

Bioinformatics · Practical notes

Single-Cell Transcriptomics

From Droplets and Barcodes to Cell Types, States,
Differential Expression, and Figures

A practical guide to droplet-based single-cell RNA-seq, from library structure and count-matrix generation to quality control, normalisation, exploratory representations, reviewed cell-type annotation and sample-aware inference. The core example uses independent donors and preserves raw counts for pseudobulk differential expression. Cell Ranger, STARsolo and alevin-fry are alternative counting routes; log-normalisation, SCTransform and scran illustrate different normalisation approaches. Integration is an optional decision assessed against uncorrected structure and relevant biology. Additional sections cover relative composition, trajectory and RNA-velocity assumptions, figures, nuclei and larger data sets. Readable companion scripts provide an R route through inference and figures and a Python exploratory route through clusters and markers. Optional demonstrations and verification limits are distinguished from the core execution contract.

Code and environments:
github.com/tunabirgun/bioinformatics-notes

Tuna Birgun
github.com/tunabirgun

Original note · 15 January 2026

Contents

1 Overview & scope **3**

2 How to use this worksheet **3**

3 Pipeline at a glance **4**

4 Software & packages **5**

5 Environment setup **6**
- 5.1 Operating system and resources . . . 6
- 5.2 Project directories and sample metadata . . . 6
- 5.3 Counting and Python environments . . . 7
- 5.4 R and Bioconductor . . . 7

6 From reads to a count matrix **7**
- 6.1 Demultiplexing Optional . . . 7
- 6.2 Cell Ranger count . . . 8
- 6.3 Alternative: STARsolo . . . 8
- 6.4 Alternative: alevin-fry . . . 9
- 6.5 Read the standardised matrix . . . 9

7 Alternative entry points **10**
- 7.1 From a 10x count matrix . . . 10
- 7.2 From an existing analysis object . . . 11
- 7.3 Public data . . . 11

8 Quality control & cell calling **11**
- 8.1 Call cells from unfiltered droplets Optional . . . 11
- 8.2 Assess ambient RNA Optional . . . 12
- 8.3 Inspect per-cell metrics and choose filters . . . 13
- 8.4 Detect doublets . . . 13

9 Normalisation & feature selection **14**
- 9.1 Choose one normalisation route . . . 14
- 9.2 Select highly variable genes . . . 15
- 9.3 Scaling and unwanted variation . . . 16

10 Dimensionality reduction & integration **17**
- 10.1 PCA . . . 17
- 10.2 Decide whether to integrate . . . 17
- 10.3 Integration alternatives . . . 17
- 10.4 Build a graph and visualise it . . . 18

11 Clustering & annotation **19**
- 11.1 Graph-based clustering . . . 19
- 11.2 Descriptive marker genes . . . 20
- 11.3 Reference-based annotation . . . 20
- 11.4 Confirm and store reviewed labels . . . 21

12 Differential expression across conditions **21**
- 12.1 Pseudobulk: aggregate raw counts within samples . . . 22
- 12.2 Why pooled per-cell tests fail for condition inference . . . 24
- 12.3 Differential abundance: relative composition . . . 24

13 Trajectory & RNA velocity Optional **24**
- 13.1 Pseudotime and trajectory inference . . . 25
- 13.2 RNA velocity . . . 25

14 Figures for single-cell **26**
- 14.1 UMAP by cell type and condition . . . 27

14.2 Marker feature plots 27
14.3 Dot plots and violins of canonical markers 27
14.4 Cell-type proportions across samples 28
14.5 Pseudobulk DE: volcano and heatmap 28

15 Supplementary **29**
15.1 Multi-sample workflow 29
15.2 Single-nucleus RNA-seq 30
15.3 Scale to larger data sets 30
15.4 Reproducibility 31
15.5 Troubleshooting 31

References **32**

1 Overview & scope

Single-cell RNA sequencing (scRNA-seq) profiles individual captured cells, making cell-type composition and within-tissue variation accessible alongside average expression differences [1, 2]. The result is a noisy, incomplete measurement of RNA abundance; a barcode is not by itself proof of a viable single cell or a distinct biological identity.

This note focuses on droplet-based 10x Genomics Chromium 3′ gene-expression counting. Barcodes identify droplets and unique molecular identifiers (UMIs) help collapse amplified observations of the same captured molecule. UMI deduplication also depends on the assigned gene and the counting algorithm [4, 5, 6].

Questions addressed. The workflow moves from a count matrix to cell identities and condition comparisons. Cell calling, quality filtering and doublet detection establish the cells to analyse; an exploratory representation then helps identify candidate populations. After reviewing cell-type labels, compare expression or relative composition across conditions. Optional integration, trajectory and RNA-velocity demonstrations introduce additional assumptions.

Questions outside this scope. Standard 3′ UMI counting is not a full-length isoform analysis, a spatial assay or a direct measurement of absolute RNA or tissue cell numbers. The choice between bulk and single-cell RNA-seq depends on the question, independent replication, tissue sampling and budget. Single-cell data are particularly useful when cell-type composition or within-type heterogeneity matters; bulk remains useful for a whole-tissue expression contrast.

Practical note

Running design. The core example uses human 10x 3′ v3 data, two conditions and independent donors, with one capture and one library per donor. The sample sheet records `sample`, `condition`, `donor` and `capture`. Three example donors per condition illustrate the layout; the required replication should follow the scientific question and expected variability. Paired, repeated, multiplexed or technical-replicate designs need an explicit extension before inference.

Independent units. Cells from one donor are nested observations: they share a donor and do not provide additional independent donor replicates. Use sample-aware inference for condition effects and preserve the donor/library/capture relationships throughout. Doublet detection uses the droplet capture as its processing unit; this is the same as the sample only under the running design (Section 8) [37].

Toolkits. Seurat (R) [20] and Scanpy/AnnData (Python) [21, 22] provide the main examples. Bioconductor's SingleCellExperiment framework and OSCA workflow provide alternative implementations for several stages [3]. Examples explain comparable tasks, but the routes are not numerically identical or interchangeable without checking their data representations.

2 How to use this worksheet

Read the experimental design and input contract before running a numbered phase. Start from FASTQ only when counting is needed; a verified filtered matrix can enter at QC, and an existing object can enter later once its counts, processing and metadata are understood (Section 7). Choose one counting and one normalisation route. Integration is a separate decision, not a compulsory correction.

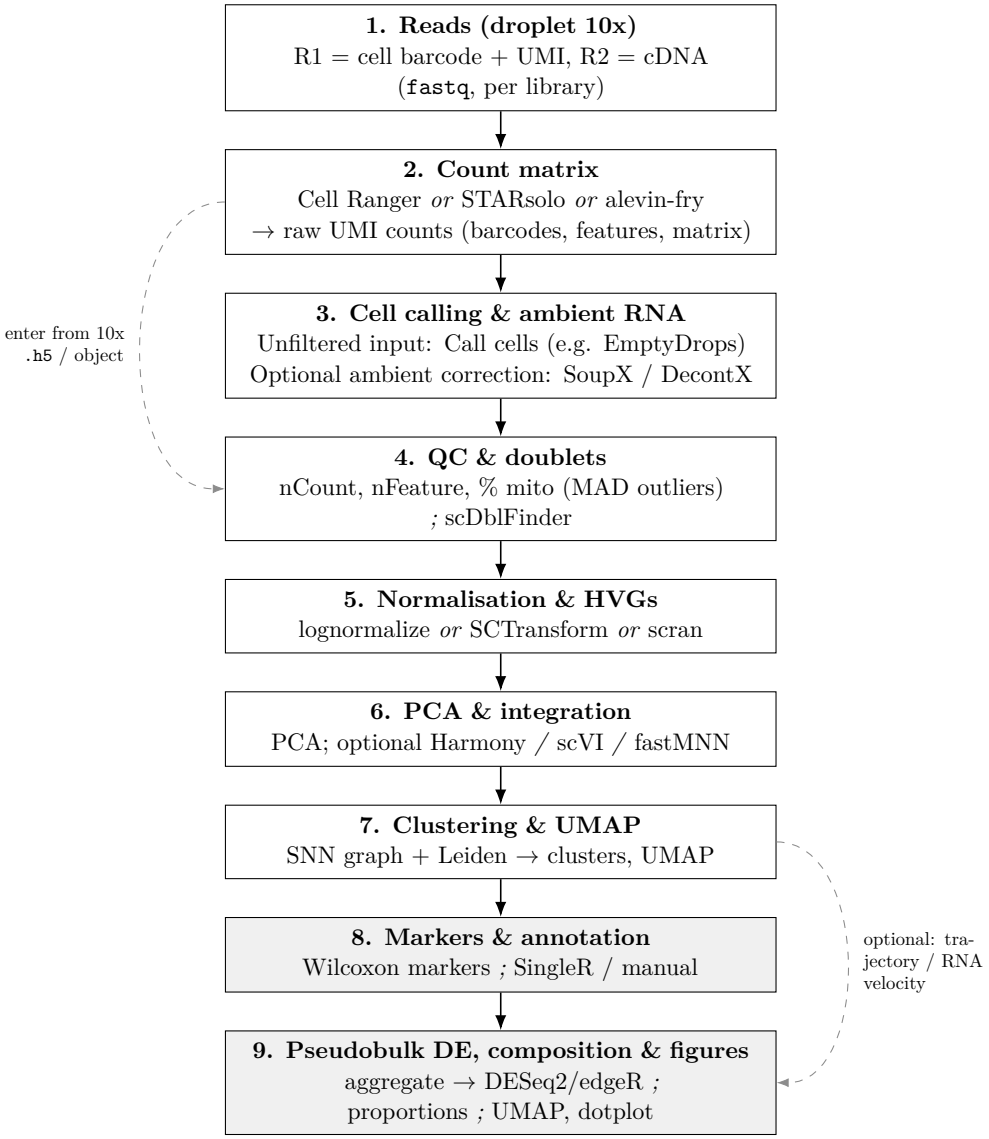
Code boxes are labelled SHELL, R or PYTHON. **Key parameters** identifies settings to choose deliberately; **Inputs & outputs** records the data contract; **Watch out for** explains failure modes; **Tip** adds practical guidance. Optional marks an optional branch. Replace placeholders such as `<sample>` before execution; illustrative snippets may require objects explicitly named in their prerequisites.

Practical note

Matrix orientation. A 10x Matrix Market file and Seurat RNA counts use genes as rows and barcodes/cells as columns. AnnData uses cells as rows and genes as columns. These encode the same conceptual measurements with opposite orientations. Raw UMI counts are finite, non-negative integers; preserve them separately from normalised expression.

The companion folder provides readable numbered scripts. The R route covers QC through annotation, pseudobulk, composition and core figures; the Python route covers QC through exploratory clustering and markers. Optional EmptyDrops, ambient correction, scVI, trajectory and velocity examples need their own prerequisites and are not automatically executed by those scripts. See the companion README for commands and verification status: syntax and synthetic contract checks are distinct from reproducing a biological data set.

3 Pipeline at a glance



4 Software & packages

Command-line tools install through conda, using the bioconda and conda-forge channels; Cell Ranger is installed separately from 10x Genomics. R packages come from CRAN and Bioconductor; Python packages from PyPI or conda-forge. The Bioconductor single-cell workflow (OSCA) documents a complementary R/Bioconductor approach [3].

