12 Bioinformatics

R. Heller · Date: 2026-05-11 · Last changed: 2026-10-30

RNA-seq differential expression, pathway and GO enrichment, and single-cell RNA-seq. The chapter assumes basic familiarity with `Bioconductor` and points back to the regression and ML chapters for the underlying machinery.

This chapter contains **50 method pages** and **3 labs**. If you are not sure which method to read, return to Chapter 0 and follow the decision tree to the right node.

12.1 Method pages

Method	Source slug
Alignment with BWA and Bowtie	`bwa-bowtie-alignment`
ATAC-Seq Analysis	`atac-seq-analysis`
Batch Correction with ComBat	`batch-correction-combat`
Bulk RNA-seq Differential Expression with DESeq2	`bulk-rnaseq-differential-expression`
Cell-Type Annotation	`scrna-cell-type-annotation`
ChIP-Seq Analysis	`chip-seq-analysis`
Copy Number Variation Analysis	`cnv-analysis`
Counting Reads with featureCounts	`counting-reads-feature-counts`
Differential Expression with edgeR	`edger-differential-expression`
Differential Expression with limma-voom	`limma-voom`
DNA Methylation Analysis	`methylation-analysis`
Drug-Target Interaction Mining	`drug-target-interaction`
FASTQ Quality Control	`fastq-quality-control`
Finding Marker Genes	`scrna-marker-genes`
Gene Annotation with biomaRt	`biomart-annotation`
Gene Ontology Enrichment	`gene-ontology-enrichment`
GSEA Preranked Analysis	`gsea-preranked`
GSVA Single-Sample Enrichment	`gsva-single-sample`
Heatmaps for RNA-seq	`heatmaps-rnaseq`
Integration with Harmony	`scrna-integration-harmony`
KEGG Pathway Enrichment	`pathway-enrichment-kegg`
MA Plots	`ma-plots`
Metagenomic Profiling	`metagenomic-profiling`
Microbiome Analysis with DADA2	`microbiome-dada2`
Microbiome Diversity Metrics	`microbiome-diversity`

Method	Source slug
Multi-Omics Integration	`multiomics-integration`
Multiple Sequence Alignment	`msa-biostrings`
PCA of RNA-seq Samples	`pca-of-rnaseq`
Phylogenetic Trees with ape	`phylogenetic-trees-ape`
Population Genetics Basics	`population-genetics-basics`
Protein Structure Prediction	`protein-structure-prediction`
Proteomics with MSstats	`proteomics-msstats`
Pseudoalignment with Salmon and Kallisto	`salmon-kallisto-pseudoalignment`
Read Trimming and Adapter Removal	`read-trimming-adapters`
RNA-seq Normalisation	`rnaseq-normalization`
scRNA Clustering	`scrna-clustering`
scRNA Normalisation	`scrna-normalization`
scRNA QC and Filtering	`scrna-qc-filtering`
Sequence Alignment: Overview	`sequence-alignment-overview`
Single-Cell with Seurat	`single-cell-seurat-intro`
Spatial Transcriptomics	`spatial-transcriptomics`
STAR: Spliced Alignment	`star-spliced-alignment`
Structural Variant Detection	`structural-variant-detection`
Surrogate Variable Analysis (SVA)	`surrogate-variable-analysis-sva`
Trajectory Analysis	`scrna-trajectory-analysis`
Transcript-to-Gene Summarisation	`tx-to-gene-summarization`
Variant Annotation with VEP	`annotation-variant-vep`
Variant Calling with GATK	`variant-calling-gatk`
VCF Manipulation	`vcf-manipulation`
Volcano Plots	`volcano-plots`

12.2 Labs

Lab
RNA-seq with DESeq2 / edgeR
Enrichment analysis (GSEA / ORA)
scRNA-seq with Seurat

12.3 Variant Annotation with VEP

12.3.1 Introduction

Ensembl's Variant Effect Predictor (VEP) assigns each variant a predicted consequence (missense, synonymous, splice-donor, intronic, etc.), gene and transcript IDs, and pathogenicity scores from predictors (SIFT, PolyPhen, CADD, REVEL). It is the standard annotator for variant interpretation in clinical genetics and population genomics.

12.3.2 Prerequisites

VCF file; Ensembl or RefSeq transcript database.

12.3.3 Theory

VEP traverses each variant, intersects it with transcript structures, and emits one record per overlapping transcript with the predicted functional consequence. Gene-level summaries pick the most severe consequence across transcripts (MANE-select or canonical transcript is standard for clinical use).

12.3.4 Assumptions

Reference build matches the VCF (GRCh37 vs GRCh38); transcript database is current.

12.3.5 R Implementation

VEP runs on the command line (Perl). For programmatic annotation in R, use `VariantAnnotation::locateVariants()` against a `TxDb` plus `predictCoding()` for protein consequences.

```
library(VariantAnnotation)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(BSgenome.Hsapiens.UCSC.hg19)

vcf_file <- system.file("extdata", "chr22.vcf.gz", package = "VariantAnnotation")
vcf <- readVcf(vcf_file, "hg19")
seqlevelsStyle(vcf) <- "UCSC"

txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
loc  <- locateVariants(rowRanges(vcf), txdb, CodingVariants())
table(loc$LOCATION)

pc <- predictCoding(vcf, txdb, Hsapiens)
head(mcols(pc)[, c("CONSEQUENCE", "REFAA", "VARAA")])
```

A representative VEP command (not run here):

```
vep -i input.vcf -o output.vcf --cache --sift p --polyphen p --canonical
```

12.3.6 Output & Results

Per-variant, per-transcript annotation with consequence (missense_variant, synonymous_variant, etc.) and amino-acid substitution information.

12.3.7 Interpretation

“VEP annotation identified 12 missense variants predicted deleterious by SIFT and PolyPhen; one nonsense variant in BRCA1 was classified as likely pathogenic.”

12.3.8 Practical Tips

- Always check assembly/transcript version match between caller and annotator.
- For clinical work, use VEP’s `--pick` or `--flag_pick_allele_gene` to obtain a single canonical annotation per variant.
- Add pathogenicity plugins (CADD, REVEL, SpliceAI) for deleteriousness prediction.
- `ANNOVAR` is a competing annotator with a different gene-model approach; results can differ at splice boundaries.
- For batch annotation of millions of variants, VEP’s offline cache mode is essential; REST/web queries do not scale.

12.3.9 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.3.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.4 ATAC-Seq Analysis

12.4.1 Introduction

ATAC-seq (Assay for Transposase-Accessible Chromatin using sequencing) measures genome-wide chromatin accessibility by tagging accessible DNA regions with a hyperactive Tn5 transposase. Peaks in the resulting alignment density correspond to open chromatin — regulatory elements such as promoters and enhancers — and differential accessibility analysis between conditions reveals cell-state-specific regulatory changes. ATAC-seq has largely replaced DNase-seq for accessibility profiling because of its lower input requirement (10{,}000 cells vs millions) and simpler protocol.

12.4.2 Prerequisites

A working understanding of read alignment, BAM files, peak-calling concepts (typically MACS2), and regulatory genomics (promoters, enhancers, transcription factor binding).

12.4.3 Theory

The standard ATAC-seq pipeline:

1. **QC**: TSS enrichment score, fragment-length distribution with characteristic nucleosomal peaks (mono-, di-, tri-nucleosomal at approximately 150, 300, 450 bp).
2. **Duplicate and mitochondrial-read removal**: typical mitochondrial fraction is 20–50 % of reads and must be removed before peak calling.
3. **Peak calling**: MACS2 with Tn5-specific flags (`--nomodel --shift -75 --extsize 150`) treats each end of the fragment as a Tn5 cut site.
4. **Consensus peak set**: combine peaks across replicates and conditions for joint quantitative analysis.
5. **Differential accessibility**: DiffBind, edgeR, or DESeq2 on peak-level read counts.
6. **Peak annotation**: ChIPseeker assigns peaks to nearest gene and annotates promoter vs distal categories.

12.4.4 Assumptions

Tn5 tagmentation is approximately uniform across accessible regions; sequencing depth is sufficient (> 50 million reads per sample is typical); mitochondrial reads have been removed before peak calling.

12.4.5 R Implementation

```
library(ATACseqQC); library(ChIPseeker)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)

# Fragment-length diagnostic from an example ATAC bam shipped with ATACseqQC
bam ← system.file("extdata", "GL1.bam", package = "ATACseqQC")
# fragSize ← fragSizeDist(bam, bamFiles.labels = "example")

# Peak annotation example with a toy peak GRanges
set.seed(2026)
peaks ← GRanges(
  seqnames = rep("chr1", 50),
  ranges = IRanges(start = sort(sample(1e7:1.05e7, 50)),
                   width = 500)
)
anno ← annotatePeak(peaks,
                    TxDb = TxDb.Hsapiens.UCSC.hg19.knownGene,
                    level = "gene",
                    verbose = FALSE)
as.data.frame(anno)[1:5, c("seqnames", "start", "annotation", "geneId")]
```

12.4.6 Output & Results

`ATACseqQC::fragSizeDist()` produces the diagnostic fragment-size distribution; visible peaks at the nucleosomal lengths confirm successful tagmentation.
`ChIPseeker::annotatePeak()` assigns each peak to the nearest gene and categorises it as promoter, intronic, intergenic, etc.

12.4.7 Interpretation

A reporting sentence: “ATAC-seq on CD8 and regulatory T cells (n = 4 per group) identified 25{,}412 reproducible peaks at IDR < 0.05; ChIPseeker annotation showed 35 % promoter-proximal and 65 % distal-regulatory locations. Differential accessibility analysis identified 1{,}240 peaks at FDR < 0.05, enriched for AP-1 family motifs (Jun, Fos).” Always report TSS enrichment, peak counts, and the consensus peak strategy.

12.4.8 Practical Tips

- Remove duplicates and mitochondrial reads with `samtools markdup` and chromosome filtering before peak calling; mitochondrial reads dominate ATAC libraries and confound peak calling if retained.
- Check TSS enrichment score (computed by `ATACseqQC` or by `chrom.sizes`-aware tools); scores above 7 are typical for fresh tissue, scores above 4 are acceptable for frozen samples.
- For differential accessibility, define a consensus peak set across replicates (DiffBind-style) and quantify all samples on the same regions; calling peaks separately per sample produces unstable comparisons.
- Motif enrichment via `chromVAR` provides per-sample transcription-factor activity scores; pair with differential accessibility to identify TFs driving the observed changes.
- For single-cell ATAC-seq (scATAC-seq), use `Signac` (Seurat-flavoured) or `ArchR`; the workflow differs substantially from bulk ATAC.
- Use BlackList regions (ENCODE) to exclude artefact-prone genomic regions before downstream analysis.

12.4.9 R Packages Used

`ATACseqQC` for fragment-size distribution and TSS enrichment QC; `ChIPseeker` for peak annotation; `DiffBind` for differential-binding analysis; `chromVAR` for motif accessibility scores; `Signac` and `ArchR` for single-cell ATAC-seq; `Rsubread` and `GenomicAlignments` for upstream alignment and quantification.

12.4.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.4.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.5 Batch Correction with ComBat

12.5.1 Introduction

Batch effects — systematic technical variation introduced by different sequencing dates, instruments, library-prep operators, or reagent lots — bias gene-expression measurements in ways that can swamp biological signal if not addressed. ComBat is the most widely used batch-correction method, applying an empirical-Bayes adjustment to centre and scale each gene's distribution per batch. It is appropriate when batches are known and condition is not confounded with batch; the alternative — including batch as a covariate in differential-expression models — preserves more information when downstream analysis explicitly models batch.

12.5.2 Prerequisites

A working understanding of normalisation, the distinction between technical and biological variation, and basic empirical-Bayes shrinkage.

12.5.3 Theory

ComBat models each gene's expression as

$$Y_{ij} = \alpha_j + X\beta_j + \gamma_{bj} + \delta_{bj}\varepsilon_{ij},$$

with sample i, gene j, batch b, additive batch effect γ_{bj}, and multiplicative batch effect δ_{bj}. The empirical-Bayes step shrinks per-gene batch-effect estimates toward a common distribution across genes, stabilising estimation when batches contain few samples. The output is an expression matrix with batch effects removed but biological condition (specified via `mod`) preserved.

ComBat-Seq (in `sva`) is the count-data analogue, designed for raw RNA-seq counts before normalisation; standard ComBat assumes log-normalised input.

12.5.4 Assumptions

Batches are known; biological condition is not perfectly confounded with batch (otherwise correction is impossible); sufficient samples per batch for stable empirical-Bayes shrinkage.

12.5.5 R Implementation

```r
library(sva)

set.seed(2026)
# Simulated: 10 genes x 20 samples, 2 batches, 2 conditions
expr ← matrix(rnorm(10 * 20), 10, 20)
batch ← c(rep(1, 10), rep(2, 10))
condition ← rep(c("A", "B"), 10)

# Add batch effect
expr[, batch == 2] ← expr[, batch == 2] + 2

mod ← model.matrix(~ condition)
expr_corrected ← ComBat(dat = expr, batch = batch, mod = mod)

# PCA before/after
# prcomp(t(expr))$x[, 1:2]; prcomp(t(expr_corrected))$x[, 1:2]
```

12.5.6 Output & Results

`ComBat()` returns the batch-corrected expression matrix; `ComBat_seq()` returns batch-corrected counts. PCA before and after correction is the standard QC check: condition-driven clustering should remain or strengthen, while batch-driven separation should diminish substantially.

12.5.7 Interpretation

A reporting sentence: "ComBat correction with condition included as a covariate (`mod = model.matrix(~ condition)`) reduced batch variance from 35 % of PC1 to under 5 %, while preserving the condition-driven separation that explains 28 % of the remaining variance. The corrected matrix was used for downstream visualisation and clustering." Always report which covariates were preserved during correction.

12.5.8 Practical Tips

- Do *not* run ComBat when condition is fully confounded with batch — the correction will remove the biological signal along with the batch effect.
- Use `ComBat_seq()` for raw RNA-seq counts; standard ComBat is for log-normalised data.
- For differential-expression analysis, consider including batch as a covariate in the DE design (`design = ~ batch + condition` in DESeq2/edgeR/limma) rather than running ComBat first; the model-based approach is generally preferred.
- Always report batch-correction methods explicitly in publications; reviewers expect clarity about how batch was handled.
- Alternative methods include RUVSeq (latent factor extraction), svaseq (surrogate variable analysis), and `limma::removeBatchEffect()` (simple linear adjustment); the choice depends on whether batches are known.
- Pair correction with PCA before and after as a routine QC step; visible improvement is the validation for whether the correction worked.

12.5.9 R Packages Used

`sva::ComBat()` and `ComBat_seq()` for the canonical empirical-Bayes batch correction; `RUVSeq` for unsupervised factor-based correction; `limma::removeBatchEffect()` for simple linear batch removal; `harmony` for batch correction in single-cell RNA-seq workflows.

12.5.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.5.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.6 Gene Annotation with biomaRt

12.6.1 Introduction

`biomaRt` provides a programmatic interface to Ensembl BioMart for converting between gene identifier systems (Ensembl gene IDs, Entrez IDs, HGNC symbols, RefSeq, UniProt), retrieving genomic coordinates, transcript structures, GO term annotations, and orthologues across species. It is indispensable whenever a gene list — from differential expression, GWAS, or any other genomic analysis — needs to be enriched with curated annotation. The `biomaRt` workflow builds queries that filter, attribute, and merge in a single call, replacing dozens of manual lookups across web interfaces.

12.6.2 Prerequisites

A working understanding of gene identifier types (Ensembl vs Entrez vs HGNC symbol vs RefSeq), the difference between gene-level and transcript-level annotation, and basic familiarity with R data frames.

12.6.3 Theory

A `biomaRt` query targets a Mart (database) and dataset (species), specifying:

- **Filters**: which genes to query (input IDs and their type).
- **Attributes**: which annotations to return.
- **Values**: the actual identifier values.

The core call `getBM(attributes, filters, values, mart)` issues the query and returns a data frame. For reproducibility, pin the Ensembl release with `host = "<month> <year>.archive.ensembl.org"`; the default `host = "www.ensembl.org"` points to the current release, which changes over time.

`biomaRt` also supports cross-species queries via `getLDS()` (linked datasets) for orthologue lookup, transcript-level queries for splicing analyses, and SNP queries against Ensembl variation.

12.6.4 Assumptions

Internet connectivity to the Ensembl BioMart server (rare downtime can interrupt workflows); input identifiers match the chosen Mart's species and identifier type.

12.6.5 R Implementation

```
library(biomaRt)

# Illustrative (requires internet; swap to offline cache in production):
# mart ← useMart("ensembl", dataset = "hsapiens_gene_ensembl",
#                 host = "www.ensembl.org")
#
# genes_symbol ← c("TP53", "BRCA1", "EGFR")
# annot ← getBM(
#   attributes = c("hgnc_symbol", "ensembl_gene_id",
#                  "entrezgene_id", "chromosome_name",
#                  "start_position", "end_position"),
#   filters    = "hgnc_symbol",
#   values     = genes_symbol,
#   mart       = mart
# )
# annot

# Simulated equivalent for offline illustration
set.seed(2026)
annot ← data.frame(
  hgnc_symbol      = c("TP53", "BRCA1", "EGFR"),
  ensembl_gene_id  = c("ENSG00000141510", "ENSG00000012048", "ENSG00000146648"),
  entrezgene_id    = c(7157, 672, 1956),
  chromosome_name  = c("17", "17", "7"),
  start_position   = c(7668402, 43044295, 55019032),
  end_position     = c(7687550, 43125483, 55211628)
)
annot
```

12.6.6 Output & Results

A data frame mapping the input identifiers to all requested attributes. In practice, the output is merged with differential-expression results or other gene-level tables to attach genomic coordinates, alternative identifiers, and functional annotation.

12.6.7 Interpretation

A reporting sentence: “Gene annotation was retrieved from Ensembl release 110 via biomaRt with `host = 'jul2023.archive.ensembl.org'` for reproducibility; TP53 was localised to chr17:7{,}668{,}402–7{,}687{,}550, with Entrez ID 7157 for downstream KEGG/Reactome pathway analyses.” Always pin the Ensembl release version when attaching annotation to published results.

12.6.8 Practical Tips

- Pin the release host (`host = "jul2023.archive.ensembl.org"`) for reproducibility; the default points to the current release, which silently drifts over time.
- `biomaRt` queries can time out for large requests; split into chunks of a few thousand IDs and combine the results.
- For high-throughput annotation (re-running annotations on large gene sets), cache the results locally; re-querying 50{,}000 genes is slow.
- For offline workflows, use `org.Hs.eg.db` (and species-specific equivalents) or `AnnotationHub` for pre-packaged annotation.
- For cross-species orthologue lookup, `getLDS()` queries the homology Mart; this is the standard approach for translating findings between human and mouse studies.
- When the BioMart server is down, fall back to `AnnotationHub` or local packages; never let a transient network failure block downstream analysis.

