computing Euclidean distances across genes *(https://bioconductor.org/books/3.15/OSCA.basic/dimensionality-reduction.html#overview)*. However, scRNA-Seq experiments include highly dimensional data, with each cell associated with the expression of thousands of genes, which are impacted by biological and technical factors.

Principal component analysis (PCA) is a linear dimension reduction method applied to highly dimensional data. The goal of PCA is to reduce the dimensionality of the data by transforming the data in a way that maximizes the variance explained. The first PC explains the most variance, followed by the second PC, and so on so on.

> To overcome the extensive technical noise in any single feature (gene) for scRNA-seq data, Seurat clusters cells based on their PCA scores, with each PC essentially representing a 'metafeature' that combines information across a correlated feature set. The top principal components therefore represent a robust compression of the dataset. --- Seurat vignette *(https://satijalab.org/seurat/articles/pbmc3k_tutorial#determine-the-dimensionality-of-the-dataset)*.

Linear dimension reduction via PCA thus eliminates noise and speeds up clustering. Learn more about PCA here *(https://bioconductor.org/books/3.15/OSCA.basic/dimensionality-reduction.html#principal-components-analysis)* and here *(https://github.com/hbctraining/scRNA-seq_online/blob/master/lessons/05_theory_of_PCA.m)*.

We run PCA using `RunPCA()` on our SCtransformed data.

```
adp_filt <- RunPCA(adp_filt, verbose = FALSE,assay="SCT")
```

Important

Use scaled data here.

After we run PCA, we need to decide which PCs to include in downstream analyses. We want to include enough PCs to retain the biological signal, but few enough PCs to avoid interference by noise in the data.

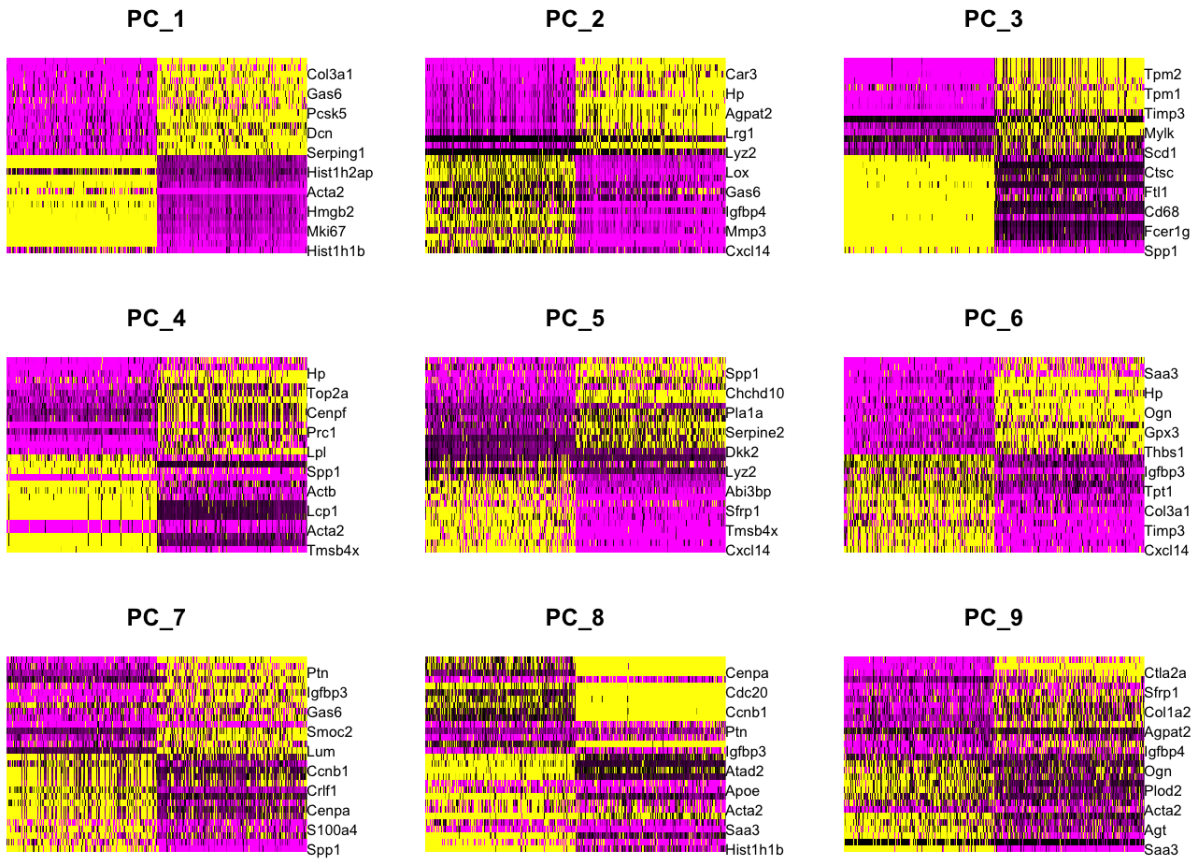
Exploring the PCA results

There are a number of ways to explore the PCA results. Two of the more useful visualizations include the `DimHeatmap()` and `ElbowPlot()`.

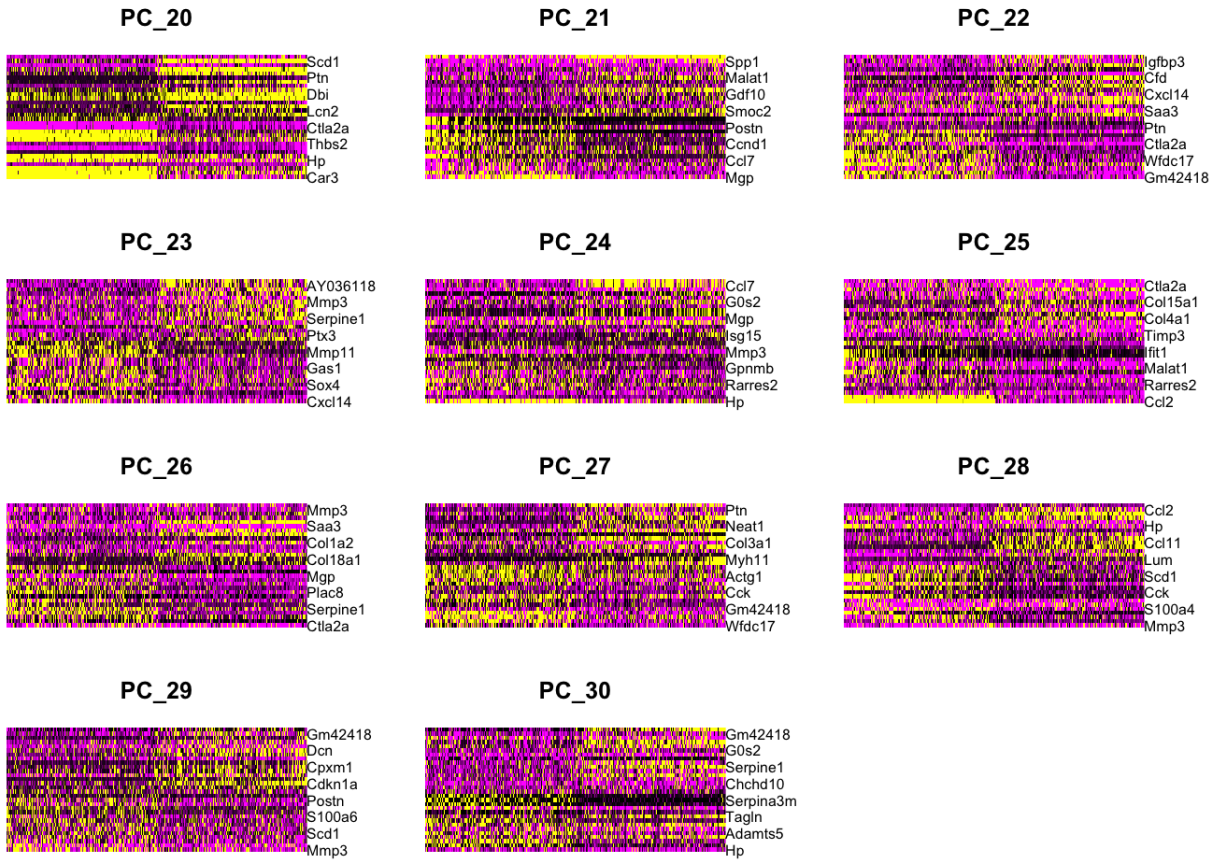
DimHeatmap() allows us to visualize the top genes contributing to each PC. Both cells and genes are sorted by their principal component scores. By default it includes the top 30 genes with the highest and lowest PC loadings.

Here you can see how well a PC separates populations of cells.

```
# dimensions 1 to 9
DimHeatmap(adp_filt, dims = 1:9, cells = 500, balanced = TRUE,ncol=3)
```

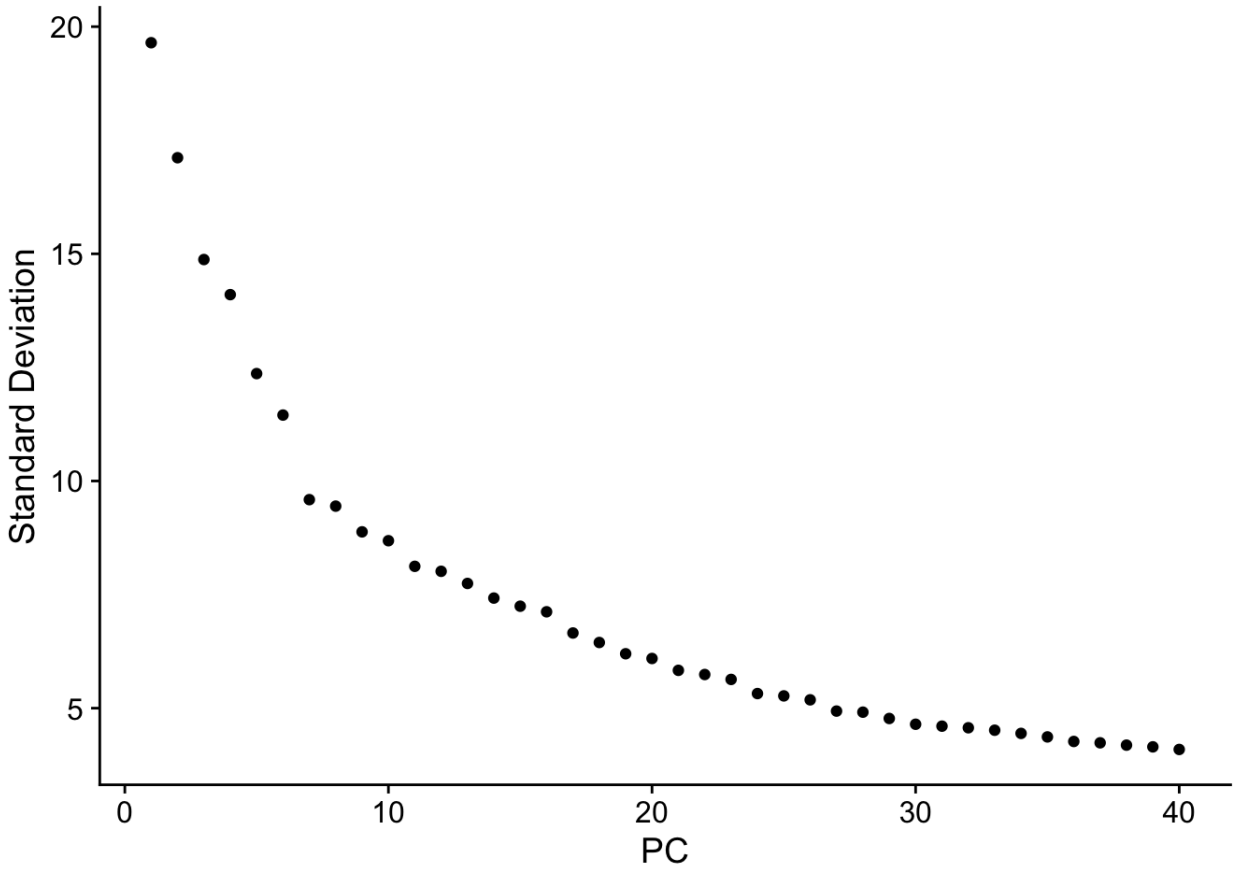



```
#dimensions 20 to 30
DimHeatmap(adp_filt, dims = 20:30, cells = 500, balanced = TRUE,ncol=
```



Complementing the above heatmap, we can also create an elbow plot, which produces a ranking of principle components based on the percentage of variance explained by each one *(https://satijalab.org/seurat/articles/pbmc3k_tutorial#determine-the-dimensionality-of-the-dataset)*.

```
ElbowPlot(adp_filt, ndims = 40)
```



We can see the amount of variance explained tapers off around 30. Perhaps this is a good starting point?

Of note, the Seurat developers indicate that it is safer to use a greater number of PCs when using SCTransform, which is more efficient at eliminating technical effects.

Check out the Seurat - Guided Clustering Tutorial *(https://satijalab.org/seurat/articles/pbmc3k_tutorial)* for more examples of visualizing PCA results.

Clustering

Clustering is used to group cells by similar transcriptomic profiles. Seurat uses a graph based clustering method. You can read more about it here *(https://bioconductor.org/books/3.15/OSCA.basic/clustering.html#background)*.

The first step is to compute the nearest neighbors of each cell based on similarity. This is done using `FindNeighbors()`, which will construct a KNN graph based on the euclidean distance in PCA space, and refine the edge weights between any two cells based on the shared overlap in their local neighborhoods (Jaccard similarity) *(https://satijalab.org/seurat/articles/pbmc3k_tutorial#cluster-the-cells)*. Therefore, an important argument is the dimensions of our PCA that we would like to use.

This then informs `FindClusters()`, which uses an unsupervised learning technique (modularity optimization with the Louvain algorithm) to group multiple data points into clusters based on their similarities. Importantly, this function includes a resolution parameter that sets the 'granularity' of the downstream clustering, with increased values leading to a greater number of clusters *(https://satijalab.org/seurat/articles/pbmc3k_tutorial#cluster-the-cells)*.

> **Choosing the "right" resolution**
>
> It is difficult to know whether the clustering is "correct". This will largely depend on your goals and your experience. You may want more or less resolution based on your scientific question. Are you interested in major types of cells, sub-types, cell states? Novel sub-types and cell states will require greater resolution.

Here, we keep resolution fairly low to get a preliminary idea of the broad cell types. Feel free to play with the resolution to see how this impacts clustering. The Seurat developers suggest a resolution of 0.4-1.2 for data sets of ~3,000 cells.

```
adp_filt <- FindNeighbors(adp_filt, dims = 1:30)
```

```
Computing nearest neighbor graph
```

```
Computing SNN
```

```
adp_filt <- FindClusters(adp_filt, resolution = 0.1)
```

```
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van E

Number of nodes: 42114
Number of edges: 1480064

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9546
Number of communities: 8
Elapsed time: 9 seconds
```

At this point, the `active.ident` is now seurat.clusters.

Now we can visualize our clusters using a non-linear dimension reduction method. UMAPs (uniform manifold approximation and projection) and t-SNE (t -stochastic neighbor embedding) are common reduction / visualization techniques for single cell data sets. UMAP has recently become the gold standard for this type of analysis due to its computational efficiency and ability

to better maintain global structure; though, like t-SNE it is likely much more accurate at local distances. Read more about non-linear methods for visualization here *(https://bioconductor.org/books/3.15/OSCA.basic/dimensionality-reduction.html#non-linear-methods-for-visualization)*.