Command-line tools. The table below lists the aligners/quantifiers that turn raw FASTQ into a gene-by-barcode UMI count matrix, plus supporting utilities.

Tool	Ver.	Purpose
Cell Ranger	—	Count matrix from 10x FASTQ; downloaded from 10x, not on conda [4, 7]
STARsolo	via STAR 2.7.x	Alternative count matrix [8]
salmon + alevin-fry	—	Alternative counting; USA mode for spliced/unspliced [9]
samtools	—	BAM/SAM indexing, sorting, inspection
bcl2fastq / bcl-convert	—	Demultiplex BCL → FASTQ; from Illumina

Practical note

The companion separates counting tools from Python analysis (Section 5); Cell Ranger ships as a self-contained distribution added to `PATH`.

R / Bioconductor packages. The table below covers the R side of the worksheet, from object handling through integration, annotation, and pseudobulk DE.

Package	Purpose
Seurat	Primary object (`Seurat`), QC, clustering, integration, DE [20]
SingleCellExperiment / OSCA workflow	Bioconductor object class and end-to-end workflow [3]
scran	Normalisation (pooling/deconvolution), variance modelling [13, 14]
scater	QC metrics, outlier detection, visualisation [15]
DropletUtils	Reading 10x matrices, empty-droplet calling (`emptyDrops`) [10]
sctransform	Regularised-NB normalisation, variance stabilisation [23]
SoupX **Optional**	Ambient-RNA contamination estimate and correction [11]
celda / DecontX **Optional**	Alternative ambient-RNA decontamination [12]
scDblFinder	Doublet detection via simulated artificial doublets [16]
DoubletFinder	Alternative doublet detection [17]
batchelor (fastMNN)	Batch integration by mutual nearest neighbours [25]
harmony	Batch/embedding integration [24]
SingleR	Reference-based automated cell-type annotation [34]
Azimuth	Reference-mapping annotation (Seurat ecosystem) [35]
presto	Fast Wilcoxon rank-sum marker testing [33]
muscat	Pseudobulk aggregation and DE across conditions [38]
DESeq2	Bulk/pseudobulk differential expression [39]
edgeR	Bulk/pseudobulk differential expression [40]
miloR	Neighbourhood-level differential abundance [41]
speckle / propeller	Cell-type proportion differential abundance [42]
slingshot **Optional**	Trajectory inference on a reduced space [43]
clusterProfiler **Optional**	Functional enrichment of DE gene lists [48]

Python packages. The table below lists the Scanpy/AnnData-based analysis stack.

Package	Purpose
scanpy	Primary analysis API: QC, normalisation, clustering, DE [21]
anndata	Annotated data matrix (`adata`), the object backing scanpy [22]
scvi-tools	Deep generative integration and modelling (scVI) [27, 26]
harmonypy	Python port of Harmony integration [24]
leidenalg / igraph	Leiden community detection on the neighbour graph [31]
scrublet	Doublet detection via simulated artificial doublets [49]
celltypist	Reference-based automated cell-type annotation [36]
scVelo Optional	RNA velocity, dynamical model [46]
umap-learn	UMAP embedding, used internally by `sc.tl.umap` [29]

Watch out for

Record and lock a solved environment for each analysis, including platform, channels and package versions. The supplied `environment.yml` is an unsolved manifest, not a lock. Results can change across package versions; a seed alone does not resolve that difference.

5 Environment setup

Choose the required route before installing software. Counting and downstream analysis have different platform and resource requirements; optional modelling packages need not be part of a minimal environment. The supplied manifests are installation recipes, not solved portable lockfiles.

5.1 Operating system and resources

Cell Ranger is distributed for supported Linux systems; check its version-specific hardware and operating-system requirements before counting [7]. A Linux server or HPC allocation is a common choice. WSL2 needs sufficient assigned memory and compatibility checks; its availability does not establish vendor support. STARsolo and salmon/alevin-fry have their own platform builds and index requirements. Do not assume that a single counting environment will solve on every macOS architecture.

Downstream R and Python analysis may run on Linux, macOS or Windows when compatible package builds are available. Memory demand depends on cell number, feature number, intermediate representations and algorithm choices. Inspect free RAM and disk space before loading large objects or launching parallel workers. Match an explicit thread allocation to both CPU and memory; more threads do not reduce the size of a reference index.

5.2 Project directories and sample metadata

Keep the companion source separate from an analysis root containing references, data and results. The commands below assume an initialized bash shell; replace the companion path with its actual location.

Shell

```
export SC_WORKFLOW="/path/to/main_scripts/singlecell_transcriptomics"
export SCRNA_PROJECT="$HOME/scrna_project"
export SCRNA_THREADS=8   # example allocation: choose for your machine
bash "$SC_WORKFLOW/scripts/00_setup/01_project_dirs.sh"
cd "$SCRNA_PROJECT"
```

Setup creates `ref/`, `fastq/`, `counts/`, `config/` and `results/`. It copies a template `config/samples.tsv` only if that file is absent. Replace its example identifiers and conditions with the actual design before analysis. The core scripts reject repeated donors or captures because they implement the independent one-donor/one-capture-per-sample example, not paired or multiplexed designs.

5.3 Counting and Python environments

Shell

```
# Python exploratory route; use a separate environment for optional packages.
conda env create -f "$SC_WORKFLOW/environment.yml"
conda activate scrna

# Alternative counting tools, normally on Linux; activate separately.
conda env create -f "$SC_WORKFLOW/environment-counting.yml"
# conda activate scrna-counting
```

Cell Ranger is installed separately from 10x Genomics and added to `PATH`; record its actual version, matching reference and chemistry. Optional scVI, CellTypist and scVelo demonstrations have additional dependencies listed in `requirements-optional.txt`; resolve and test these in an isolated environment rather than changing a validated analysis environment without review.

5.4 R and Bioconductor

The minimal R route uses Seurat, scDblFinder, SingleR/celldex, DESeq2 and speckle; Harmony is needed only for the optional integration branch. The installer also requests an SVG device. Optional chapter examples need the packages named in their code boxes.

Shell

```
Rscript "$SC_WORKFLOW/R/install_packages.R"
```

Record the solved R and Bioconductor versions and package dependencies. For a maintained analysis, restore a tested lock when one exists; otherwise create a project environment only after checking compatibility and preserve the resulting record. An environment export describes one solution on one platform and is not proof that another platform can restore it.

Watch out for

Keep the full single-cell matrix sparse: sparse Matrix classes in R, or CSR/CSC arrays in Python. Scaling, model fitting or interconversion may still allocate dense intermediates. Dense conversion of a small pseudobulk matrix is different from densifying thousands of cells. Fix seeds near stochastic operations and record versions; seeds do not guarantee identical results across software versions, hardware or algorithm implementations.

6 From reads to a count matrix

Counting assigns cDNA reads to genes, corrects cell barcodes and UMIs, and collapses duplicate observations within barcode/gene groups. For the running 10x 3′ v3 example, R1 contains a 16-base cell barcode and 12-base UMI, while R2 contains cDNA. Other chemistries use different layouts; match the library's documented geometry rather than inferring it from a filename [4, 7].

6.1 Demultiplexing Optional

If the input is a sequencer's BCL output, generate sample FASTQ files with a supported Illumina BCL Convert workflow and the correct index/sample sheet. Existing `cellranger mkfastq` workflows wrap bcl2fastq, but 10x now deprecates that command in favour of BCL Convert [7]. Do not trim barcode or UMI sequence during demultiplexing. If a sequencing core supplies FASTQ files, verify read pairing, sample assignment and library chemistry before counting.

6.2 Cell Ranger count

Cell Ranger aligns reads, performs barcode/UMI processing, calls cells and writes raw and filtered matrices plus a per-library QC report. Run once for each independent library in this example; sequencing lanes of the same library are inputs to the same counting run, not biological replicates.

Shell

```
cellranger count --id=sample1 \
  --transcriptome=refdata-gex-GRCh38-2024-A \
  --fastqs=fastq/ --sample=sample1 --create-bam=true
```

Inputs and outputs

Inputs: compatible FASTQ files and a species/build-matched 10x reference. Outputs under `sample1/outs/`: `filtered_feature_bc_matrix/` for called cells; `raw_feature_bc_matrix/` including uncalled droplets; and `web_summary.html`. Matrix Market counts are genes×barcodes. Review mapping, sequencing saturation, cell calling and contamination diagnostics before continuing.

Key parameters

Record the actual Cell Ranger version, chemistry, reference and annotation release, FASTQ selection and relevant counting options. Intronic reads are included by default in Cell Ranger 7 and later; confirm the selected version's setting and keep it consistent across samples. Cell-calling parameters depend on the assay and observed data; do not change them solely to recover a desired number of cells.

The companion `01_cellranger_count.sh` stores run directories under the analysis root's `count_runs/` and copies only the filtered triplet to `counts/<sample>/`. It refuses an existing destination. Retain the original raw output for cell-calling or ambient-RNA work; the copied filtered directory cannot replace it.

6.3 Alternative: STARsolo

STARsolo performs barcode/UMI counting within STAR [8]. Matching reference, read geometry, UMI rules, intron handling and cell-calling settings matters when comparing it with Cell Ranger; changing the mapper does not guarantee numerical identity.

Shell

```
STAR --runMode alignReads --genomeDir star_index \
  --soloType CB_UMI_Simple --soloCBwhitelist 3M-february-2018.txt \
  --soloCBstart 1 --soloCBlen 16 --soloUMIstart 17 --soloUMIlen 12 \
  --soloFeatures GeneFull --soloCellFilter EmptyDrops_CR \
  --readFilesIn R2.fastq.gz R1.fastq.gz --readFilesCommand zcat \
  --outFileNamePrefix sample1_
```

R2 precedes R1 in `--readFilesIn` because STAR expects cDNA first and barcode/UMI sequence second. The companion pairs lane files explicitly. `GeneFull` includes intronic signal; `Gene` is an exon-based alternative with a different target. Verify the STAR index and annotation against the intended comparison. A v2 whitelist/UMI layout is not interchangeable with v3.

The standalone command above writes `sample1_Solo.out/` in the working directory. From that same analysis root, import its filtered triplet once into `counts/sample1/` with the companion adapter:

Shell

```
Rscript "$SC_WORKFLOW/scripts/01_count_matrix/04_import_matrix.R" \
  10x sample1_Solo.out/GeneFull/filtered sample1
```

6.4 Alternative: alevin-fry

salmon alevin can map reads to a spliced-plus-intronic (splici) reference, followed by alevin-fry permit-list generation, collation and quantification [9]. The index, transcript-to-gene map and gene-symbol map must come from the same annotation. Build these using the selected version's documented reference workflow before running this example; an index path alone is not a complete reference contract.

Shell

```
salmon alevin -i salmon_splici -l ISR \
  -1 R1.fastq.gz -2 R2.fastq.gz --chromiumV3 --sketch -o map
alevin-fry generate-permit-list -i map -d fw -o quant \
  --unfiltered-pl 3M-february-2018.txt
alevin-fry collate -i quant -r map -t 8
alevin-fry quant -i quant -m t2g_3col.tsv -r cr-like \
  -o quant/res --use-mtx -t 8
```

A three-column transcript/gene/status map enables USA (unspliced, spliced, ambiguous) quantification. Preserve those states if velocity will be explored. They are not an ordinary 10x filtered matrix: use a USA-aware loader, choose the expression representation explicitly, and distinguish barcode correction from cell calling.