12.6.9 R Packages Used

`biomaRt` for Ensembl BioMart queries; `org.Hs.eg.db` and species-specific equivalents for offline annotation; `AnnotationHub` for pre-curated annotation packages; `ensembldb` for offline Ensembl annotation in SQLite form; `tximeta` for metadata-aware annotation in RNA-seq workflows.

12.6.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.6.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.7 Bulk RNA-seq Differential Expression with DESeq2

12.7.1 Introduction

Bulk RNA-seq asks which genes change in expression between conditions. DESeq2 is the most widely used framework for the analysis: it models counts with a negative binomial distribution, shares dispersion information across genes to stabilise estimates in small samples, and tests each gene's log-fold change for significance with multiple-testing correction. A standard DESeq2 workflow goes from a raw count matrix to a list of differentially expressed genes in a dozen lines of R.

12.7.2 Prerequisites

The reader should understand what a count matrix is (rows = genes, columns = samples), the basic biology of RNA-seq (reads mapped to transcripts, summed per gene), and the idea of a generalised linear model. Familiarity with Bioconductor data structures (`SummarizedExperiment`) is helpful.

12.7.3 Theory

DESeq2 assumes that the read count K_{ij} for gene i in sample j follows a negative binomial distribution

$$K_{ij} \sim \mathrm{NB}(\mathrm{mean} = \mu_{ij},\ \mathrm{dispersion} = \alpha_i),$$

where $\mu_{ij} = s_j q_{ij}$ is the product of a sample-specific size factor s_j (for sequencing depth normalisation) and a gene-specific expected count q_{ij} that depends on the experimental design through a log-linear model $\log_2 q_{ij} = \sum_k x_{jk}\beta_{ik}$.

Three ideas make DESeq2 work well on small-sample studies:

1. **Size factors** are computed by the median-of-ratios method, which is robust to highly expressed genes dominating the normalisation.
2. **Dispersion shrinkage** borrows strength across genes: each gene's dispersion estimate is shrunk toward a mean-dispersion trend fitted across all genes, so that noisy per-gene estimates in small samples do not produce wild inference.
3. **Log-fold-change shrinkage** (via `apeglm` or `ashr`) pulls LFCs of low-count genes toward zero, preventing them from dominating ranked gene lists.

The Wald test compares the estimated coefficient of interest to zero, with standard errors that reflect both the design and the dispersion. The Benjamini-Hochberg procedure controls the false discovery rate across the thousands of genes tested.

12.7.4 Assumptions

- Read counts follow a negative binomial distribution – usually reasonable for bulk mRNA-seq; may fail for very-low-coverage or heavily PCR-amplified libraries.
- The dispersion-mean trend is smooth across genes.

- Replicates within a condition are exchangeable after accounting for modelled covariates.
- No confounding between the biological condition and technical covariates (batch, library prep date). If there is, include the technical covariate in the design.

12.7.5 R Implementation

```r
library(DESeq2)
library(apeglm)
library(ggplot2)

counts <- matrix(rnbinom(20000 * 6, mu = 100, size = 10),
                 nrow = 20000, ncol = 6)
rownames(counts) <- paste0("gene", seq_len(20000))
colnames(counts) <- paste0("sample", seq_len(6))
coldata <- data.frame(condition = factor(rep(c("control", "treated"), each = 3)),
                      row.names = colnames(counts))

counts[1:200, 4:6] <- counts[1:200, 4:6] * 3

dds <- DESeqDataSetFromMatrix(countData = counts,
                              colData = coldata,
                              design = ~ condition)

dds <- dds[rowSums(counts(dds) >= 10) >= 3, ]

dds <- DESeq(dds)

res <- lfcShrink(dds, coef = "condition_treated_vs_control",
                 type = "apeglm")

summary(res)
head(res[order(res$padj), ])

plotMA(res, ylim = c(-3, 3))
```

The toy simulation above creates 20,000 genes and 6 samples, upregulates the first 200 genes in the treated condition, and runs the standard DESeq2 pipeline: construct the DESeqDataSet, filter very-low-count genes, estimate size factors and dispersion, fit the model, and shrink LFCs for reporting.

12.7.6 Output & Results

The `summary(res)` output reports the number of genes up- and down-regulated at the default alpha (0.1): typically a few hundred in the simulated upregulated set, with a small number of false positives at FDR 10%. The MA plot shows that the shrunken LFCs are compressed toward zero at low mean expression, as desired.

12.7.7 Interpretation

For a manuscript: “Differential expression between treated and control samples was assessed with DESeq2 (v1.44). Size factors were estimated by the median-of-ratios method; dispersions were shared across genes via the DESeq2 trend; log-fold changes

were shrunk using `apeglm`. 212 genes were differentially expressed at an FDR of 5% (154 up, 58 down)."

12.7.8 Practical Tips

- Use `vst()` or `rlog()` transformations for downstream visualisation (PCA, heatmaps); use raw counts in DESeq2's own pipeline.
- Filter low-count genes *before* running DESeq (e.g., keep genes with ≥ 10 counts in $\geq$ the smallest group size). This improves multiple-testing adjusted p-values without inflating false discoveries.
- Always inspect the PCA of normalised samples before differential expression. If samples cluster by batch rather than by condition, revise the design formula before interpreting results.
- `apeglm` shrinkage is the recommended default for LFC reporting. `ashr` handles contrasts; `normal` is the legacy method.
- Report shrunken LFCs for ranking and visualisation, but use the Wald test p-values for significance calling.

12.7.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.7.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.8 Alignment with BWA and Bowtie

12.8.1 Introduction

BWA-MEM and Bowtie2 are the dominant short-read DNA aligners in modern sequencing pipelines. Both use a Burrows-Wheeler-transformed (BWT) index of the reference genome to achieve rapid exact and approximate matching. The BWT compresses the reference while preserving the ability to query for substrings efficiently — the algorithmic insight that brought genome-scale alignment from hours per sample to minutes. Production alignment runs at the command line; R wrappers (`Rbowtie2`, `BWA` via `Rsubread`) integrate with downstream Bioconductor analyses.

12.8.2 Prerequisites

A working understanding of FASTQ format, reference genomes, and the SAM/BAM alignment format.

12.8.3 Theory

BWA-MEM (Burrows-Wheeler Aligner, Maximal Exact Matches) seeds local alignments by finding maximal exact matches, then extends via banded Smith-Waterman. It is the default aligner for whole-genome DNA sequencing because of its accuracy on long-insert paired-end reads and its handling of indels.

Bowtie2 offers end-to-end alignment (the read must align contiguously) or local alignment (soft-clipping allowed at read ends). It is fast and accurate for short reads and is a common choice for ChIP-seq, ATAC-seq, and small-genome DNA sequencing.

Both write SAM (text) or BAM (binary) output, including alignment scores, mapping quality, mate-pair information, and tags for soft-clipping, mismatches, and other features. For long reads (PacBio, Nanopore), use minimap2 instead — it is designed for the higher error rates and longer alignment lengths.

12.8.4 Assumptions

Short reads (typically below ~300 bp; for longer, switch to minimap2); reference genome appropriate to the species and assembly version of the experiment.

12.8.5 R Implementation

```
library(Rbowtie2); library(Rsamtools)

# Build index (one-off)
# bowtie2_build(references = "reference.fa", bt2Index = "idx")

# Align
# bowtie2(bt2Index = "idx", samOutput = "aln.sam",
#         seq1 = "reads.fastq", overwrite = TRUE)

# Convert SAM to sorted, indexed BAM
# asBam("aln.sam", destination = "aln_sorted")
# indexBam("aln_sorted.bam")

# Read a short segment
# scanBam("aln_sorted.bam", param = ScanBamParam(which = GRanges("chr1", IRanges(1,
1000))))
```

12.8.6 Output & Results

The primary output is a BAM file containing aligned reads with mapping quality, CIGAR string, and tags. `samtools flagstat` (or `Rsamtools::quickBamFlagSummary()`) reports alignment statistics: total reads, reads aligned, properly paired, duplicates, secondary alignments. A 95–98 % alignment rate is typical for high-quality WGS data.

12.8.7 Interpretation

A reporting sentence: “BWA-MEM (v0.7.17) alignment to GRCh38 achieved 97 % overall alignment rate on a 30× whole-genome library, with <1 % ambiguously mapped reads (multi-mappers); after Picard MarkDuplicates, the duplicate rate was 8 %, consistent with a well-optimised PCR-based library prep.” Always state the aligner version, reference assembly, and alignment / duplicate rates.

12.8.8 Practical Tips

- Use minimap2 for long reads (PacBio HiFi, Oxford Nanopore); BWA-MEM and Bowtie2 are short-read tools and lose accuracy beyond about 300 bp.
- Report alignment statistics (`flagstat`) after alignment as a routine QC step; declines in alignment rate signal contamination, library-prep problems, or reference mismatch.
- Index BAM files (`samtools index` or `Rsamtools::indexBam()`) for random-access by downstream tools (variant calling, peak calling, browser visualisation).
- Deduplicate after alignment with Picard MarkDuplicates or `samtools markdup` ; PCR duplicates inflate variant-calling false positives if not removed.
- For paired-end reads, specify both mate files; the aligner uses insert-size constraints to improve mapping accuracy near repeats.
- For ChIP-seq and ATAC-seq, Bowtie2 with `--very-sensitive` is the standard; for WGS variant calling, BWA-MEM is preferred.

12.8.9 R Packages Used

`Rbowtie2` for `bowtie2()` and `bowtie2_build()` ; `Rsubread::align()` for BWA-style alignment; `Rsamtools` for BAM manipulation, indexing, and flagstat-style statistics; `GenomicAlignments` for downstream alignment-aware analysis.

12.8.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.8.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.9 ChIP-Seq Analysis

12.9.1 Introduction

ChIP-seq (chromatin immunoprecipitation followed by sequencing) localises protein-DNA binding events genome-wide — transcription factor binding sites, histone modifications, chromatin remodellers. The resulting alignment density forms peaks at the binding loci; sharp peaks correspond to point-source binders (TFs), and broad domains correspond to histone-mark territories (H3K27me3, H3K9me3 spanning kilobases). The standard pipeline is alignment, peak calling with MACS2, annotation with ChIPseeker, motif enrichment with HOMER or `motifmatchr`, and differential binding with DiffBind.

12.9.2 Prerequisites

A working understanding of read alignment (BWA, Bowtie2), peak calling, and regulatory genomics including transcription factor binding sites and histone modifications.

12.9.3 Theory

Two peak-calling modes:

- **Narrow peaks** (TF ChIP, transcription start sites): MACS2 defaults are tuned for sharp, point-source binding events.
- **Broad peaks** (H3K27me3, H3K9me3, H3K36me3): use `--broad` to call diffuse domains spanning kilobases.

A matched input control (IgG or total chromatin) is essential for background normalisation; ChIP enrichment is measured as ChIP-vs-input fold enrichment per peak region. Standard QC includes:

- **FRiP** (fraction of reads in peaks): > 1 % indicates successful IP; lower values suggest weak antibody or insufficient enrichment.
- **TSS enrichment**: read coverage at transcription start sites; high values confirm chromatin accessibility was preserved.
- **Cross-correlation**: relative strand cross-correlation should peak at the fragment length, indicating successful IP.

12.9.4 Assumptions

Cross-linking and immunoprecipitation are efficient and specific; the control input matches ChIP depth; duplicates have been removed (essential for ChIP-seq peak calling).

12.9.5 R Implementation

```r
library(ChIPseeker)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)

# Example: annotate a toy peak set
set.seed(2026)
peaks <- GRanges(
  seqnames = rep("chr1", 100),
  ranges = IRanges(start = sort(sample(1e7:5e7, 100)), width = 400)
)

anno <- annotatePeak(peaks,
                     TxDb = TxDb.Hsapiens.UCSC.hg19.knownGene,
                     level = "gene",
                     verbose = FALSE)
as.data.frame(anno)[1:5, c("seqnames", "annotation", "distanceToTSS")]

# Feature-type distribution
plotAnnoBar(anno)

# (Motif-enrichment example would require sequence extraction with
#  BSgenome and motifmatchr::matchMotifs against a JASPAR PWM set.)
```

12.9.6 Output & Results

`annotatePeak()` assigns each peak to its nearest gene and categorises as promoter, intron, intergenic, etc. `plotAnnoBar()` summarises the feature-type distribution; typical TF ChIP-seq shows ~30 % promoter, ~40 % intron, ~25 % distal-intergenic.

12.9.7 Interpretation

A reporting sentence: "MACS2 (with input control, $q < 0.01$) called 18{,}230 high-confidence STAT1 ChIP peaks in IFN-γ-stimulated cells; 42 % were promoter-proximal, 35 % intronic, 18 % distal-intergenic. De novo motif analysis (HOMER) recovered the canonical STAT1 GAS motif (TTCN3GAA) at $p < 10^{-100}$, confirming antibody specificity. FRiP was 12 %, TSS enrichment 16, both well within QC standards." Always report FRiP, TSS enrichment, and motif validation.

12.9.8 Practical Tips

- Always use a matched input control (IgG, sonicated input, or no-antibody mock); without it, MACS2 falls back to generic background assumptions and inflates false positives.
- Check FRiP per sample; values below 1 % indicate weak IP, incorrect peak-calling mode, or sample failure.
- For histone marks (especially H3K27me3, H3K36me3, H3K9me3), use `--broad`; sharp peak callers miss diffuse domains entirely.
- Validate antibody specificity with de novo motif enrichment (HOMER's `findMotifsGenome.pl` or MEME-ChIP); recovering the expected motif is the gold standard for ChIP success.

- For differential binding across conditions, use DiffBind which builds consensus peak sets and runs DESeq2 or edgeR on peak-level read counts.
- For CUT&RUN or CUT&Tag (the modern higher-resolution alternatives), the same downstream tools apply; specialised peak callers (SEACR) handle the lower-background data more accurately.

12.9.9 R Packages Used

`ChIPseeker` for peak annotation and visualisation; `DiffBind` for differential-binding analysis; `motifmatchr` for motif enrichment with PWM databases; `chromVAR` for chromatin-accessibility-style motif activity scores; `ChIPQC` for quality control on ChIP-seq experiments.

12.9.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.9.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.10 Copy Number Variation Analysis

12.10.1 Introduction

Copy number variations (CNVs) are large-scale gains or losses of genomic regions, critical in oncology (tumour aneuploidy), developmental disorders (trisomies, deletions), and population genetics. WGS, exome, or array-based readouts feed segmentation algorithms that partition the genome into constant-copy-number segments.

12.10.2 Prerequisites

Coverage or intensity data (read depth, log R ratio); genome windowing.

12.10.3 Theory

Read-depth CNV calling: normalised read counts in windows scale linearly with copy number. The log-ratio series is smoothed and segmented via **Circular Binary Segmentation (CBS)** (DNAcopy), HMMs (`HMMcopy`), or Bayesian segmentation.

For tumour sequencing, methods like `ASCAT` and `FACETS` jointly estimate purity, ploidy, and allele-specific copy number from B-allele frequencies.

12.10.4 Assumptions

Coverage uniformity (controlled by GC / mappability correction); matched normal for tumour calling (ideal); segments have consistent copy number within.

12.10.5 R Implementation

```
library(DNAcopy)

set.seed(2026)
n_probes ← 1000
chrom ← rep(1:5, each = n_probes / 5)
maploc ← rep(seq(1, 1e7, length.out = n_probes / 5), 5)

# Simulated log ratios with two CNV segments
logR ← rnorm(n_probes, 0, 0.2)
logR[201:300] ← logR[201:300] + 0.8    # duplication on chr2
logR[601:680] ← logR[601:680] - 1.0    # deletion   on chr4

cna ← CNA(genomdat = logR, chrom = chrom, maploc = maploc,
          data.type = "logratio", sampleid = "tumor1")
smoothed ← smooth.CNA(cna)
seg ← segment(smoothed, verbose = 0)
seg$output                     # segment means and boundaries

plot(seg)
```

12.10.6 Output & Results

A segment table with chromosome, start, end, number of probes, and mean log ratio; the simulated duplication and deletion are recovered as distinct segments.

12.10.7 Interpretation

“CNV segmentation of the tumour identified gain on 17q (log2 ratio 0.85) and loss on 13q (log2 ratio -1.1); the 13q deletion spans RB1, consistent with the retinoblastoma diagnosis.”

12.10.8 Practical Tips

- GC bias correction (`QDNAseq`) is essential for low-coverage sequencing; uncorrected data produces spurious segments.
- For tumour studies, always use a matched normal to control germline CNVs and sequencing artefacts.
- Integer copy-number calls require ploidy and purity estimates (ASCAT, FACETS); log ratios alone give only relative copy number.
- Report CNV frequencies across cohorts with GISTIC2, which controls for background aneuploidy.
- For rare-CNV studies, focus on large events (> 100 kb); small events are dominated by noise in array data.

12.10.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.10.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.11 Counting Reads with featureCounts

12.11.1 Introduction

After read alignment, the next step in most RNA-seq pipelines is summarising aligned reads into per-gene (or per-feature) count matrices for differential-expression analysis. `featureCounts` (in `Rsubread`) and HTSeq-count are the two standard tools; both consume BAM files plus a GTF annotation and produce a gene-by-sample count matrix. The choices that matter — strand-specificity, multi-mapper handling, paired-end mode, and meta-feature aggregation — directly affect downstream conclusions and must match the experimental library prep.

12.11.2 Prerequisites

A working understanding of BAM alignment files, GTF/GFF annotation formats, and the difference between transcript-level and gene-level expression quantification.

12.11.3 Theory

Each aligned read is assigned to a feature (gene, exon, or other annotated region) it overlaps. Key configuration choices:

- **Strand-specificity**: `strandSpecific = 0` (unstranded), `= 1` (forward-stranded, dUTP), `= 2` (reverse-stranded, TruSeq SR). Wrong setting halves the assigned counts.
- **Multi-mapper handling**: usually counted once at the primary alignment; fractional counting (`fraction = TRUE`) splits credit across all alignments of a multi-mapper.
- **Paired-end**: count fragments rather than reads; both mates must align consistently.
- **Meta-feature**: aggregate exon-level counts into gene-level by `gene_id`; alternatively, count at the exon level for splicing analyses.

For transcript-level counts, pseudo-alignment tools (Salmon, Kallisto) skip the BAM-producing alignment step and provide isoform-level abundance estimates directly.