Here, we run UMAP to embed the complexity of the data within a two-dimensional plot.

```
adp_filt <- RunUMAP(adp_filt,dims = 1:30)
```

```
Warning: The default method for RunUMAP has changed from calling Pyth
To use Python UMAP via reticulate, set umap.method to 'umap-learn' ar
This message will be shown once per session
```

```
21:10:28 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
21:10:28 Read 42114 rows and found 30 numeric columns
```

```
21:10:28 Using Annoy for neighbor search, n_neighbors = 30
```

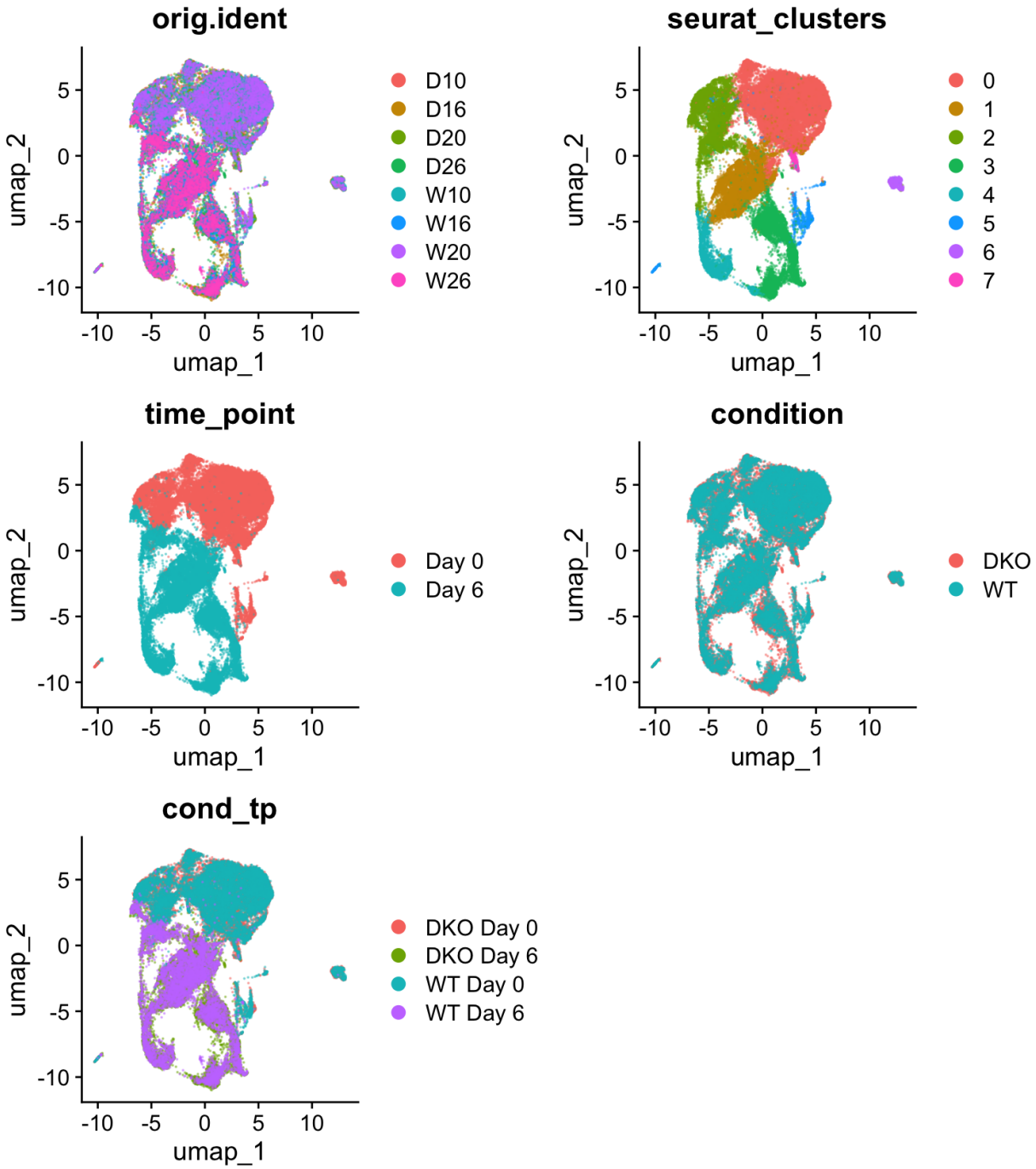
```
21:10:28 Building Annoy index with metric = cosine, n_trees = 50
```

```
0%   10   20   30   40   50   60   70   80   90   100%
```

```
[----|----|----|----|----|----|----|----|----|----|
```

```
**************************************************|
21:10:33 Writing NN index file to temp file /var/folders/hv/m8yfpqmn4
21:10:33 Searching Annoy index using 1 thread, search_k = 3000
21:10:47 Annoy recall = 100%
21:10:49 Commencing smooth kNN distance calibration using 1 thread wi
21:10:51 Initializing from normalized Laplacian + noise (using RSpect
21:10:52 Commencing optimization for 200 epochs, with 1854092 positiv
21:11:15 Optimization finished
```

```
DimPlot(adp_filt, reduction = "umap", group.by = c("orig.ident", "seu
        alpha=0.4, ncol=2)
```



These data do not exhibit batch effects by sample or condition, but show major differences by time point. This is expected given the biology of the samples.

Take a moment to save your object at this point.

```
saveRDS(adp_filt, "../outputs/adp_merge_filt_sctran_clust0.1.rds")
```

Note

This saved R object (`adp_merge_filt_sctran_clust0.1.rds`) will be used as the starting point in the next seminar.

A word about integration.

Integration is the process of aligning the same cell types across samples, treatments, data sets, batches, etc. Clustering should represent biological differences and not technical artifacts. Integration is not always necessary. You should run through the standard workflow and decide whether integration is required post-clustering. If cells are clustering by cell type rather than other factors (e.g., sample), you can likely skip integration. See the integration vignette *(https://satijalab.org/seurat/articles/seurat5_integration)*.

Here, we aren't integrating.

Finding Cluster Biomarkers

Now, that we have clusters, we can use differential expression analysis to uncover markers that define our clusters. These markers can be used to assign cell types to our clusters.

First, because we are working with a merged data set (with multiple SCT models), we need to run `PrepSCTFindMarkers()` prior to differential expression testing.

> This allows us to use the 'corrected counts' that are stored in the data slot of the the SCT assay. Corrected counts are obtained by setting the sequencing depth for all the cells to a fixed value and reversing the learned regularized negative-binomial regression model. `PrepSCTFindMarkers()` ensures that the fixed value is set properly. --- Archived Seurat Vignette *(https://satijalab.org/seurat/archive/v4.3/sctransform_v2_vignette#:~:text=Prior%20to%20performing%20differential%20expression,stimulated%*

```
adp_filt<-PrepSCTFindMarkers(adp_filt,verbose=T)
```

```
Found 8 SCT models. Recorrecting SCT counts using minimum median cour
```

Once this is done, we can perform differential expression testing for each cluster compared to all other clusters using `FindAllMarkers()`. `only.pos = TRUE` will only return marker genes with an avg_log2FC.

Note

There are many available options for the method used for differential expression testing. See the `test.use` parameter. By default, a Wilcoxon Rank Sum test is applied.

```
#requires presto installation to speed up the next step
# devtools::install_github('immunogenomics/presto')
library(presto)
# find all markers
adp_filt_markers <- FindAllMarkers(adp_filt, only.pos = TRUE)
```

```
Calculating cluster 0
```

```
Calculating cluster 1
```

```
Calculating cluster 2
```

```
Calculating cluster 3
```

```
Calculating cluster 4
```

```
Calculating cluster 5
```

```
Calculating cluster 6
```

```
Calculating cluster 7
```

```
#ordering the results
adp_filt_markers <- adp_filt_markers %>%
  arrange(cluster,desc(avg_log2FC), desc(p_val_adj))

#examine a small subset
adp_filt_markers %>%
    group_by(cluster) %>%
    slice_max(n = 5, order_by = avg_log2FC)
```

```
# A tibble: 41 × 7
# Groups:   cluster [8]
       p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
```

```
       <dbl>      <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
 1 3.96e- 46       4.02 0.01  0.001 7.57e- 42 0       Erv3
 2 2.29e-144       3.96 0.032 0.002 4.37e-140 0       Pdlim3
 3 0               3.70 0.159 0.014 0         0       Mlana
 4 1.47e-140       3.55 0.033 0.003 2.80e-136 0       Nat8f3
 5 0               3.47 0.158 0.02  0         0       Scg2
 6 0               3.50 0.257 0.044 0         1       Cxcl13
 7 1.28e- 66       3.15 0.016 0.002 2.45e- 62 1       Dlx3
 8 0               3.15 0.38  0.054 0         1       Serpina3m
 9 1.64e- 32       3.13 0.013 0.003 3.13e- 28 1       Pyy
10 0               3.05 0.427 0.056 0         1       Scara5
# ℹ 31 more rows
```

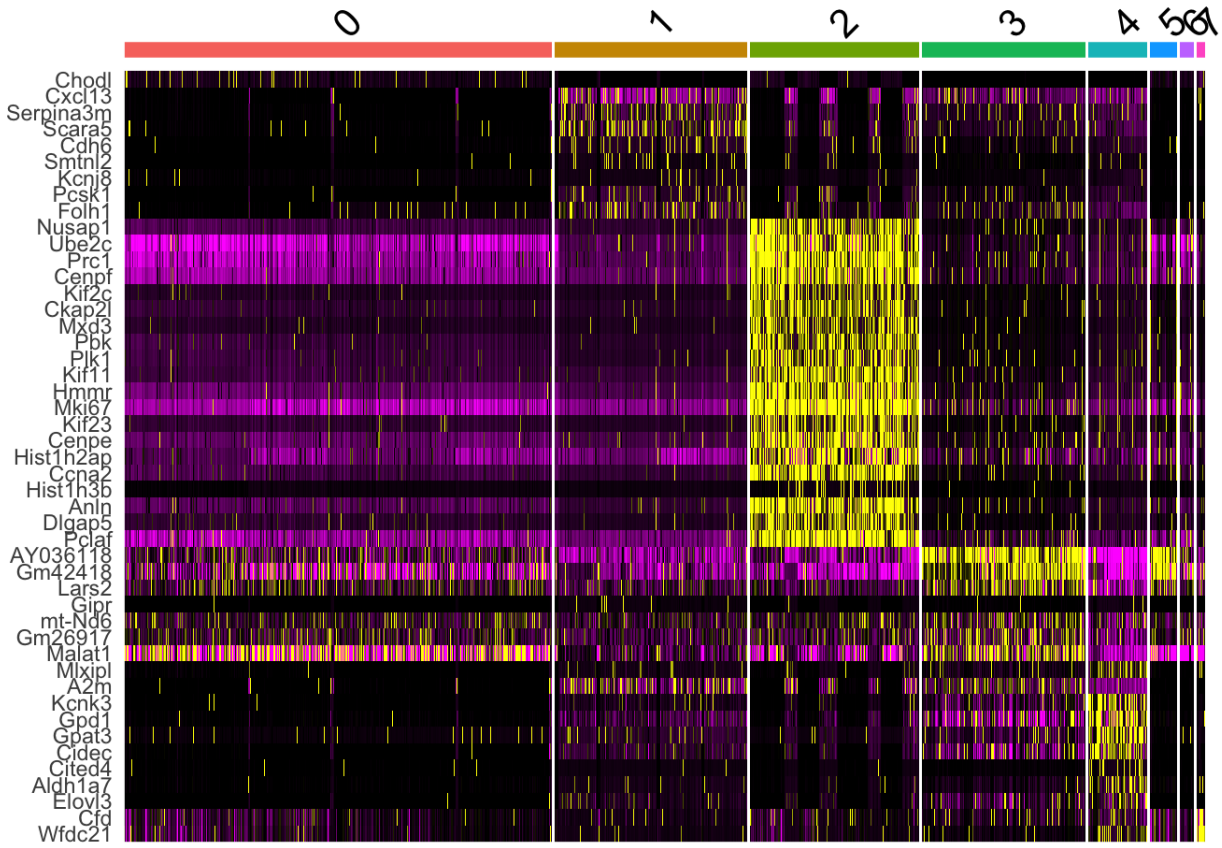
In the output of `adp_filt_markers`,

> pct.1 is the percentage of cells in the cluster where the gene is detected, while pct.2 is the percentage of cells on average in all the other clusters where the gene is detected. A gene to be considered as an IDEAL cluster marker is expected to be expressed exclusively in that cluster and silenced in all others and thus pct.1 will be more towards 1 and pct.2 towards 0.---Biostars Post *(https://www.biostars.org/p/410170/)*

We can use a heatmap to visualize potential marker genes.

```
top20<-adp_filt_markers %>%
    group_by(cluster) %>%
    dplyr::filter(avg_log2FC > 1) %>%
    slice_head(n = 20) %>%
    ungroup()
DoHeatmap(adp_filt, features = top20$gene) + NoLegend()
```

```
Warning in DoHeatmap(adp_filt, features = top20$gene): The following
were omitted as they were not found in the scale.data slot for the SC
Gm13709, Vnn1, Igfals, Pla2g5, Sctr, C6, Spink13, Igsf21, Myl1, Atp1a
Gm37800, Mup3, 2010106E10Rik, Vnn3, Saa4, Rnase2b, Smim5, Spi1, C1qc
Mgl2, C1qb, Dock2, Ly9, Slamf7, Clec4e, Bcl2a1a, Bcl2a1d, Ly86, Fcrls
Tifab, Slc28a2, Cst7, Gm5150, Myo1g, Gm26740, Tie1, Icam2, Gimap8, Da
Zfp366, Mall, Gimap1, Kcne3, Gm14207, Gimap4, Gpihbp1, Gimap5, Emcn,
C1qtnf9, Nts, Gm32688, Myct1, Sp5, Sox17, Sox6os, Apoc3, Paqr9, Ucp1
Thrsp, Gm45607, Acvr1c, Adora1, Acot5, Ffar4, Acot3, A530016L24Rik, C
A830018L16Rik, Olfr810, Amt, Glt1d1, Masp2, Gm17638, Gm43585, Gm4315C
4930565N06Rik, Gm17300, Cbln2, Snap25, Galnt15, Agtr2, Sptssb, Gm501
Hcn4, Gm45924, Hmgcs2, Pyy, Dlx3, Slc22a18, Nat8f5, Cyb5r2, Cfi, Ppef
Gm15866, Tnfsf8, Kcnma1, Sync, 2810433D01Rik, Mamdc2, Apba2, 8030423F
Scg2, Nat8f3, Mlana, Pdlim3, Erv3
```

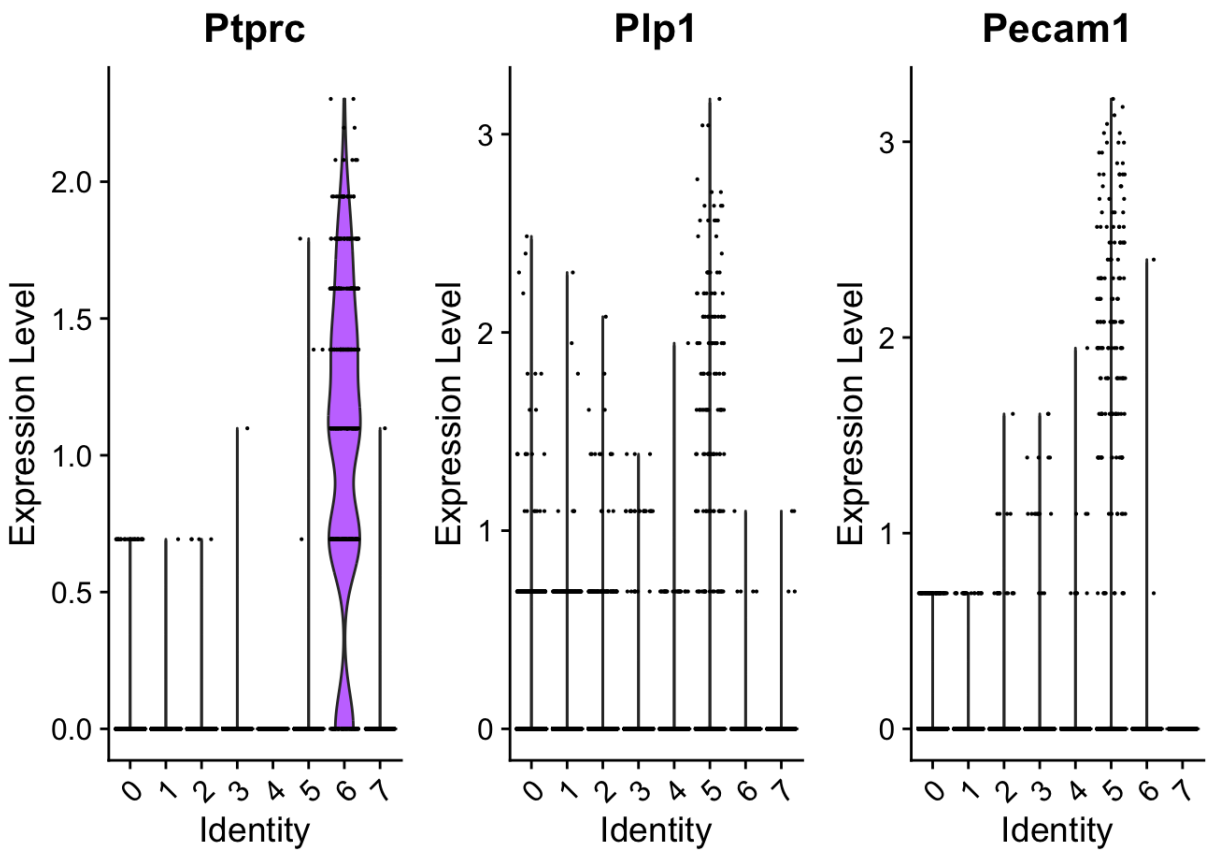


We can also use `VlnPlot()` and `FeaturePlot()` to compare marker expression. VlnPlot() shows expression probability distributions across clusters, and FeaturePlot() visualizes feature expression on a tSNE or PCA plot. *(https://satijalab.org/seurat/articles/pbmc3k_tutorial#finding-differentially-expressed-features-cluster-biomarkers)*