The companion import branch uses `fishpond::loadFry(outputFormat="all")` to combine U+S+A once per gene, retaining the original USA files separately. It requires `config/gene_symbols.tsv` with unique `gene_id` and `gene_symbol` columns from the counting annotation, plus a reviewed one-barcode-per-line list of called cells. Perform cell calling on the unfiltered barcode matrix with an appropriate method before supplying that list (Section 8).

The standalone commands write `quant/res`; the companion counting wrapper instead writes `counts/<sample>_quant/res`. Use the location produced by the chosen route. For the standalone commands above:

Shell

```
Rscript "$SC_WORKFLOW/scripts/01_count_matrix/04_import_matrix.R" \
  fry quant/res sample1 config/sample1_called_barcodes.tsv
```

6.5 Read the standardised matrix

The companion contract is a compressed `matrix.mtx.gz`, `features.tsv.gz` and `barcodes.tsv.gz` under `counts/<sample>/`. Its reader verifies matrix dimensions, count values and identifiers, selects Gene Expression features and retains stable gene IDs alongside display symbols. Reference/feature order must agree across samples.

R

```
source(file.path(Sys.getenv("SC_WORKFLOW"), "R/contracts.R"))
input <- read_counts("counts/sample1")
so <- Seurat::CreateSeuratObject(input$counts, min.cells = 0, min.features = 0)
# Alternative for an original raw 10x directory:
sce <- DropletUtils::read10xCounts("sample1/outs/raw_feature_bc_matrix")
```

PYTHON

```
import scanpy as sc
adata = sc.read_10x_mtx("counts/sample1", var_names="gene_symbols")
adata.var_names_make_unique()
```

Watch out for

Filtered matrices have already undergone cell calling; do not rerun EmptyDrops on that restricted set. Use the original unfiltered matrix for an ambient-profile cell-calling test or SoupX background estimation. Different intron assignments, gene annotations or barcode/UMI rules can change counts even when dimensions and filenames agree. The import adapter verifies format preservation; it does not certify mapper equivalence or cell identity.

7 Alternative entry points

This section covers three alternative entry points: a count matrix produced by a counter, a saved analysis object, or a public data set. A raw USA matrix needs its own loading and cell-calling route; it is not automatically a filtered 10x matrix. Pick the one that matches what you have and enter at QC (Section 8) or a later verified stage.

7.1 From a 10x count matrix

A 10x `filtered_feature_bc_matrix` directory and its single-file `.h5` equivalent encode the same gene-by-barcode sparse UMI counts. Seurat reads either; Bioconductor reads the directory form into a `SingleCellExperiment`.

R

```r
library(Seurat)

# from the .h5 file (single file, convenient for sharing)
m <- Read10X_h5("sample1/outs/filtered_feature_bc_matrix.h5")
if (is.list(m)) m <- m[["Gene Expression"]]
so <- CreateSeuratObject(m, project = "sample1", min.cells = 0, min.features = 0)

# from the mtx/ directory (barcodes.tsv.gz, features.tsv.gz, matrix.mtx.gz)
m <- Read10X("sample1/outs/filtered_feature_bc_matrix")
if (is.list(m)) m <- m[["Gene Expression"]]
so <- CreateSeuratObject(m, project = "sample1", min.cells = 0, min.features = 0)

# OSCA / Bioconductor entry point
library(DropletUtils)
sce <- read10xCounts("sample1/outs/filtered_feature_bc_matrix", col.names = TRUE)
```

PYTHON

```python
import scanpy as sc

# from the .h5 file
adata = sc.read_10x_h5("sample1/outs/filtered_feature_bc_matrix.h5")
adata.var_names_make_unique()

# from the mtx/ directory
adata = sc.read_10x_mtx("sample1/outs/filtered_feature_bc_matrix", var_names="gene_symbols")
```

Inputs and outputs

Input: a 10x `outs/` directory or `.h5` file (filtered or raw). Output: a `Seurat` object, a `SingleCellExperiment`, or an `AnnData` with raw UMI counts in `counts/so[["RNA"]]$counts/adata.X` and no other normalisation applied. The illustrated constructors retain the imported features and cells; QC and metadata assignment are separate steps.

7.2 From an existing analysis object

A colleague's saved object – an `.rds` Seurat object or an `.h5ad` AnnData – can be loaded directly, or converted to work in the other ecosystem. `sceasy` and `SeuratDisk` convert Seurat↔AnnData; `zellkonverter` converts `SingleCellExperiment`↔AnnData.

R

```r
library(Seurat)
so <- readRDS("sample1_processed.rds")

# convert to AnnData for a Python session (sceasy)
library(sceasy)
sceasy::convertFormat(so, from = "seurat", to = "anndata", outFile = "sample1.h5ad")
# Version-dependent demonstration: verify v5 layer support before using it.

# SingleCellExperiment <-> AnnData (zellkonverter)
library(zellkonverter)
sce <- readH5AD("sample1.h5ad")
```

PYTHON

```python
import anndata as ad

adata = ad.read_h5ad("sample1_processed.h5ad")
```

Watch out for

Interconversion needs verification of both values and conventions: Seurat's `scale.data` slot, Seurat v5 layers, and AnnData's `.obsm`/`.uns` do not map one-to-one. After conversion, check that raw counts, normalised data and embeddings retain the expected values, locations and identifier order before proceeding.

7.3 Public data

Published matrices come from GEO supplementary files, the 10x Genomics public datasets page, CELLxGENE Discover, or the Human Cell Atlas (HCA) data portal. Formats vary (mtx triplets, loom, `.h5ad`, `.rds`), so read the accompanying README before assuming raw counts.

Watch out for

Deposited objects are frequently already normalised, log-transformed, or fully annotated with cell types. Inspect the object before re-running the pipeline: `adata.X` (and `adata.raw`) in AnnData, or `so[["RNA"]]$counts` versus `so[["RNA"]]$data` in Seurat. Do not re-normalise an already-normalised representation. Curated labels can be reused when they match the new question; if reclustering is scientifically justified, preserve the original labels and explain the new analysis.

8 Quality control & cell calling

Droplet calling, ambient-RNA correction, cell-quality filtering and doublet detection answer different quality questions. A barcode called as a cell is not necessarily a high-quality singlet, so inspect each capture's results before combining samples [1, 2].

8.1 Call cells from unfiltered droplets Optional

A filtered Cell Ranger or STARsolo matrix already contains called cells. Skip additional cell calling when using that input. For an unfiltered matrix, examine barcode ranks and test candidate droplets against an estimated ambient profile with EmptyDrops; it uses information beyond a single count threshold [10].

R

```
library(DropletUtils)
sce <- read10xCounts("sample1/outs/raw_feature_bc_matrix", col.names = TRUE)
br <- barcodeRanks(counts(sce))
set.seed(42)
e <- emptyDrops(counts(sce))
keep <- which(e$FDR <= 0.001)
sce <- sce[, keep]
```

Inputs and outputs

Input: unfiltered counts, including droplets used to estimate ambient RNA. Output: a called-cell subset and diagnostics. The illustrated FDR threshold is 0.001; inspect the barcode-rank distribution, ambient model and simulation resolution before accepting the call set. Export the retained barcodes explicitly when using the USA import branch.

Watch out for

NA statistics for very low-count droplets excluded from testing are intentional; they do not usually indicate insufficient Monte Carlo sampling. Inspect the `Limited` flag for tested droplets near the decision threshold and increase iterations using the argument documented for the installed version when simulation resolution is limiting. Set a seed. A filtered input cannot supply the background population this test requires.

8.2 Assess ambient RNA Optional

RNA released into the suspension can contribute counts to cell-containing droplets. SoupX estimates contamination using the raw droplet background and cluster-associated expression; DecontX uses a probabilistic mixture approach [11, 12]. Assess whether contamination could explain apparent low expression of another population's markers or a condition-associated signal.

R

```
# Demonstration: define these matrices and preliminary cluster labels first.
# Matrices are genes x barcodes from the same capture and annotation.
library(SoupX)
sc <- SoupChannel(tod = raw_matrix, toc = filtered_matrix)
sc <- setClusters(sc, preliminary_clusters)
sc <- autoEstCont(sc)
corrected <- adjustCounts(sc, roundToInt = TRUE)
```

The preliminary clusters support contamination estimation; they are not final biological labels. Preserve original counts and diagnostics separately from the inferred corrected counts. Correction changes the data model, so validate the chosen representation and its suitability for later count-based inference rather than silently replacing every raw-count layer. The core companion route does not apply ambient correction automatically.

Practical note

Estimate ambient profiles by capture. Pooling unrelated captures before correction mixes backgrounds. For multiplexed designs, capture and donor are different fields; decide which cells share the same ambient pool before analysis.

8.3 Inspect per-cell metrics and choose filters

Compute total counts, detected genes, mitochondrial fraction and other relevant feature summaries. Low counts, high mitochondrial fractions or unusually high feature counts can identify problematic cells, but none is a universal biological diagnosis. Tissue, preparation, depth and cell type affect these distributions [1, 15].

R

```r
so <- CreateSeuratObject(m, min.cells = 0, min.features = 0)
so[["percent.mt"]] <- PercentageFeatureSet(so, pattern = "^MT-")
so[["percent.ribo"]] <- PercentageFeatureSet(so, pattern = "^RP[SL]")
# Illustrative human-cell thresholds; inspect distributions first.
so <- subset(so, subset = nFeature_RNA > 200 & nFeature_RNA < 6000 & percent.mt < 15)
```

These symbol patterns apply to human-style gene symbols, not Ensembl IDs or every species. The companion retains a stable-ID/symbol table and stops when its mitochondrial pattern matches no features. A zero mitochondrial percentage caused by an identifier mismatch must not be interpreted as excellent quality.

R

```r
# Bioconductor alternative; read10xCounts stores symbols in rowData.
mito <- grep("^MT-", rowData(sce)$Symbol)
qc <- scuttle::perCellQCMetrics(sce, subsets = list(Mito = mito))
discard <- scuttle::perCellQCFilters(qc, sub.fields = "subsets_Mito_percent")
sce <- sce[, !discard$discard]
```

PYTHON

```python
adata.var["mt"] = adata.var_names.str.startswith("MT-")
sc.pp.calculate_qc_metrics(adata, qc_vars=["mt"], percent_top=None, inplace=True)
adata = adata[(adata.obs.pct_counts_mt < 15) &
              (adata.obs.n_genes_by_counts > 200) &
              (adata.obs.n_genes_by_counts < 6000)].copy()
```

Key parameters

Inspect metrics per sample and relevant cell population; MAD-based outliers can help identify unusual values, but should not automatically remove legitimate rare types. Record retained/discarded cell counts and reasons. The companion first writes per-sample QC metrics and stops until `SC_QC_REVIEWED=yes`; its threshold variables are an explicit starting configuration, not a recommendation for all tissues.

8.4 Detect doublets

Doublets can combine distinct expression profiles and create misleading hybrid populations. scDblFinder and Scrublet simulate artificial doublets and compare them with observed profiles [16, 49]. They are less informative for doublets whose component cells are very similar; predicted singlets are not a guarantee that no doublets remain. DoubletFinder is another R alternative [17].