12.11.4 Assumptions

The GTF annotation matches the genome reference used for alignment; the strand-specificity setting matches the library prep; aligned BAMs are coordinate-sorted (or sortable by `featureCounts`).

12.11.5 R Implementation

```
library(Rsubread)

# Build counts from BAM + GTF
# counts ← featureCounts(files = c("s1.bam", "s2.bam"),
#                        annot.ext = "annotation.gtf",
#                        isGTFAnnotationFile = TRUE,
#                        GTF.featureType = "exon",
#                        GTF.attrType = "gene_id",
#                        strandSpecific = 2,
#                        isPairedEnd = TRUE)

# counts$counts: gene x sample matrix
```

12.11.6 Output & Results

`featureCounts()` returns a list with `$counts` (gene-by-sample matrix), `$annotation` (per-gene metadata including length used for FPKM/TPM normalisation), and `$stat` (per-sample summary of assigned vs ambiguous vs unassigned reads).

12.11.7 Interpretation

A reporting sentence: “featureCounts (Rsubread v2.16) assigned 82 % of mapped reads to GENCODE v44 gene features (`strandSpecific = 2` , `isPairedEnd = TRUE`); 8 % were ambiguous due to multi-gene overlap and 10 % were unassigned to any feature. The high assignment rate confirms the strand setting matches the dUTP library prep.” Always quote the assignment rate; sub-50 % rates signal serious problems (wrong strand, annotation mismatch).

12.11.8 Practical Tips

- Report `$stat` summaries alongside the count matrix; low assignment rates (below 60 % for typical RNA-seq) signal alignment, annotation, or strand-setting problems.
- Strand-specificity must match library protocol exactly; check with `infer_experiment.py` from RSeQC or visual inspection in IGV when uncertain.
- Use `primaryOnly = TRUE` for ChIP-seq and ATAC-seq, where multi-mappers are typically excluded.
- For transcript-level (isoform) counts, switch to Salmon, Kallisto, or RSEM; gene-level counting collapses isoform-resolved information.
- Bundle counts with metadata in a `SummarizedExperiment` object for downstream Bioconductor workflows (DESeq2, edgeR, limma all accept `SummarizedExperiment`).
- Include feature length in the output and use it for normalisation when reporting TPM/FPKM; raw counts alone do not control for gene length differences.

12.11.9 R Packages Used

`Rsubread::featureCounts()` for the canonical implementation; `GenomicAlignments::summarizeOverlaps()` for an alternative Bioconductor-style approach; `tximport` for importing transcript-level estimates from Salmon/Kallisto into `SummarizedExperiment`; `tximeta` for metadata-aware imports.

12.11.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.11.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.12 Drug-Target Interaction Mining

12.12.1 Introduction

Drug-target interaction data – which compounds bind which proteins at what affinity – underpins target discovery and drug repositioning. Public databases (ChEMBL, DrugBank, PubChem, BindingDB, PharmGKB) archive millions of bioassay measurements; programmatic access lets users assemble custom interaction tables.

12.12.2 Prerequisites

Familiarity with compound identifiers (SMILES, InChI, ChEMBL ID) and protein identifiers (UniProt).

12.12.3 Theory

Key bioassay quantities: - **IC50** – inhibitor concentration halving a biological response. - **Ki** – equilibrium inhibition constant. - **Kd** – dissociation constant. - **EC50** – effect concentration (agonist activity).

Lower = more potent. Values are typically reported in nM or uM; low-nM and sub-nM bindings are considered high-affinity drug-like hits.

12.12.4 Assumptions

Assays are comparable across studies (unreliable in practice – format, conditions, species differ); target and compound annotations are correct.

12.12.5 R Implementation

```
library(httr); library(jsonlite); library(dplyr)

# Query ChEMBL's REST API for bioactivities of a target (ERBB2 / HER2 = CHEMBL1824)
# This illustrative snippet is internet-free pseudo-code:
chembl_url ← paste0("https://www.ebi.ac.uk/chembl/api/data/activity.json",
                    "?target_chembl_id=CHEMBL1824&limit=100")
# res ← fromJSON(content(GET(chembl_url), "text"))
# activities ← res$activities

# Instead, simulate a small bioactivity table for illustration
set.seed(2026)
df ← data.frame(
  compound = paste0("CMP", 1:200),
  target   = sample(c("HER2", "EGFR", "MET"), 200, replace = TRUE),
  type     = sample(c("IC50", "Ki", "Kd"), 200, replace = TRUE),
  value_nM = round(10^runif(200, 0, 4))
)

# Potent binders (< 100 nM) per target
df %>%
  filter(value_nM < 100) %>%
  count(target, type, sort = TRUE)

# Rank compounds by median potency across multiple assays
df %>%
  group_by(compound, target) %>%
  summarise(median_pot = median(value_nM), n = n(), .groups = "drop") %>%
  arrange(median_pot) %>%
  head(5)
```

12.12.6 Output & Results

A ranked table of potent drug-target interactions per target and per compound. In practice this replaces the simulated data with a ChEMBL pull of > 10,000 measurements.

12.12.7 Interpretation

“ChEMBL mining identified 26 HER2 inhibitors with IC50 < 10 nM across independent assays; the most potent (CMP_X, median IC50 = 2 nM) was selected for in-silico docking.”

12.12.8 Practical Tips

- Aggregate across assays using median or geometric mean to smooth inter-lab variance.
- Convert IC50 to pIC50 ($-\log_{10}$) for symmetric, Gaussian-like distributions.
- Filter by assay confidence score when using ChEMBL (confidence ≥ 7).
- Use `BindingDB` for affinity-focused data; `DrugBank` for approved-drug information.

- Cross-reference compounds via InChIKey to merge records across databases (SMILES is not unique).

12.12.9 Reporting

A reproducible drug-target-interaction analysis names the database snapshot, the date of access, the activity types retained, and the confidence or quality filter applied to the raw assay records. State whether replicate measurements on the same compound-target pair were aggregated by median, mean of pIC50, or geometric mean, since these choices materially shift downstream rankings. When potency thresholds such as the conventional sub-100-nanomolar cutoff are used to define hits, justify the threshold in terms of the assay distribution rather than treating it as a universal constant, and report how compounds with mixed agonist or antagonist annotations were resolved before the ranking step.

12.12.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.12.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.13 Differential Expression with edgeR

12.13.1 Introduction

edgeR is one of the three dominant RNA-seq differential-expression frameworks (alongside DESeq2 and limma-voom). It fits a negative-binomial generalised linear model to each gene's counts, with empirical-Bayes shrinkage borrowing information across genes to stabilise per-gene dispersion estimates. The combination of NB likelihood and shrinkage gives edgeR its characteristic strength on small samples and its competitive accuracy on standard RNA-seq experiments.

12.13.2 Prerequisites

A working understanding of negative-binomial regression, RNA-seq normalisation (TMM), generalised linear models with offsets, and the empirical-Bayes shrinkage approach.

12.13.3 Theory

Per gene i and sample j:

$$Y_{ij} \sim \mathrm{NB}(\mu_{ij}, \phi_i), \qquad \log \mu_{ij} = \log S_j + x_j^\top \beta_i,$$

with S_j the TMM-normalised library size offset and ϕ_i the gene-wise dispersion. Empirical-Bayes shrinkage (`estimateDisp`) pulls per-gene dispersions toward a common trend across genes, stabilising estimation.

Three test choices:

- **exactTest**: NB-exact test for two-group comparisons; rarely used today.
- **glmQLFTest**: quasi-likelihood F-test; the recommended default for nearly all designs because it accounts for uncertainty in dispersion estimation.
- **glmLRT**: classical likelihood-ratio test; more aggressive but anti-conservative when sample sizes are very small.

12.13.4 Assumptions

Counts approximately negative-binomial conditional on covariates; the design matrix encodes the biological and technical factors correctly; sufficient sample size for stable dispersion estimation (typically ≥ 3 per group).

12.13.5 R Implementation

```r
library(edgeR)

set.seed(2026)
counts ← matrix(rpois(20000 * 12, lambda = 30), 20000, 12)
group ← factor(rep(c("A", "B"), each = 6))

y ← DGEList(counts = counts, group = group)
y ← calcNormFactors(y)

design ← model.matrix(~ group)
y ← estimateDisp(y, design)

fit ← glmQLFit(y, design)
qlf ← glmQLFTest(fit, coef = 2)

topTags(qlf)
summary(decideTests(qlf))
```

12.13.6 Output & Results

`topTags()` returns the top differentially-expressed genes ordered by FDR-adjusted p-value, with $\log_2$ fold change, log-CPM, F-statistic, and BH-adjusted p-value. `decideTests()` summarises significance calls per direction (up / down / not significant) for cross-method comparison.

12.13.7 Interpretation

A reporting sentence: "edgeR's quasi-likelihood F-test identified 342 differentially expressed genes at FDR < 0.05 between conditions A and B (n = 6 per group), with 180 upregulated and 162 downregulated in condition B; the median absolute $\log_2$ fold change among significant genes was 1.4. Filtering with `filterByExpr` retained 14{,}500 of 20{,}000 genes for testing." Always state the FDR threshold, sample sizes, and pre-filtering decisions.

12.13.8 Practical Tips

- Use `glmQLFTest()` as the default for nearly all designs; it has better Type I error control than `glmLRT()` at small sample sizes.
- Filter low-count genes with `filterByExpr()` before dispersion estimation; testing 20{,}000 mostly-zero genes wastes power and produces unstable shrinkage.
- For paired or blocked designs, include the blocking factor in the `design` matrix; the QLF test handles the corresponding contrast correctly.
- Use `glmTreat()` for fold-change-thresholded inference (testing $|\text{LFC}| >$ a chosen cutoff rather than LFC ≠ 0); this is more biologically meaningful than testing strict zero.
- Visualise with MA plot, volcano plot, and a heatmap of top genes; `Glimma::glimmaMA()` provides interactive versions for exploration.
- Compare edgeR results with DESeq2 and limma-voom on the same data; concordance across methods is reassuring, while disagreement signals model-sensitive genes that warrant investigation.

12.13.9 R Packages Used

`edgeR` for `DGEList`, `calcNormFactors`, `estimateDisp`, `glmQLFit`, `glmQLFTest`, and the canonical NB workflow; `limma` for the voom alternative; `DESeq2` for the third major DE method; `Glimma` for interactive MA and volcano plots; `EnhancedVolcano` for publication-ready volcano plots.

12.13.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.13.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.14 FASTQ Quality Control

12.14.1 Introduction

Every high-throughput sequencing analysis starts with FASTQ quality control. Before alignment, before quantification, before any downstream interpretation, you should inspect per-base quality, adapter contamination, GC content, and duplicate rate to decide on trimming and filtering strategies. The cost of skipping this step is high: low-quality data quietly degrade alignment accuracy, inflate variant-calling false positives, and bias differential-expression conclusions in ways that are hard to detect retrospectively.

12.14.2 Prerequisites

A working understanding of the FASTQ format, Phred quality scores, and basic Illumina sequencing biology.

12.14.3 Theory

The Phred quality score is $Q = -10 \log_{10} P(\text{error})$, so Q30 means 1 in 1000 probability of base-call error and Q40 means 1 in 10{,}000. Modern Illumina runs aim for Q30 or higher across most of each read, with quality typically dropping toward the 3′ end.

Standard FastQC-style metrics:

- **Per-base sequence quality**: boxplot of Phred scores at each position; declines toward 3′ are normal but should not drop below Q20.
- **Per-sequence quality**: histogram of mean read quality; bi-modal histograms suggest mixed-quality libraries.
- **GC content**: should match expected genomic GC; deviations indicate contamination or biased library prep.
- **Adapter contamination**: presence of known adapter sequences at 3′ ends.
- **Overrepresented sequences**: top recurring sequences often pinpoint adapters or contamination.

12.14.4 Assumptions

Modern Illumina FASTQ files with Sanger-encoded Phred (Phred+33). Older Illumina encodings (Phred+64) require explicit specification.

12.14.5 R Implementation

```
library(ShortRead); library(Rqc)

# Assume demo FASTQ file path
fq_file ← system.file("extdata", "ex1.fq", package = "ShortRead")

# Basic inspection
reads ← readFastq(fq_file)
length(reads)
head(width(reads))
mean(as.numeric(encoding(quality(reads))))   # mean Phred base

# Rqc for systematic QC
qc ← rqcQA(fq_file, workers = 1)
rqcReport(qc)
```

12.14.6 Output & Results

`Rqc::rqcQA()` produces an interactive HTML report covering per-position quality, GC content, duplicate rate, length distribution, and adapter flags. `ShortRead::readFastq()` provides programmatic access for custom diagnostics.

12.14.7 Interpretation

A reporting sentence: “FastQC-style QC on the 12 RNA-seq samples showed mean per-base Phred ranging from 35 to 38 across the read length, adapter contamination below 1 %, and GC content centred at 49 % (within the expected range for human transcriptome). After fastp trimming at Q15 with adapter removal, approximately 95 % of bases were retained per sample.” Always report pre- and post-trim QC.

12.14.8 Practical Tips

- Use `fastp` or `trim_galore` at the command line for production QC and trimming; R tools (`Rqc`, `ShortRead`) are useful for exploration but are slower than the dedicated command-line utilities on large datasets.
- Report pre- and post-trim QC; the QC improvement validates the trimming step.
- GC content far from the expected genomic value suggests contamination, library prep biases, or species mismatch; investigate before alignment.
- Overrepresented sequences often pinpoint adapter sequences, rRNA contamination, or PCR duplicates of single dominant transcripts.
- For multiple samples, MultiQC aggregates per-sample reports into a single dashboard; this is essential for reviewing dozens or hundreds of samples consistently.
- Phred Q20 is the conventional threshold for “good” bases; trimming at Q15 retains more data while removing the worst calls.

12.14.9 R Packages Used

`ShortRead` for general FASTQ manipulation in R; `Rqc` for systematic QC reports; `Rfastp` as a wrapper around fastp for fast trimming and QC; `ngsReports` for parsing and aggregating MultiQC and FastQC outputs into R objects.

12.14.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.14.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.15 Gene Ontology Enrichment

12.15.1 Introduction

Gene Ontology (GO) enrichment asks whether a list of differentially-expressed genes is disproportionately annotated to specific GO terms — biological processes, molecular functions, or cellular components. It is the most common first-pass functional interpretation of an RNA-seq, proteomics, or single-cell DE list and is the entry point to pathway-level analysis. The dominant method is over-representation analysis (ORA) using Fisher's exact or hypergeometric tests; gene-set enrichment analysis (GSEA) extends to ranked gene lists without requiring a significance threshold.

12.15.2 Prerequisites

A working understanding of differential-expression analysis, GO annotations and their three branches (biological process, molecular function, cellular component), and basic Fisher's exact test framework.

12.15.3 Theory

ORA constructs a 2×2 contingency table for each GO term:

	DE	not DE
In term	a	b
Not in term	c	d

Fisher's exact test (or the hypergeometric distribution) tests whether more DE genes fall in the term than expected under random sampling from the universe. The universe is the set of all genes that *could* have been tested — typically the genes that passed pre-filtering — not the entire genome.

Multiple testing across thousands of GO terms is controlled via Benjamini-Hochberg FDR; nested GO term hierarchies introduce correlated tests, but BH adjustment remains conservative under dependence.

12.15.4 Assumptions

Genes are sampled uniformly from the universe given non-DE status; GO annotations are correct (recently updated); the universe matches the tested set (testing all genes from the genome inflates apparent enrichment when most are not in the tested set).

12.15.5 R Implementation

```r
library(clusterProfiler); library(org.Hs.eg.db)

# Example with a small simulated DE list
set.seed(2026)
all_genes ← keys(org.Hs.eg.db, keytype = "SYMBOL")
de_genes  ← sample(all_genes, 200)

ego ← enrichGO(gene = de_genes,
               universe = all_genes,
               OrgDb = org.Hs.eg.db,
               keyType = "SYMBOL",
               ont = "BP",
               pAdjustMethod = "BH",
               pvalueCutoff = 0.05,
               qvalueCutoff = 0.2)

head(as.data.frame(ego))[, c("ID", "Description", "p.adjust", "Count")]
dotplot(ego, showCategory = 15)
```

12.15.6 Output & Results

`enrichGO()` returns a ranked table of GO terms with BH-adjusted p-values, gene counts, and the contributing gene symbols. `dotplot()` visualises the top hits as a horizontal dot plot with gene-count size and adjusted-p-value colour.

12.15.7 Interpretation

A reporting sentence: "GO over-representation analysis (BH-FDR < 0.05) identified 28 biological-process terms among the 200 DE genes; the top terms — `response to stimulus` (45 genes), `immune response` (32), `cell cycle regulation` (28) — reflect the experimental manipulation. `simplify()` reduced the redundant hierarchy to 12 representative terms." Always state the universe, the multiple-testing correction, and any term-redundancy reduction.

12.15.8 Practical Tips

- Define the universe correctly — always the set of genes that passed pre-filtering and were tested for DE, never the whole genome; using the latter inflates apparent enrichment.
- Use `simplify()` on the `enrichResult` object to collapse redundant nested GO terms; the unsimplified output often contains many parent-child duplications.
- Direction-agnostic ORA loses signal; for biology-aware interpretation, run up- and down-regulated lists separately.
- For ranked gene lists with continuous statistics, prefer GSEA (`gseGO()`) over ORA; GSEA does not require an arbitrary significance threshold.
- `goseq` adjusts for gene-length bias in RNA-seq (longer genes have more power for DE detection); use it when length bias is a concern.
- Pair GO with KEGG (`enrichKEGG`) and Reactome (`enrichPathway`) for complementary functional coverage.

12.15.9 R Packages Used

`clusterProfiler::enrichGO()`, `gseGO()`, `simplify()` for canonical GO enrichment; `org.Hs.eg.db` and species-specific packages for annotation; `topGO` and `goseq` for alternative implementations; `enrichplot` for `dotplot()`, `cnetplot()`, and `emapplot()` visualisations of enrichment results.

12.15.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.15.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.16 GSEA Preranked Analysis

12.16.1 Introduction

Gene Set Enrichment Analysis (GSEA) tests whether a gene set is concentrated at the top or bottom of a ranked gene list (e.g., by log fold change or signed p-value). Unlike over-representation analysis, it uses the full ranking and is more sensitive to coordinated modest changes within a pathway.

12.16.2 Prerequisites

Ranked gene statistic (LFC or signed -log10 p); a curated gene-set database (MSigDB, Reactome, KEGG).