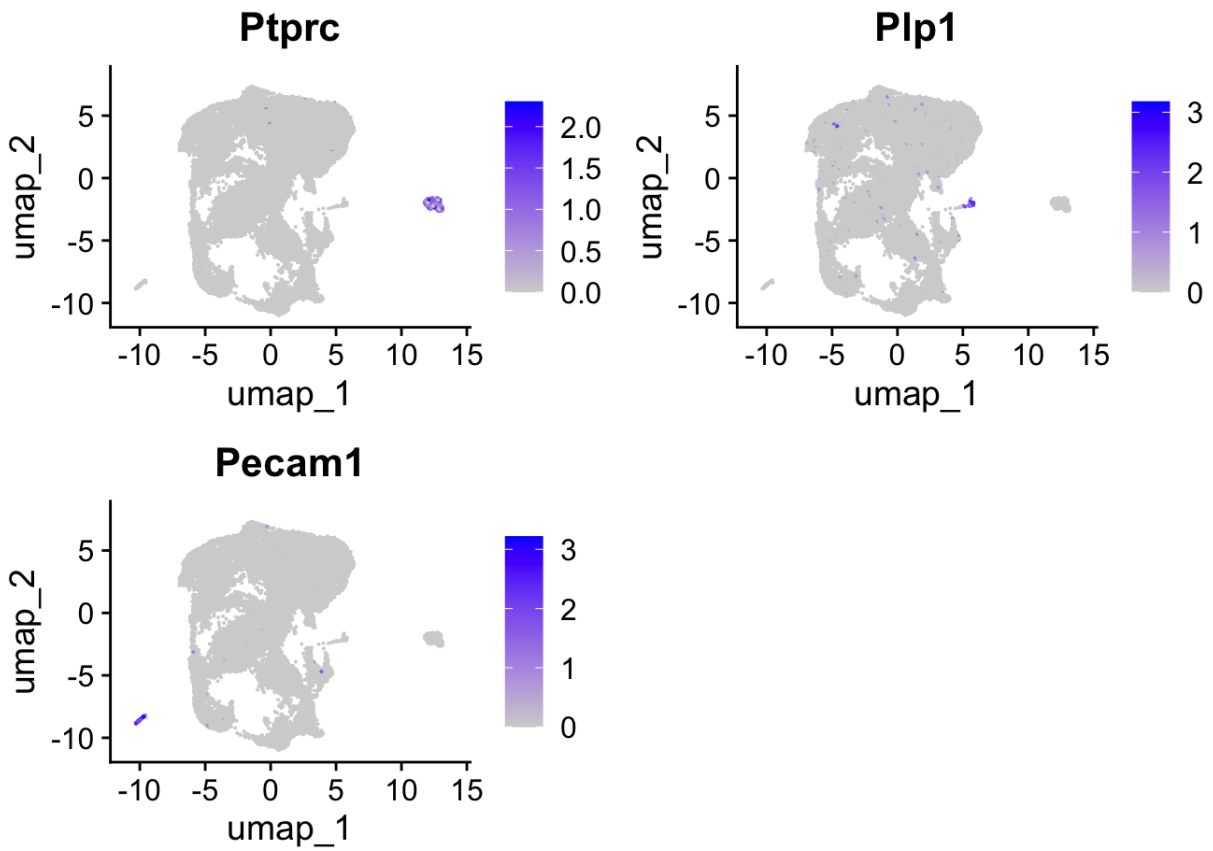
In the case of this example data set, we can identify our clusters using published markers associated with the data (from the original publication).

```
#contaminants
contam=c("Ptprc", "Plp1", "Pecam1")
#beige adipocytes
beige=c("Ucp1","Ppargc1a","Elovl3","Cidea")
#preadipocytes
preadip<-c('Mmp3', 'Cd142', 'Itgb1')
#proliferating
prolif<-'Mki67'
#differentiating adipocytes
diffadip<-c('Col5a3', 'Serpina3n')

VlnPlot(adp_filt, features=contam)
```




```
FeaturePlot(adp_filt,features=contam)
```



For example, clusters 5 and 6 are likely contaminant cell populations, which could be removed.

- CD45+ (gene = Ptprc)
- Pecam+ (gene = Pecam1) = endothelial cells
- Plp1+ = neuronal / glial

The next seminar will discuss differential expression analysis in greater detail.

Cell Annotation

There are many tools and strategies available for cell annotation. I've included a non-comprehensive list for your convenience:

- Azimuth *(https://satijalab.github.io/azimuth/index.html)*
- SingleR *(https://bioconductor.org/packages/release/bioc/html/SingleR.html)*
- scType *(https://github.com/IanevskiAleksandr/sc-type)*
- Garnet *(https://cole-trapnell-lab.github.io/garnett/)*
- Digital Cell Sorter *(https://github.com/sdomanskyi/DigitalCellSorter)*.
- Cell Marker 2.0 *(http://117.50.127.228/CellMarker/index.html)*

Here is a list of resources that also may be helpful from 10X genomics:

- https://www.10xgenomics.com/analysis-guides/automated-cell-type-annotation-from-r-to-loupe-using-louper
- https://www.10xgenomics.com/analysis-guides/web-resources-for-cell-type-annotation *(https://www.10xgenomics.com/analysis-guides/web-resources-for-cell-type-annotation)*
- https://www.10xgenomics.com/blog/annotating-single-cell-datasets-a-q-a-covering-challenges-of-automated-and-manual-methods *(https://www.10xgenomics.com/blog/annotating-single-cell-datasets-a-q-a-covering-challenges-of-automated-and-manual-methods)*

As well as this chapter *(https://bioconductor.org/books/3.15/OSCA.basic/cell-type-annotation.html#cell-type-annotation)* from OSCA.

Acknowledgements:

The following sources inspired this content:

- https://www.sc-best-practices.org *(https://www.sc-best-practices.org)*
- https://hbctraining.github.io/scRNA-seq_online/ *(https://hbctraining.github.io/scRNA-seq_online/)*
- https://bioconductor.org/books/3.15/OSCA.basic/ *(https://bioconductor.org/books/3.15/OSCA.basic/)*

Getting Started with Seurat: Differential Expression and Classification

1. Introduction and Learning Objectives

This tutorial has been designed to demonstrate common secondary analysis steps in a scRNA-Seq workflow. We will start with a merged Seurat Object with multiple data layers representing multiple samples that have already been filtered and undergone preliminary clustering.

Throughout this tutorial we will

1. Explore setting and visualizing identities in a single cell dataset\
2. Perform differential expression analysis through Seurat\
3. Use differentially expressed genes to classify cells\
4. Run a case test of cell type annotation using SingleR

This tutorial largely follows the standard unsupervised clustering workflow by Seurat *(https://satijalab.org/seurat/articles/pbmc3k_tutorial)* and the differential expression testing vignette *(https://satijalab.org/seurat/articles/de_vignette)*, with slight deviations and a different data set.

> **Warning**
>
> This document in no way represents a complete analysis. Here, we are demonstrating many of the standard steps in Seurat. Results here would be considered preliminary, requiring greater care and consideration of biology.

The data

In this tutorial, we will continue to use data from Nanduri et al. 2022, *Epigenetic regulation of white adipose tissue plasticity and energy metabolism by nucleosome binding HMGN proteins*, published in *Nature Communications (https://www.nature.com/articles/s41467-022-34964-5)*.

As a reminder, this study examined "the role of HMGNs in white adipocyte browning by comparing wild-type (WT) mice and cells to genetically derived mice and cells lacking the two major members of the HMGN protein family, HMGN1 and HMGN2 (DKO mice)." *(https://www.nature.com/articles/s41467-022-34964-5)*. The experimental design included WT vs DKO mice, 2 replicates each, at two time points, 0 and 6 days. At day 0, cells were in a preadipocyte state, while at day 6 they had differentiated into adipocytes. Each time point had two replicates of each condition (WT vs DKO), for a total of 8 samples.

An R project directory containing these data and associated files can be downloaded here.

Consider the biology in your analysis.

The biology of your experiment will inform your analysis. While scRNA-Seq can be exploratory, you should have some general idea of the cell types that you expect. Knowing some characteristics of expected cells, for example, cell size or whether cells are complex or more energetically active is helpful for data processing.

For example, in this experiment, we expect cells transitioning from preadipocytes to adipocytes and then beige adipocytes.

Load the packages

```
library(tidyverse) # dplyr and ggplot2; CRAN
library(Seurat) # Seurat toolkit; CRAN
library(hdf5r) # for data import; CRAN
library(patchwork) # for plotting; CRAN
library(presto) # for differential expression; Github
library(glmGamPoi) # for sctransform; Bioconductor
library(ggplot2) # for visualization; installed with tidyverse
library(dplyr) #just to make sure it was called; installed with tidy


library(SingleR) # for cell type annotation; Bioconductor
library(celldex) # for cell type annotation reference; Bioconductor
library(MAST) # for differential expression; Bioconductor
```

Load the Seurat Object

Here, we will start with the data stored in a Seurat object. For instructions on data import and creating the object, see an *Introduction to scRNA-Seq with R (Seurat) (https://bioinformatics.ccr.cancer.gov/docs/getting-started-with-scrna-seq/IntroToR_Seurat/)* and *Getting Started with Seurat: QC to Clustering (https://bioinformatics.ccr.cancer.gov/docs/getting-started-with-scrna-seq/Seurat_QC_to_Clustering)*.

```
adp <- readRDS("../outputs/adp_merge_filt_sctran_clust0.1.rds")
mem.maxVSize()
```

```
[1] 16384
```

```
mem.maxVSize(vsize=16384*4)
```

```
[1] 65536
```

Tip

Many of the steps used in differential expression can be very memory-intensive, and on some computers, this could lead to errors or warnings that look like this:

```
Found 8 SCT models. Recorrecting SCT counts using minimum median counts: 8146
  |++++++++++++++++++++++++++++++++++++++++++++++++++| 100% elapsed=01m 58s
Error: vector memory limit of 16.0 Gb reached, see mem.maxVSize()
Error during wrapup: vector memory limit of 16.0 Gb reached, see mem.maxVSize()
Error: no more error handlers available (recursive errors?); invoking 'abort' resta
```

This particular error indicates that there isn't enough virtual memory allocated to run a process (in this case, `PrepSCTFindMarkers`). This can be corrected by adjusting the memory that R is allowed to access.

```
mem.maxVSize()
#[1] 16384
mem.maxVSize(vsize=16384*4)
#[1] 65536
```

For more details, please see the documentation on memory allocation *(https://stat.ethz.ch/R-manual/R-devel/library/base/html/memlimits.html)*.

Examine the object:

```
glimpse(adp)
```

```
Formal class 'Seurat' [package "SeuratObject"] with 13 slots
  ..@ assays      :List of 2
  .. ..$ RNA:Formal class 'Assay5' [package "SeuratObject"] with 8 s
  .. ..$ SCT:Formal class 'SCTAssay' [package "Seurat"] with 9 slots
  ..@ meta.data   :'data.frame':    42114 obs. of  11 variables:
  .. ..$ orig.ident     : chr [1:42114] "D10" "D10" "D10" "D10" ...
  .. ..$ nCount_RNA     : num [1:42114] 11188 32832 28515 18954 2718
  .. ..$ nFeature_RNA   : int [1:42114] 3383 5823 5002 4305 4586 501
  .. ..$ condition      : chr [1:42114] "DKO" "DKO" "DKO" "DKO" ...
  .. ..$ time_point     : chr [1:42114] "Day 0" "Day 0" "Day 0" "Day
  .. ..$ cond_tp        : chr [1:42114] "DKO Day 0" "DKO Day 0" "DKO
  .. ..$ percent.mt     : num [1:42114] 3.16 1.8 1.39 1.89 1.33 ...
  .. ..$ nCount_SCT     : num [1:42114] 23551 25705 25491 23790 2536
  .. ..$ nFeature_SCT   : int [1:42114] 3843 5809 5002 4306 4586 501
  .. ..$ SCT_snn_res.0.1: Factor w/ 8 levels "0","1","2","3",..: 8 1
  .. ..$ seurat_clusters: Factor w/ 8 levels "0","1","2","3",..: 8 1
  ..@ active.assay: chr "SCT"
  ..@ active.ident: Factor w/ 8 levels "0","1","2","3",..: 8 1 1 1 1
  .. ..- attr(*, "names")= chr [1:42114] "D10_AAACCCAAGATGCTTC-1" "D
  ..@ graphs      :List of 2
```

```
.. ..$ SCT_nn :Formal class 'Graph' [package "SeuratObject"] with 7
.. ..$ SCT_snn:Formal class 'Graph' [package "SeuratObject"] with 7
..@ neighbors   : list()
..@ reductions  :List of 2
.. ..$ pca :Formal class 'DimReduc' [package "SeuratObject"] with 9
.. ..$ umap:Formal class 'DimReduc' [package "SeuratObject"] with 9
..@ images      : list()
..@ project.name: chr "Adipose"
..@ misc        : list()
..@ version     :Classes 'package_version', 'numeric_version'  hidd
.. ..$ : int [1:3] 5 0 1
..@ commands    :List of 5
.. ..$ SCTransform.RNA      :Formal class 'SeuratCommand' [package
.. ..$ RunPCA.SCT           :Formal class 'SeuratCommand' [package
.. ..$ FindNeighbors.SCT.pca:Formal class 'SeuratCommand' [package
.. ..$ FindClusters         :Formal class 'SeuratCommand' [package
.. ..$ RunUMAP.SCT.pca      :Formal class 'SeuratCommand' [package
..@ tools       : list()
```

Reviewing the Data

As we can see above, the `glimpse` command shows the metadata that can be used to classify the cells. Within Seurat, the metadata is used to define the "identity" of the dataset. This is critical, as the labels within the categories will be used to determine which groups of cells are being compared. One method of viewing the available labels is to use the `table` function, which lists the frequency of each label:

```
table(adp$cond_tp)
```

```
DKO Day 0 DKO Day 6  WT Day 0  WT Day 6
    11655     10512     11807      8140
```