R

```r
library(scDblFinder)
# sce$capture must identify independent droplet captures.
set.seed(42)
sce <- scDblFinder(sce, samples = "capture",
                   BPPARAM = BiocParallel::SerialParam(RNGseed = 42))
sce <- sce[, sce$scDblFinder.class == "singlet"]
```

PYTHON

```python
import scrublet as scr
scrub = scr.Scrublet(adata.X, random_state=42)  # raw counts, one capture
scores, predicted = scrub.scrub_doublets()
adata.obs["doublet_score"] = scores
adata = adata[~predicted].copy()
```

The companion processes each capture separately before merge under its one-capture-per-sample contract. A merged SingleCellExperiment can also be processed correctly when `samples` explicitly identifies captures. Check score distributions and threshold behaviour; use assay-specific loading/recovery guidance when specifying an expected rate. There is no single rate per thousand recovered cells that applies to all chemistries.

Watch out for

Remove or investigate likely doublets before interpreting final clusters or testing conditions. Do not declare an intermediate biological state solely because mixed markers or a connecting UMAP bridge remain. Conversely, a computational doublet score alone does not establish that a rare population is an artefact; inspect its evidence and consistency across independent samples.

9 Normalisation & feature selection

Normalisation and feature selection prepare the data for exploratory comparisons and a low-dimensional representation. Normalisation makes cells with different sequencing depths easier to compare; feature selection identifies the genes used to build that representation. These steps do not remove every technical effect, and normalised expression does not preserve an absolute RNA scale. Keep the full gene set and the original UMI counts for annotation and sample-level inference [1, 2].

Inputs and outputs

Input: per-sample QC'd singlets, with sample, donor, capture and condition metadata. Output: preserved raw counts, a normalised expression representation, an HVG mask and PCA input. The companion R phase reads `results/objects/qc.rds` and writes `results/objects/pca.rds`; the Python exploratory route saves both count and normalised representations in its final `.h5ad`.

9.1 Choose one normalisation route

Global log-normalisation divides each cell's counts by its total, multiplies by a chosen scale factor and applies `log1p`; the example uses 10,000. This is a practical exploratory default, with the assumption that relative library totals provide a useful depth correction. Large composition changes or differences in total RNA can complicate that assumption.

SCTransform instead fits a regularised negative-binomial model and uses Pearson residuals for variance stabilisation [23]. It replaces the usual Seurat normalisation, feature-selection and scaling calls; its RNA input counts must remain available separately. scran pools cells to estimate size factors, then deconvolves those factors and constructs log-normalised expression [13, 14]. These are alternative workflows, not three sequential corrections.

R

```r
# Minimal Seurat route; counts remain in the RNA assay.
so <- NormalizeData(so, normalization.method = "LogNormalize", scale.factor = 10000)
so <- FindVariableFeatures(so, selection.method = "vst", nfeatures = 2000)
so <- ScaleData(so, features = VariableFeatures(so))
```

Seurat v5 keeps per-sample layers when the companion QC phase merges objects. Normalisation and variable-feature selection can therefore operate on those layers directly; do not split an already split assay

again. Rejoin the RNA layers before marker testing or extracting a single count matrix. A single-sample object does not require splitting or integration.

R

```r
# Alternative Seurat route, starting again from QC'd RNA counts.
so <- SCTransform(so, assay = "RNA", verbose = FALSE)
# Prepare RNA data separately for the later RNA-based examples.
so <- NormalizeData(so, assay = "RNA", normalization.method = "LogNormalize",
                    scale.factor = 10000)
DefaultAssay(so) <- "SCT"  # PCA uses SCT; original RNA counts remain unchanged.
```

Here SCT supplies the variable features and scaled residuals for PCA. The separate `NormalizeData(assay="RNA")` call creates log-normalised RNA data for the later RNA marker tests, SingleR input and expression plots; it does not normalise SCT again. Neither step replaces the original RNA counts used for pseudobulk. Setting the default assay alone would not create a missing RNA data layer.

R

```r
# Alternative Bioconductor route, starting from counts(sce).
library(scran)
library(scater)
set.seed(42)
clusters <- quickCluster(sce)
sce <- computeSumFactors(sce, clusters = clusters)
sce <- logNormCounts(sce)
```

PYTHON

```python
import scanpy as sc

# Save before any operation that overwrites X; keep all genes.
adata.layers["counts"] = adata.X.copy()
sc.pp.normalize_total(adata, target_sum=1e4)
sc.pp.log1p(adata)
```

Watch out for

A counts layer contains finite, non-negative integer UMI counts, not log-normalised expression or Pearson residuals. AnnData's `.raw` is a saved representation chosen by the analyst; its name does not establish that it contains raw counts. Rounding transformed values cannot recover the original counts. Use the preserved count layer for scVI and pseudobulk, and the appropriate normalised representation for descriptive marker plots.

9.2 Select highly variable genes

About 2,000 HVGs is a starting choice for exploratory PCA, not a biological constant. Including more genes can retain weaker state signals but also introduce additional technical or uninformative variation; too few can omit a rare population's distinguishing features. The selected genes determine PCA, whose retained components determine distances and neighbours, so feature choices can propagate to clusters and UMAP. Inspect whether the selected genes and PCs reflect the cell states of interest, sequencing depth or another unwanted effect. Seurat's `vst` selection models the mean–variance relationship. Scanpy's `seurat_v3` flavour expects raw counts and requires the additional `scikit-misc` dependency; its input differs from Scanpy's log-data-based `seurat` flavour.

R

```r
# Already included in the minimal Seurat route above.
top10 <- head(VariableFeatures(so), 10)

# scran alternative, using log-normalised expression in sce.
dec <- modelGeneVar(sce)
hvgs <- getTopHVGs(dec, n = 2000)
```

PYTHON

```python
sc.pp.highly_variable_genes(
    adata, n_top_genes=2000, flavor="seurat_v3",
    layer="counts", batch_key="sample",
)
# Preserve non-HVGs; PCA uses a feature mask instead of slicing them away.
sc.pp.pca(adata, n_comps=30, mask_var="highly_variable", random_state=42)
```

The Python example selects HVGs within samples and combines their rankings. This can reduce selection driven only by sample-specific variation; it does not establish that all sample differences are technical. The Seurat and Scanpy implementations need not select identical genes or yield numerically identical embeddings.

A 2025 benchmark compared 24 feature-selection methods across ten datasets and found variance-stabilising HVG methods effective for integration and querying. Its support for roughly 2,000 features is task-dependent, not a requirement to discard other genes or a universal optimum. Reassess feature choice when rare populations, states or the integration task differ [50].

9.3 Scaling and unwanted variation

The Seurat log-normalisation route centres and scales the selected features before PCA. SCTransform already supplies its residual representation. The Python companion uses PCA on log-normalised HVGs without an additional variance-scaling step; this is an explicit alternative representation, so its PCs are not an exact recreation of the Seurat route. If scaling is chosen in Python, apply it only to a suitable working representation and preserve counts and log expression separately.

R

```r
# Optional regression in a log-normalised Seurat analysis.
# so <- ScaleData(so, features = VariableFeatures(so),
#                 vars.to.regress = "percent.mt")
```

Key parameters

Record the normalisation method, scale factor, feature-selection method, HVG count and any batch variable or regression covariates. The companion uses 2,000 requested HVGs and up to 30 PCs, reducing the PC count when the matrix is smaller. Compare the uncorrected representation before choosing integration (Section 10).

Watch out for

Regress a covariate only when removing its signal serves the scientific question. Mitochondrial expression and cell-cycle activity can reflect biology as well as quality or technical variation. Routine regression can remove real differences in stressed, proliferating or differentiating populations. Reassess the resulting representation rather than assuming that regression improves it.

10 Dimensionality reduction & integration

PCA summarises variation in the selected expression representation through components that can be used to build an exploratory neighbourhood graph. These components can reflect biology, technical variation or both, so decide whether integration is needed before constructing the final shared representation [1, 28].

10.1 PCA

To interpret the PCs, inspect their explained variance and contributing genes alongside sample, quality and biological annotations. A range such as 20–50 components is a starting exploration, not an automatic cutoff. A visible elbow alone cannot establish which variation is biological.

R

```
set.seed(42)
so <- RunPCA(so, features = VariableFeatures(so), npcs = 30, seed.use = 42)
ElbowPlot(so, ndims = 30)
reduction <- "pca"
dims <- seq_len(ncol(Embeddings(so, reduction)))
```

PYTHON

```
sc.pp.pca(adata, n_comps=30, mask_var="highly_variable", random_state=42)
sc.pl.pca_variance_ratio(adata, n_pcs=30, log=True)
representation = "X_pca"
```

The inline examples request 30 PCs and require enough cells and selected features for that request. The companion instead caps the request at the smaller of 30, the variable-feature count minus one and the cell count minus one, and stops below two PCs. Including additional PCs may retain subtle states or add depth/quality variation; excluding them may discard either. Inspect loadings and sensitivity of the graph and annotations rather than assuming that more PCs improve the analysis. Seurat's scaled-feature PCs and the Python companion's unscaled log-expression PCs are different representations; numerical agreement is not their validation target.

10.2 Decide whether to integrate

First construct an uncorrected representation and inspect it by sample, known technical variables and plausible cell types. Sample separation may reflect a technical effect, a real condition response, different cell populations or inadequate QC. UMAP appearance alone cannot identify the cause. Complete confounding between batch and condition cannot be repaired by integration.

Practical note

The companion defaults to uncorrected PCA. After evaluating the design and uncorrected structure, `SC_INTEGRATION_KEY` can request Harmony using an existing metadata field. Preserve the original PCA and its diagnostics. Choosing `sample` aligns sample-associated variation for exploratory mapping; it does not establish that all removed variation was technical.

10.3 Integration alternatives

Harmony iteratively adjusts a PCA representation [24]. Seurat's anchor-based CCA/RPCA methods identify correspondences between data sets [19]; fastMNN matches mutual nearest neighbours in a reduced space [25]. scVI fits a generative count model with batch covariates and supplies a latent representation [26, 27]. Their assumptions, compute requirements and preservation of biological variation differ. Benchmarks show that performance depends on the data and evaluation target [28]; no one method is universally best.

R

```r
# Optional Harmony branch after normalization and PCA.
# Confirm that this field is the intended grouping variable.
so <- JoinLayers(so, assay = "RNA")
set.seed(42)
so <- harmony::RunHarmony(so, group.by.vars = "sample", reduction.use = "pca")
reduction <- "harmony"
# Alternative: IntegrateLayers on appropriately split Seurat v5 layers.
```

PYTHON

```python
# Optional scVI demonstration; requires preserved integer counts.
import scvi
scvi.settings.seed = 42
model_data = adata[:, adata.var.highly_variable].copy()
scvi.model.SCVI.setup_anndata(model_data, batch_key="sample", layer="counts")
model = scvi.model.SCVI(model_data)
model.train()
assert model_data.obs_names.equals(adata.obs_names)
adata.obsm["X_scVI"] = model.get_latent_representation()
representation = "X_scVI"
```

Key parameters

Evaluate both removal of unwanted technical structure and preservation of relevant biological differences. Strong batch mixing alone is insufficient: overcorrection can also mix genuinely different states. Record the selected method, grouping variables, feature set, parameters and comparison with uncorrected data. Keep original RNA counts for sample-level expression inference (Section 12).