12.16.3 Theory

GSEA walks down the ranked list, incrementing a running-sum statistic when encountering a set member (weighted by the rank statistic) and decrementing otherwise. The **enrichment score (ES)** is the maximum deviation from zero; a permutation-based null distribution yields p-values. `fgsea` uses a fast adaptive multi-level permutation scheme.

Normalised ES (NES) adjusts for gene-set size and allows cross-set comparison.

12.16.4 Assumptions

The ranking reflects biological signal; gene sets are well-curated; permutation null captures the null distribution of ES.

12.16.5 R Implementation

```
library(fgsea); library(msigdbr)

set.seed(2026)
n_genes ← 5000
genes ← paste0("g", 1:n_genes)
stats ← rnorm(n_genes); names(stats) ← genes

# Inject a pathway signal
pathway_A ← sample(genes, 30); stats[pathway_A] ← stats[pathway_A] + 1.5
pathway_B ← sample(genes, 20); stats[pathway_B] ← stats[pathway_B] - 1.2

pathways ← list(pathway_A = pathway_A, pathway_B = pathway_B,
                pathway_C = sample(genes, 40))

fg ← fgsea(pathways = pathways, stats = sort(stats, decreasing = TRUE),
           minSize = 10, maxSize = 500)
fg[, .(pathway, NES, pval, padj)]
plotEnrichment(pathways$pathway_A, sort(stats, decreasing = TRUE))
```

12.16.6 Output & Results

A table with NES, p-value, and adjusted p-value per pathway. The enrichment plot for pathway_A shows a strong positive leading edge.

12.16.7 Interpretation

“GSEA against MSigDB Hallmark revealed positive enrichment for inflammatory response (NES = 1.8, BH-FDR < 0.01) and negative enrichment for oxidative phosphorylation (NES = -1.5).”

12.16.8 Practical Tips

- Use the `stat` column from DESeq2 or `t` from limma as the ranking; signed -log10(p-value) is an alternative.
- Remove NAs from the ranking; fgsea is sensitive to ties and NAs.
- Report the **leading-edge subset** for biologically driven interpretation.
- MSigDB has Hallmark (50 curated), C2 (curated pathways), C5 (GO), C7 (immune); match collection to biological question.
- GSEA tests concentrated enrichment, not direction-specific effects; combine with ORA for complementary evidence.

12.16.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.16.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.17 GSVA Single-Sample Enrichment

12.17.1 Introduction

GSVA (Gene Set Variation Analysis) transforms a gene-by-sample expression matrix into a pathway-by-sample matrix by assigning each sample a per-pathway enrichment score. The transformation enables pathways to be treated as features in any downstream analysis — Cox regression for survival, clustering, classification, or differential analysis at the pathway level. Where over-representation analysis (GO, KEGG over-representation) tells you which pathways are enriched in a hit list, GSVA tells you how each individual sample stands relative to a panel of pathways.

12.17.2 Prerequisites

A working understanding of gene-set enrichment, log-normalised expression matrices, and the difference between sample-level and contrast-level enrichment.

12.17.3 Theory

For each sample, GSVA computes an empirical CDF of expression values across genes (rank-based) and derives a Kolmogorov-Smirnov-like statistic for each gene set. The result is a per-sample, per-pathway score, typically on a roughly $[-1, 1]$ scale. ssGSEA (single-

sample GSEA) is a related method differing primarily in the weighting scheme; both are implemented in the `GSVA` package.

The transformed pathway-by-sample matrix can be analysed with any standard tool: limma for differential pathway expression, Cox regression for survival, hierarchical clustering, supervised classification.

12.17.4 Assumptions

Expression is consistently normalised across samples; gene sets are accurately curated and relevant to the biology; scores are interpretable as relative within a study but not on an absolute scale across studies.

12.17.5 R Implementation

```
library(GSVA)

set.seed(2026)
n_genes ← 500; n_samples ← 20
expr ← matrix(rnorm(n_genes * n_samples), n_genes, n_samples)
rownames(expr) ← paste0("g", 1:n_genes)

pathways ← list(
  path_A = paste0("g", sample(1:n_genes, 30)),
  path_B = paste0("g", sample(1:n_genes, 40)),
  path_C = paste0("g", sample(1:n_genes, 25))
)

gsva_scores ← gsva(expr, pathways, method = "gsva", kcdf = "Gaussian",
                   verbose = FALSE)
dim(gsva_scores)        # pathways x samples
round(gsva_scores[, 1:4], 2)
```

12.17.6 Output & Results

`gsva()` returns a pathway-by-sample matrix of enrichment scores. Each column is one sample's pathway profile; high scores indicate the pathway's genes are coordinately up-regulated in that sample relative to others, low scores indicate the opposite.

12.17.7 Interpretation

A reporting sentence: "GSVA pathway scores (Hallmark v7.5 gene sets, log-CPM input, Gaussian kernel) served as features in a Cox regression of overall survival across 200 patients; the inflammatory-response pathway emerged as the strongest independent predictor (HR 1.7 per SD increase, 95 % CI 1.2 to 2.4, $p = 0.002$) after adjustment for age, stage, and treatment arm." Always state the gene-set source, the kernel, and the input data.

12.17.8 Practical Tips

- Use `kcdf = "Gaussian"` for log-CPM, `kcdf = "Poisson"` for raw counts; the wrong setting biases scores systematically.

- Downstream differential-pathway analysis works with a standard limma fit on the GSVA score matrix with the same design used for gene-level analysis.
- For single-cell expression, per-cell GSVA is very noisy; aggregate cells into pseudobulk first, or use signed-rank alternatives (`AUCell` , `UCell`) designed for the single-cell context.
- Always compute GSVA on a single, consistently normalised expression matrix; mixing normalisations across the input matrix produces invalid scores.
- Scores are relative within a study; do not compare raw GSVA scores across independent datasets without re-running on the combined matrix.
- For pathway exploration, the MSigDB Hallmark, KEGG, and Reactome gene sets (via `msigdbr::msigdbr()`) are the standard starting libraries.

12.17.9 R Packages Used

`GSVA::gsva()` for the canonical GSVA and ssGSEA implementation; `msigdbr` for MSigDB gene-set retrieval; `AUCell` and `UCell` for single-cell-friendly enrichment alternatives; `singscore` for an alternative rank-based single-sample method; `escape` for a `Seurat` -integrated GSVA workflow.

12.17.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.17.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.18 Heatmaps for RNA-seq

12.18.1 Introduction

Heatmaps compactly display an expression matrix as a colour grid with rows (genes) and columns (samples) optionally clustered hierarchically. For RNA-seq they are the standard visualisation for a differential-expression signature, a curated marker panel, or a pathway-of-interest gene set: colour intensity encodes scaled expression, dendrograms reveal structure, and annotation bars link sample columns to experimental metadata. Done well, a heatmap communicates more about the biological structure of an experiment than dozens of pages of statistical tables.

12.18.2 Prerequisites

A working understanding of log-CPM or variance-stabilised counts, hierarchical clustering, and the role of row scaling (z-scoring) in making heatmaps interpretable.

12.18.3 Theory

Raw counts are unsuitable for heatmaps because one strongly expressed gene dominates the colour range and obscures the pattern in lower-expression genes. The standard pre-processing is:

1. **Log-transform** counts (or use VST/rlog from DESeq2).
2. **Row z-score** each gene to unit variance and zero mean across samples; this scales each gene's display range comparably.
3. **Cluster rows and columns** by hierarchical clustering, typically Euclidean distance with Ward or complete linkage.

Annotation bars (sample group, batch, time point) overlay metadata on the column dendrogram, making it immediately visible whether sample clustering corresponds to biology or to technical artefacts.

12.18.4 Assumptions

Row scaling assumes comparable behaviour of genes (one z-score unit means the same thing across genes); column clustering assumes samples are comparable after normalisation; the chosen distance metric and linkage match the cluster structure.

12.18.5 R Implementation

```
library(pheatmap)

set.seed(2026)
n_genes ← 50; n_samples ← 20
mat ← matrix(rnorm(n_genes * n_samples), n_genes, n_samples)
rownames(mat) ← paste0("gene_", 1:n_genes)
colnames(mat) ← paste0("S", 1:n_samples)

# Introduce a DE pattern
group ← rep(c("ctrl", "trt"), each = 10)
mat[1:10, group == "trt"] ← mat[1:10, group == "trt"] + 3

annot ← data.frame(group = factor(group))
rownames(annot) ← colnames(mat)

pheatmap(mat, scale = "row",
         clustering_distance_rows = "euclidean",
         clustering_method = "ward.D2",
         annotation_col = annot,
         color = colorRampPalette(c("#2A9D8F", "white", "#E76F51"))(50),
         main = "Top DE genes (row-scaled z-scores)")
```

12.18.6 Output & Results

`pheatmap()` produces a coloured grid with row and column dendrograms, an annotation strip for the `group` factor, and a colour legend. The two sample clusters separate the control and treatment groups based on the 10 simulated DE genes; the row dendrogram groups the DE genes into a coherent block distinct from the noise rows.

12.18.7 Interpretation

A reporting sentence (figure caption): "Row-scaled heatmap of the top 50 differentially-expressed genes (z-scores per row, blue-white-red diverging colour scale, Ward linkage on Euclidean distance); samples cluster into two distinct groups corresponding to the experimental conditions, and the top 10 row-cluster shows coherent up-regulation in the treatment group." Always describe the scaling, distance metric, and linkage method.

12.18.8 Practical Tips

- Always row-scale (z-score) for DE heatmaps; otherwise high-expression genes swamp the colour range and the pattern in lower-expression genes is invisible.
- Use `ComplexHeatmap::Heatmap()` for richer annotations, row/column splits by cluster, and multi-panel composition; `pheatmap` is simpler for one-off plots.
- Truncate extreme values (e.g., cap z-scores at ± 3) to prevent one outlier dominating the colour scale.
- Annotate samples with batch, condition, and time-point information in column annotations; the visual separation should match the biology, not the technical structure.
- Use variance-stabilised counts (DESeq2 `vst()` or `rlog()`) rather than raw log-CPM for more robust heatmaps when the dataset has wide expression range.
- Save the figure at the final publication size (`ggsave()` if using `ggplot`, `pdf()` / `png()` for base plots); shrinking heatmaps in a word processor produces unreadable cell labels.

12.18.9 R Packages Used

`pheatmap` for quick standard heatmaps; `ComplexHeatmap` for advanced layouts including multiple annotations, row/column splits, and composition with other panels; `DESeq2::vst()` and `rlog()` for variance-stabilising input transformations; `RColorBrewer` and `viridis` for colour palettes.

12.18.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.18.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.19 Differential Expression with limma-voom

12.19.1 Introduction

limma-voom is one of the three dominant differential-expression methods for RNA-seq (alongside DESeq2 and edgeR). The voom step transforms raw counts onto the log-CPM scale and computes per-observation precision weights that account for the mean-variance trend characteristic of count data. The result is suitable input to limma's full linear-modelling framework, which adds the empirical-Bayes shrinkage of variance estimates that drives limma's power on small samples and complex designs.

12.19.2 Prerequisites

A working understanding of linear models, RNA-seq normalisation (TMM via edgeR), the mean-variance relationship of count data, and the empirical-Bayes shrinkage approach.

12.19.3 Theory

`voom()` transforms raw counts via $\log_2(\mathrm{CPM} + 0.5)$ and fits a smooth mean-variance trend across genes. The fitted trend produces a precision weight for each observation; downstream `lmFit()` regression uses these weights so that low-expression genes (with high relative variance) contribute less to the joint estimation. The empirical-Bayes step (`eBayes()`) shrinks per-gene variance estimates toward the global trend, yielding moderated t-statistics that have better small-sample properties than ordinary t-tests.

The full limma-voom pipeline supports arbitrarily complex designs (interactions, blocking, batch adjustment) through standard `model.matrix` notation, and the contrast machinery handles multi-condition comparisons cleanly via `contrasts.fit()`.

12.19.4 Assumptions

Counts have been TMM-normalised (via `calcNormFactors` from edgeR); the chosen design matrix correctly encodes the biological and technical factors; the mean-variance trend is approximately monotone (true for typical bulk RNA-seq).

12.19.5 R Implementation

```r
library(limma); library(edgeR)

set.seed(2026)
counts ← matrix(rpois(20000 * 12, lambda = 30), 20000, 12)
group ← factor(rep(c("A", "B"), each = 6))

dge ← DGEList(counts = counts); dge ← calcNormFactors(dge)
design ← model.matrix(~ group)

v ← voom(dge, design, plot = FALSE)
fit ← lmFit(v, design)
fit ← eBayes(fit)

topTable(fit, coef = 2, n = 10)
summary(decideTests(fit))
```

12.19.6 Output & Results

`topTable()` returns the top differentially-expressed genes with $\log_2$ fold change, average expression, moderated t-statistic, raw p-value, and Benjamini-Hochberg adjusted p-value. `decideTests()` summarises significance calls per contrast, useful for cross-method comparison.

12.19.7 Interpretation

A reporting sentence: “limma-voom identified 380 genes at FDR < 0.05 between conditions A and B (n = 6 per group), with mean fold change among significant genes of 1.7 (range 0.4 to 3.8); the corresponding edgeR analysis on the same data identified 342 genes, with 320 genes (84 %) shared between methods, indicating substantial concordance.” Always state the FDR threshold and the design matrix structure.

12.19.8 Practical Tips

- limma-voom is competitive with edgeR’s `glmQLFit` and DESeq2’s negative-binomial Wald test; choose based on workflow conventions and personal preference rather than expected accuracy gains.
- `voomWithQualityWeights()` additionally down-weights noisy samples, useful when one or two samples have visibly poor QC; combine the two weight types when sample-level variability is substantial.
- For complex designs (interactions, multiple blocking factors), the standard `model.matrix` and `contrasts.fit()` machinery applies; limma’s design language is among the most flexible.
- Voom is well-suited to scRNA-seq pseudobulk: aggregate cells per cluster into pseudo-bulk counts, then run voom on the resulting matrix.
- For very few samples per group (n < 3), variance estimates are unstable; consider DESeq2 with its variance-stabilising shrinkage, or rely heavily on the empirical Bayes step in limma.
- Pair the analysis with `voomQCPlot` or `meanvariancetrend()` diagnostic plots to confirm the variance trend is well-fitted.

12.19.9 R Packages Used

`limma` for `voom()`, `lmFit()`, `eBayes()`, `topTable()`, and the broader linear-modelling framework; `edgeR` for `DGEList`, `calcNormFactors` (TMM normalisation); `Glimma` for interactive MA and volcano plots; `EnhancedVolcano` for publication-ready volcano plots from limma output.

12.19.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.19.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.20 MA Plots

12.20.1 Introduction

An MA plot — also known as a mean-difference plot or Bland-Altman-style plot for genomics — displays the log fold change (M) on the y-axis against the mean expression intensity (A) on the x-axis. It is the standard diagnostic for differential-expression results: if the bulk of points (the non-DE genes) clusters around zero independent of mean intensity, normalisation has succeeded; systematic curvature or vertical drift indicates intensity-dependent bias that contaminates the DE calls. The MA plot complements the volcano plot by emphasising the relationship between effect size and expression level rather than effect size and statistical significance.

12.20.2 Prerequisites

A working understanding of log fold change, average expression on the log scale, and differential-expression output (DESeq2, edgeR, limma).

12.20.3 Theory

For two samples (or two groups) X, Y:

$$M = \log_2(Y/X), \qquad A = \tfrac{1}{2}\log_2(XY).$$

A well-normalised comparison shows M scattered around zero independently of A; the lowess curve of M vs A should be approximately flat at zero. Curvature, drift, or systematic offset indicates scaling or compositional bias that normalisation has failed to remove. The “trumpet” shape — wider scatter at low expression, narrower at high — is expected because low-expression genes have noisier fold-change estimates; this is information about precision, not bias.

12.20.4 Assumptions

Counts have been normalised (TMM, RLE, or similar); the majority of genes are non-differentially expressed (so the lowess of M should pass through zero); the dataset is sufficiently large to estimate the lowess curve stably.

12.20.5 R Implementation

```r
library(limma); library(edgeR)

set.seed(2026)
counts ← matrix(rpois(10000 * 6, lambda = 30), 10000, 6)
group  ← factor(rep(c("A", "B"), each = 3))
dge    ← DGEList(counts); dge ← calcNormFactors(dge)

design ← model.matrix(~ group)
v ← voom(dge, design)
fit ← lmFit(v, design); fit ← eBayes(fit)

plotMD(fit, coef = 2, status = decideTests(fit)[, 2],
       values = c(-1, 1), hl.col = c("#2A9D8F", "#E76F51"),
       main = "MA plot (limma)")
abline(h = 0, lty = 2)
```

12.20.6 Output & Results

`limma::plotMD()` plots logFC vs mean log-CPM with significant genes highlighted in colour and a horizontal reference line at zero. `DESeq2::plotMA()` produces the same plot from a results object in a single call. Most non-DE genes should cluster tightly around zero with the trumpet shape opening toward low mean expression.

12.20.7 Interpretation

A reporting sentence: “The MA plot (Fig. X) confirms that after TMM normalisation the log fold changes are centred on zero across the full mean-expression range; the lowess curve does not deviate from zero, indicating no intensity-dependent bias and supporting the validity of the differential-expression calls. Significant genes (highlighted) are distributed across the expression range.” Always pair MA plots with volcano plots when reporting DE results.

12.20.8 Practical Tips

- Always produce an MA plot alongside the volcano plot when reporting DE results; the two communicate complementary information.

- Overlay a lowess curve (`lines(lowess(A, M))`) to diagnose intensity-dependent bias formally; visible drift signals normalisation problems.
- For DESeq2, `plotMA(res)` produces the standard MA plot in a single call; the function automatically applies LFC shrinkage if the results object contains shrunken estimates.
- Use shrinkage (`DESeq2::lfcShrink` with `apeglm` or `ashr`) to stabilise noisy log-fold-change estimates at low expression; the shrunken MA plot is more interpretable than the raw version.
- Label only the top few genes (5–10); labelling all significant hits makes the plot unreadable.
- For very large MA plots, alpha-blending or hexbin (`smoothScatter`) prevents overplotting in the dense central column.

12.20.9 R Packages Used

`limma::plotMD()` for MA plots from limma fits; `DESeq2::plotMA()` for MA plots from DESeq2 results; `edgeR::plotMD()` for the edgeR analogue; `Glimma::glimmaMA()` for interactive MA plots embeddable in HTML; `ggplot2` for hand-built versions with custom styling.

12.20.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.20.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.21 Metagenomic Profiling

12.21.1 Introduction

Shotgun metagenomics sequences all DNA in a sample rather than a marker gene, enabling species-level taxonomic and functional profiling. Two dominant approaches: MetaPhlAn (marker-gene alignment to species-specific genes) and Kraken (k-mer matching against a reference database).