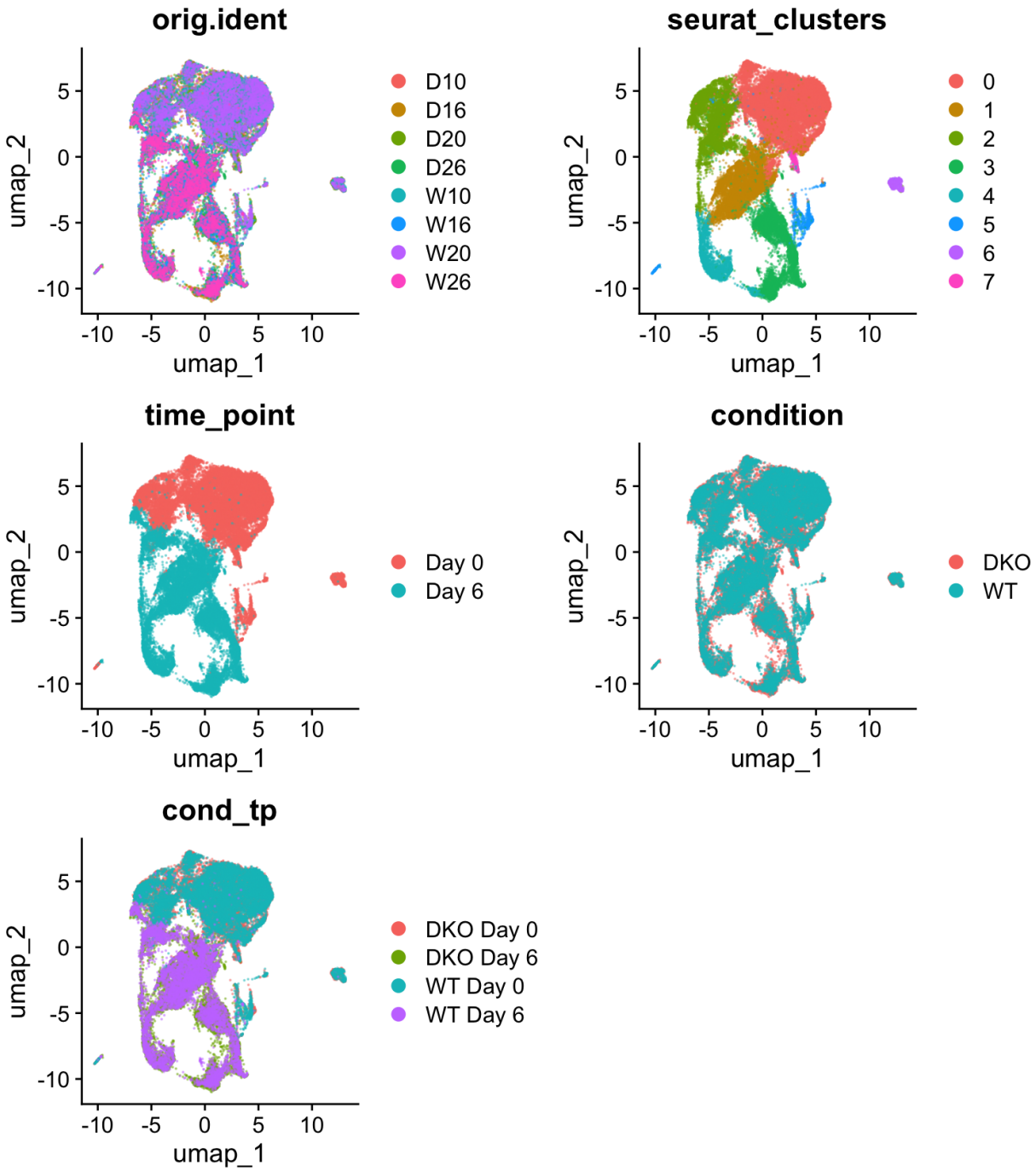
The `table` function can also be used in two dimensions:

```
table(adp$cond_tp,adp$orig.ident)
```

```
            D10  D16  D20  D26  W10  W16  W20  W26
DKO Day 0  6990    0 4665    0    0    0    0    0
DKO Day 6     0 4674    0 5838    0    0    0    0
WT Day 0      0    0    0    0 6780    0 5027    0
WT Day 6      0    0    0    0    0 3781    0 4359
```

Before running any sort of differential expression, the identity of the Seurat object needs to be explicitly defined. Different identities can be assigned, based on what needs to be analyzed. Recall this visualization of the original identities, clusters, time point, experimental condition, and combined time point and condition:

```
DimPlot(adp, reduction = "umap", group.by = c("orig.ident", "seurat_c
        alpha=0.4, ncol=2)
```



Here we will set the identity to the clusters, and view the frequency of the resulting cell identities:

```
Idents(adp) = "SCT_snn_res.0.1"
table(Idents(adp))
```

```
    0     1     2     3     4     5     6     7
16954  7635  6731  6481  2326  1085   556   346
```

Alternate identity classification

If there are identities defined in another vector, the vector values can be explicitly assigned to the dataset, without adding the identities to the metadata itself:

```
clusters=adp$SCT_snn_res.0.1
Idents(adp)=clusters
```

However, if the identities are overwritten, you will need to call the vector again, unless the vector is added to the metadata.

2. Differential Expression

What is differential expression analysis?

Differential expression analysis is the process of identifying genes that have a significant difference in expression between two or more groups. For many sequencing experiments, regardless of methodology, differential analysis lays the foundation of the results and any biological interpretation.

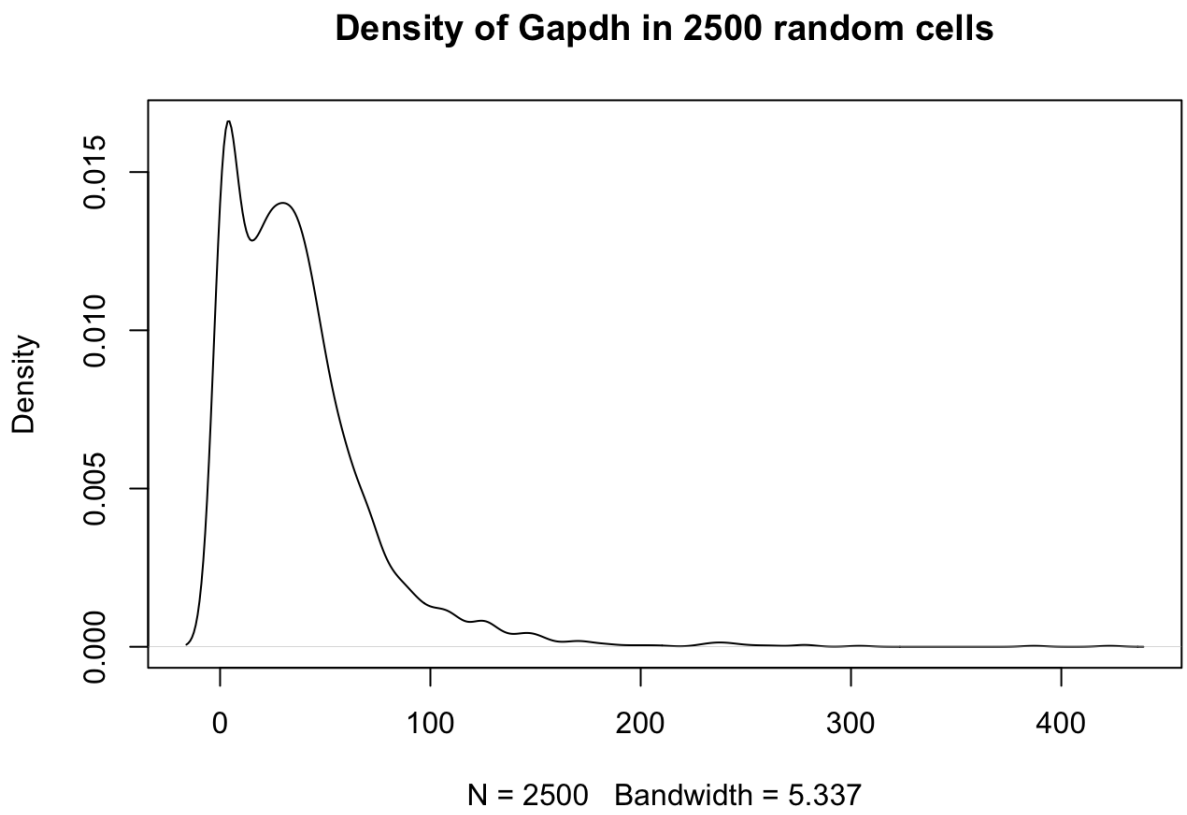
Different tools can be used for differential gene expression analysis, depending on the underlying statistics. One of the best known statistical tests is the Student's t-test *(https://en.wikipedia.org/wiki/Student%27s_t-test)*, which tests for how likely it is that two sets of measurements come from the same normal distribution. However, this does assume that the data is normally distributed *(https://en.wikipedia.org/wiki/Normal_distribution)*, meaning that the data has the distinctive bell curve shape. Should this assumption be inconsistent, a non-parametric version of this test is the Wilcoxon rank-sum test *(https://en.wikipedia.org/wiki/Mann%E2%80%93Whitney_U_test)*, which tests how likely it is that one population is larger than the other, regardless of probability distribution size. Alternative distribution shapes can include bimodal or multimodal distributions, with multiple distinct peaks, and skewed distributions with data that favors one side or another (examples *(https://onlinelibrary.wiley.com/doi/pdf/10.1016/j.pmrj.2012.10.013)*).

For many sequencing experiments, the assumption of normality is rarely guaranteed. Some of the more popular tools for bulk RNASeq experiments, such as DESeq2 *(https://bioconductor.org/packages/release/bioc/html/DESeq2.html)*, limma *(https://bioconductor.org/*

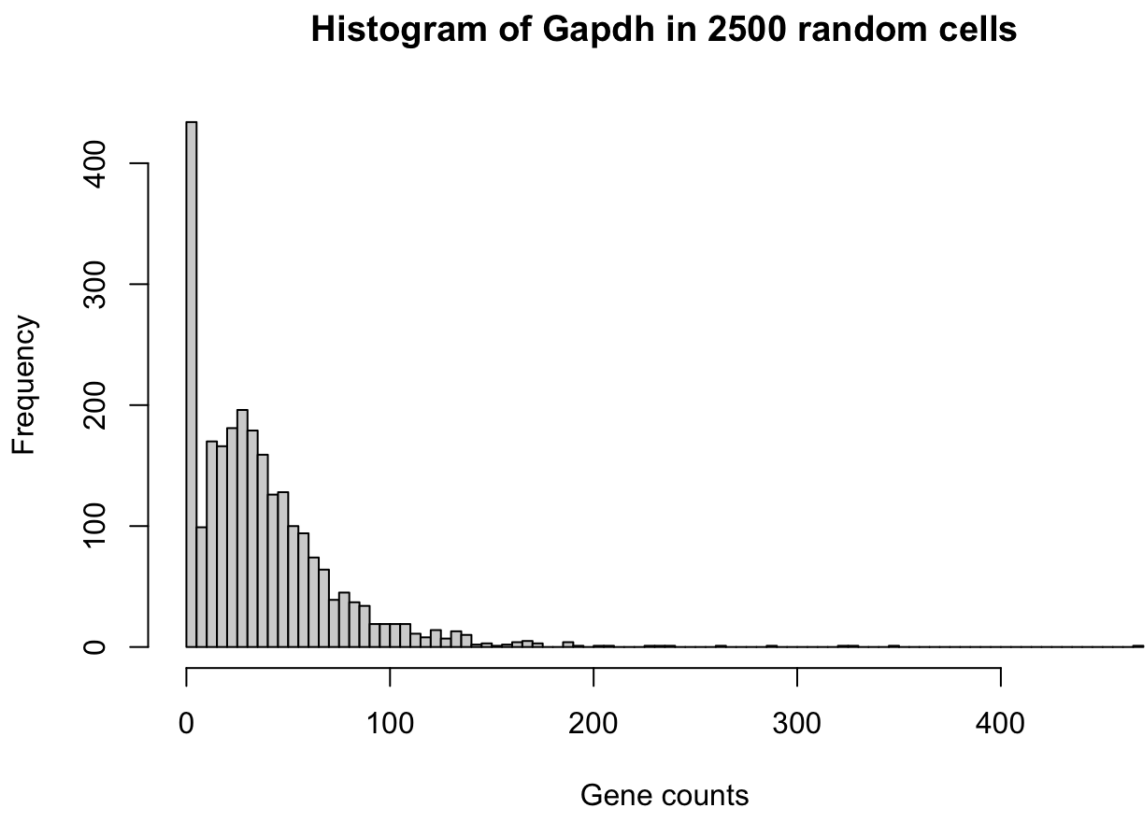
packages/release/bioc/html/limma.html), and edgeR *(https://bioconductor.org/packages/release/bioc/html/edgeR.html%7Btarget=%22_blank%22%7D)*, acknowledge this, and use different statistical models to identify and interpret differences in gene expression.

Single cell RNASeq is notorious for not having a normal distribution of gene expression. For technical reasons, scRNASeq has a transcript recovery rate between 60%-80%, which naturally skews the gene expression towards the non-expressed end.

```
plot(density(sample(JoinLayers(adp@assays$RNA)$count["Gapdh",],2500)
```




```
hist(sample(JoinLayers(adp@assays$RNA)$count["Gapdh",],2500),breaks=
```



Note

Viewing the code, `JoinLayers` was called to link all the data sets together. This is a new feature of Seurat5, and is required for analyzing data after integration and batch correction (ref *(https://satijalab.org/seurat/articles/seurat5_integration)*).

The Seurat tool acknowledges this, and by default uses the Wilcoxon rank-sum test to identify differentially expressed genes, via the `presto` package. Another algorithm that is often used is MAST, which implements a hurdle model to determine the likelihood that a gene is genuinely not expressed in a cell, or if it is more likely a technical dropout.

Running Differential Expression in Seurat

The primary means of running differential expression in Seurat is through the `FindMarkers` function. The main usage for this function is as follows:

```
FindMarkers(object,ident.1=   ...,   ident.2=...,   test.use="wilcox",
min.pct = 0.01, logfc.threshold = 0.1)
```

- `object`: The Seurat object being examined
- `ident.1`: The first or main label in the comparison. Positive fold change values indicate upregulation in `ident.1`

- `ident.2`: The second label in the comparison. If `ident.2` is not defined, it is automatically set to all remaining cells in the comparison. In contrast to `ident.1`, negative fold changes indicated upregulation in `ident.2`.
- `test.use`: The statistical test that is used to compare genes between `ident.1` and `ident.2`. The default values is the Wilcoxon rank-sum test
- `min.pct`: The minimum fraction of cells in at least one of the two groups that must express the gene in order to be compared. If this value is not exceeded in *both* populations, the gene is excluded from the differential expression analysis.
- `logfc.threshold`: The minimum difference in average fold change between the two populations before the gene is evaluated. If the gene is below the threshold, i.e. the expression between the populations is too similar, the gene will excluded from the DE analysis.

In the event that a user wants to recover DE statistics for all genes, `min.pct` and `logfc.threshold` can be set to 0. Additional parameters, including other statistical tests, can be found in the Help menu for `FindMarkers`.

Prepare the dataset for analysis

In Seurat (since version 4), differential analysis requires a preprocessing step to appropriately scale the normalized SCTransform assay across samples:

```
adp = PrepSCTFindMarkers(adp)
```

```
Found 8 SCT models. Recorrecting SCT counts using minimum median cour
```

As covered earlier, the identities of the Seurat object will also need to be defined. To start, we will use the clusters that were generated in the previous workshop:

```
Idents(adp) = "SCT_snn_res.0.1"
```

It is also strongly recommended to explicitly set the `SCT` assay as the basis for the differential expression. Earlier workflows that do not use SCTransform for normalizaton use log-normalization and scaling on the `RNA` assay, and should be treated as such.

```
DefaultAssay(adp) = "SCT"
```

Setting up differential expression on a different assay

In the event that a different assay is being used for differential expression (protein expression, ATACSeq data, initial RNA data), the assay is defined with the same command. In this case, the `PrepSCTFindMarkers` function will not need to be run.

```
DefaultAssay(adp) = "RNA"
```

or

```
DefaultAssay(adp)="CITESeq"
```

Warning

Under no circumstances should the batch corrected or `integrated` assay be used for differential expression. The batch correction is used primarily for dimensionality reduction, UMAP/T-SNE visualization, and clustering. In order to optimize this process, only a select subset of highly variable genes (~3000) is used, and this is reflected in the size of the assay. If differential expression is run on this assay, the differences will be accentuated, and only return the ~3000 genes that were selected for batch correction.

2a. FindAllMarkers

The `FindAllMarkers` function is particularly useful in identifying the differentially expressed genes that distinguish several groups, such as seen here in the clusters. What makes this unique is that none of the identities are initially defined, as each identity will be compared with the rest of the cohort.

```
de_allClusters = FindAllMarkers(adp, test.use="wilcox", min.pct=0.1,
```

```
Calculating cluster 0
```

```
Calculating cluster 1
```

```
Calculating cluster 2
```

```
Calculating cluster 3
```

```
Calculating cluster 4
```

```
Calculating cluster 5
```

```
Calculating cluster 6
```

```
Calculating cluster 7
```

Note

For expediency, this is run with a stricter `min.pct` threshold on the default Wilcoxon rank-sum test, and turns on the function-specific parameter `only.pos`, which is used to return only positively expressed markers from each cluster, as opposed to markers that are both significantly upregulated and downregulated.