Assess conservation within cell types as well as separation between types. A benchmark using lung and breast atlases showed that coarse label-based integration scores can miss losses of within-type structure [51]. Compare uncorrected and corrected representations using expected states and sample-level expression evidence; do not select the result solely by attractive mixing in UMAP. A 2026 perspective also identifies overcorrection and undercorrection as continuing limitations [52].

10.4 Build a graph and visualise it

UMAP visualises local structure in the chosen PCA or latent representation [29]; t-SNE is another display option [30]. Use the same representation for the intended graph-based clustering, and label which representation each figure shows.

R

```r
dims <- seq_len(ncol(Embeddings(so, reduction)))
set.seed(42)
so <- FindNeighbors(so, reduction = reduction, dims = dims)
set.seed(42)
so <- RunUMAP(so, reduction = reduction, dims = dims, seed.use = 42)
DimPlot(so, reduction = "umap", group.by = "sample")
```

PYTHON

```
sc.pp.neighbors(adata, use_rep=representation, random_state=42)
sc.tl.umap(adata, random_state=42)
sc.pl.umap(adata, color="sample")
```

The Seurat calls above use the same PCs but construct separate neighbourhoods: `FindNeighbors` uses `k.param` for the clustering graph, whereas `RunUMAP` uses `n.neighbors` for its display neighbourhood and `min.dist` for how tightly points may pack. Increasing `min.dist` generally spreads the display; it does not change the previously computed clusters. A common representation therefore does not imply the same graph. Scanpy's UMAP instead uses the graph stored by `sc.pp.neighbors`; its display still need not preserve global distances.

Watch out for

Global distances, density, cluster area and apparent bridges in UMAP can be distorted by the display algorithm. Do not cluster the UMAP coordinates or treat their geometry as a quantitative expression effect. Cluster on an appropriate high-dimensional graph and test expression using gene-level data with the required replicate-aware model. Fix seeds and record versions without promising bit-identical embeddings across platforms.

11 Clustering & annotation

Clustering organises a chosen expression representation into groups; annotation assigns biological meaning using markers, references and context. Neither a cluster number nor an automated label establishes a cell identity by itself. Begin with the uncorrected or deliberately integrated representation selected in Section 10 [1, 2].

11.1 Graph-based clustering

Seurat constructs a nearest-neighbour graph and typically uses it to build a shared nearest-neighbour (SNN) graph that weights shared neighbours; Scanpy constructs a weighted neighbourhood graph. Community detection operates on these graphs, not on a two-dimensional UMAP. Leiden addresses connectivity problems of Louvain partitions [31]; Louvain remains Seurat's default unless another algorithm is selected [32].

R

```
set.seed(42)
so <- FindNeighbors(so, reduction = reduction, dims = dims)
set.seed(42)
so <- FindClusters(so, resolution = 0.5, algorithm = 4, random.seed = 42)
```

PYTHON

```
sc.pp.neighbors(adata, use_rep=representation, random_state=42)
sc.tl.leiden(adata, resolution=0.5, key_added="leiden", random_state=42,
             flavor="igraph", directed=False, n_iterations=2)
```

Check the chosen Seurat version's Leiden backend and dependencies; older releases may use Python via reticulate, while current interfaces also expose R backends. The companion requests Leiden explicitly. A package change can alter the graph or partition even when the seed and resolution remain unchanged.

Key parameters

Neighbourhood size changes the scale of the graph: Smaller neighbourhoods emphasise local structure and may fragment it; larger ones connect broader regions and can merge fine distinctions. Seurat exposes `k.param`, while Scanpy exposes `n_neighbors`; record each explicitly when assessing sensitivity. Resolution controls partition granularity; increasing it generally favours finer partitions for a fixed graph, but 0.5 is a teaching starting value, not an equivalent partition across software, graphs or datasets. Explore a suitable range, examine stability and marker coherence, and ask whether candidate populations recur across independent samples. More clusters do not necessarily provide more biological information. Record the representation, neighbours, graph settings, clustering algorithm and seed; avoid selecting resolution solely to obtain an expected number of cell types.

11.2 Descriptive marker genes

To find candidate markers, a one-versus-rest test compares each cluster with all remaining cells and ranks enriched genes. Seurat and Scanpy can use a Wilcoxon rank-sum test on normalised gene expression; presto accelerates this calculation when the selected Seurat version uses it [33]. These are descriptive annotation tools rather than the donor-level condition test.

R

```r
so <- JoinLayers(so, assay = "RNA")
markers <- FindAllMarkers(so, assay = "RNA", only.pos = TRUE,
                          min.pct = 0.25, logfc.threshold = 0.25, test.use = "wilcox")
```

The RNA assay must contain the log-normalised data prepared in Section 9, including after the SCT alternative; joining layers does not create those values. Here `min.pct=0.25` keeps genes detected in at least 25% of cells in either comparison group, while `logfc.threshold=0.25` screens by average log fold change. Increasing either filter reduces the candidates and computation but can exclude sparse or weaker markers. These are prefilters for descriptive markers, not donor-level effect thresholds; `only.pos=TRUE` reports positive markers only.

PYTHON

```python
sc.tl.rank_genes_groups(adata, groupby="leiden", method="wilcoxon", use_raw=False)
markers = sc.get.rank_genes_groups_df(adata, group=None)
```

R

```r
# Bioconductor alternative, after assigning cluster labels to sce.
markers <- scran::findMarkers(sce, groups = colLabels(sce), test.type = "wilcox")
```

Watch out for

The same data helped define the clusters and identify their markers. Standard marker p-values do not account for that selection or necessarily for donor dependence. Use effect sizes, detection frequencies, known marker panels and independent samples to assess annotation; do not reinterpret a cluster-marker table as condition-level differential expression (Section 12).

11.3 Reference-based annotation

The following SingleR conversion uses RNA counts and its prepared log-normalised data, rather than SCT residuals. SingleR compares query expression with labelled reference profiles, using references such as those supplied by celldex [34]. Azimuth provides reference mapping in the Seurat ecosystem [35]; CellTypist uses trained classifiers, including immune and tissue models [36]. Choose a reference compatible with the species, tissue, preparation and expected populations, and inspect uncertain or unassigned predictions.

R

```
library(SingleR)
library(celldex)
ref <- HumanPrimaryCellAtlasData()
pred <- SingleR(test = as.SingleCellExperiment(so, assay = "RNA"),
                ref = ref, labels = ref$label.main)
so$SingleR <- pred$labels[match(colnames(so), rownames(pred))]
so$SingleR_pruned <- pred$pruned.labels[match(colnames(so), rownames(pred))]
```

PYTHON

```
# Optional demonstration: obtain a suitable model first.
import celltypist
model = celltypist.models.Model.load(model="Immune_All_Low.pkl")
pred = celltypist.annotate(adata, model=model, majority_voting=True)
adata.obs["celltypist_label"] = pred.predicted_labels["majority_voting"]
```

The CellTypist example expects the model's documented normalised input and gene identifiers. It does not run on an arbitrary scaled matrix. A reference cannot reliably assign populations absent from its training labels; a plausible label therefore needs marker and context checks rather than automatic acceptance.

11.4 Confirm and store reviewed labels

Inspect canonical positive and negative markers, detection fractions and per-sample representation. A cluster may contain a mixed population, low-quality cells or uncertain labels; preserve an unassigned category when the evidence does not support a finer identity. The companion writes a proposed `config/cluster_labels.tsv`, stops for review, and applies the edited map only with `SC_LABELS_REVIEWED=yes`.

R

```
# Example only: replace with a map covering all observed clusters exactly.
new_ids <- c("0" = "CD4 T", "1" = "CD14 Mono", "2" = "B", "3" = "NK")
stopifnot(setequal(names(new_ids), levels(so)))
so$celltype <- unname(new_ids[as.character(Idents(so))])
Idents(so) <- "celltype"
DotPlot(so, assay = "RNA", features = c("CD3D", "CD4", "CD14", "MS4A1", "NKG7"))
```

PYTHON

```
new_ids = {"0": "CD4 T", "1": "CD14 Mono", "2": "B", "3": "NK"}
assert set(new_ids) == set(adata.obs["leiden"].astype(str))
adata.obs["celltype"] = adata.obs["leiden"].astype(str).map(new_ids)
sc.pl.dotplot(adata, ["CD3D", "CD4", "CD14", "MS4A1", "NKG7"], groupby="celltype")
```

Use the metadata field `celltype` consistently in figures and inference. Cluster numbers can change across reruns; record the reviewed map, reference version and supporting marker evidence with the object. The Python companion supplies clusters and markers but requires this annotation step separately.

12 Differential expression across conditions

A condition comparison asks whether an effect generalises across independent experimental units, usually donors. Cells from one donor do not supply additional independent donor replicates. The running example has one library and one droplet capture per independent donor; paired donors, repeated measures, technical libraries and multiplexed captures require a design that represents those relationships [37, 38].

Connect the question to bulk RNA-seq. A whole-tissue bulk contrast combines expression within cell populations with their contributions to the RNA mixture. A cell-type pseudobulk contrast instead asks about relative expression within the annotated population; a composition analysis asks about its share of recovered cells. These quantities can change independently. Agreement is not required, and a within-type effect does not by itself explain a whole-tissue effect [37, 38].

12.1 Pseudobulk: aggregate raw counts within samples

Pseudobulk analysis makes the sample the unit of comparison by summing original UMI counts across cells of one reviewed type within each sample. The resulting genes×samples matrix is the input to DESeq2 or edgeR. A correctly specified sample-level model can account for between-sample variability; aggregation alone does not guarantee valid inference when replication, annotation, confounding or the model design is inadequate [37, 38, 39, 40].

A small aggregation example. The following invented UMI counts are for two genes in five cells of one reviewed type. A1–A3 belong to donor A; B1–B2 belong to donor B. In this example each donor supplies one sample.

Gene	A1	A2	A3	B1	B2	Donor A	Donor B
G1	2	0	4	1	3	6	4
G2	0	3	1	2	5	4	7

For G1, the donor columns are $2+0+4=6$ and $1+3=4$; for G2, they are $0+3+1=4$ and $2+5=7$. The expression matrix has two donor columns, while the separate cell-count diagnostic is three cells for A and two for B. These are two independent donors, not five replicates; adding cells changes the sums and sampling coverage, not the number of donors. This toy table illustrates aggregation only and is not a condition-testing design.

Inputs and outputs

Input: `annotated.rds` with joined, untransformed RNA counts, reviewed `celltype` labels and a sample sheet. Output: one cell type's DE result and fitted DESeq2 object, plus cell-count diagnostics. The companion command takes `<celltype> <numerator> <denominator>`; choose labels that actually occur in the reviewed object.