12.21.2 Prerequisites

Shotgun sequencing basics; read QC; familiarity with amplicon methods.

12.21.3 Theory

MetaPhlAn maps reads to ~1M clade-specific markers; species abundances are inferred as relative marker coverage. Output is a per-species relative abundance table.

Kraken2 classifies reads via k-mer matches to a database (e.g., RefSeq Bacteria). Fast and comprehensive but prone to false positives for low-abundance species; pair with **Bracken** for abundance re-estimation.

Functional profiling: HUMAnN3 for pathway-level abundances (MetaCyc); eggNOG-mapper for gene-function annotation.

12.21.4 Assumptions

Reads are adapter-trimmed and host-decontaminated; reference database covers the expected taxa; relative abundances are interpretable only within a sample.

12.21.5 R Implementation

MetaPhlAn and Kraken run on the command line; R ingests the results for downstream diversity and DA analysis:

```
library(phyloseq)

set.seed(2026)
# Simulated MetaPhlAn-style output
species ← paste0("s__Species_", 1:50)
samples ← paste0("S", 1:20)
abund ← matrix(rgamma(length(species) * length(samples), 0.5, 2),
               length(species), length(samples))
abund ← sweep(abund, 2, colSums(abund), "/")   # relative abundances
rownames(abund) ← species; colnames(abund) ← samples

# Coerce to a phyloseq object for diversity analyses
tax_tab ← matrix(species, length(species), 1,
                 dimnames = list(species, "Species"))
ps ← phyloseq(otu_table(round(abund * 1e6), taxa_are_rows = TRUE),
              tax_table(tax_tab))
ps

# Top species by mean abundance
head(sort(rowMeans(abund), decreasing = TRUE), 5)
```

Example command-line steps (not run here):

```
metaphlan sample.fastq --input_type fastq --bowtie2db metaphlan_db \
          -o sample_profile.txt
kraken2 --db kraken_db --paired R1.fq R2.fq --report sample.kreport
```

12.21.6 Output & Results

A species-by-sample abundance table; in phyloseq, all standard diversity and DA methods apply.

12.21.7 Interpretation

“MetaPhlAn profiling revealed 420 species; top differentially abundant species between gut inflammation cases and controls included Faecalibacterium prausnitzii (depleted) and Escherichia coli (enriched).”

12.21.8 Practical Tips

- Host-read removal (Bowtie2 vs the host genome) is essential; residual host DNA biases classification.
- Kraken2 + Bracken gives per-species quantification with reduced false positives vs Kraken alone.
- For functional analysis, HUMAnN3 provides pathway abundances suitable for differential pathway analysis with MaAsLin2 or LEfSe.
- ONT long reads enable better species resolution in complex communities.
- Do not mix relative and absolute abundances across analyses; compositional data require log-ratio transforms (CLR) or specific methods (ANCOM-BC).

12.21.9 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.21.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.22 DNA Methylation Analysis

12.22.1 Introduction

DNA methylation at CpG dinucleotides is a stable epigenetic mark that regulates gene expression and varies with age, tissue type, environmental exposures, and disease state. The two dominant assays are Illumina arrays (450K, 850K EPIC) measuring methylation at hundreds of thousands of CpGs and whole-genome bisulfite sequencing (WGBS) covering nearly all CpGs at higher cost. Differential methylation analysis identifies CpG positions or regions whose methylation differs between groups, often serving as biomarkers (e.g., cancer subtype classification, age estimation).

12.22.2 Prerequisites

A working understanding of bisulfite conversion chemistry (unmethylated C → T, methylated C → C), the difference between beta-values and M-values as methylation summaries, and basic linear-modelling on continuous outcomes.

12.22.3 Theory

Two methylation summaries:

- **Beta-value**: $\beta = M/(M + U + \alpha)$, the proportion of methylated reads (with α a small offset to stabilise low-coverage estimates). Intuitive (0 = fully unmethylated, 1 = fully methylated) but heteroscedastic — variance is largest near 0.5.
- **M-value**: $\log_2(M/U)$, the log-ratio of methylated to unmethylated. Linear and approximately homoscedastic; preferred for regression.

The standard pipeline:

1. Read raw IDAT files (`minfi::read.metharray.exp`).
2. Normalise (SWAN, functional normalisation, BMIQ).
3. Detect and remove outlier samples and cross-reactive probes.
4. Differential methylation at probe level via limma on M-values.
5. Region-level analysis (DMRcate, bumphunter) for smoother small-effect detection.

12.22.4 Assumptions

Array probes are well-designed (no major cross-reactivity); batch effects are identified and modelled; cell-type composition is adjusted for in blood or other heterogeneous samples — failure to do so creates spurious associations.

12.22.5 R Implementation

```
library(limma)

set.seed(2026)
n_cpgs ← 1000; n_samples ← 12
M_values ← matrix(rnorm(n_cpgs * n_samples, mean = 0, sd = 2),
                  n_cpgs, n_samples)
rownames(M_values) ← paste0("cg", 1:n_cpgs)
colnames(M_values) ← paste0("S", 1:n_samples)

group ← factor(rep(c("ctrl", "case"), each = 6))
# Inject differential methylation in 50 CpGs
M_values[1:50, group == "case"] ← M_values[1:50, group == "case"] + 1.5

design ← model.matrix(~ group)
fit ← lmFit(M_values, design)
fit ← eBayes(fit)

topTable(fit, coef = 2, n = 5)
summary(decideTests(fit))
```

12.22.6 Output & Results

`topTable()` returns a per-CpG table with log fold change in M-value, raw p-value, and BH-adjusted p-value. The 50 injected CpGs should appear at the top of the ranked list with the largest absolute fold changes and most significant p-values.

12.22.7 Interpretation

A reporting sentence: "Epigenome-wide differential methylation analysis (limma on M-values, design adjusting for age and estimated cell-type composition) identified 1{,}240 differentially methylated CpGs at Bonferroni $p < 10^{-7}$; 62 % were hypermethylated in cases, concentrated in CpG islands of known tumour-suppressor loci. Region-level analysis (DMRcate) identified 87 differentially methylated regions overlapping 64 unique genes." Always state the multiple-testing correction, cell-type adjustment strategy, and use of M-values vs beta-values.

12.22.8 Practical Tips

- Fit regression models on M-values (homoscedastic), but report effect sizes in beta-values (more interpretable as "% methylation difference") for clinical audiences.
- Cell-type deconvolution (`EpiDISH`, `FlowSorted.Blood.450k`) is mandatory for blood-based EWAS; unadjusted results predominantly reflect composition differences.
- Region-level analysis (DMRcate, bumphunter) is more powerful than probe-level for small consistent effects across nearby CpGs; it also reduces multiple-testing burden.
- Batch effects (array plate, chip, scan date) are very strong in methylation data; include as covariates or run ComBat.
- For age-related studies, the Horvath, Hannum, or PhenoAge clocks give DNA methylation age estimates that are biologically interpretable; report alongside chronological age.
- Standardised EWAS pipelines (`meffil`, `PaCES`) offer one-call QC and analysis with sensible defaults.

12.22.9 R Packages Used

`minfi` for IDAT reading, normalisation, and basic differential analysis; `limma` for linear-modelling on M-values; `DMRcate` for region-level differential methylation; `EpiDISH` and `FlowSorted.Blood.450k` for cell-type deconvolution; `bumphunter` for an alternative region detector; `meffil` for an end-to-end EWAS pipeline; `methylKit` for WGBS analysis.

12.22.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.22.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.23 Microbiome Analysis with DADA2

12.23.1 Introduction

DADA2 (Divisive Amplicon Denoising Algorithm 2) infers exact amplicon sequence variants (ASVs) from 16S rRNA, 18S, ITS, or other amplicon sequencing data. It replaces the older OTU-clustering approach (which lumps similar sequences together at, typically, 97 % identity) with single-nucleotide resolution. ASVs are reproducible across studies — the same sequence is the same variant regardless of who runs the analysis — and allow finer taxonomic assignment than the threshold-based OTU representation.

12.23.2 Prerequisites

A working understanding of amplicon sequencing (paired-end or single-end Illumina), primer trimming, and basic microbial ecology including the OTU vs ASV distinction.

12.23.3 Theory

DADA2 fits a position- and quality-aware error model from the observed FASTQ data, then uses it to distinguish real biological variants from sequencing errors. The intuition: a sequence appearing more often than the error model predicts is a candidate ASV; sequences explained by errors from a more abundant variant are merged into it. The final output is a per-sample ASV count table (analogous to a gene count matrix in RNA-seq) that feeds the same downstream pipeline — alpha and beta diversity, differential-abundance testing, ordination.

The standard pipeline: filter and trim FASTQs, learn error rates, denoise per sample, merge paired-end reads, construct a sequence table, remove chimeras, assign taxonomy.

12.23.4 Assumptions

Primers have been removed before DADA2 input; paired-end reads overlap sufficiently for merging; error rates are sample-specific (DADA2 estimates them per run).

12.23.5 R Implementation

DADA2 processes FASTQs on disk; here we simulate a post-DADA2 ASV count table for downstream analysis:

```
library(phyloseq)

set.seed(2026)
n_asv <- 80; n_samples <- 20
otu_tab <- matrix(rpois(n_asv * n_samples, lambda = 10), n_asv, n_samples)
rownames(otu_tab) <- paste0("ASV", 1:n_asv)
colnames(otu_tab) <- paste0("S", 1:n_samples)

tax_tab <- matrix(sample(c("Firmicutes", "Bacteroidetes", "Proteobacteria"),
                         n_asv, replace = TRUE), n_asv, 1)
rownames(tax_tab) <- rownames(otu_tab); colnames(tax_tab) <- "Phylum"

meta <- data.frame(group = factor(rep(c("ctrl", "case"), each = 10)),
                   row.names = colnames(otu_tab))

ps <- phyloseq(otu_table(otu_tab, taxa_are_rows = TRUE),
               tax_table(tax_tab),
               sample_data(meta))

sort(sample_sums(ps), decreasing = TRUE)[1:5]   # per-sample depth
taxa_sums(ps)[1:5]                              # per-ASV total
```

The actual DADA2 calls (commented for the simulation example):

```
# filterAndTrim(fnFs, filtFs, fnRs, filtRs, truncLen = c(240, 160),
#               maxEE = c(2, 2), truncQ = 2, compress = TRUE)
# errF <- learnErrors(filtFs); errR <- learnErrors(filtRs)
# dadaFs <- dada(filtFs, err = errF); dadaRs <- dada(filtRs, err = errR)
# mergers <- mergePairs(dadaFs, filtFs, dadaRs, filtRs)
# seqtab <- makeSequenceTable(mergers)
# seqtab.nochim <- removeBimeraDenovo(seqtab, method = "consensus")
```

12.23.6 Output & Results

The DADA2 output is an ASV-by-sample count matrix together with taxonomy assignments from SILVA or GTDB. Wrapping the count table, taxonomy, and sample metadata in a `phyloseq` object provides the standard input for downstream community ecology analyses.

12.23.7 Interpretation

A reporting sentence: “DADA2 inferred 2{,}148 amplicon sequence variants across 50 samples after chimera removal; 980 ASVs were assigned to phylum level using the SILVA v138 reference, with median read depth of 42{,}300 reads per sample. The resulting `phyloseq` object served as input to alpha-diversity (Shannon, Simpson) and beta-diversity (Bray-Curtis NMDS) analyses.” Always state the reference database version and the read-retention rate.

12.23.8 Practical Tips

- Always run `removeBimeraDenovo()` after sequence-table construction to strip chimeric ASVs; they accumulate during PCR and can be a substantial fraction of the count.

- Assign taxonomy with SILVA (v138 or later for 16S) or GTDB; species-level assignment for short amplicons is usually unreliable due to limited resolution.
- Filter ASVs present in fewer than 2 samples to reduce noise; very rare ASVs are often sequencing artefacts.
- `DECIPHER::IdTaxa()` provides a faster alternative to DADA2's `assignTaxonomy()` and supports the same SILVA reference.
- Always check per-sample read-retention at each DADA2 step (filtering, denoising, merging, chimera removal); large drops at any step usually mean over-strict parameters.
- For 16S V4 region with overlapping 250 bp paired-end reads, `truncLen = c(240, 160)` is a sensible starting point; tune based on quality profiles.

12.23.9 R Packages Used

`dada2` for ASV inference and the canonical workflow; `phyloseq` for downstream community-ecology analysis on the ASV count table; `DECIPHER` for fast taxonomy assignment; `vegan` for alpha and beta diversity; `ANCOMBC` and `ALDEx2` for compositional differential-abundance testing.

12.23.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.23.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.24 Microbiome Diversity Metrics

12.24.1 Introduction

Diversity metrics summarise microbial community composition. **Alpha diversity** measures richness and evenness within a sample; **beta diversity** measures compositional dissimilarity between samples. Both are essential for comparing communities and detecting treatment, disease, or niche effects.

12.24.2 Prerequisites

ASV or OTU count table; phylogenetic tree (for UniFrac); sample metadata.

12.24.3 Theory

Alpha diversity: - **Richness** (# ASVs observed). - **Shannon index** $H = -\sum p_i \log p_i$ – evenness-weighted. - **Simpson** $1 - \sum p_i^2$ – inverse concentration. - **Faith's PD** – phylogenetic tree branch length sum.

Beta diversity: - **Bray-Curtis** – abundance-weighted Manhattan dissimilarity. - **Jaccard** – presence/absence. - **UniFrac (weighted / unweighted)** – phylogeny-aware.

Permutational MANOVA (`adonis2`) tests whether group membership explains beta diversity.

12.24.4 Assumptions

Counts are non-negative; rarefaction or equivalent normalisation is applied before alpha diversity if depths vary widely; phylogenetic tree matches ASV labels.

12.24.5 R Implementation

```
library(vegan)

set.seed(2026)
n_samples ← 20; n_asv ← 50
counts ← matrix(rpois(n_asv * n_samples, lambda = 10),
                n_samples, n_asv)   # samples x ASVs

# Inject group differences
group ← factor(rep(c("ctrl", "case"), each = 10))
counts[group == "case", 1:15] ← counts[group == "case", 1:15] + 20

shannon ← diversity(counts, index = "shannon")
simpson ← diversity(counts, index = "simpson")
richness ← rowSums(counts > 0)

by(shannon, group, median)

bray ← vegdist(counts, method = "bray")
adonis_res ← adonis2(bray ~ group, permutations = 999)
adonis_res
```

12.24.6 Output & Results

Shannon / Simpson / richness per sample (group medians printed), and a PERMANOVA on Bray-Curtis showing whether group explains dissimilarity.

12.24.7 Interpretation

"Case samples had lower Shannon diversity (median 2.1 vs 2.9, Wilcoxon p = 0.004) and PERMANOVA on Bray-Curtis revealed significant community-level differences (R^2 = 0.18, p = 0.001)."

12.24.8 Practical Tips

- Rarefy to equal depth before alpha diversity to avoid depth-driven confounding; alternatively use rarefaction curves to confirm coverage.
- `phyloseq::estimate_richness` computes many alpha metrics in one call.
- Weighted UniFrac captures abundance-weighted phylogenetic distance; unweighted is presence-only.
- PERMANOVA is sensitive to dispersion heterogeneity; run `betadisper` to confirm equal dispersion before interpreting.
- PCoA visualisation on the beta dissimilarity matrix complements PERMANOVA.

12.24.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.24.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.25 Multiple Sequence Alignment

12.25.1 Introduction

Multiple sequence alignment (MSA) is the precursor to phylogeny, motif discovery, and structural modelling: it arranges residues so homologous positions align in columns. The `msa` Bioconductor package offers ClustalW, ClustalOmega, and Muscle under a unified R interface.

12.25.2 Prerequisites

FASTA sequences; substitution scoring (PAM, BLOSUM for proteins; match/mismatch for DNA).

12.25.3 Theory

Progressive alignment (ClustalW): build a guide tree from pairwise distances, then align sequences following the tree from leaves to root. **Iterative refinement** (Muscle, MAFFT) revisits partial alignments to improve scores. **Consistency-based** methods (T-Coffee) combine information from multiple library alignments.

Accuracy depends on sequence divergence: for highly divergent sequences, structural alignment (DALI for proteins) outperforms sequence-only methods.

12.25.4 Assumptions

Sequences are homologous; a single substitution model approximates the evolutionary process; insertions/deletions are rare enough for gap parameters to be meaningful.

12.25.5 R Implementation

```
library(Biostrings); library(msa)

# Example amino-acid sequences (hemoglobin across species)
aas ← readAAStringSet(
  system.file("examples", "exampleAA.fasta", package = "msa"))
aas
aln ← msa(aas, method = "ClustalW", verbose = FALSE)
aln

# Summary of alignment
print(aln, show = "complete")

# Convert to phangorn for phylogeny
library(phangorn)
aln_phy ← as.phyDat(as(aln, "AAStringSet"), type = "AA")
dm ← dist.ml(aln_phy)
tree ← NJ(dm)
plot(tree, main = "NJ tree from MSA", cex = 0.8)
```

12.25.6 Output & Results

An MSA object with aligned sequences and gap characters; a consensus is derivable via `consensusString`.

12.25.7 Interpretation

“ClustalW alignment of the hemoglobin orthologs reveals conserved haem-binding residues and lineage-specific substitutions on the surface helices.”

12.25.8 Practical Tips

- For > 1000 sequences, MAFFT (outside R) is substantially faster than Biostrings/msa.
- Alignment quality matters more than tree-building algorithm; a bad MSA produces a bad phylogeny.
- Trim alignment columns with many gaps (e.g., `trimAl`) before tree inference.
- Visualise alignments with `ggmsa` to identify problematic columns.
- Amino-acid alignments are often more reliable than codon alignments for distantly related sequences; use back-translation when codon-level analysis is needed.

12.25.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.25.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.26 Multi-Omics Integration

12.26.1 Introduction

Multi-omics experiments measure transcriptome, methylome, proteome, etc. on the same samples. Joint analysis exposes coordinated variation across layers that single-layer analyses miss – cis-regulatory effects of methylation, mRNA-protein correspondence, or integrated disease subtypes.

12.26.2 Prerequisites

Single-omic analysis (DE, methylation, proteomics); PCA / factor analysis basics.

12.26.3 Theory

MOFA (Multi-Omics Factor Analysis) learns latent factors explaining variation across omics layers. Each factor has per-layer weights: factors loading strongly on multiple layers capture shared signal; single-layer factors reflect layer-specific variation. MOFA2 adds sparsity for interpretability.