We can peek at the results using the following commands:

```
head(de_allClusters)
```

```
       p_val avg_log2FC pct.1 pct.2 p_val_adj cluster    gene
Emp3       0   1.437535 0.866 0.290         0       0   Emp3
Tagln      0   1.647233 0.852 0.299         0       0  Tagln
Mfap5      0   1.877743 0.958 0.418         0       0  Mfap5
Igfbp6     0   2.028028 0.806 0.281         0       0 Igfbp6
Cd9        0   1.678611 0.760 0.261         0       0    Cd9
Acta2      0   1.559173 0.792 0.295         0       0  Acta2
```

```
nrow(de_allClusters)
```

```
[1] 25435
```

The `head` function lets us look at the first 10 of so lines of the table, where we can see the genes are in the rows, with relevant statistics. The `nrow` function then tells us how many genes have been compared that matched the `logfc.threshold` and `min.pct` criteria. It should be noted that these criteria do not necessarily mean that only the significant genes have been returned; the table may still contain genes that are not significantly dysregulated.

The differential expression table from `FindAllMarkers` contains the following statistics:

- `p_val`: The raw p-value as calculated by the chosen algorithm.

- `avg_log2FC`: The average log2 fold change between the first group and the second group. In this case, it is the cluster in question and the remainder of the cells, respectively.
- `pct.1`: The fraction of cells in the first group (cluster in question) that express the gene.
- `pct.2`: The fraction of cells in the second group (all other cells) that express the gene.
- `p_val_adj`: The adjusted p-value, as determined using False Discovery Rate (also known as Benjamini-Hochberg) algorithm.
- `cluster`: The cluster that was being evaluated as "Population 1". - `gene`: Reiterates the gene that is being evaluated. This is necessary because R requires unique row names.

Here we will take a look at the top 5 positively differentially expressed genes of each cluster:

```
top5PerCluster = matrix(ncol=7)
colnames(top5PerCluster) = colnames(de_allClusters)
for (i in 0:7){
  top5PerCluster = rbind(top5PerCluster, head(de_allClusters[which(de
}
top5PerCluster=top5PerCluster[-1,]
top5PerCluster
```

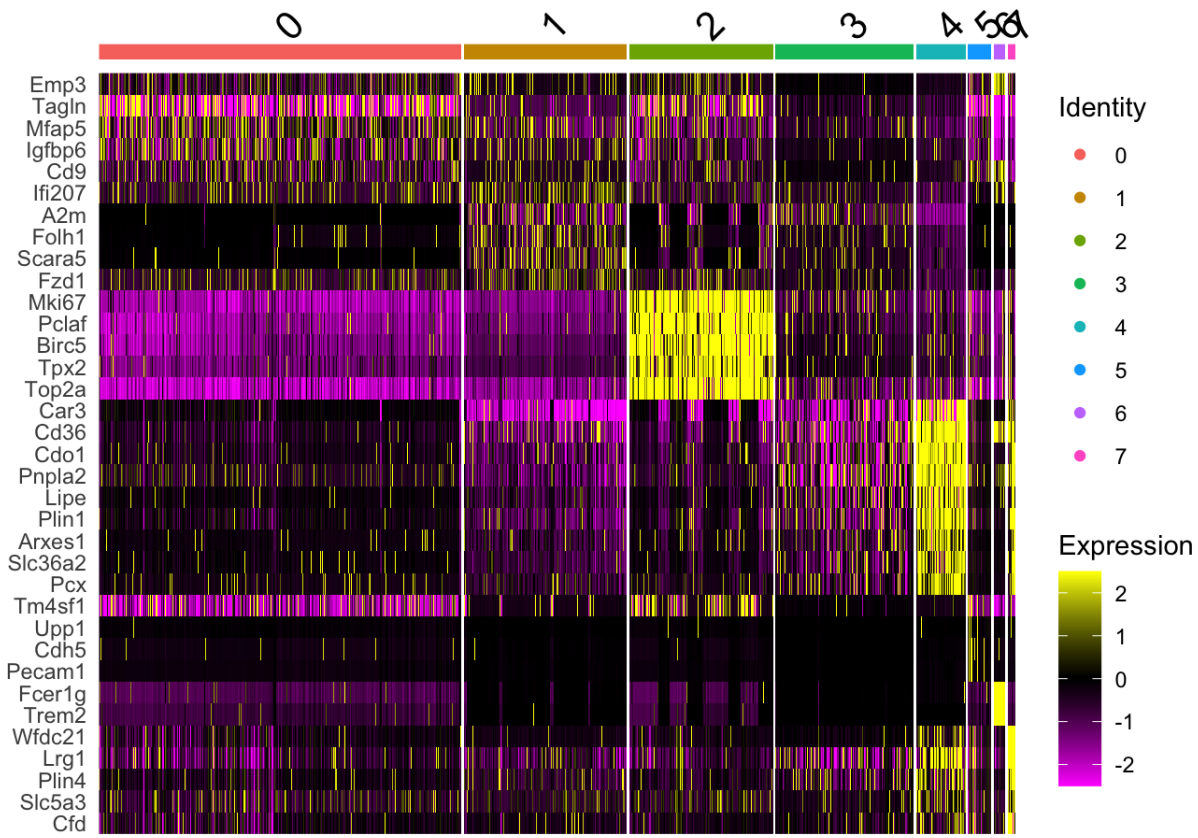
```
         p_val avg_log2FC pct.1 pct.2 p_val_adj cluster    gene
Emp3         0  1.4375351 0.866 0.290         0       0    Emp3
Tagln        0  1.6472332 0.852 0.299         0       0   Tagln
Mfap5        0  1.8777432 0.958 0.418         0       0   Mfap5
Igfbp6       0  2.0280281 0.806 0.281         0       0  Igfbp6
Cd9          0  1.6786106 0.760 0.261         0       0     Cd9
Ifi207       0  1.8002046 0.751 0.370         0       1  Ifi207
A2m          0  1.8799582 0.613 0.238         0       1     A2m
Folh1        0  2.5556965 0.458 0.085         0       1   Folh1
Scara5       0  3.0511756 0.427 0.056         0       1  Scara5
Fzd1         0  1.5380712 0.597 0.247         0       1    Fzd1
Mki67        0  4.8278546 0.930 0.071         0       2   Mki67
Pclaf        0  4.7049029 0.935 0.078         0       2   Pclaf
Birc5        0  4.6066362 0.929 0.077         0       2   Birc5
Tpx2         0  4.6715031 0.899 0.079         0       2    Tpx2
Top2a        0  4.5592408 0.927 0.127         0       2   Top2a
Car3         0  0.4065029 0.917 0.185         0       3    Car3
A2m1         0  1.7465643 0.906 0.197         0       3     A2m
Cd361        0  1.1648713 0.957 0.252         0       3    Cd36
Cdo1         0  1.2141175 0.913 0.238         0       3    Cdo1
Pnpla2       0  1.4005127 0.934 0.290         0       3  Pnpla2
Lipe1        0  4.2099355 0.947 0.156         0       4    Lipe
```

```
Plin11       0  4.0682072 0.945 0.159          0       4   Plin1
Arxes11      0  3.9698014 0.899 0.126          0       4  Arxes1
Slc36a21     0  4.4037697 0.833 0.069          0       4 Slc36a2
Pcx1         0  3.7673292 0.890 0.148          0       4     Pcx
Tm4sf12      0  1.9998484 0.836 0.280          0       5  Tm4sf1
Pdgfb        0  6.0929653 0.180 0.005          0       5   Pdgfb
Upp1         0  6.8186440 0.161 0.003          0       5    Upp1
Cdh5         0  7.7923303 0.157 0.007          0       5    Cdh5
Pecam1       0  8.0957976 0.150 0.003          0       5  Pecam1
Tyrobp       0 11.2901172 0.996 0.006          0       6  Tyrobp
C1qc         0 11.4203660 0.996 0.007          0       6    C1qc
Fcer1g       0 11.0144789 1.000 0.012          0       6  Fcer1g
Trem2        0 10.4536558 0.993 0.006          0       6   Trem2
Ctss         0 10.9417908 0.991 0.005          0       6    Ctss
Wfdc211      0  7.6043603 0.850 0.035          0       7  Wfdc21
Lrg11        0  5.2555628 0.974 0.169          0       7    Lrg1
Plin41       0  3.7513090 0.893 0.115          0       7   Plin4
Slc5a31      0  4.3232241 0.893 0.160          0       7  Slc5a3
Cfd1         0  9.0781154 0.749 0.043          0       7     Cfd
```

This can be visualized with a heatmap:

```
DoHeatmap(adp,features = top5PerCluster$gene,slot="scale.data")
```

```
Warning in DoHeatmap(adp, features = top5PerCluster$gene, slot = "sca
The following features were omitted as they were not found in the sca
slot for the SCT assay: Ctss, C1qc, Tyrobp, Pdgfb
```



Tip

Here we are specifying that we are using the `scale.data` slot in the `SCT` assay. During the SCTransform process, the `SCT` assay has the slot for `scale.data` filled; however, this may not contain all genes as it contains only the most highly variable genes. In the example above, Ctss, C1qc, Tyrobp, and Pdgfb have been excluded for this reason. In order to ensure that all genes are represented, the `data` slot can be directly called as the data reference, but it will not be z-scaled.

2b. FindMarkers

Many experiments look to compare to distinct populations, and scRNASeq is no exception. The two populations being compared can vary wildly from experiment to experiment; some look to draw comparisons based on the experimental condition, while others may look to compare different clusters within the experiment itself.

Here, we will be drawing a comparison between the two timepoints. First, we need to set the identities of the object:

```
Idents(adp) = "time_point"
table(Idents(adp))
```

```
Day 0 Day 6
23462 18652
```

And then we set up `FindMarkers` to compare our two conditions:

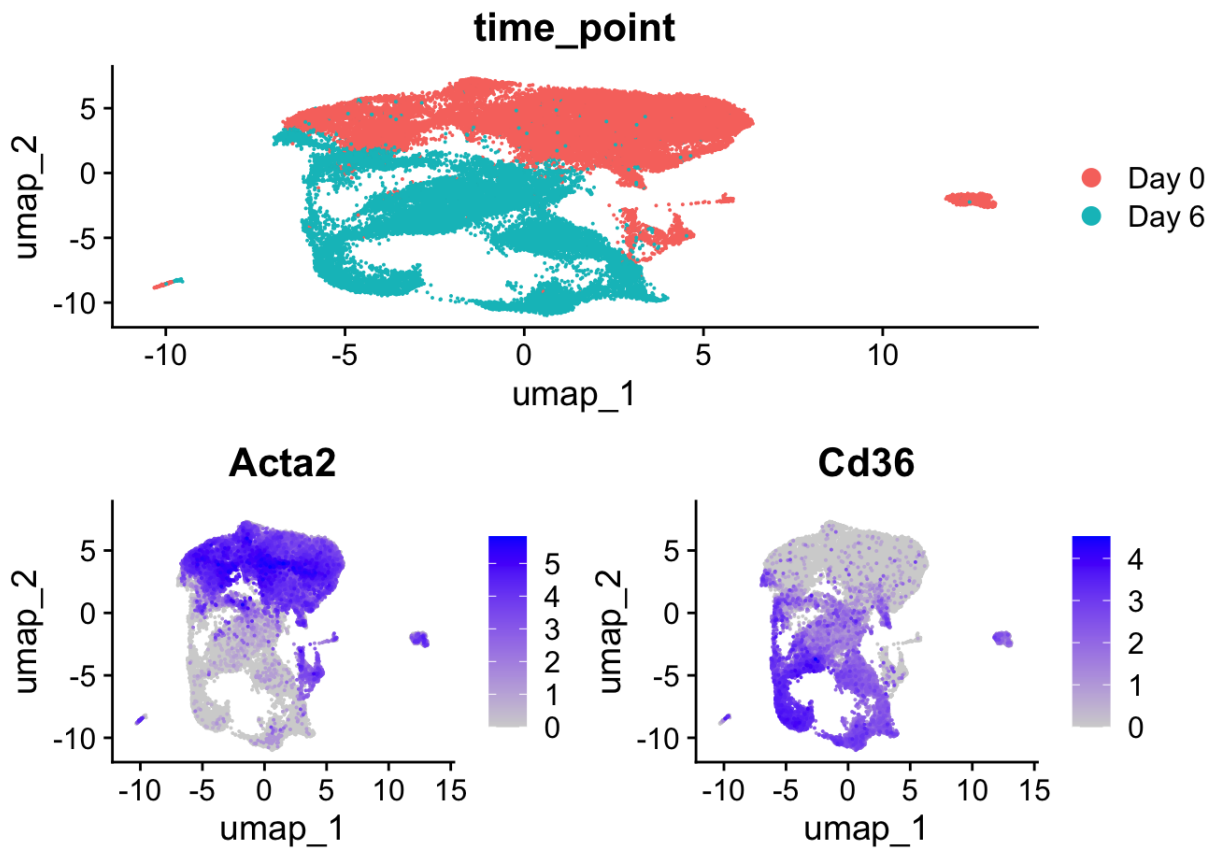
```
day6_day0_de=FindMarkers(adp,ident.1="Day 6",ident.2="Day 0",test.use
head(day6_day0_de, 10)
```

```
        p_val avg_log2FC pct.1 pct.2 p_val_adj
Emp3        0  -5.413609 0.060 0.889         0
Tagln       0  -5.992922 0.080 0.873         0
Acta2       0  -6.098134 0.079 0.825         0
Tpm2        0  -5.040596 0.084 0.805         0
Cd36        0   5.349745 0.762 0.042         0
A2m         0   9.465772 0.687 0.004         0
Cd9         0  -4.134289 0.082 0.764         0
Rbp4        0   3.769205 0.841 0.160         0
Mfap5       0  -3.605865 0.257 0.937         0
Igfbp6      0  -3.786182 0.115 0.792         0
```

As you can see, most of the column headers are the same as the results from FindAllMarkers; in fact, there are fewer because the system does not need to account for which cluster is being compared against the rest and all the gene IDs should be unique. As mentioned previously, `ident.1` will have the placeholder of the positive fold change. Another visualization of the differential expression can be performed using the `FeaturePlot` function:

```
fig1 = DimPlot(adp,group.by="time_point")
fig2 = FeaturePlot(adp,features="Acta2",order=T)
fig3 = FeaturePlot(adp,features="Cd36",order=T)

fig1/(fig2|fig3)
```



2c. Pseudobulk Differential Expression

One particular critique of differential expression in single cell RNASeq analysis is p-value "inflation," where the p-values get so small that there are far too many genes exist with p-values below 0.05, even after adjustment. This is caused by the nature of scRNASeq, where each individual cell essentially consists of a single replicate; as the replicate count increases, the p-values tend to shrink. Generally, we recommend using the DE values as a guide for significant differential gene identification, as opposed to using the p-value for a hard cutoff. The p-value, which is an expression of a probability of a real result, should be examined in context with the remaining statistics, such as fraction expression (`pct.1` and `pct.2`) and fold change. The entire picture of the statistical comparison will aid in determining which events are likely to be real and therefore actionable pursuits.

One method to counter the p-value inflation is to run a pseudobulk analysis. In this process, the total gene counts for each population of interest is aggregated across cells, and then these total counts can be treated in the same manner as a bulk RNASeq experiment. Through this process, the replicate inflation is overcome, as well as any cell-to-cell variation that may be excessively skewing results.