R

```r
# Run from the analysis root; SC_WORKFLOW points to this companion folder.
library(Seurat)
library(DESeq2)
source(file.path(Sys.getenv("SC_WORKFLOW"), "R/contracts.R"))
samples <- read_samples("config/samples.tsv")
so <- readRDS("results/objects/annotated.rds")
ct <- "T_cells"  # replace with a reviewed label present in this object
sub <- subset(so, cells = colnames(so)[so$celltype == ct])
raw <- LayerData(sub, assay = "RNA", layer = "counts")
pb <- aggregate_counts(raw, sub@meta.data, group = "sample")
meta <- align_samples(pb, samples, "treated", "control")
stopifnot(setequal(colnames(pb), samples$sample))

dds <- DESeqDataSetFromMatrix(as.matrix(pb), meta, design = ~ condition)
dds <- dds[rowSums(counts(dds) >= 10) >= 2, ]
dds <- DESeq(dds)
res <- results(dds, contrast = c("condition", "treated", "control"))
```

The shared aggregation helper preserves sample identifiers literally and checks metadata order and count conservation. It avoids reconstructing sample metadata from compound column names: Seurat's `AggregateExpression` also sums counts, but its generated group names can change underscores to dashes.

Never infer condition from a sample-name prefix. Dense conversion above is of the small pseudobulk matrix, not the full cells-by-genes data. The companion DE table includes explicit stable `gene_id`, original `gene_symbol` and object `row_name` columns from the retained feature map; display symbols alone are not stable mapping identifiers.

muscat provides an alternative multi-cell-type workflow through `aggregateData` and `pbDS` [38]. Use the same experimental units, raw-count requirement and design checks when changing the aggregation interface.

PYTHON

```python
# Demonstration after reviewed labels have been added to the full-gene object.
import os
import sys
from pathlib import Path
import numpy as np
from scipy import sparse
import scanpy as sc
sys.path.insert(0, str(Path(os.environ["SC_WORKFLOW"]) / "py"))
from workflow_contracts import read_samples, align_metadata, check_cell_metadata
samples = read_samples()
check_cell_metadata(adata.obs, samples)

assert "counts" in adata.layers
pb = sc.get.aggregate(
    adata, by=["sample", "celltype"], func="sum", layer="counts",
)
pb_ct = pb[pb.obs["celltype"] == "T_cells"].copy()
pb_ct.X = pb_ct.layers["sum"]
values = pb_ct.X.data if sparse.issparse(pb_ct.X) else np.asarray(pb_ct.X)
assert np.isfinite(values).all() and (values >= 0).all()
assert (values == np.floor(values)).all()
# R count matrices use genes as rows and samples as columns.
counts_table = pb_ct.to_df().T
counts_table.columns = pb_ct.obs["sample"].to_numpy()
counts_table.to_csv("pseudobulk_T_cells_counts.csv")
meta = align_metadata(pb_ct.obs["sample"].tolist(), samples)
assert counts_table.columns.tolist() == meta.index.tolist()
meta.to_csv("pseudobulk_T_cells_metadata.csv", index_label="sample_id")
```

This Python block prepares an export; it does not fit a condition model. Check that the exported sample columns and metadata rows match exactly before using DESeq2, edgeR or a separately verified PyDESeq2 workflow. The companion Python script stops at descriptive cluster markers and does not supply reviewed cell-type labels or condition-level DE automatically.

Key parameters

Plan enough independent donors for the expected effect and heterogeneity; three or more per condition is a practical starting recommendation, not a power calculation or a guarantee. The companion requires at least two retained independent samples per condition and requires the selected type to be present in every listed sample. Inspect cells and library size per sample/type before fitting. The example count filter is illustrative. A paired design or a design with technical batches needs a suitable full-rank model, not automatic reuse of `~condition`.

Eligibility and diagnostics. No captured cells of a type means its expression was not observed in that sample; it is not an all-zero expression profile. Do not insert a zero library or silently remove that sample. Investigate recovery, QC and annotation, then define sample/type eligibility without selecting on the desired DE result. Among eligible samples, inspect pseudobulk library sizes, transformed sample-level PCA, count profiles and influential donors. A fine label can still mix different substates across conditions; its DE then includes changes in that within-label mixture [2, 38].

12.2 Why pooled per-cell tests fail for condition inference

A Wilcoxon or t-test that pools all cells by condition while ignoring their samples treats correlated observations as independent. It underestimates the variation relevant to a donor-level question and can produce many false discoveries [37]. Adding more cells from the same few donors cannot replace independent donors. Descriptive cluster-marker testing answers a different question and does not establish a condition effect (Section 11).

Watch out for

Use sample-aware inference: pseudobulk with an appropriate design, or a justified mixed model that represents the repeated observations. Technical libraries from the same donor must not be counted as independent biological replicates. No model can separate a condition effect from a technical effect that is completely confounded with it.

12.3 Differential abundance: relative composition

A condition may change the proportion of captured cells assigned to a type even when expression within that type does not change. Propeller computes per-sample proportions, transforms them and tests group differences using a moderated linear-model framework [42]. Milo instead tests cell counts in overlapping neighbourhoods of a kNN graph, with sample-level design and normalisation [41]. These are alternative resolutions of the abundance question, not interchangeable tests of an absolute number of cells.

A denominator example. In an invented recovered-cell sample, 20 X cells and 80 Y cells give shares of $20/100 = 20\%$ and $80/100 = 80\%$. In another sample, 40 X cells and 80 Y cells give $40/120 \approx 33.3\%$ and $80/120 \approx 66.7\%$. Y's recovered count is unchanged, yet its share falls because the denominator grows. These are cell numbers, not summed gene UMIs; neither example establishes absolute tissue abundance or a condition effect.

R

```
library(speckle)
out <- propeller(clusters = so$celltype, sample = so$sample,
                 group = so$condition, transform = "logit")
```

Watch out for

Proportions sum to one: an increase in one population can lower other populations' relative proportions even if their absolute counts do not change. Transformation does not remove this dependence or establish absolute abundance. Interpret propeller and Milo relative to their denominator and sampling assumptions; dissociation, capture efficiency and unequal cell recovery can also change the observed composition. Include independent samples and inspect per-sample proportions before interpreting a group test.

Optional Functional enrichment of a pseudobulk gene list can use over-representation analysis or a ranked-list method with clusterProfiler [48]. Define the tested gene universe from the genes eligible for that cell type's differential analysis, preserve identifier mappings, and report selection and multiple-testing rules. This optional step is not implemented by the core single-cell scripts.

13 Trajectory & RNA velocity **Optional**

These analyses require an additional biological question about a continuous or dynamic process. They are not automatic extensions of a treated-versus-control comparison. A static collection of cell types need not contain the intermediate states, temporal information or sampling structure needed to infer a lineage. Keep the original cell identities, donors and condition labels available when exploring a proposed process.

13.1 Pseudotime and trajectory inference

Trajectory methods infer a path or graph through expression states and assign a relative ordering. Slingshot combines cluster-level lineage structure with simultaneous principal curves [43]; Monocle3 fits a principal graph [44]; PAGA summarises neighbourhood connectivity between groups [45]. Connectivity can be consistent with a continuum but does not independently demonstrate differentiation, direction or ancestry.

R

```r
# Demonstration: choose a biologically justified subset and root first.
library(slingshot)
sce <- as.SingleCellExperiment(so)
root_cluster <- "0"  # replace using independent progenitor-state evidence
stopifnot(root_cluster %in% sce$seurat_clusters)
sce <- slingshot(sce, clusterLabels = "seurat_clusters",
                 reducedDim = "PCA", start.clus = root_cluster)
pseudotime <- slingPseudotime(sce)
```

PYTHON

```python
# Topology exploration on the previously selected neighbour graph.
sc.tl.paga(adata, groups="celltype")
sc.pl.paga(adata, threshold=0.05)
# Optional graph layout for display, not evidence of temporal direction.
sc.tl.draw_graph(adata, init_pos="paga", random_state=42)
```

Key parameters

Choose cells, features, reduction, clustering and root using the biological question, then assess sensitivity to those choices. A star-shaped graph may represent branching; a disconnected graph may reflect sampling or analysis choices. Neither shape alone decides whether a biological trajectory exists. Prefer a justified high-dimensional representation for inference; the apparent geometry of a UMAP can distort distances. Pseudotime is not elapsed time and has no inherent direction without rooting evidence.

13.2 RNA velocity

RNA velocity uses spliced and unspliced RNA with a kinetic model to estimate the direction and rate of local expression change [47, 46]. An unspliced/spliced ratio is informative only relative to the fitted relationship and model assumptions; a high ratio alone does not prove that a gene or lineage is moving in a particular direction. USA-mode quantification or a compatible velocyto workflow can supply the required layers.

PYTHON

```python
# Demonstration after exact barcode/gene harmonisation and UMAP construction.
import scvelo as scv
velocity = sc.read_loom("sample1_velocyto.loom")
assert adata.obs_names.isin(velocity.obs_names).all()
assert adata.var_names.isin(velocity.var_names).all()
matched = velocity[adata.obs_names, adata.var_names]
adata_v = adata.copy()
for layer in ("spliced", "unspliced"):
    adata_v.layers[layer] = matched.layers[layer].copy()
adata_v.X = adata_v.layers["spliced"].copy()
scv.pp.filter_and_normalize(adata_v, min_shared_counts=20, n_top_genes=2000)
scv.pp.moments(adata_v, n_pcs=30, n_neighbors=30)
scv.tl.recover_dynamics(adata_v, n_jobs=1)
scv.tl.velocity(adata_v, mode="dynamical")
scv.tl.velocity_graph(adata_v, n_jobs=1)
scv.pl.velocity_embedding_stream(adata_v, basis="umap")
```

This example preserves the existing object's UMAP and metadata while adding aligned velocity layers. The strict identifier checks deliberately stop if a loom file uses incompatible barcode prefixes, gene keys or a different feature set; establish an explicit one-to-one mapping before proceeding. Do not silently replace the processed object with an unrelated loom object and then plot an embedding that it does not contain. Large dynamical fits need a measured memory/CPU budget; serial workers are an explicit starting allocation.

Watch out for

The dynamical model imposes kinetic assumptions, including simple transcriptional regimes and splicing/degradation rates. Complex regulation, inadequate sampling, weak kinetic signal or preprocessing can violate them. Inspect gene-level fits and sensitivity, not only projected arrows. Treat direction as a model-based hypothesis and seek independent time-course, perturbation, lineage or marker evidence before making a biological lineage claim. Agreement with a pseudotime derived from the same data is not fully independent validation.

14 Figures for single-cell

A figure's interpretation depends on the expression representation and cell annotations it displays. Build figures from the reviewed object and record both choices. Integration is optional; a figure should not imply that technical correction was compulsory or biologically harmless. Export a raster PNG and vector SVG at explicit physical dimensions, then inspect both at the intended reading size. Code alone does not establish legibility, correct labels or preserved scientific meaning.

R

```r
library(ggplot2)
# Reuse the project's figure theme and font configuration.
save_pair <- function(plot, stem, width, height) {
  ggsave(paste0(stem, ".png"), plot, width = width, height = height,
         units = "in", dpi = 300)
  ggsave(paste0(stem, ".svg"), plot, width = width, height = height,
         units = "in", device = svglite::svglite)
}
```

Use Times New Roman with opaque black text for standalone figures. The Computer Modern/Latin Modern exception applies to the LaTeX note and its compiled PDF. Purposeful colour may encode cell types, conditions or expression. Keep category mappings consistent across panels, avoid colour as the only identifying cue where labels or shapes can help, and place captions outside plots rather than duplicating

them as overall titles. Retain necessary gene names and panel identifiers. The companion exports three core figure pairs; additional examples below require their named packages and inputs.