Similarity network fusion (SNF) and **iCluster+** offer alternative approaches; DIABLO (`mixOmics`) builds supervised multi-omic classifiers.

12.26.4 Assumptions

Samples are matched across layers; layer-specific noise and scaling are handled by the model; sufficient sample size for factor inference (typically > 50).

12.26.5 R Implementation

```
library(MOFA2)

set.seed(2026)
n ← 50
# Latent factor drives correlated variation across 2 layers
z ← rnorm(n)

# Layer 1: 200 features (e.g., RNA)
m_rna ← matrix(rnorm(200 * n, 0, 1), 200, n)
m_rna[1:20, ] ← m_rna[1:20, ] + outer(rep(1.5, 20), z)

# Layer 2: 100 features (e.g., methylation)
m_meth ← matrix(rnorm(100 * n, 0, 1), 100, n)
m_meth[1:10, ] ← m_meth[1:10, ] - outer(rep(1.2, 10), z)

colnames(m_rna) ← colnames(m_meth) ← paste0("S", 1:n)
rownames(m_rna)  ← paste0("rna_",  1:200)
rownames(m_meth) ← paste0("meth_", 1:100)

mofa ← create_mofa(list(RNA = m_rna, Methylation = m_meth))
# Illustrative: actual fitting requires Python backend (reticulate + mofapy2).
# model_opts ← get_default_model_options(mofa)
# model_opts$num_factors ← 3
# mofa_trained ← run_mofa(mofa, training_data_options = list(),
#                         model_options = model_opts, outfile = tempfile())

mofa
```

12.26.6 Output & Results

A trained MOFA model with latent-factor scores per sample, per-layer feature weights, and variance decomposition per factor x layer.

12.26.7 Interpretation

“MOFA identified 5 latent factors; factor 1 explained 32 % of RNA and 18 % of methylation variance, linked to cancer subtype; factor 2 was RNA-specific and reflected proliferation signature.”

12.26.8 Practical Tips

- Standardise each layer before MOFA (mean = 0, SD = 1) so no one layer dominates.
- Layer-specific factors often reflect technical artefacts; use them to diagnose batch effects.
- `DIABLO` in `mixOmics` is the better choice when integration is supervised by an outcome.
- MOFA requires Python via `reticulate`; test the install with `run_mofa2_docs()` before large runs.
- Interpret factor weights with domain knowledge; algorithmic factors without biology are uninformative.

12.26.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.26.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.27 KEGG Pathway Enrichment

12.27.1 Introduction

KEGG (Kyoto Encyclopedia of Genes and Genomes) is a curated database of metabolic and signalling pathways represented as directed graphs of genes and gene products. KEGG pathway enrichment asks whether the genes in a differential-expression hit list are over-represented in any of the curated pathways, identifying functional themes in the differential signal. The companion visualisation tool `pathview` overlays log-fold-change values directly onto the canonical KEGG pathway diagrams, making it straightforward to communicate which specific genes within a pathway drive the enrichment signal.

12.27.2 Prerequisites

A working understanding of differential-expression analysis, Entrez gene identifiers, and basic over-representation analysis using hypergeometric or Fisher's exact tests.

12.27.3 Theory

KEGG enrichment uses the same statistical framework as GO over-representation analysis: a hypergeometric (Fisher's exact) test for each pathway compares the proportion of DE genes assigned to the pathway against the proportion expected by chance from the universe of all annotated genes. Adjusted p-values (BH or Benjamini-Hochberg) control the false-discovery rate across the many pathways tested.

Pathway visualisation via `pathview` rendering colours each gene node in the canonical KEGG diagram by user-supplied log fold change (or any continuous statistic), producing an annotated PNG/PDF of the pathway map. The visual immediately exposes whether the enrichment is driven by a coordinated up- or down-regulation across the pathway versus a few isolated hits.

12.27.4 Assumptions

DE genes are correctly mapped to Entrez IDs; KEGG pathway annotations are current (KEGG is updated regularly; old annotations miss recent additions); tests are independent across pathways (BH adjustment is conservative for the typical correlated-pathway structure).

12.27.5 R Implementation

```
library(clusterProfiler); library(org.Hs.eg.db)

set.seed(2026)
all_entrez ← keys(org.Hs.eg.db, keytype = "ENTREZID")
de_entrez  ← sample(all_entrez, 200)

ek ← enrichKEGG(gene = de_entrez,
                organism = "hsa",
                pvalueCutoff = 0.1)
head(as.data.frame(ek))[, c("ID", "Description", "p.adjust", "Count")]

# Visualise a pathway with LFC overlay
# library(pathview)
# lfc_vec ← setNames(rnorm(length(de_entrez)), de_entrez)
# pathview(gene.data = lfc_vec, pathway.id = "hsa04110", species = "hsa")
```

12.27.6 Output & Results

`enrichKEGG()` returns an `enrichResult` object with one row per pathway, including the pathway ID and description, the count of DE genes, BH-adjusted p-value, and the comma-separated list of contributing gene symbols. `pathview()` writes a PNG of the pathway diagram with each gene node coloured by its supplied log-fold-change value.

12.27.7 Interpretation

A reporting sentence: “KEGG enrichment with `clusterProfiler::enrichKEGG` (organism = hsa, BH-adjusted $p < 0.05$) identified the cell cycle (hsa04110, $p_{\text{adj}} = 0.001$, 23 genes) and p53 signalling (hsa04115, $p_{\text{adj}} = 0.008$, 14 genes) as top enriched pathways; pathview overlay on hsa04110 showed consistent up-regulation of cyclins and CDKs alongside down-regulation of CDK inhibitors.” Always state the species code, the universe of background genes, and the multiple-testing correction.

12.27.8 Practical Tips

- KEGG functions in `clusterProfiler` require Entrez IDs; convert from gene symbols via `bitr(symbols, fromType = "SYMBOL", toType = "ENTREZID", OrgDb = org.Hs.eg.db)`.
- `enrichMKEGG()` uses KEGG modules (smaller, more specific functional units); it is preferable for metabolic studies where the broader KEGG pathway granularity is too coarse.
- `pathview()` requires internet access and writes output files to the working directory; treat each rendering as a one-off rather than running it in an automated

pipeline.

- KEGG organism codes: `hsa` for human, `mmu` for mouse, `rno` for rat, `dme` for fly, `cel` for worm; a complete list is at the KEGG website.
- Combine KEGG (broad pathways) with Reactome (`ReactomePA::enrichPathway`) for complementary coverage; the two databases overlap but emphasise different aspects of biology.
- For genes with associated fold-change estimates, GSEA on KEGG pathways (`gseKEGG()`) is more powerful than over-representation analysis because it uses the full ranked gene list rather than thresholding into hit and non-hit sets.

12.27.9 R Packages Used

`clusterProfiler` for `enrichKEGG()`, `enrichMKEGG()`, `gseKEGG()`, and the broader enrichment-analysis framework; `pathview` for KEGG pathway visualisation with LFC overlay; `org.Hs.eg.db` (and species-specific equivalents) for Entrez ID conversion; `ReactomePA` for Reactome alternatives; `enrichplot` for ggplot-based visualisations of enrichment results.

12.27.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.27.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.28 PCA of RNA-seq Samples

12.28.1 Introduction

Sample-level PCA on transformed RNA-seq counts is the standard quality-control step before differential-expression analysis. It reveals whether biological samples cluster by experimental condition (the desired outcome), by batch or technical covariates (a confounding signal that needs correction), or by neither (which can suggest excessive noise or sample mix-ups). Running PCA early — before any modelling — prevents the most consequential downstream errors and motivates the design choices for the differential-expression step.

12.28.2 Prerequisites

A working understanding of principal components analysis, RNA-seq count data, and the role of variance-stabilising transformations in equalising the variance-mean relationship of count data.

12.28.3 Theory

Raw RNA-seq counts have variance that grows with the mean, violating the assumption of equal variance that PCA implicitly relies on. Variance-stabilising transformations (VST and the older `rlog` in DESeq2) apply a non-linear transformation that approximately equalises variance across the expression range; the result is suitable for PCA, clustering, and other downstream methods that assume homoscedastic input.

PCA on the top-variance genes (typically the top 500 or so) highlights the dominant sources of variation. The proportion of variance explained by each PC indicates how much of the total variability is captured, and the score plot — samples coloured by metadata variables — exposes which design factors drive that variability.

12.28.4 Assumptions

Sufficient samples (rule of thumb: at least 6, ideally more) for meaningful PCA; counts have been variance-stabilised; the chosen number of top-variance genes is large enough to capture the structure but small enough to suppress noise.

12.28.5 R Implementation

```
library(DESeq2); library(ggplot2)

set.seed(2026)
counts ← matrix(rpois(20000 * 12, lambda = 30), nrow = 20000)
colnames(counts) ← paste0("s", 1:12)
coldata ← data.frame(condition = rep(c("A", "B"), each = 6),
                     batch = rep(c("X", "Y"), 6))

dds ← DESeqDataSetFromMatrix(counts, colData = coldata, design = ~ condition)
vsd ← vst(dds, blind = TRUE)

plotPCA(vsd, intgroup = c("condition", "batch"))
```

12.28.6 Output & Results

`plotPCA()` returns a ggplot of samples projected onto PC1 and PC2 with axis labels showing the proportion of variance explained. Colouring by `condition` and shape by `batch` (or vice versa) makes both factors visible simultaneously.

12.28.7 Interpretation

A reporting sentence: “Sample-level PCA on VST-transformed counts (top 500 variance genes) showed PC1 capturing 40 % of variance and clearly separating treatment conditions, while PC2 (12 %) showed a weaker batch effect. We included `batch` in the

differential-expression design to adjust for it." Always report the proportion of variance explained and what each PC corresponds to substantively.

12.28.8 Practical Tips

- Use `blind = TRUE` for QC PCA so the variance-stabilisation does not shrink toward the design; for downstream DE, the design-aware (`blind = FALSE`) transformation is appropriate.
- If batch dominates PC1, include batch in the DESeq2 (or limma) design; if batch overlaps perfectly with condition, the experiment is confounded and cannot be salvaged.
- Plot multiple metadata colourings (sex, age, RIN, library prep date) on the same PCA scatter to reveal hidden structure; outlier samples often correspond to specific covariates.
- Complement PCA with a sample-sample distance heatmap (`pheatmap(as.matrix(dist(t(assay(vsd)))))`); the two views agree on the broad story but emphasise different details.
- Outliers on PCA often indicate QC failures (low library complexity, contamination); investigate and consider exclusion before differential analysis.
- Compare PCA on raw counts, log-counts, and VST; raw counts have variance dominated by highly expressed genes, log-counts overcorrect, VST is calibrated for downstream analysis.

12.28.9 R Packages Used

`DESeq2` for `vst()`, `rlog()`, `plotPCA()` and the broader RNA-seq workflow; `pheatmap` for sample-distance heatmaps; `ggplot2` for custom PCA plots; `PCAtools` for richer PCA visualisations including biplots and loading plots.

12.28.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.28.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.29 Phylogenetic Trees with ape

12.29.1 Introduction

Phylogenetic trees depict the evolutionary relationships among sequences (species, strains, cells). They are central to taxonomy, molecular epidemiology, and cancer-clone tracing. `ape` is the standard R package for tree construction and manipulation, and `phangorn` adds maximum-likelihood and parsimony methods.

12.29.2 Prerequisites

Multiple sequence alignment; nucleotide or amino-acid substitution models.

12.29.3 Theory

Distance-based methods (neighbour-joining, UPGMA) compute pairwise distances from the alignment and build a tree that approximates those distances; fast but statistically crude.

Maximum-likelihood (ML) methods posit a substitution model (JC69, K80, GTR + Gamma + I) and find the tree topology + branch lengths maximising the likelihood of the observed alignment.

Bootstrapping resamples alignment columns to assess topology confidence; bootstrap values $\geq$ 70 % are commonly considered well-supported.

12.29.4 Assumptions

Alignment columns are homologous and independent; the substitution model captures key features (transition/transversion bias, rate heterogeneity).

12.29.5 R Implementation

```r
library(ape)

# Load an example alignment shipped with ape
data(woodmouse)          # DNAbin alignment of 15 sequences, 965 sites

# Distance matrix under K80 model
d <- dist.dna(woodmouse, model = "K80")

# Neighbour-joining tree
nj_tree <- nj(d)
plot(nj_tree, main = "Neighbour-joining tree (woodmouse)",
     cex = 0.6, font = 1)
add.scale.bar()

# Bootstrap support
bp <- boot.phylo(nj_tree, woodmouse,
                 function(x) nj(dist.dna(x, model = "K80")),
                 B = 100, quiet = TRUE)
plot(nj_tree, main = "NJ tree with bootstrap support",
     cex = 0.6, font = 1)
nodelabels(bp, cex = 0.5, frame = "none", adj = c(1.3, -0.3))
```

12.29.6 Output & Results

A neighbour-joining tree with branch lengths in substitutions per site; bootstrap values annotated at internal nodes.

12.29.7 Interpretation

“The neighbour-joining tree of the cytochrome b sequences recovers the regional subclades with bootstrap support > 80 %, consistent with the geographic sampling structure.”

12.29.8 Practical Tips

- For genome-scale data, use ML tools like `RAxML-NG` or `IQ-TREE` externally and import trees with `ape::read.tree()`.
- NJ is fast but sensitive to model misspecification; ML is the gold standard for small-moderate alignments.
- Use `phangorn::modelTest()` to pick a substitution model by AIC/BIC before running ML.
- Annotate trees with `ggtree` for publication-ready figures including metadata overlays.
- For large strain trees (SARS-CoV-2), use Nextstrain / `treetime` rather than base ape.

12.29.9 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.29.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.30 Population Genetics Basics

12.30.1 Introduction

Population genetics quantifies genetic variation within and between populations. Allele frequencies, Hardy-Weinberg equilibrium (HWE), and F-statistics form the foundational toolkit for diagnosing genotyping errors, detecting population structure, and assessing selection or drift.

12.30.2 Prerequisites

Mendelian inheritance; genotype coding (0/1/2 for minor-allele dosage).

12.30.3 Theory

Allele frequency. For a biallelic SNP with minor-allele counts n_{AA}, n_{Aa}, n_{aa}, $p = (2n_{AA} + n_{Aa})/(2n)$.

Hardy-Weinberg equilibrium (HWE). Under random mating and no selection, expected genotype frequencies are $p^2, 2pq, q^2$. Exact HWE tests (`HWExact`) are preferred over the chi-square for rare variants.

F-statistics. $F_{ST} = (H_T - H_S)/H_T$ partitions heterozygosity between and within populations, quantifying structure (Wright, 1951). Typical human F_{ST} between continents ≈ 0.1.

12.30.4 Assumptions

Random mating, no selection, no mutation, no migration, infinite population size; genotyping is unbiased.

12.30.5 R Implementation

```r
library(HardyWeinberg)

set.seed(2026)
# Simulate HWE: p = 0.3, n = 200 diploid individuals
p ← 0.3
n_AA ← rbinom(1, 200, p^2)
n_aa ← rbinom(1, 200, (1 - p)^2)
n_Aa ← 200 - n_AA - n_aa

gt ← c(AA = n_AA, AB = n_Aa, BB = n_aa)
HWExact(gt, verbose = TRUE)

# Simulate HWE violation (excess heterozygotes)
gt_violate ← c(AA = 20, AB = 160, BB = 20)
HWExact(gt_violate, verbose = TRUE)

# F_ST between two populations (manual calculation)
p1 ← 0.2; p2 ← 0.5
p_bar ← mean(c(p1, p2))
H_T ← 2 * p_bar * (1 - p_bar)
H_S ← mean(c(2 * p1 * (1 - p1), 2 * p2 * (1 - p2)))
Fst ← (H_T - H_S) / H_T
Fst
```

12.30.6 Output & Results

HWE exact test p-value (large for the simulated HWE case; ~0 for the violated case). $F_{ST} \approx 0.1$ for the two-population toy example.

12.30.7 Interpretation

"HWE exact tests excluded 1,240 variants (p < 1e-6) from association analysis as likely genotyping artefacts. Between-cohort $F_{ST} = 0.012$ indicates minimal population stratification after ancestry matching."

12.30.8 Practical Tips

- Always HWE-filter variants in GWAS pre-processing; strong HWE violations usually indicate genotyping errors.
- Test HWE in controls only (not cases) because true disease associations can cause HWE departures in cases.
- For low-MAF variants (<5%), asymptotic chi-square is unreliable; use the exact test.
- `adegenet` and `hierfstat` provide comprehensive F-statistics and population-structure utilities.
- F_{ST} between samples can be inflated by small sample size; apply Weir-Cockerham estimator and bootstrap CIs.

12.30.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.30.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.31 Protein Structure Prediction

12.31.1 Introduction

AlphaFold 2 (2021) achieved near-experimental accuracy in predicting protein 3-D structure from amino-acid sequence, transforming structural biology. RoseTTAFold and ESMFold offer competitive alternatives with different speed/accuracy trade-offs. Downstream analyses include structural alignment, docking, and mutation impact prediction.

12.31.2 Prerequisites

Amino-acid sequence; familiarity with protein structure (secondary/tertiary elements).

12.31.3 Theory

AlphaFold uses attention-based deep learning trained on PDB structures and multiple-sequence-alignment (MSA) features. The model outputs per-residue (x, y, z) coordinates plus a per-residue confidence score (**pLDDT**, 0-100) and an inter-residue confidence (**PAE**).

pLDDT > 70 = confident; 50-70 = low confidence; < 50 = disordered/unreliable.

12.31.4 Assumptions

The protein has evolutionary neighbours (sufficient MSA) or is not unusually divergent; post-translational modifications and co-factors are not modelled.

12.31.5 R Implementation

AlphaFold prediction itself runs outside R (ColabFold, AlphaFold Docker, OpenFold). R is used for structural analysis via `bio3d`:

```r
library(bio3d)

# Read a PDB (here a built-in example: human CDK2)
pdb <- read.pdb(system.file("examples/hivp.pdb", package = "bio3d"))
summary(pdb)

# Secondary structure, residue count, chains
head(pdb$atom[, c("resno", "resid", "chain", "elety")])

# Torsion angles
tor <- torsion.pdb(pdb)
head(tor$phi)

# Structural alignment of two chains (illustrative)
chain_a <- atom.select(pdb, chain = "A", elety = "CA")
chain_b <- atom.select(pdb, chain = "B", elety = "CA")
xyz_a <- pdb$xyz[chain_a$xyz]
xyz_b <- pdb$xyz[chain_b$xyz]
c(n_a = length(xyz_a) / 3, n_b = length(xyz_b) / 3)
```

12.31.6 Output & Results

A parsed PDB structure with atoms, residues, and chains accessible programmatically; torsion angles (phi, psi) define the backbone conformation.