Here, the pseudobulk DE is run between timepoints, as before. It starts with the `AggregateExpression` function in Seurat, and collapsing the counts by replicate. Be sure to include all potential metadata that will be of interest.

```
pseudo_adp=AggregateExpression(adp,assays="RNA", return.seurat=T, gr

head(pseudo_adp@assays$RNA$counts)
```

```
6 x 8 sparse Matrix of class "dgCMatrix"
        D10_Day 0_DKO_DKO Day 0 D16_Day 6_DKO_DKO Day 6 D20_Day 0_DK
Xkr4                       340                      27
Gm19938                    338                      60
Sox17                      128                       9
Mrpl15                   21040                    5015
Lypla1                    4339                    1779
Tcea1                    31828                    4905
        D26_Day 6_DKO_DKO Day 6 W10_Day 0_WT_WT Day 0 W16_Day 6_WT_W
Xkr4                        31                   411
Gm19938                     71                   302
Sox17                        .                   128
Mrpl15                    7359                 16171
Lypla1                    2608                  4610
Tcea1                     7095                 26408
        W20_Day 0_WT_WT Day 0 W26_Day 6_WT_WT Day 6
Xkr4                      278                    37
Gm19938                   196                    82
Sox17                      31                     5
Mrpl15                  12725                  5861
Lypla1                   3020                  1995
Tcea1                   17656                  6041
```

```
pseudo_adp@meta.data
```

```
                                    orig.ident time_point condition
D10_Day 0_DKO_DKO Day 0 D10_Day 0_DKO_DKO Day 0      Day 0       DKO
D16_Day 6_DKO_DKO Day 6 D16_Day 6_DKO_DKO Day 6      Day 6       DKO
D20_Day 0_DKO_DKO Day 0 D20_Day 0_DKO_DKO Day 0      Day 0       DKO
D26_Day 6_DKO_DKO Day 6 D26_Day 6_DKO_DKO Day 6      Day 6       DKO
W10_Day 0_WT_WT Day 0     W10_Day 0_WT_WT Day 0      Day 0        WT
W16_Day 6_WT_WT Day 6     W16_Day 6_WT_WT Day 6      Day 6        WT
W20_Day 0_WT_WT Day 0     W20_Day 0_WT_WT Day 0      Day 0        WT
W26_Day 6_WT_WT Day 6     W26_Day 6_WT_WT Day 6      Day 6        WT
```

```
#just to clean up the look a little bit
pseudo_adp = RenameCells(pseudo_adp,new.names=gsub("_.*","",pseudo_ad
```

```
pseudo_adp$orig.ident=gsub("_.*","",pseudo_adp$orig.ident)
head(pseudo_adp@assays$RNA$counts)
```

```
6 x 8 sparse Matrix of class "dgCMatrix"
          D10  D16   D20  D26   W10  W16   W20  W26
Xkr4      340   27   239   31   411   33   278   37
Gm19938   338   60   188   71   302   55   196   82
Sox17     128    9    61    .   128    .    31    5
Mrpl15  21040 5015 10944 7359 16171 4475 12725 5861
Lypla1   4339 1779  3024 2608  4610 1709  3020 1995
Tcea1   31828 4905 15522 7095 26408 4715 17656 6041
```

```
pseudo_adp@meta.data
```

```
    orig.ident time_point condition   cond_tp
D10        D10      Day 0       DKO DKO Day 0
D16        D16      Day 6       DKO DKO Day 6
D20        D20      Day 0       DKO DKO Day 0
D26        D26      Day 6       DKO DKO Day 6
W10        W10      Day 0        WT  WT Day 0
W16        W16      Day 6        WT  WT Day 6
W20        W20      Day 0        WT  WT Day 0
W26        W26      Day 6        WT  WT Day 6
```

The data now is a matrix of counts by sample, as opposed to counts by cell. DESeq2 may now be run on the pseudobulk data:

```
Idents(pseudo_adp)="time_point"
bulk_adp_de = FindMarkers(pseudo_adp, ident.1="Day 6", ident.2="Day 0
```

```
converting counts to integer mode
```

```
gene-wise dispersion estimates
```

```
mean-dispersion relationship
```

```
final dispersion estimates
```

```
head(bulk_adp_de)
```

```
       p_val avg_log2FC pct.1 pct.2 p_val_adj
Nabp1      0   2.249765     1     1         0
Prelp      0  -2.519092     1     1         0
Arl8a      0  -1.180647     1     1         0
Glul       0   2.291869     1     1         0
Fcer1g     0  -1.648088     1     1         0
Atp1a2     0   1.528571     1     1         0
```

Comparing the results between the single cell and pseudobulk analysis shows that there are more genes identified as significantly dysregulated via scRNASeq DE; the overall concordance sits at about 70% of the pseudobulk genes.

```
scDE.genes = rownames(day6_day0_de)[which(day6_day0_de$p_val_adj<0.05
bulkDE.genes = rownames(bulk_adp_de)[which(bulk_adp_de$p_val_adj<0.05
length(scDE.genes)
```

```
[1] 10337
```

```
length(bulkDE.genes)
```

```
[1] 9201
```

```
length(intersect(scDE.genes,bulkDE.genes))
```

```
[1] 6779
```

Checking our two targeted genes above shows that the pseudobulk analysis still identifies them as significantly differentially expressed:

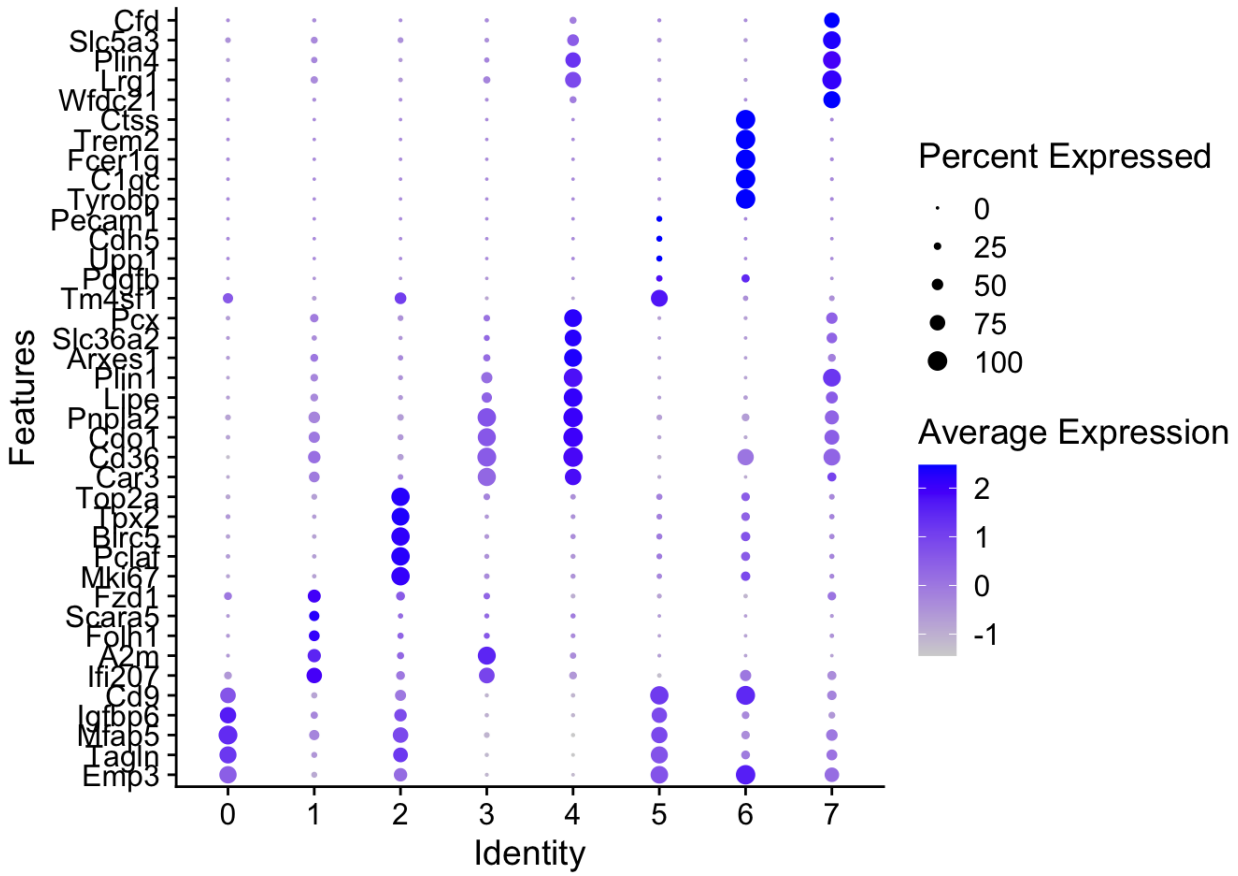
```
bulk_adp_de[c("Acta2","Cd36"),]
```

```
              p_val avg_log2FC pct.1 pct.2     p_val_adj
Acta2 1.769929e-268  -5.490767     1     1 3.655611e-264
Cd36  1.986865e-126   4.560422     1     1 4.103671e-122
```

3. Visualizing Differentially Expressed Genes

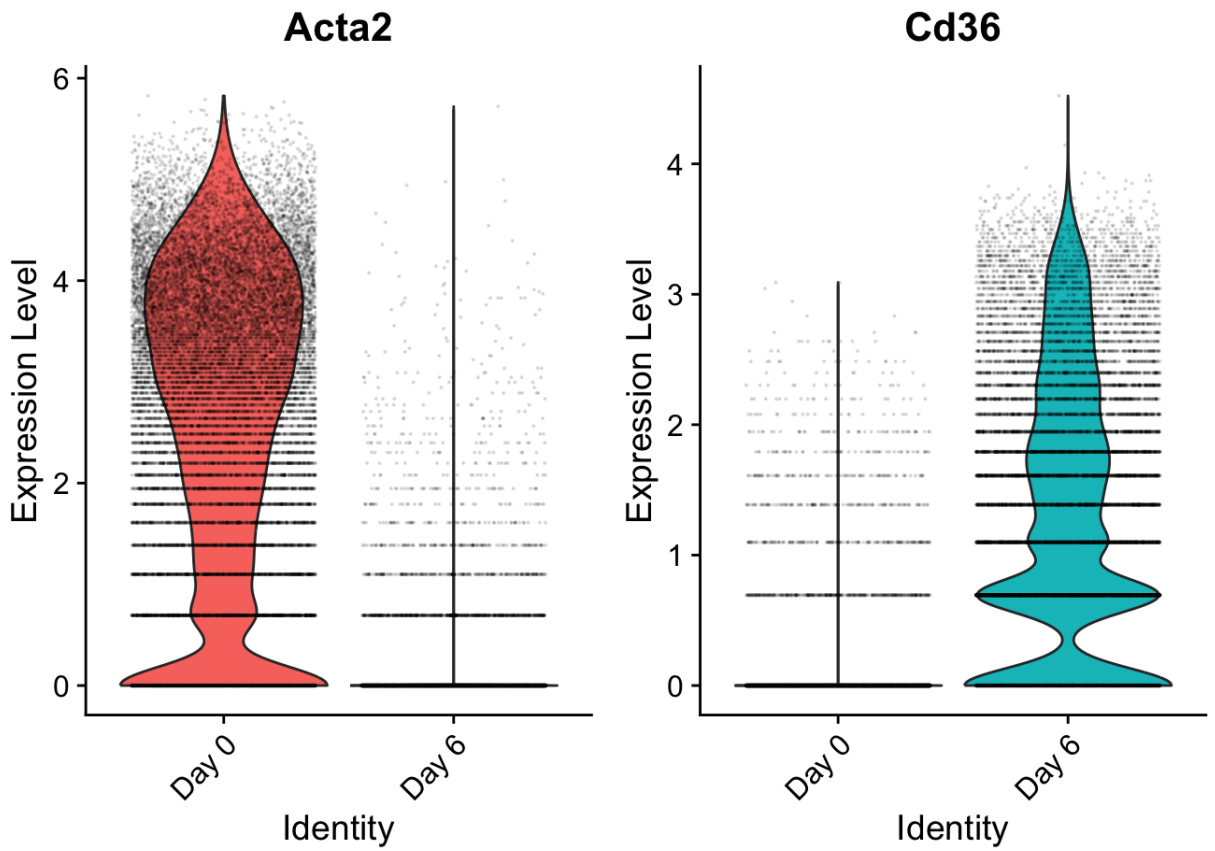
In addition to the `Heatmap` and the `FeaturePlot` shown previously, two other options easily accessible through Seurat are the Dot Plot and Violin Plot. The dot plot can visualize relative expression level and expression fraction. Using the same genes from the cluster-derived DE results:

```
Idents(adp)="SCT_snn_res.0.1"
DotPlot(adp,features=unique(top5PerCluster$gene),dot.scale = 3)+coord
```



The violin plot is another way to visualize top genes across a smaller subset of genes. Taking the two genes from the timepoint analysis:

```
Idents(adp) = "time_point"
VlnPlot(adp,features=c("Acta2","Cd36"),alpha = 0.1)
```



Note

Striation is normal. The SCTransform technique normalizes the counts themselves at the per-gene and per-sample levels, while the older log-normalization/scaling approach created a more continuous count distribution.

4. Cell Annotation

4a. Annotation by Differential Expressed Gene Markers

From the paper where this data was obtained, the following (incomplete) list of gene markers was obtained:

```
Mmp3: preadipocytes
Mki67: proliferating cells
Fabp4: differentiating beige adipocytes and differentiated beige adi
Scd1: differentiated beige adipocytes
Ucp1: differentiated beige adipocytes
Ppargc1a: differentiated beige adipocytes
Elovl3: differentiated beige adipocytes
Cidea: differentiated beige adipocytes
```

Looking at the average expression of these genes across the clusters can provide insight into annotating the cells

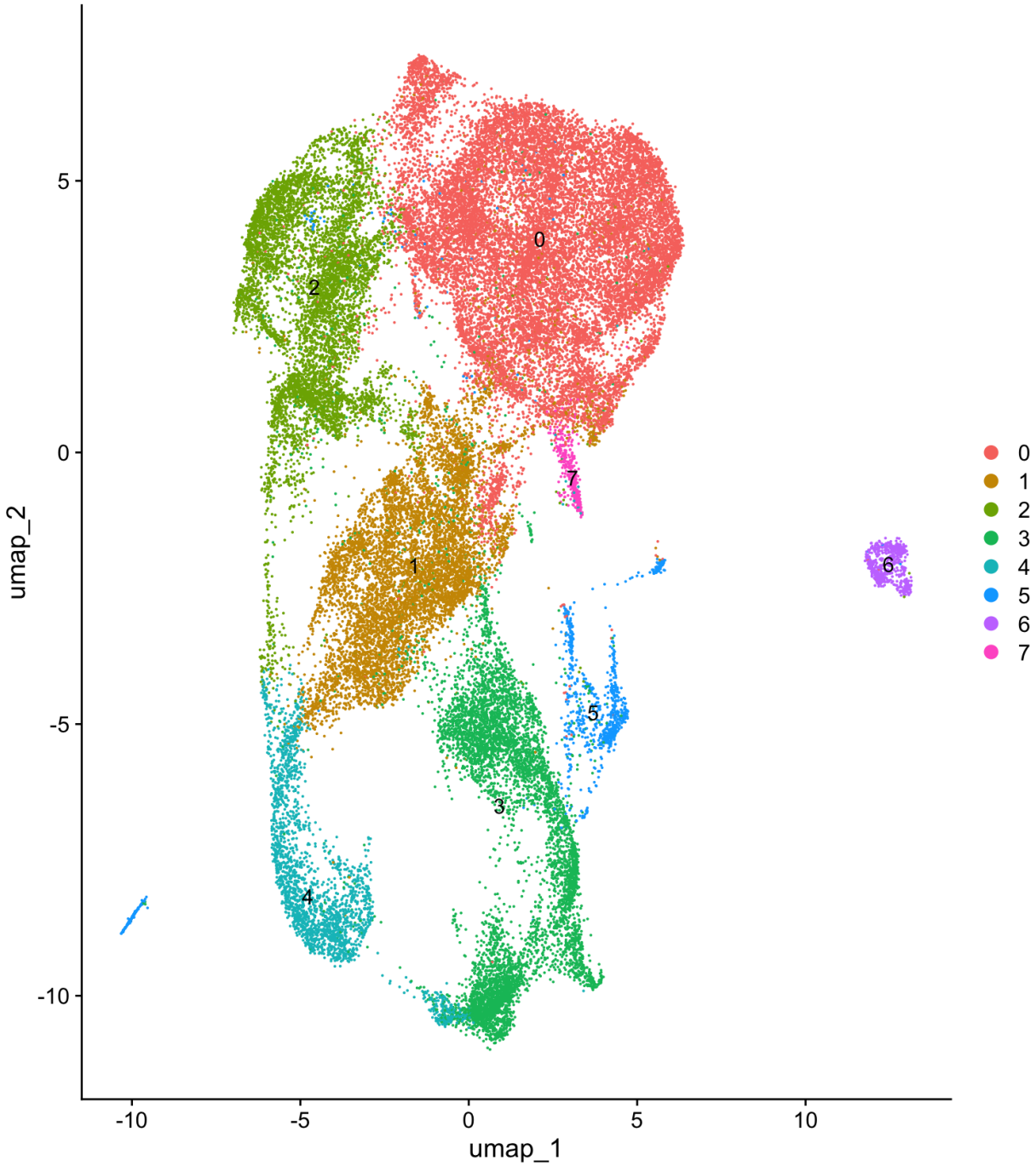
```
markers=c("Mmp3","Mki67","Fabp4","Scd1","Ucp1","Ppargc1a","Elovl3","C
Idents(adp)="SCT_snn_res.0.1"
avgExp = AverageExpression(adp,markers,assay="SCT")$SCT
```

```
As of Seurat v5, we recommend using AggregateExpression to perform ps
First group.by variable `ident` starts with a number, appending `g` t
This message is displayed once per session.
```