14.1 UMAP by cell type and condition

Show the same coordinates coloured by reviewed cell type and by condition so the two displays can be compared. UMAP summarises a chosen representation; visual proximity or apparent mixing does not by itself demonstrate a condition effect, batch correction success or lineage relationship.

R

```r
p1 <- DimPlot(so, group.by = "celltype", label = TRUE, repel = TRUE) +
  scale_colour_viridis_d() + labs(title = NULL)
p2 <- DimPlot(so, group.by = "condition") +
  scale_colour_viridis_d() + labs(title = NULL)
save_pair(p1 + p2, "umap_celltype_condition", width = 11, height = 5)
```

PYTHON

```python
fig = sc.pl.umap(adata, color=["celltype", "condition"], palette="viridis",
                 wspace=0.4, return_fig=True)
fig.set_size_inches(11, 5)
fig.savefig("umap_celltype_condition.png", dpi=300, bbox_inches="tight")
fig.savefig("umap_celltype_condition.svg", bbox_inches="tight")
```

Label cell types explicitly once reviewed; numeric cluster IDs can change across reruns. Check that rare populations remain visible, label repulsion has not hidden points or text, and colour categories match the stored annotation.

14.2 Marker feature plots

Feature plots show an expression representation on the embedding. State whether values are log-normalised expression or another quantity; use a common scale when comparing expression of the same gene across groups, and avoid comparing unrelated genes through independently scaled colours.

R

```r
DefaultAssay(so) <- "RNA"  # log-normalized RNA data for expression display
p <- FeaturePlot(so, features = c("CD3D", "MS4A1", "LYZ"), ncol = 3,
                 cols = c("lightgrey", "#440154"), order = TRUE)
p <- p & theme(plot.title = element_text(colour = "black"))  # gene panel labels
save_pair(p, "feature_markers", width = 11, height = 4)
```

PYTHON

```python
fig = sc.pl.umap(adata, color=["CD3D", "MS4A1", "LYZ"], ncols=3,
                 cmap="viridis", vmin=0, use_raw=False, return_fig=True)
fig.set_size_inches(11, 4)
fig.savefig("feature_markers.png", dpi=300, bbox_inches="tight")
fig.savefig("feature_markers.svg", bbox_inches="tight")
```

14.3 Dot plots and violins of canonical markers

Dot size can encode detection frequency and colour average expression. Confirm the plotting function's exact averaging/scaling convention; a scaled colour is not an absolute expression measurement. A violin plot can additionally show the distribution. These descriptive summaries support annotation and do not replace a sample-level condition model.

R

```
markers <- c("CD3D", "CD8A", "MS4A1", "LYZ", "NKG7", "PPBP")
p <- DotPlot(so, assay = "RNA", features = markers, group.by = "celltype") +
  scale_colour_viridis_c() + RotatedAxis() + labs(title = NULL)
save_pair(p, "dotplot_markers", width = 8, height = 5)
```

PYTHON

```
markers = ["CD3D", "CD8A", "MS4A1", "LYZ", "NKG7", "PPBP"]
dot = sc.pl.dotplot(adata, var_names=markers, groupby="celltype",
                    cmap="viridis", return_fig=True)
dot.savefig("dotplot_markers.png", dpi=300, bbox_inches="tight")
dot.savefig("dotplot_markers.svg", bbox_inches="tight")
```

14.4 Cell-type proportions across samples

Plot each independent sample rather than pooling all cells within condition. A stacked bar describes the relative captured composition; it does not establish absolute tissue abundance. Consider showing sample-wise points per cell type alongside the overview when evaluating a group comparison.

R

```
prop <- so@meta.data |> dplyr::count(sample, condition, celltype)
p <- ggplot(prop, aes(x = sample, y = n, fill = celltype)) +
  geom_col(position = "fill") + facet_wrap(~condition, scales = "free_x") +
  scale_fill_viridis_d() + labs(x = NULL, y = "Proportion of captured cells")
save_pair(p, "sample_composition", width = 9, height = 5)
```

PYTHON

```
prop = (adata.obs.groupby(["sample", "celltype"], observed=True).size()
        .groupby(level=0).transform(lambda x: x / x.sum()).rename("prop").reset_index())
ax = prop.pivot(index="sample", columns="celltype", values="prop").fillna(0).plot(
    kind="bar", stacked=True, colormap="viridis", figsize=(9, 5))
ax.set_ylabel("Proportion of captured cells")
ax.figure.savefig("sample_composition.png", dpi=300, bbox_inches="tight")
ax.figure.savefig("sample_composition.svg", bbox_inches="tight")
```

14.5 Pseudobulk DE: volcano and heatmap

Use the fitted sample-level model for one cell type and an explicit contrast. Report the multiple-testing and effect-size selection rules. A volcano plot's fold-change estimates and adjusted p-values have different roles; zero, missing or non-finite values need an explicit plotting treatment rather than silent clipping.

R

```
# dds is the saved DESeq2 object for this cell type.
res_df <- as.data.frame(results(dds, contrast = c("condition", "treated", "control"))) |>
  tibble::rownames_to_column("feature") |>
  dplyr::mutate(sig = !is.na(padj) & padj < 0.05 & abs(log2FoldChange) > 1)
plot_df <- dplyr::filter(res_df, is.finite(log2FoldChange), is.finite(padj), padj > 0)
p <- ggplot(plot_df, aes(log2FoldChange, -log10(padj), colour = sig)) +
  geom_point(size = 0.8) + scale_colour_manual(values = c("grey70", "#440154")) +
  ggrepel::geom_text_repel(data = dplyr::filter(plot_df, sig), aes(label = feature),
                           max.overlaps = 15, size = 3) +
  labs(x = "log2 fold change (treated / control)", y = "-log10 adjusted p-value")
save_pair(p, "pseudobulk_volcano", width = 7, height = 5)

vsd <- DESeq2::varianceStabilizingTransformation(dds, blind = FALSE)
top_genes <- res_df |> dplyr::filter(sig) |> dplyr::slice_min(padj, n = 30) |>
  dplyr::pull(feature)
if (length(top_genes) >= 2) {
  mat <- assay(vsd)[top_genes, , drop = FALSE]
  mat <- mat[apply(mat, 1, sd) > 0, , drop = FALSE]
  if (nrow(mat) >= 2) {
    ht <- pheatmap::pheatmap(mat, scale = "row", silent = TRUE,
      annotation_col = as.data.frame(colData(dds)[, "condition", drop = FALSE]))
    save_pair(ht$gtable, "pseudobulk_heatmap", width = 7, height = 8)
  }
}
```

Document any excluded zero/non-finite adjusted p-values. Heatmap row scaling displays relative sample patterns per gene, not between-gene abundance; identify the transformation and selection rule in the caption. If too few genes meet the rule, do not manufacture a significant-gene heatmap. Exported DE tables retain stable gene IDs for annotation; use the validated feature map when replacing object keys with readable labels. Inspect all final raster/vector outputs for clipping, overlap and font substitution before sharing them.

15 Supplementary

Multi-sample processing, nuclei and larger data sets require choices that affect several phases of the workflow. The following sections explain those choices, together with reproducibility and troubleshooting, while retaining the teaching example's independent-donor assumptions.

15.1 Multi-sample workflow

Use the sample sheet to connect library, condition, donor and capture. Run counting per library, QC/doublet detection by capture and condition inference by independent experimental unit. Those units coincide only in the running one-library/one-capture-per-donor example. Technical resequencing lanes are not new donors; repeated donors require a suitable model rather than a duplicate row treated as independent.

Shell

```
# Explicit illustrative library list; use actual IDs from the reviewed sheet.
for s in control_rep1 control_rep2 control_rep3 treated_rep1 treated_rep2 treated_rep3; do
  bash "$SC_WORKFLOW/scripts/01_count_matrix/01_cellranger_count.sh" "$s"
done
```

The companion QC phase processes each sample before merging and retains all sample metadata. The following snippets instead demonstrate combining already processed per-sample objects; they do not perform the missing QC/doublet steps on newly loaded matrices.

R

```r
# Each file is a saved, QC'd singlet object with sample metadata.
samples <- read.delim("config/samples.tsv", colClasses = "character")
obj_list <- lapply(samples$sample, function(s) {
  readRDS(file.path("results/per_sample", paste0(s, ".rds")))
})
so <- merge(obj_list[[1]], obj_list[-1], add.cell.ids = samples$sample)
# Seurat v5 merge retains per-input layers; avoid splitting them again.
```

PYTHON

```python
import anndata as ad
samples = read_samples()
adatas = {s: ad.read_h5ad(f"results/per_sample/{s}.h5ad") for s in samples.index}
# Require identical gene IDs, ordering and processing before concatenation.
reference = next(iter(adatas.values())).var
assert all(a.var.equals(reference) for a in adatas.values())
adata = ad.concat(adatas, label="sample_key", index_unique="-", merge="same")
```

Watch out for

A merged object can be processed capture-wise when a method explicitly receives capture labels. What fails is treating all merged cells as one loading or treating every technical library as an independent donor. Verify the sample/cell relationships before fitting any condition model.

15.2 Single-nucleus RNA-seq

Nuclei can enable analysis of frozen or difficult-to-dissociate tissues and often contain substantial intronic RNA. Use an assay-appropriate reference and intron-counting policy consistently across samples. Cell Ranger includes intronic reads by default from version 7; verify the actual setting and library type [7].

Nuclear and whole-cell preparations have different RNA distributions, detection patterns and contamination mechanisms. Reassess nCount/nFeature distributions and relevant quality metrics rather than copying whole-cell thresholds. Mitochondrial fractions are often low in nuclei; unexpectedly high values can still reveal contamination or preparation issues, so low expected baseline does not make the metric universally useless. Assess ambient RNA and the effect of any correction instead of applying one correction routinely to every data set.

15.3 Scale to larger data sets

Keep count matrices sparse and account for dense intermediates during scaling, model fitting and conversion. For larger data, BPCells can supply on-disk Seurat matrices, and HDF5-backed Bioconductor representations can defer loading. AnnData backed mode limits some in-memory loading, but it does not make every Scanpy operation backed-compatible [3].

R

```r
# Optional: a pre-existing BPCells matrix directory, not a 10x directory.
library(BPCells)
mat <- open_matrix_dir("bpcells_matrix_dir")
so <- CreateSeuratObject(mat)
```

PYTHON

```python
adata_backed = sc.read_h5ad("all_samples.h5ad", backed="r")
# Inspect metadata; materialise only a planned subset when RAM permits.
# adata_small = adata_backed[selected_cells, selected_genes].to_memory()
adata_backed.file.close()
```

Choose supported operations for the selected backend/version. Do not assume that standard Scanpy QC calculations or normalisation calls stream through a backed matrix. Measure memory and use a verified chunked workflow where required. Approximate-neighbour methods can reduce compute cost, but their availability and scaling depend on the selected implementation and data.

15.4 Reproducibility

Record inputs, reference/annotation, chemistry, sample relationships, retained-cell decisions, parameters and actual environment versions. Set seeds near stochastic steps, including doublet simulation, PCA initialisation, neighbour search where applicable, clustering, UMAP and model training [2]. A seed is part of a reproducibility record, not a guarantee across implementations.