12.31.7 Interpretation

“AlphaFold prediction of the novel kinase scored pLDDT 88.3 overall; the activation loop (residues 185-210) had pLDDT < 60, consistent with conformational flexibility seen in homologous kinases.”

12.31.8 Practical Tips

- Always check pLDDT per residue; low-confidence regions are unreliable for structural analysis.
- PAE plots reveal inter-domain uncertainty; two confident domains with high PAE between them means the relative orientation is unreliable.
- For mutation impact, combine AlphaFold + molecular-dynamics simulation with Gromacs or AMBER.
- AlphaFold-Multimer handles complexes; standard AlphaFold is monomer-focused.
- ESMFold is faster but less accurate for novel folds; use for screening large sequence sets.

12.31.9 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.31.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.32 Proteomics with MSstats

12.32.1 Introduction

MSstats is the standard Bioconductor package for statistical analysis of bottom-up mass-spectrometry proteomics. It ingests peptide-level intensity from any search engine (MaxQuant, Skyline, OpenMS, Spectronaut) and fits mixed-effects models with protein-level contrasts for label-free (LFQ), TMT, and SRM/PRM workflows.

12.32.2 Prerequisites

Peptide-level MS data; experimental-design matrix (condition, bioreplicate, run).

12.32.3 Theory

For each protein, MSstats fits a linear mixed-effects model with run/feature random effects and condition fixed effects; the contrast produces a log-fold change and a moderated t-test analogue. Missing-value handling uses accelerated failure-time imputation or censoring (AFT). LOG2 intensity is the response scale.

12.32.4 Assumptions

Peptide intensity is a valid surrogate for protein abundance; missingness is either completely at random (MCAR) or left-censored at a detection limit (MNAR).

12.32.5 R Implementation

```
library(MSstats)

# Simulated MSstats-format table: peptide intensity x run x condition
set.seed(2026)
proteins ← paste0("P", 1:20)
pep_per_prot ← 3
runs ← paste0("R", 1:6)
conditions ← rep(c("ctrl", "trt"), each = 3)

df ← expand.grid(Protein = proteins,
                 Feature = 1:pep_per_prot,
                 Run = runs,
                 stringsAsFactors = FALSE)
df$Condition    ← conditions[match(df$Run, runs)]
df$BioReplicate ← df$Run
df$Intensity    ← 2^(rnorm(nrow(df), mean = 20, sd = 1))

# Inject DE in first 5 proteins
trt_rows ← df$Condition == "trt" & df$Protein %in% proteins[1:5]
df$Intensity[trt_rows] ← df$Intensity[trt_rows] * 2

# Mock up the MSstats-required column names and minimum columns
msdf ← data.frame(
  ProteinName = df$Protein,
  PeptideSequence = paste0("PEP", df$Feature),
  PrecursorCharge = 2,
  FragmentIon = NA, ProductCharge = NA,
  IsotopeLabelType = "L",
  Condition = df$Condition,
  BioReplicate = df$BioReplicate,
  Run = df$Run,
  Intensity = df$Intensity
)

# MSstats workflow: processing + protein-level model + contrast
# processed ← dataProcess(msdf)
# comparison ← matrix(c(-1, 1), nrow = 1,
#                     dimnames = list("trt_vs_ctrl", c("ctrl", "trt")))
# results ← groupComparison(contrast.matrix = comparison, data = processed)
# head(results$ComparisonResult)
head(msdf, 3)
```

12.32.6 Output & Results

Per-protein log-fold change, standard error, p-value, and adjusted p-value; MSstats also outputs processed data diagnostic plots (profile plots, condition plots) for QC.

12.32.7 Interpretation

“MSstats identified 142 proteins differentially abundant at BH-FDR < 0.05 between treatment and control; the top hit (log2FC = 3.2) corresponds to the direct drug target.”

12.32.8 Practical Tips

- Provide replicate structure via `BioReplicate`; technical replicates must be distinct from biological replicates.

- Normalise within MSstats (`dataProcess(normalization = "equalizeMedians")`) rather than pre-normalising, to keep the pipeline single-entry.
- For TMT, use `MSstatsTMT`; the mixed-effects model differs due to isobaric design structure.
- Filter peptides to unique-to-protein before modelling; shared peptides inflate between-protein correlation.
- Report both LFC and adjusted p-value; very high LFC with wide CI (few peptides) is weak evidence.

12.32.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.32.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.33 Read Trimming and Adapter Removal

12.33.1 Introduction

After FASTQ-level QC has identified low-quality bases and adapter contamination, read trimming removes them before alignment. The motivation is straightforward: sequencing-quality scores typically degrade toward the 3′ end of reads, and adapter sequences appear when the insert is shorter than the read length. Both produce mismatched bases that bias alignment and increase mapping rates incorrectly. The `Rfastp` package provides an R-friendly interface to `fastp`, the fast C++ tool that handles trimming, adapter removal, quality filtering, and polytail cleanup in a single pass.

12.33.2 Prerequisites

A working understanding of FASTQ format, Phred quality scores, and Illumina adapter sequences (TruSeq, Nextera).

12.33.3 Theory

Standard trimming operations:

- **Quality trimming**: scan from the 3′ end (and optionally 5′) and remove bases falling below a Phred threshold; sliding-window variants trim when the mean quality in a

window drops below a cutoff.
- **Adapter removal**: detect known adapter sequences via overlap matching with the read 3′ end and trim them out. Auto-detection (matching to a library of common adapters) is usually reliable.
- **Length filtering**: discard reads that fall below a minimum length after trimming; very short reads align ambiguously and waste downstream resources.
- **Polytail trimming**: remove poly-A or poly-G contamination, common in NextSeq/NovaSeq dark-cycle artefacts.

12.33.4 Assumptions

Adapter sequences are known or auto-detectable; quality scores are on the standard Phred+33 encoding (true for all modern Illumina output).

12.33.5 R Implementation

```
library(Rfastp)

in_file  ← system.file("extdata", "reads1.fastq.gz", package = "Rfastp")
out_file ← tempfile(fileext = ".fastq.gz")

# Trim with fastp default heuristics
out ← rfastp(read1 = in_file, outputFastq = out_file,
             thread = 2,
             trimAdapter = TRUE, overlapDiff = 5)

out$summary
```

12.33.6 Output & Results

`rfastp()` returns a summary list with before/after statistics: reads passing filter, reads with adapters trimmed, mean read length before and after, base content. The HTML report (auto-generated by `fastp`) provides a richer visualisation of the trimming impact.

12.33.7 Interpretation

A reporting sentence: “Read trimming with fastp removed Illumina TruSeq adapters from 3.2 % of reads and quality-trimmed bases below Phred 15; 98 % of reads were retained after trimming, with mean read length decreasing from 150 to 145 bp.” Always report the adapter set, quality threshold, and retention rate.

12.33.8 Practical Tips

- Keep trimming settings mild — aggressive quality filtering removes data without proportional gains in downstream accuracy.
- Adapter auto-detection in `fastp` is usually reliable; specify adapter sequences explicitly only when working with non-standard library prep kits.
- For paired-end reads, trim both mates simultaneously and retain only properly-paired survivors; `Rfastp::rfastp()` with `read1` and `read2` arguments handles this automatically.

- Validate downstream QC (FastQC, MultiQC) after trimming; a successful trim improves per-base quality, per-base content, and adapter-content panels.
- Record trimming parameters in a methods document or workflow file (Snakemake, Nextflow); reproducibility requires exact tool versions and parameters.
- For UMI-tagged libraries (single-cell, low-input), use UMI-aware tools (`umi_tools`, fastp's UMI mode) before quality trimming to preserve UMI integrity.

12.33.9 R Packages Used

`Rfastp` for the canonical fastp wrapper; `ShortRead` for general FASTQ manipulation in R; `Biostrings` for adapter-sequence handling; `QuasR` for an integrated alignment workflow that includes trimming as a step.

12.33.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.33.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.34 RNA-seq Normalisation

12.34.1 Introduction

Raw RNA-seq read counts depend on library size and library composition; normalisation removes these technical sources of variation so that samples can be compared on equal footing. The standard methods — TMM (edgeR), RLE (DESeq2), upper-quartile, and CPM — differ in how aggressively they handle compositional differences and which assumption they make about the bulk of genes. Choosing the right normalisation is foundational; downstream differential-expression tools incorporate the normalisation factors as offsets in their GLM, so a poor choice undermines every conclusion that follows.

12.34.2 Prerequisites

A working understanding of count data, library size, and the difference between technical (sequencing depth) and biological (true expression) variation between samples.

12.34.3 Theory

The four standard methods:

- **CPM (counts per million)**: simple division of each count by the total library size in millions. Adjusts only for sequencing depth; does not address compositional differences (where one sample has a few highly expressed genes pulling resources from others).
- **TMM (Trimmed Mean of M-values)**: edgeR's default. Computes log-fold-changes between each sample and a reference, trims extreme values, and uses the trimmed mean as the normalisation factor. Robust to highly expressed compositional outliers.
- **RLE (Relative Log Expression)**: DESeq2's `estimateSizeFactors`. Takes the median of the ratio of each gene's count to its geometric mean across samples. Equivalent to TMM in spirit; both adjust for composition.
- **Upper-quartile**: scaling by the 75th-percentile count; less robust than TMM/RLE but useful for very sparse data.

TMM and RLE are the de facto standards; CPM is appropriate only for visualisation, not for statistical comparison.

12.34.4 Assumptions

Most genes are not differentially expressed (the "majority is unchanged" assumption that motivates trimmed-mean and median-based normalisation); samples are technical replicates of comparable biological material.

12.34.5 R Implementation

```
library(edgeR); library(DESeq2)

counts ← matrix(rpois(1000, lambda = 10), 100, 10)
rownames(counts) ← paste0("g", 1:100)

# edgeR TMM
dge ← DGEList(counts = counts)
dge ← calcNormFactors(dge, method = "TMM")
dge$samples$norm.factors

# DESeq2 size factors
dds ← DESeqDataSetFromMatrix(counts, colData = DataFrame(id = 1:10),
                             design = ~ 1)
dds ← estimateSizeFactors(dds)
sizeFactors(dds)
```

12.34.6 Output & Results

`calcNormFactors()` returns per-sample TMM normalisation factors (centred near 1, with deviations indicating compositional differences); `estimateSizeFactors()` returns DESeq2 size factors with similar interpretation. Both are used as offsets in downstream GLMs by edgeR, DESeq2, and limma-voom.

12.34.7 Interpretation

A reporting sentence: “TMM normalisation factors ranged from 0.85 to 1.18 across the 10 samples, with no factor exceeding the conventional ±20 % concern threshold; the corresponding DESeq2 RLE size factors agreed within 5 %, confirming the normalisation is robust to method choice.” Quote the range of normalisation factors; extreme values signal compositional anomalies.

12.34.8 Practical Tips

- TMM and RLE adjust for compositional differences; CPM does not. For statistical comparison, always use TMM or RLE; CPM is only appropriate for visualisation.
- Extreme normalisation factors (above 2 or below 0.5) warrant investigation; they often indicate a few very highly expressed genes dominating a sample.
- For single-cell RNA-seq, different normalisation methods apply (scran’s pooling-based size factors, sctransform’s regularised regression); bulk methods underperform on the sparser, lower-coverage single-cell data.
- Normalisation factors enter the downstream GLM as an offset: `log(\hat\mu_{ij}) = \log(\mathrm{NF}_j \cdot \ell_j) + X_i^\top \beta`. The DE method handles this automatically.
- TMM with `refColumn` set to a specific reference sample is useful when one sample is known to be representative; the default chooses an automatic reference.
- For mixed-tissue or very heterogeneous experiments, the “majority unchanged” assumption can fail; consider quantile normalisation (`limma::normalizeQuantiles`) or condition-aware alternatives.

12.34.9 R Packages Used

`edgeR::calcNormFactors()` for TMM, upper-quartile, and RLE alternatives; `DESeq2::estimateSizeFactors()` for the canonical RLE; `limma::cpm()` and `voom()` for normalisation in the limma framework; `scran` and `sctransform` for single-cell normalisation.

12.34.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.34.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.35 Pseudoalignment with Salmon and Kallisto

12.35.1 Introduction

Pseudo-alignment tools — Salmon, Kallisto, and the related selective-alignment variant in newer Salmon versions — quantify transcript-level expression without producing full BAM-file alignments. They match read k-mers to transcript fingerprints in an indexed reference and resolve multi-mappers via expectation-maximisation. The result is a 10-to-100-fold speedup over STAR or HISAT2 with comparable accuracy at the gene level, making pseudo-alignment the standard fast-path for RNA-seq quantification when isoform-level alignment is not the goal.

12.35.2 Prerequisites

A working understanding of RNA-seq, the difference between transcript-level and gene-level quantification, and basic familiarity with FASTQ inputs and gene-annotation formats.

12.35.3 Theory

Pseudo-aligners build an index of all transcripts in the reference, splitting them into k-mers (typically $k = 31$). At quantification time, each read is broken into k-mers, the compatible transcripts are identified, and an expectation-maximisation algorithm distributes ambiguous reads across compatible transcripts in proportion to expected expression. The output is per-transcript counts and TPM; gene-level aggregation uses a transcript-to-gene map via `tximport` or `tximeta`.

Salmon adds bias-correction flags (`--gcBias`, `--seqBias`, `--posBias`) that calibrate counts for GC content, hexamer priming bias, and positional sequencing bias. Decoy-aware indexing (using the genome as a decoy alongside the transcriptome) reduces spurious alignment to paralogues and improves accuracy for difficult regions.

12.35.4 Assumptions

A transcript annotation matched to the genome; polyA-selected or otherwise enriched mRNA library; reads predominantly map to known transcripts (pseudo-alignment cannot detect novel transcripts or junctions).

12.35.5 R Implementation

```
library(tximport); library(tximeta)

# Salmon output: quant.sf files per sample
# files ← c("s1/quant.sf", "s2/quant.sf")
# tx_map ← read.table("tx2gene.tsv", header = TRUE)
#
# txi ← tximport(files, type = "salmon", tx2gene = tx_map)
# dim(txi$counts)
#
# library(DESeq2)
# dds ← DESeqDataSetFromTximport(txi, colData = coldata, design = ~ condition)
```

12.35.6 Output & Results

Salmon and Kallisto produce per-sample `quant.sf` (Salmon) or `abundance.tsv` (Kallisto) files containing transcript-level counts, TPM, and effective length. `tximport()` aggregates these to gene level with optional length-scaled TPM correction; the resulting matrix flows directly into DESeq2, edgeR, or limma-voom.

12.35.7 Interpretation

A reporting sentence: "Salmon (v1.10) with selective alignment, decoy-aware indexing, and `--gcBias --seqBias` flags quantified 45{,}000 transcripts per sample in approximately 3 minutes per sample (vs. 30 minutes for STAR); tximport aggregation yielded 22{,}000 protein-coding genes for DESeq2 analysis." Always state the bias-correction flags and the decoy strategy.

12.35.8 Practical Tips

- Use decoy-aware indexing (Salmon) by including the genome as a decoy alongside the transcriptome; this reduces spurious mapping to paralogous regions.
- Enable bias-correction flags (`--gcBias` , `--seqBias`) routinely; they improve accuracy without meaningful runtime cost.
- For transcript-level differential expression with uncertainty, use bootstrap replicates (Salmon's `--numBootstraps` or Kallisto's `bootstrap-samples`) and analyse with sleuth or swish (`fishpond`).
- `tximeta` extends `tximport` with automatic transcript-to-gene mapping based on the Salmon index's metadata; use it to avoid annotation-mismatch errors.
- Pseudo-alignment is unsuitable for variant calling, novel-junction detection, or splicing analysis; use STAR with the genome reference for these tasks.
- For single-cell RNA-seq, use alevin (in Salmon) or kallisto-bustools; they extend pseudo-alignment to the cell-barcode and UMI structure of droplet-based single-cell data.

12.35.9 R Packages Used

`tximport` for transcript-to-gene aggregation; `tximeta` for metadata-aware aggregation; `fishpond::swish()` for transcript-level DTE with bootstrap inferential replicates; `DRIMSeq` and `DEXSeq` for differential transcript usage. Salmon and Kallisto are command-line tools; their output integrates with all standard Bioconductor RNA-seq workflows.

12.35.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.35.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.36 Cell-Type Annotation

12.36.1 Introduction

After clustering, each cluster must be labelled with a biological identity: T cell, B cell, macrophage, tumour epithelial cell, neuron subtype. Cell-type annotation is the bridge between unsupervised clustering and biological interpretation. Two main approaches compete: reference-based methods (SingleR, Azimuth, scType) that score each cell or cluster against curated reference profiles, and manual annotation that inspects marker genes against literature and reference atlases.

12.36.2 Prerequisites

A working understanding of clustered scRNA-seq output, marker-gene analysis, and the available cell-type reference atlases (Human Cell Atlas, Tabula Muris, Monaco Immune, ENCODE).

12.36.3 Theory

SingleR computes per-cell Spearman correlations against each reference profile over reference-defined marker genes; cells are labelled with the highest-scoring reference label. A pruning step removes calls below a confidence threshold. The method is fast, automated, and produces both per-cell and per-cluster summaries.

Azimuth uses a transfer-learning approach: it projects query cells into the reference's latent space (typically a PCA + harmony-corrected representation) and transfers labels via nearest-neighbour matching. It is the modern Seurat-team-recommended approach when a curated reference atlas exists for the tissue.

Manual annotation inspects each cluster's top marker genes against curated marker lists (canonical T-cell markers: CD3D, CD3E; B cells: CD19, MS4A1; macrophages: CD68, CD163). Cluster-by-cluster manual annotation is essential when references are missing or the tissue is non-standard.

12.36.4 Assumptions

The reference atlas matches the tissue and species of interest; cell types present in the data are represented in the reference (rare or unusual states may be absent and require manual annotation).

12.36.5 R Implementation

```
library(SingleR); library(celldex)

# Reference: built-in Monaco immune cells
ref ← MonacoImmuneData()

# Toy query: simulate a cell-by-gene expression matrix
set.seed(2026)
query ← matrix(rpois(200 * 100, lambda = 2), 200, 100)
rownames(query) ← sample(rownames(ref), 200)   # match gene names
colnames(query) ← paste0("c", 1:100)

pred ← SingleR(test = query, ref = ref,
               labels = ref$label.main,
               de.method = "wilcox")

table(pred$labels)
head(pred[, c("labels", "pruned.labels")])
```

12.36.6 Output & Results

`SingleR()` returns per-cell predicted labels, pruned labels (high-confidence subset), and the underlying per-reference-label scores. Summary tables show predicted cell-type counts; `pruneScores()` allows fine-tuning of the confidence threshold.