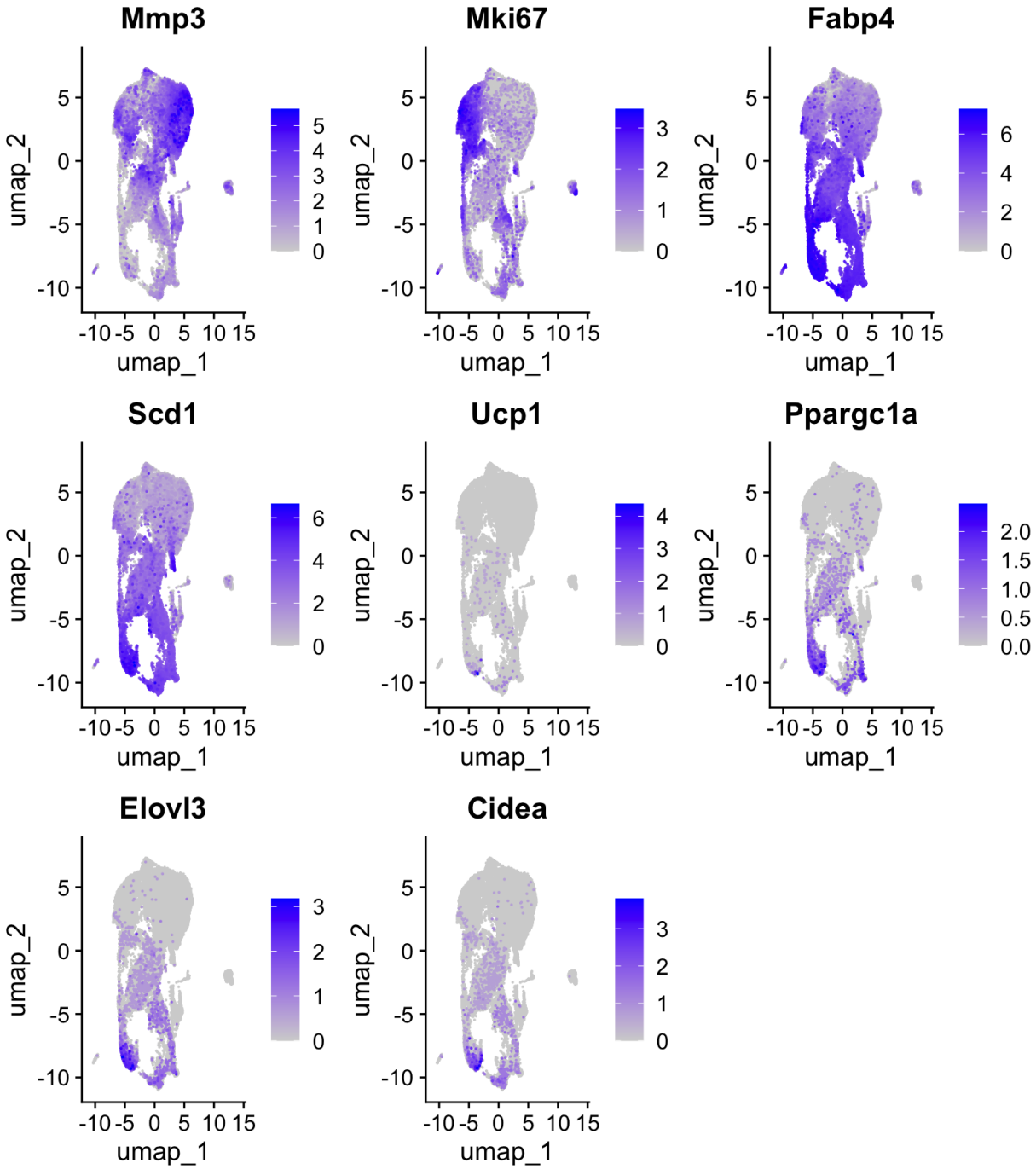
```
avgExp
```

```
8 x 8 sparse Matrix of class "dgCMatrix"
                  g0           g1           g2           g3
Mmp3     1.007001e+01  2.666666667  2.024810578   0.63200123   0.1474
Mki67    5.520821e-02  0.081859856  5.157777448   0.38975467   0.2622
Fabp4    3.586233e+00 50.806941716 20.006834051 158.68677673 450.8907
Scd1     7.791082e-01  7.064178127  2.212895558  26.46952631  76.7214
Ucp1     1.179663e-04  0.008382449  0.002971327   0.01481253   0.1294
Ppargc1a 4.895600e-03  0.039816634  0.011142475   0.10862521   0.7923
Elovl3   1.828477e-03  0.064047151  0.023919180   0.17636167   1.2682
Cidea    2.595258e-03  0.041519319  0.014410934   0.12976393   0.6977
                  g5          g6           g7
Mmp3      2.429493088 1.446043165   1.07803468
Mki67     0.459907834 1.937050360   0.43930636
Fabp4    11.935483871 2.183453237 137.46531792
Scd1      1.003686636 0.208633094  70.86416185
Ucp1      .           .             .
Ppargc1a  0.003686636 .             0.05202312
Elovl3    0.005529954 .             .
Cidea     0.007373272 0.001798561   .
```

```
DimPlot(adp,label=T)
```




```
FeaturePlot(adp,features=markers,ncol=3,order=T)
```



Comparing the genes to the clusters, the following correlations are apparent:

- Cluster 0: Mmp3
- Cluster 1: Moderate Fabp4, slight Scd1
- Cluster 2: Mki67, mild Fabp4
- Cluster 3: Moderate Fabp4, Moderate Scd1
- Cluster 4: Fabp4, Scd1, Ucp1, Ppargc1a, Cidea

- Cluster 5: Moderate Mmp3, slight Fabp4
- Cluster 6: Moderate Mki67
- Cluster 7: Fabp4, Scd1

Based on these gene markers, the following cell identities could be assigned:

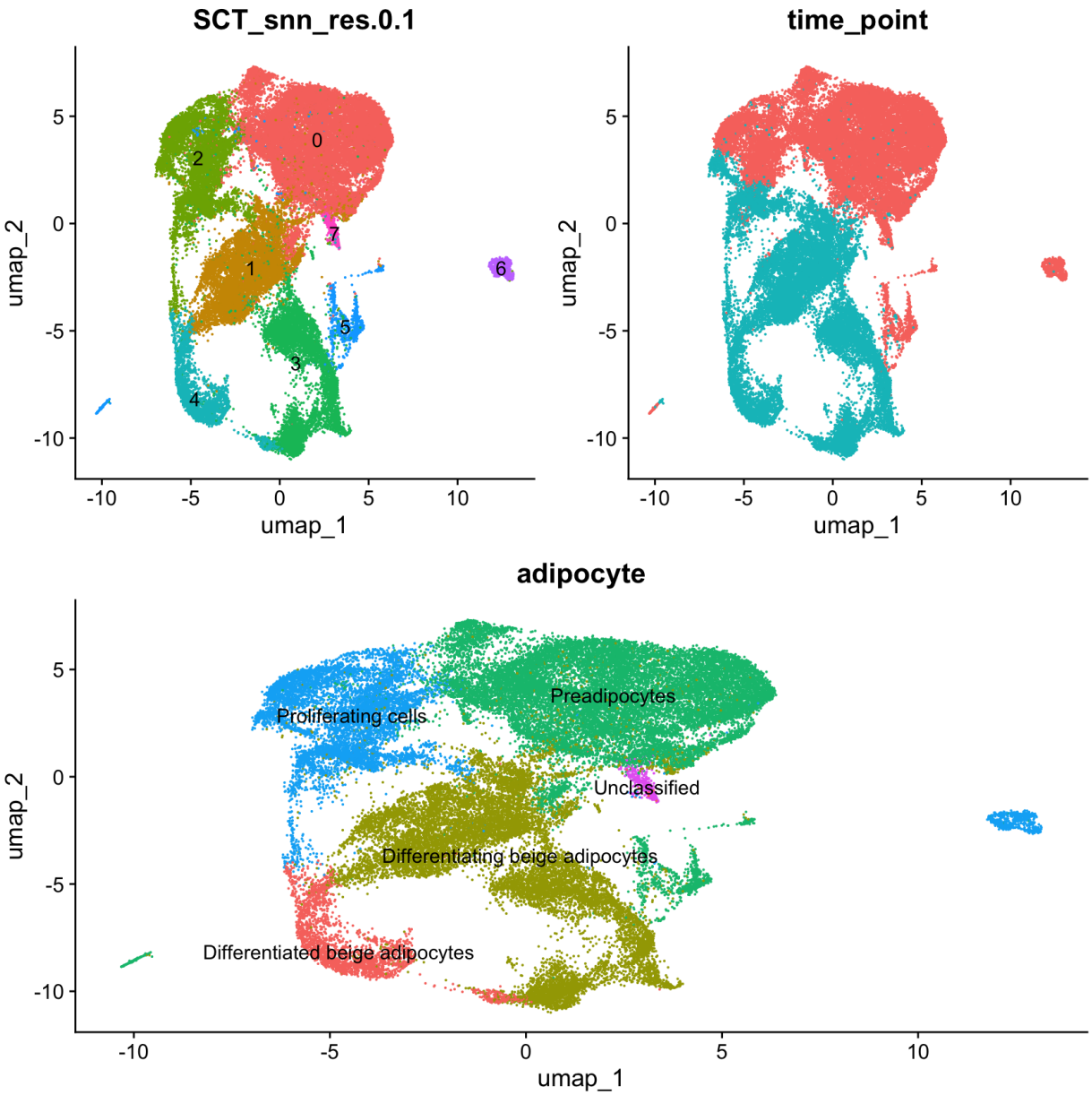
- Preadipocytes: Clusters 0 and 5
- Proliferating cells: Clusters 2 and 6
- Differentiating beige adipocytes: Cluster 1 and 3
- Differentiated beige adipocytes: Cluster 4
- Unclassified: Cluster 7

These can be relabeled via vector assignment:

```
adipocyte=vector(length=ncol(adp))
adipocyte[which(adp$SCT_snn_res.0.1 %in% c(0,5))]="Preadipocytes"
adipocyte[which(adp$SCT_snn_res.0.1 %in% c(2,6))]="Proliferating cel
adipocyte[which(adp$SCT_snn_res.0.1 %in% c(1,3))]="Differentiating be
adipocyte[which(adp$SCT_snn_res.0.1 %in% c(4))]="Differentiated beige
adipocyte[which(adp$SCT_snn_res.0.1 %in% c(7))]="Unclassified"
adp$adipocyte = adipocyte

f1 = DimPlot(adp,group.by="SCT_snn_res.0.1",label=T) + NoLegend()
f2 = DimPlot(adp,group.by="time_point") + NoLegend()
f3 = DimPlot(adp,group.by="adipocyte",label=T)+NoLegend()

(f1|f2)/f3
```



4b. Annotation using SingleR

Some tools have been described in the previous session (see here *(https://bioinformatics.ccr.cancer.gov/docs/getting-started-with-scrna-seq/Seurat_QC_to_Clustering/#cell-annotation)*). Today, we will be focusing on the SingleR tool, which also requires the `celldex` package *(https://bioconductor.org/packages/release/data/experiment/html/celldex.html)*. In short, SingleR operates by comparing your current dataset against a reference dataset, mostly through correlative gene expression. It then assigns a score for the expression set (i.e. each cell or cluster being compared) for all possible reference labels. This walkthrough will explore using one of the default references from `celldex` to try to assign annotations to our current dataset.

SingleR requires that the data be mildly transformed into a different data structure. It is also slightly more efficient to retrieve and store the reference dataset from `celldex` preemptively. This also gives the opportunity to peek into the reference data itself.

```
adp.sce = as.SingleCellExperiment(adp,assay="SCT") #This selects *on
mouseRNASeq = celldex::MouseRNAseqData()
head(mouseRNASeq)
```

```
class: SummarizedExperiment
dim: 6 358
metadata(0):
assays(1): logcounts
rownames(6): Xkr4 Rp1 ... Lypla1 Tcea1
rowData names(0):
colnames(358): ERR525589Aligned ERR525592Aligned ... SRR1044043Aligne
  SRR1044044Aligned
colData names(3): label.main label.fine label.ont
```

```
table(mouseRNASeq$label.main)
```

```
      Adipocytes        Astrocytes          B cells    Cardiomyocyt
              13                27                5
 Dendritic cells Endothelial cells  Epithelial cells      Erythrocyt
               2                12                2
     Fibroblasts      Granulocytes       Hepatocytes       Macropha
              45                15                4
       Microglia         Monocytes          Neurons          NK ce
              72                 6               64
Oligodendrocytes           T cells
              12                18
```

```
table(mouseRNASeq$label.fine)
```

```
          Adipocytes                aNSCs           Astrocytes
                  13                    8                   24
Astrocytes activated              B cells        Cardiomyocytes
                   3                    5                    8
     Dendritic cells     Endothelial cells            Ependymal
                   2                   12                    2
        Erythrocytes          Fibroblasts Fibroblasts activated
```

```
                     3                     27                      9
Fibroblasts senescent          Granulocytes           Hepatocytes
                     9                     15                      4
          Macrophages Macrophages activated             Microglia
                    26                      6                     56
  Microglia activated             Monocytes               Neurons
                    16                      6                     39
    Neurons activated              NK cells                  NPCs
                     4                     18                     11
     Oligodendrocytes                   OPCs                  qNSCs
                    10                      2                      2
              T cells
                    18
```

For many of the `celldex` datasets, there are two separate categories of labels. The `main` labels capture the family of cell types available (e.g. T cells), while the `fine` category captures the specific subtypes (e.g. Cd4 effector T cells). The references in `celldex` are species-specific, which has to be accounted for in any reference-based cell type annotation.

Your annotation is only as good as your reference!

In many cases, one of the default reference datasets from celldex will be a sufficient starting point. However, some studies will require more specific cell types than what can be retrieved from `celldex`. In these cases, one may consider retrieving a dataset from the Bioconductor package `scRNAseq` *(https://bioconductor.org/packages/release/data/experiment/vignettes/scRNAseq/inst/doc/scRNAseq.html)* or from the Chan-Zuckerberg Biohub, which has collected and annotated cell types from multiple organs from both mouse (Tabula muris *(https://www.czbiohub.org/sf/tabula-muris/)*) and human (Tabula sapiens *(https://tabula-sapiens-portal.ds.czbiohub.org/)*) samples.