R

```r
set.seed(42)
so <- RunUMAP(so, reduction = reduction, dims = dims, seed.use = 42)
set.seed(42)
so <- FindClusters(so, resolution = 0.5, algorithm = 4, random.seed = 42)
sessionInfo()
```

PYTHON

```python
from importlib.metadata import version
sc.tl.umap(adata, random_state=42)
sc.tl.leiden(adata, resolution=0.5, random_state=42,
             flavor="igraph", directed=False, n_iterations=2)
print({p: version(p) for p in ("scanpy", "anndata", "numpy")})
```

The companion exports session records per R phase and Python package versions. The environment export script requires the named conda environment and activated analysis interpreter; it fails visibly if export fails. Archive the solved environment and platform locally, and verify restoration before describing it as a portable lock. No full biological reproduction is established by parsing scripts or by passing small synthetic format/count checks.

15.5 Troubleshooting

Symptom	**Possible causes**	**Check before changing analysis**
Samples separate on UMAP	Technical effects, real biology, population differences or QC	Inspect design, markers and uncorrected PCs; evaluate optional integration, Section 10
Clusters track mitochondrial fraction	Damaged cells, preparation or real tissue/state variation	Verify symbols and per-sample metrics; review filters, Section 8
Many condition-associated genes	Strong biology, confounding, pseudoreplication or count-input errors	Check independent units, raw counts, metadata and model, Section 12
Little visible structure	Feature/PC/graph choices, QC or a genuine continuum	Inspect diagnostics rather than forcing clusters, Sections 9–11
Mixed-marker populations	Doublets, contamination, mixed clusters or real states	Compare scores, markers and recurrence across samples, Section 8

References

[1] Luecken MD, Theis FJ. Current best practices in single-cell RNA-seq analysis: a tutorial. *Mol Syst Biol.* 2019;15(6):e8746.

[2] Heumos L, Schaar AC, Lance C, et al. Best practices for single-cell analysis across modalities. *Nat Rev Genet.* 2023;24(8):550–572.

[3] Amezquita RA, Lun ATL, Becht E, et al. Orchestrating single-cell analysis with Bioconductor. *Nat Methods.* 2020;17(2):137–145.

[4] Zheng GXY, Terry JM, Belgrader P, et al. Massively parallel digital transcriptional profiling of single cells. *Nat Commun.* 2017;8:14049.

[5] Macosko EZ, Basu A, Satija R, et al. Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets. *Cell.* 2015;161(5):1202–1214.

[6] Klein AM, Mazutis L, Akartuna I, et al. Droplet barcoding for single-cell transcriptomics applied to embryonic stem cells. *Cell.* 2015;161(5):1187–1201.

[7] 10x Genomics. *Cell Ranger.* Software; 2024. `https://www.10xgenomics.com/support/software/cell-ranger`

[8] Kaminow B, Yunusov D, Dobin A. *STARsolo: accurate, fast and versatile mapping/quantification of single-cell and single-nucleus RNA-seq data.* bioRxiv. 2021. doi:10.1101/2021.05.05.442755

[9] He D, Zakeri M, Sarkar H, Soneson C, Srivastava A, Patro R. Alevin-fry unlocks rapid, accurate and memory-frugal quantification of single-cell RNA-seq data. *Nat Methods.* 2022;19(3):316–322.

[10] Lun ATL, Riesenfeld S, Andrews T, et al. EmptyDrops: distinguishing cells from empty droplets in droplet-based single-cell RNA sequencing data. *Genome Biol.* 2019;20:63.

[11] Young MD, Behjati S. SoupX removes ambient RNA contamination from droplet-based single-cell RNA sequencing data. *GigaScience.* 2020;9(12):giaa151.

[12] Yang S, Corbett SE, Koga Y, et al. Decontamination of ambient RNA in single-cell RNA-seq with DecontX. *Genome Biol.* 2020;21:57.

[13] Lun ATL, McCarthy DJ, Marioni JC. A step-by-step workflow for low-level analysis of single-cell RNA-seq data with Bioconductor. *F1000Research.* 2016;5:2122.

[14] Lun ATL, Bach K, Marioni JC. Pooling across cells to normalize single-cell RNA sequencing data with many zero counts. *Genome Biol.* 2016;17:75.

[15] McCarthy DJ, Campbell KR, Lun ATL, Wills QF. Scater: pre-processing, quality control, normalization and visualization of single-cell RNA-seq data in R. *Bioinformatics.* 2017;33(8):1179–1186.

[16] Germain PL, Lun A, Garcia Meixide C, Macnair W, Robinson MD. Doublet identification in single-cell sequencing data using scDblFinder. *F1000Research.* 2021;10:979.

[17] McGinnis CS, Murrow LM, Gartner ZJ. DoubletFinder: doublet detection in single-cell RNA sequencing data using artificial nearest neighbors. *Cell Syst.* 2019;8(4):329–337.e4.

[18] Satija R, Farrell JA, Gennert D, Schier AF, Regev A. Spatial reconstruction of single-cell gene expression data. *Nat Biotechnol.* 2015;33(5):495–502.

[19] Stuart T, Butler A, Hoffman P, et al. Comprehensive integration of single-cell data. *Cell.* 2019;177(7):1888–1902.e21.

[20] Hao Y, Stuart T, Kowalski MH, et al. Dictionary learning for integrative, multimodal and scalable single-cell analysis. *Nat Biotechnol.* 2024;42(2):293–304.

[21] Wolf FA, Angerer P, Theis FJ. SCANPY: large-scale single-cell gene expression data analysis. *Genome Biol.* 2018;19:15.

[22] Virshup I, Rybakov S, Theis FJ, Angerer P, Wolf FA. anndata: Access and store annotated data matrices. *J Open Source Softw.* 2024;9(101):4371.

[23] Hafemeister C, Satija R. Normalization and variance stabilization of single-cell RNA-seq data using regularized negative binomial regression. *Genome Biol.* 2019;20:296.

[24] Korsunsky I, Millard N, Fan J, et al. Fast, sensitive and accurate integration of single-cell data with Harmony. *Nat Methods.* 2019;16(12):1289–1296.

[25] Haghverdi L, Lun ATL, Morgan MD, Marioni JC. Batch effects in single-cell RNA-sequencing data are corrected by matching mutual nearest neighbors. *Nat Biotechnol.* 2018;36(5):421–427.

[26] Lopez R, Regier J, Cole MB, Jordan MI, Yosef N. Deep generative modeling for single-cell transcriptomics. *Nat Methods.* 2018;15(12):1053–1058.

[27] Gayoso A, Lopez R, Xing G, et al. A Python library for probabilistic analysis of single-cell omics data. *Nat Biotechnol.* 2022;40(2):163–166.

[28] Luecken MD, Büttner M, Chaichoompu K, et al. Benchmarking atlas-level data integration in single-cell genomics. *Nat Methods.* 2022;19(1):41–50.

[29] McInnes L, Healy J, Melville J. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv:1802.03426; 2018.

[30] Van der Maaten L, Hinton G. Visualizing data using t-SNE. *J Mach Learn Res.* 2008;9:2579–2605.

[31] Traag VA, Waltman L, van Eck NJ. From Louvain to Leiden: guaranteeing well-connected communities. *Sci Rep.* 2019;9:5233.

[32] Blondel VD, Guillaume JL, Lambiotte R, Lefebvre E. Fast unfolding of communities in large networks. *J Stat Mech Theory Exp.* 2008;2008(10):P10008.

[33] Korsunsky I, Nathan A, Millard N, Raychaudhuri S. *Presto scales Wilcoxon and auROC analyses to millions of observations*. bioRxiv. 2019. doi:10.1101/653253

[34] Aran D, Looney AP, Liu L, et al. Reference-based analysis of lung single-cell sequencing reveals a transitional profibrotic macrophage. *Nat Immunol.* 2019;20(2):163–172.

[35] Hao Y, Hao S, Andersen-Nissen E, et al. Integrated analysis of multimodal single-cell data. *Cell.* 2021;184(13):3573–3587.

[36] Domínguez Conde C, Xu C, Jarvis LB, et al. Cross-tissue immune cell analysis reveals tissue-specific features in humans. *Science.* 2022;376(6594):eabl5197.

[37] Squair JW, Gautier M, Kathe C, et al. Confronting false discoveries in single-cell differential expression. *Nat Commun.* 2021;12:5692.

[38] Crowell HL, Soneson C, Germain PL, et al. muscat detects subpopulation-specific state transitions from multi-sample multi-condition single-cell transcriptomics data. *Nat Commun.* 2020;11:6077.

[39] Love MI, Huber W, Anders S. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. *Genome Biol.* 2014;15(12):550.

[40] Robinson MD, McCarthy DJ, Smyth GK. edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics.* 2010;26(1):139–140.

[41] Dann E, Henderson NC, Teichmann SA, Morgan MD, Marioni JC. Differential abundance testing on single-cell data using k-nearest neighbor graphs. *Nat Biotechnol.* 2022;40(2):245–253.

[42] Phipson B, Sim CB, Porrello ER, Hewitt AW, Powell J, Oshlack A. propeller: testing for differences in cell type proportions in single cell data. *Bioinformatics.* 2022;38(20):4720–4726.

[43] Street K, Risso D, Fletcher RB, et al. Slingshot: cell lineage and pseudotime inference for single-cell transcriptomics. *BMC Genomics.* 2018;19:477.

[44] Cao J, Spielmann M, Qiu X, et al. The single-cell transcriptional landscape of mammalian organogenesis. *Nature.* 2019;566(7745):496–502.

[45] Wolf FA, Hamey FK, Plass M, et al. PAGA: graph abstraction reconciles clustering with trajectory inference through a topology preserving map of single cells. *Genome Biol.* 2019;20:59.

[46] Bergen V, Lange M, Peidli S, Wolf FA, Theis FJ. Generalizing RNA velocity to transient cell states through dynamical modeling. *Nat Biotechnol.* 2020;38(12):1408–1414.

[47] La Manno G, Soldatov R, Zeisel A, et al. RNA velocity of single cells. *Nature.* 2018;560(7719):494–498.

[48] Wu T, Hu E, Xu S, et al. clusterProfiler 4.0: A universal enrichment tool for interpreting omics data. *The Innovation.* 2021;2(3):100141.

[49] Wolock SL, Lopez R, Klein AM. Scrublet: Computational Identification of Cell Doublets in Single-Cell Transcriptomic Data. *Cell Syst.* 2019;8(4):281–291.e9. `https://doi.org/10.1016/j.cels.2018.11.005`

[50] Zappia L, Richter S, Ramírez-Suástegui C, et al. Feature selection methods affect the performance of scRNA-seq data integration and querying. *Nat Methods.* 2025;22:834–844. `https://doi.org/10.1038/s41592-025-02624-3`

[51] Yi C, Cheng J, Chen J, Liu W, Liu J, Li Y. Benchmarking deep learning methods for biologically conserved single-cell integration. *Genome Biol.* 2025;26:398. `https://doi.org/10.1186/s13059-025-03869-z`

[52] Li S, Lücken M, Marioni JC, Teichmann SA, He P. Toward informed batch correction for single-cell transcriptome integration. *Nat Comput Sci.* 2026;6:123–133. `https://doi.org/10.1038/s43588-025-00943-1`