Version 1: Cell-level cell type annotation

When using SingleR, the 3 primary parameters are the experimental dataset, the reference dataset, and the labels being used. Continuing with the `main` labels of the MouseRNASeq dataset on the full dataset looks like this:

```
annot = SingleR(test=adp.sce, ref=mouseRNASeq, labels=mouseRNASeq$lab
head(annot)
```

```
DataFrame with 6 rows and 4 columns
                                                  scores      labels de
                                                <matrix> <character>  <r
D10_AAACCCAAGATGCTTC-1 0.523325:0.275539:0.210645:...  Adipocytes  0
D10_AAACCCAAGCTCTTCC-1 0.499954:0.250459:0.226184:... Fibroblasts  0
D10_AAACCCAAGTCATCGT-1 0.474176:0.213157:0.189314:... Fibroblasts  0
```

```
D10_AAACCCAGTAGCGTCC-1 0.449555:0.245758:0.228256:... Fibroblasts  0
D10_AAACCCAGTGCACGCT-1 0.398923:0.179533:0.207357:... Fibroblasts  0
D10_AAACCCATCCACTGAA-1 0.500808:0.250242:0.187568:... Fibroblasts  0
                       pruned.labels
                         <character>
D10_AAACCCAAGATGCTTC-1    Adipocytes
D10_AAACCCAAGCTCTTCC-1   Fibroblasts
D10_AAACCCAAGTCATCGT-1   Fibroblasts
D10_AAACCCAGTAGCGTCC-1   Fibroblasts
D10_AAACCCAGTGCACGCT-1   Fibroblasts
D10_AAACCCATCCACTGAA-1   Fibroblasts
```

Tip

This particular step can take a while, since it compares every cell in the dataset against every cell type transcriptome in the reference. One suggestion from the developers is to divide the dataset and run the annotation on each subset independently (in a process called "scatter and gather"). A version of this that works particularly well is to run the cell type annotation on each sample prior to combining the dataset. This isn't as much a problem with this toy dataset, but will become rather pronounced as data sets increase beyond \~75k cells or 10 samples.

The results from SingleR are returned as a dataframe:

- `scores`: Correlation scores for each cell against each of the reference columns. Accessed as `annot$scores` in this case
- `labels`: Initial labels based on highest scoring cell type annotation
- `delta.next`: The difference in scores between the highest and second-highest scoring annotations
- `pruned.labels`: SingleR determines if a label should be discarded because the `delta.next` value is too small, i.e. the label can be ambiguous. Any discarded labels will be replaced with `NA`.

Here, the `pruned.labels` are added to the meta.data

```
table(annot$pruned.labels, useNA="ifany") #useNA can be used turned
```

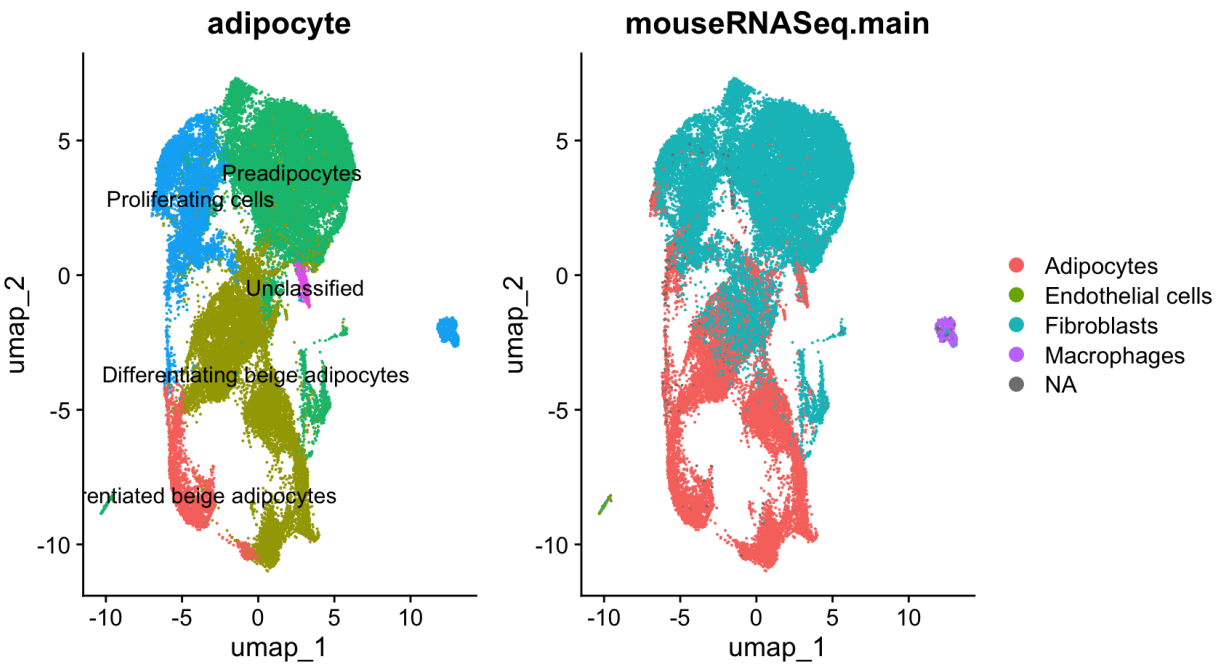
```
       Adipocytes Endothelial cells       Fibroblasts       Macropha
            13408               103             27692              
             <NA>                                                    
              431                                                    
```

```
adp$mouseRNASeq.main=annot$pruned.labels
```

This is actually a great example of the warning above. The dataset is *specifically* adipocytes, and should only have a smattering of alternative cell types. However, a number of the cells are being returned as a fibroblasts. Visualizing the two sets of annotations reveals that many of the cells identified as proliferating adipocytes or preadipocytes have been labeled as fibroblasts. Interestingly, the two clusters that are more separate from the rest of the main body received alternative annotations as endothelial cells and macrophages, which might be more accurate than the labels that were "forced" into the adipocyte pigeonholes.

```
annotFig1 = DimPlot(adp,group.by="adipocyte",label=T)+NoLegend()
annotFig2 = DimPlot(adp,group.by="mouseRNASeq.main")

annotFig1|annotFig2
```



Version 2: Cluster-level cell type annotation

Much like pseudobulk differential expression, the RNA expression can be collapsed into pre-defined components, such as the clusters, if it is believed that cell-to-cell variation is inducing too much confusion in the labeling. This collapsing can be performed using either the `AggregateExpression` or `AverageExpression` functions, as seen previously. The resulting matrix can then be fed into SingleR to produce the cluster-based annotations.

```
Idents(adp)="SCT_snn_res.0.1" #Assign clusters as the identities
avgExp = AverageExpression(adp, assays="SCT")$SCT #Run AverageExpress
```

```
clustAnnot=SingleR(test=avgExp, ref=mouseRNASeq, labels=mouseRNASeq$
clustAnnot
```

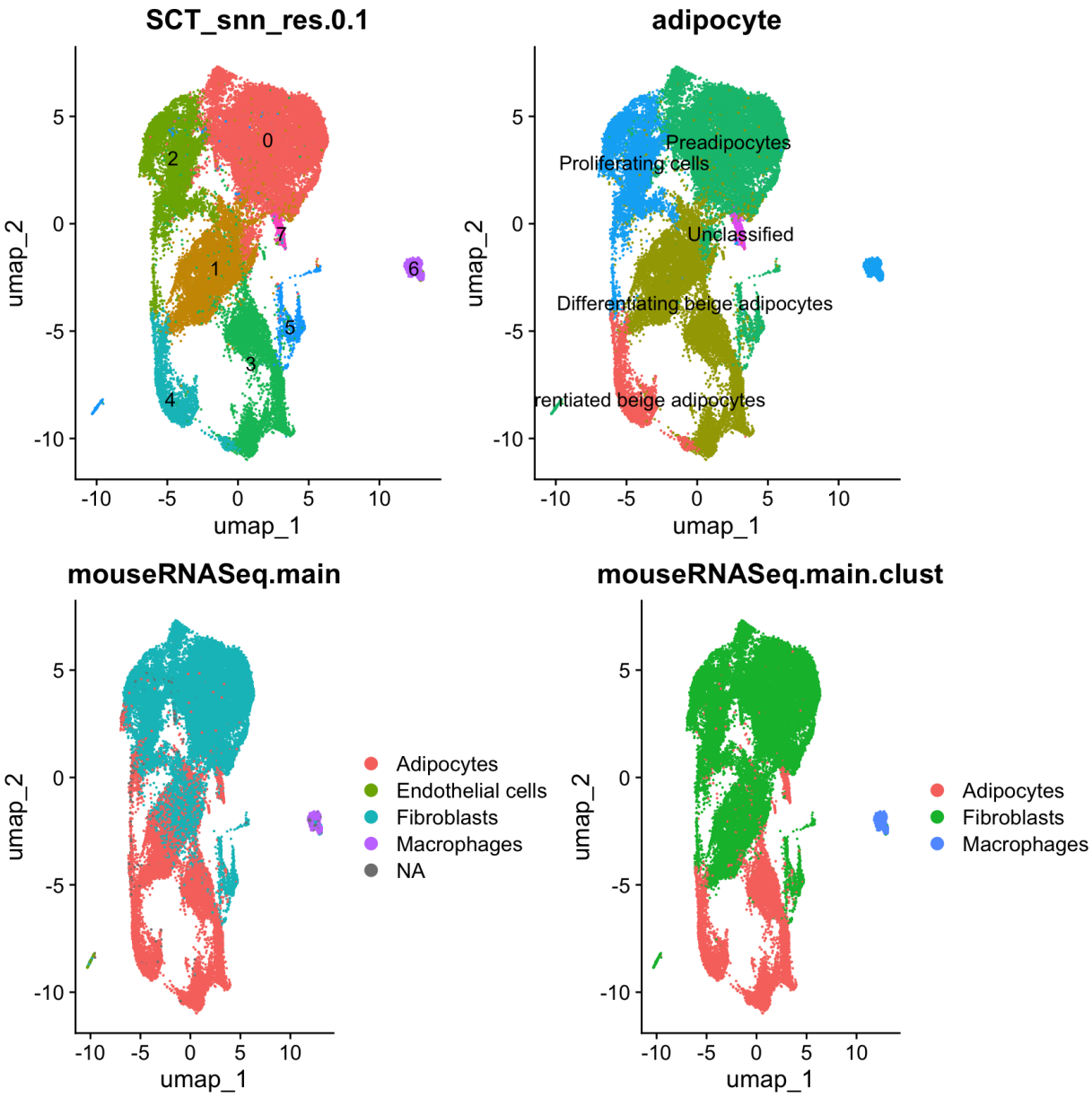
```
DataFrame with 8 rows and 4 columns
                          scores      labels delta.next pruned.labe
                        <matrix> <character>  <numeric>   <characte
g0 0.769608:0.406627:0.311650:... Fibroblasts  0.0845489   Fibroblas
g1 0.814336:0.425592:0.304490:... Fibroblasts  0.1485983   Fibroblas
g2 0.759363:0.389728:0.317296:... Fibroblasts  0.0755571   Fibroblas
g3 0.821708:0.418202:0.304515:...  Adipocytes  0.0533554    Adipocyte
g4 0.834166:0.416754:0.310953:...  Adipocytes  0.1534154    Adipocyte
g5 0.728947:0.368267:0.359848:... Fibroblasts  0.0653320   Fibroblas
g6 0.570283:0.210928:0.542144:... Macrophages  0.1005743   Macrophage
g7 0.828192:0.385496:0.314414:...  Adipocytes  0.0671988    Adipocyte
```

As before, most of the labels are assigned to either fibroblasts and adipocytes. The data takes a little massaging to assign to the scRNASeq metadata, but it is doable.

```
clustLabels = as.vector(clustAnnot$pruned.labels) #retrieve only the
names(clustLabels) = c(0:7) #assign the cluster numbers as the annota
clustLabels.vect = clustLabels[match(adp$SCT_snn_res.0.1, names(clust
names(clustLabels.vect) = colnames(adp) #ensure that the cluster iden
adp$mouseRNASeq.main.clust = clustLabels.vect #add the cluster annota

clustAnnotFig1 = DimPlot(adp,group.by="SCT_snn_res.0.1",label=T)+NoLe
clustAnnotFig2 = DimPlot(adp,group.by="adipocyte",label=T)+NoLegend()
clustAnnotFig3 = DimPlot(adp,group.by="mouseRNASeq.main")
clustAnnotFig4 = DimPlot(adp,group.by="mouseRNASeq.main.clust")

(clustAnnotFig1|clustAnnotFig2)/(clustAnnotFig3|clustAnnotFig4)
```



Generally, the annotation between the per-cell annotation coincides with the per-cluster annotation. This image is also somewhat cleaner, as the cell-to-cell variation is superseded into the clusters. However, this also provides an example where this "collapsing" process can lead to smaller populations being overlooked: The cluster that was labeled as endothelial cells (at \~ [-10,-8]) was relabeled as fibroblasts, due to its initial inclusion as part of cluster 5.

5. Conclusions

Single cell RNASeq is a remarkably powerful tool for analyzing populations of cells that can be recovered from various experiments. Clustering and cell type annotation can be used to distinguish different populations with a level of specificity not otherwise available in bulk sequencing methods. Differential expression can then be applied to identify critical genes that define and classify the cell types. Inversely, differential expression can then be used to identify

and classify the subpopulations, especially where cell type annotation falls short of the known biology. In either case, allow the biology to influence the bioinformatics, and use the bioinformatics to expand the biology.

Happy (gene) hunting!

Acknowledgements:

The following sources inspired this content:

- https://www.sc-best-practices.org *(https://www.sc-best-practices.org)*
- https://hbctraining.github.io/scRNA-seq_online/ *(https://hbctraining.github.io/scRNA-seq_online/)*
- https://bioconductor.org/books/3.15/OSCA.basic/ *(https://bioconductor.org/books/3.15/OSCA.basic/)*

This is only a small subset of tools available to single cell RNASeq. For further information and tools, please feel free to reach out to CCBR or BTEP.

The CCBR Single-cell RNA-seq Workflow on NIDAP

Josh Meyer, bioinformatics analyst (CCBR), will cover a scRNA-seq workflow available to NCI researchers on NIDAP *(https://bioinformatics.ccr.cancer.gov/ccbr/education-training/nidap-training/)*. NIDAP, the NIH Integrated Data Analysis Platform, is a cloud-based and collaborative data aggregation and analysis platform that hosts user-friendly bioinformatics workflows. This platform allows researchers to use many of the open-source tools discussed in this seminar series without necessitating coding experience. This seminar will focus on the capabilities of this workflow for QC, annotation, and visualization of single-cell RNA-seq data.

Additional Resources

1. BTEP scRNA-Seq FAQs *(https://bioinformatics.ccr.cancer.gov/btep/questions/?category=Single-Cell+RNA-Seq)*
2. Training modules available on Github *(https://glittr.org/?per_page=25&sort_by=stargazers&sort_direction=desc&search=single+cell+RNA-Seq)*
3. *Orchestrating Single Cell Analysis with Bioconductor (https://bioconductor.org/books/release/OSCA/)*
4. *Single Cell Best Practices (https://www.sc-best-practices.org/preamble.html)*
5. 2023 BTEP Single Cell Annotation Seminar Series *(https://bioinformatics.ccr.cancer.gov/btep/seminar/Single+Cell)*
    - Event recordings are located in the BTEP Video Archive *(https://bioinformatics.ccr.cancer.gov/btep/btep-video-archive-of-past-classes/)*.
6. Analysis Guides from 10X Genomics *(https://www.10xgenomics.com/analysis-guides?query=&page=1&refinementList%5Bproducts.name%5D%5B0%5D=Single%20Cell%20Gene%20Exp*
7. 10X Genomics FAQs *(https://kb.10xgenomics.com/hc/en-us)*

Contacts

Who can you contact with questions?

1. NCI CCR Single Cell Analysis Facility (SCAF) *(https://crtp.ccr.cancer.gov/labs/scaf/)*
    - provides single cell support to NCI-CCR Researchers.
2. NCI CCR Bioinformatics Training and Education Program (BTEP) *(https://bioinformatics.ccr.cancer.gov/btep/)*
    - provides bioinformatics training support.
    - Contact BTEP via email at ncibtep@nih.gov *(mailto:ncibtep@nih.gov)*.
3. Advanced Biomedical Computational Science (ABCS) *(https://frederick.cancer.gov/research/science-areas/bioinformatics-and-computational-science/advanced-biomedical-computational-science)*
    - focuses on applications of bioinformatics, computational and data science, and artificial intelligence to support NCI researchers.
    - Submit a project request here *(https://abcs-amp.nih.gov/project/request/ABCS/)*.

4. CCR Collaborative Bioinformatics Resource (CCBR) *(https://bioinformatics.ccr.cancer.gov/ccbr/)*
    - provides a mechanism for CCR researchers to obtain many different types of bioinformatics assistance to further their research goals.
    - Contact CCBR *(mailto: NCICCBRNIDAP@mail.nih.gov)* for questions related to CCBR Workflows & Training on NIDAP.
    - If you are a CCR researcher and you would like to request additional project consultation and analysis from CCBR, use this form *(https://bioinformatics.ccr.cancer.gov/ccbr/ask-for-help/)*.