12.36.7 Interpretation

A reporting sentence: "SingleR with the Monaco Immune reference assigned 62 % of cells to canonical PBMC types (T cells, B cells, NK cells, monocytes, dendritic cells); pruning removed 8 % low-confidence calls. The remaining 30 % unannotated cells were tumour-associated stromal cells confirmed manually using ACTA2, COL1A1, and CD90 markers." Always report the reference, pruning rate, and manual validation of automated calls.

12.36.8 Practical Tips

- Use a reference matching the tissue source — never annotate skin cells against a blood reference; the cell types simply do not match.
- Report pruned labels rather than raw labels; pruned labels exclude low-confidence calls that often reflect intermediate or unusual states.
- For novel tissues or rare states, manual annotation outperforms reference-based methods; the references cannot label what they have not seen.
- Azimuth (`SeuratDisk::LoadH5Seurat` + `MapQuery`) is the modern Seurat alternative; it handles batch differences between query and reference more gracefully than SingleR.
- Cross-validate automated cluster labels against marker gene expression; a cluster labelled "T cell" by SingleR but without CD3D/CD3E expression is suspect and warrants investigation.
- For very granular reference atlases (Human Cell Atlas with 100+ cell types), use cluster-level SingleR rather than per-cell to reduce noise.

12.36.9 R Packages Used

`SingleR` for the canonical reference-based annotation; `celldex` for built-in reference datasets (Monaco, Encode, Hematopoietic Cell Atlas, etc.); `Azimuth` for Seurat-flavoured transfer learning; `scType` for marker-database-based annotation; `clustifyr` for an alternative reference-based approach.

12.36.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.36.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.37 scRNA Clustering

12.37.1 Introduction

Single-cell RNA-seq clustering groups cells into putative cell types or transcriptional states based on transcriptome similarity. The dominant approach is graph-based community detection: construct a shared-nearest-neighbour (SNN) graph of cells in PCA space, then apply the Louvain or Leiden algorithm to identify densely connected communities. The approach scales to millions of cells, produces interpretable cluster boundaries, and is the standard in both Seurat and Bioconductor scRNA-seq workflows.

12.37.2 Prerequisites

A working understanding of normalised scRNA-seq counts, dimensionality reduction (PCA), and the concept of community detection in graphs.

12.37.3 Theory

The standard pipeline:

1. **K-nearest-neighbour graph** in reduced PCA space: each cell is connected to its k nearest neighbours (typically $k = 20$).
2. **Shared-nearest-neighbour (SNN) graph**: edge weights between cell pairs are determined by the number of shared nearest neighbours, emphasising robust local structure over noisy individual neighbour relationships.

3. **Community detection**: Louvain optimises modularity; Leiden refines Louvain to guarantee well-connected communities and has become the modern default.

The `resolution` parameter controls granularity: higher resolution yields more, smaller clusters; lower resolution yields fewer, larger ones. There is no universally correct resolution — the choice depends on the biological question and is typically validated by marker-gene analysis.

12.37.4 Assumptions

Cell types or states form reasonably discrete communities in PCA space; the SNN graph captures local biological structure; the chosen number of PCs (typically 10–50) captures the major biological variation.

12.37.5 R Implementation

```
library(Seurat)

set.seed(2026)
counts ← matrix(rpois(300 * 200, lambda = 2), 200, 300)
rownames(counts) ← paste0("g", 1:200)
colnames(counts) ← paste0("c", 1:300)

so ← CreateSeuratObject(counts)
so ← NormalizeData(so, verbose = FALSE)
so ← FindVariableFeatures(so, nfeatures = 100, verbose = FALSE)
so ← ScaleData(so, verbose = FALSE)
so ← RunPCA(so, npcs = 10, verbose = FALSE)
so ← FindNeighbors(so, dims = 1:10, verbose = FALSE)

so ← FindClusters(so, resolution = 0.3, verbose = FALSE)
n_low ← nlevels(Idents(so))

so ← FindClusters(so, resolution = 1.2, verbose = FALSE)
n_high ← nlevels(Idents(so))

c(low_res = n_low, high_res = n_high)
```

12.37.6 Output & Results

`FindClusters()` returns the same `Seurat` object with cluster identities added to the active idents and to per-cell metadata. Different resolutions produce different cluster counts; the `clustree` package visualises how cells move between clusters as resolution changes.

12.37.7 Interpretation

A reporting sentence: “Leiden clustering on the SNN graph (`algorithm = 4`) at resolution 0.5 produced 12 clusters spanning the major immune cell-type families; clusters were annotated by canonical marker genes (CD3 for T cells, CD19 for B cells, CD14 for monocytes) before downstream differential-expression analysis.” Always describe the algorithm, resolution, and PCs used.

12.37.8 Practical Tips

- Explore multiple resolutions and visualise cluster relationships across resolutions with `clustree::clustree()`; cluster stability across nearby resolutions is reassuring.
- Use Leiden (`algorithm = 4` in Seurat or `cluster_leiden()` in `igraph`) over Louvain — Leiden guarantees connected communities, which Louvain does not.
- Never cluster directly on UMAP coordinates; UMAP distorts global distances and clustering on it produces unstable, geometry-dependent results. Always cluster on PCA space.
- The "correct" number of clusters is a biological judgement, not a statistical one; use marker-gene analysis to confirm whether clusters correspond to known or plausible cell types.
- For very large datasets ($> 500{,}000$ cells), use Seurat v5 sketch-based clustering or Bioconductor's `bluster` package with subsampling for tractable computation.
- Compare clustering to integration-aware methods (`harmony`, `Symphony`) when batch effects span multiple datasets; clustering on uncorrected data confounds batch with biology.

12.37.9 R Packages Used

`Seurat::FindClusters()` for graph-based clustering with Louvain/Leiden; `bluster` for Bioconductor-style clustering with multiple algorithms; `clustree` for cross-resolution stability visualisation; `igraph::cluster_leiden()` for the underlying Leiden implementation; `harmony` for batch correction before clustering.

12.37.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.37.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.38 Integration with Harmony

12.38.1 Introduction

Datasets from different batches, donors, or technologies cluster by batch rather than biology without correction. Integration methods align shared cell types across datasets while preserving dataset-specific populations. Harmony iteratively centres cell embeddings by batch; Seurat's CCA/RPCA anchors cells between batches.

12.38.2 Prerequisites

Multiple scRNA-seq datasets with shared cell populations; normalised expression; PCA embeddings.

12.38.3 Theory

Harmony alternates (i) soft cluster assignment in PCA space, (ii) per-cluster batch centroid computation, (iii) shifting cells within each cluster by its batch-specific correction vector. Converges in a few iterations and scales to millions of cells.

Seurat CCA/RPCA integration finds anchor pairs across datasets via mutual nearest neighbours in a shared low-dimensional space and uses anchors to transform embeddings.

12.38.4 Assumptions

Shared cell types span batches; batch effects are corrective in PCA space; biology and batch are not perfectly confounded.

12.38.5 R Implementation

```
library(harmony); library(Seurat)

set.seed(2026)
# Simulate two batches with shared biology + batch shift
n ← 150
batch ← rep(c("A", "B"), each = n)

counts ← matrix(rpois(300 * 200, lambda = 2), 200, 300)
rownames(counts) ← paste0("g", 1:200)
colnames(counts) ← paste0("c", 1:300)
counts[, batch == "B"] ← counts[, batch == "B"] + 3   # batch effect

so ← CreateSeuratObject(counts)
so$batch ← batch
so ← NormalizeData(so, verbose = FALSE)
so ← FindVariableFeatures(so, nfeatures = 100, verbose = FALSE)
so ← ScaleData(so, verbose = FALSE)
so ← RunPCA(so, npcs = 10, verbose = FALSE)

so ← RunHarmony(so, group.by.vars = "batch", dims.use = 1:10,
                verbose = FALSE)
so ← RunUMAP(so, reduction = "harmony", dims = 1:10, verbose = FALSE)
so ← FindNeighbors(so, reduction = "harmony", dims = 1:10, verbose = FALSE)
so ← FindClusters(so, resolution = 0.3, verbose = FALSE)

table(Batch = so$batch, Cluster = Idents(so))
```

12.38.6 Output & Results

A Harmony-corrected embedding; after clustering, batches should be represented in each cluster (shared cell types) rather than segregating by batch.

12.38.7 Interpretation

“After Harmony integration, all seven major cell types contained cells from both donors (mixing LISI = 1.84 / 2.0), indicating successful batch correction without over-mixing.”

12.38.8 Practical Tips

- Use Harmony for speed; CCA/RPCA for heavier anchor-based correction when batches have unique cell types.
- Quantify integration quality with kBET, LISI, or silhouette by batch vs cell type.
- Never integrate across tissues; integrate within tissue with donors/batches as covariates.
- Over-integration destroys biology; always inspect cell-type marker expression post-integration.
- Batch as a fixed effect in the DE model is an alternative to explicit embedding integration for focused comparisons.

12.38.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.38.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.39 Finding Marker Genes

12.39.1 Introduction

Marker genes are genes differentially expressed in one cluster relative to others; they drive cell-type annotation, functional interpretation, and the connection between unsupervised clustering and known biology. Seurat's `FindMarkers` and `FindAllMarkers` implement the standard differential-expression tests (Wilcoxon rank-sum, MAST hurdle, DESeq2) on log-normalised or SCT-transformed data, per cluster, against either all other cells or a specified comparison cluster.

12.39.2 Prerequisites

A working understanding of clustered scRNA-seq output, log-fold-change interpretation, and the difference between cluster-vs-all and pairwise cluster comparisons.

12.39.3 Theory

For each cluster, the marker test compares cells in that cluster to cells in all other clusters (or a specified subset) gene-by-gene. The output for each gene includes:

- **avg_log2FC**: average log-fold change between the target cluster and the reference.
- **pct.1**: fraction of cells in the target cluster expressing the gene.
- **pct.2**: fraction of cells in the reference expressing the gene.
- **p_val_adj**: Benjamini-Hochberg adjusted p-value.

Wilcoxon rank-sum is the default test in Seurat (fast, robust); MAST handles zero-inflation more rigorously via a hurdle model; DESeq2 applies the negative-binomial framework (slow but principled).

Conventional marker criteria: $\log_2 \mathrm{FC} > 0.5$, adjusted $p < 0.05$, and `pct.1 > 0.25` (the gene is expressed in at least a quarter of the cluster's cells).

12.39.4 Assumptions

Cells within a cluster are reasonably homogeneous; the reference set is a valid comparison; multiple testing is corrected across genes within each comparison (but not across the joint cluster set, which would be over-conservative).

12.39.5 R Implementation

```
library(Seurat)

set.seed(2026)
counts ← matrix(rpois(300 * 200, lambda = 2), 200, 300)
rownames(counts) ← paste0("g", 1:200)
colnames(counts) ← paste0("c", 1:300)
# Inject a marker pattern in cells 1-50
counts[1:10, 1:50] ← counts[1:10, 1:50] + 10

so ← CreateSeuratObject(counts)
so ← NormalizeData(so, verbose = FALSE)
so ← FindVariableFeatures(so, nfeatures = 100, verbose = FALSE)
so ← ScaleData(so, verbose = FALSE)
so ← RunPCA(so, npcs = 10, verbose = FALSE)
so ← FindNeighbors(so, dims = 1:10, verbose = FALSE)
so ← FindClusters(so, resolution = 0.5, verbose = FALSE)

markers_all ← FindAllMarkers(so, only.pos = TRUE, min.pct = 0.25,
                             logfc.threshold = 0.25, verbose = FALSE)
head(markers_all[, c("cluster", "gene", "avg_log2FC", "p_val_adj", "pct.1")])
```

12.39.6 Output & Results

`FindAllMarkers()` returns a data frame with one row per significant marker per cluster, including cluster identity, gene name, log-fold change, adjusted p-value, and percentage of cells expressing in target vs reference. Filtering to `only.pos = TRUE` retains only up-regulated markers, which are typically the most interpretable for cell-type identification.

12.39.7 Interpretation

A reporting sentence: “FindAllMarkers (Wilcoxon test, `min.pct = 0.25`, `logfc.threshold = 0.25`, `only.pos = TRUE`) identified 1{,}240 unique markers across 12 clusters; cluster 2’s top markers (CD8A, GZMB, PRF1) identify cytotoxic T cells, and cluster 5’s markers (MS4A1, CD79A) identify B cells. Annotation followed manual review against the Human Cell Atlas reference.” Always state the test, thresholds, and annotation source.

12.39.8 Practical Tips

- Use `only.pos = TRUE` to retain up-regulated markers; down-regulated markers are less interpretable in scRNA-seq because most genes are dropouts in any given cluster.
- Set `min.pct = 0.25` to filter out genes with very limited expression in the target cluster; this is a quality gate that prevents spurious markers from low-detection genes.

- For publication-quality marker calls, use MAST (`test.use = "MAST"`); it handles zero-inflation more rigorously than Wilcoxon and gives more conservative p-values.
- Always annotate clusters using markers *after* clustering — never adjust clusters to fit pre-conceived markers; the latter is circular reasoning.
- Do not perform DE testing between clusters produced by the same clustering on the same data; this is a form of double-dipping that inflates significance. Use post-hoc validation or independent test data instead.
- For automated cell-type annotation, pair markers with reference-based tools (`SingleR`, `scType`, `Azimuth`); they reduce manual annotation effort substantially.

12.39.9 R Packages Used

`Seurat::FindMarkers()` and `FindAllMarkers()` for the canonical workflow; `MAST` for hurdle-based DE testing; `DESeq2` for negative-binomial DE on pseudobulk; `SingleR` and `Azimuth` for reference-based automated annotation; `presto` for fast Wilcoxon implementations on large datasets.

12.39.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.39.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.40 scRNA Normalisation

12.40.1 Introduction

Single-cell RNA-seq cells differ in total UMI count by orders of magnitude — a typical 10x Chromium experiment ranges from a few hundred to tens of thousands of UMIs per cell. This library-size variation swamps biological signal if not corrected. Normalisation removes the library-size effect, and variance stabilisation goes further by flattening the mean-variance relationship so that downstream PCA and clustering are driven by biology rather than count magnitude.

12.40.2 Prerequisites

A working understanding of count data, library-size effects, and the negative-binomial distribution as the standard model for count overdispersion.

12.40.3 Theory

Three approaches dominate:

LogNormalize (Seurat default): divide each cell's count by its total, multiply by 10{,}000, take $\log_{1p}$. Simple, fast, and well understood; does not stabilise variance.

SCTransform: regularised negative-binomial regression with sequencing depth as the explanatory variable, producing Pearson residuals as the normalised output. Variance is stabilised by construction, and the regularised parameter pooling allows rare cell types to be recovered better than LogNormalize.

Pool-based size factors (`scran::computeSumFactors`): pool cells for robust size-factor estimation in low-depth settings, addressing the dropout problem that simple library-size normalisation does not.

12.40.4 Assumptions

Library-size effects are multiplicative on count expectations; counts follow a (possibly cell-specific) negative-binomial distribution with gene-specific dispersion.

12.40.5 R Implementation

```
library(Seurat)

set.seed(2026)
counts ← matrix(rpois(300 * 200, lambda = 2), 200, 300)
rownames(counts) ← paste0("g", 1:200)
colnames(counts) ← paste0("c", 1:300)

so ← CreateSeuratObject(counts)

# Standard LogNormalize
so ← NormalizeData(so, verbose = FALSE)
so ← FindVariableFeatures(so, nfeatures = 100, verbose = FALSE)
so ← ScaleData(so, verbose = FALSE)

# Alternative: SCTransform
so_sct ← SCTransform(so, verbose = FALSE)

head(GetAssayData(so, assay = "RNA", slot = "data")[1:5, 1:4])
head(GetAssayData(so_sct, assay = "SCT", slot = "data")[1:5, 1:4])
```

12.40.6 Output & Results

Two normalised matrices: LogNormalize (log-counts-per-10k stored in `data` slot of the `RNA` assay) and SCT (Pearson residuals stored in `data` slot of the `SCT` assay). Both feed the same downstream pipeline (PCA, clustering, UMAP) but produce different clustering granularity in practice.

12.40.7 Interpretation

A reporting sentence: "After SCTransform normalisation (`vst.flavor = 'v2'`, `vars.to.regress = 'percent.mt'`), PCA separated the major immune cell types in the first 5 components; LogNormalize required 10 components to achieve comparable separation. The SCT-based clustering identified two additional rare populations (mast cells, plasmacytoid dendritic cells) that LogNormalize merged into larger clusters." Always state the normalisation method and any regressed-out covariates.

12.40.8 Practical Tips

- Use `SCTransform(vst.flavor = "v2")`, the current recommended flavour; it improves on the original SCTransform parameter regularisation.
- For rare cell types, SCTransform consistently outperforms LogNormalize; for coarse-grained analyses either method works.
- Store raw counts in the `RNA` assay and SCT residuals in the `SCT` assay; avoid re-running LogNormalize after SCTransform on the same object.
- Do not run `NormalizeData()` after `SCTransform()`; the latter already produces normalised values, and stacking is incorrect.
- For multi-sample integration, normalise each sample independently before merging (`SCTransform` per sample, then `IntegrateData` or `harmony`); pooling raw counts then normalising loses sample-specific structure.
- Regress out unwanted technical covariates (mitochondrial percentage, cell-cycle scores) via `vars.to.regress` in `SCTransform`; this often improves downstream clustering.

12.40.9 R Packages Used

`Seurat::NormalizeData()` and `SCTransform()` for the canonical scRNA-seq normalisation methods; `scran::computeSumFactors()` for pool-based size factors in the Bioconductor ecosystem; `sctransform` (the underlying engine) for low-level access to the regression model; `scater` for additional QC and normalisation diagnostics.

12.40.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.40.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.41 scRNA QC and Filtering

12.41.1 Introduction

Single-cell RNA-seq experiments routinely contain empty droplets, stressed or apoptotic cells, doublets (two cells captured in one droplet), and various technical artefacts that should not enter downstream clustering or differential-expression analysis. Per-cell QC filtering by library size, detected-gene count, and mitochondrial-read percentage removes the worst offenders. The trade-off is delicate: over-filtering excludes real biology (e.g. plasma cells with naturally low gene counts), while under-filtering produces clusters of damaged cells that mimic real cell types.

12.41.2 Prerequisites

A working understanding of scRNA-seq counts, the structure of droplet-based experiments, and the role of mitochondrial reads as a cell-health indicator.

12.41.3 Theory

Three canonical per-cell metrics dominate QC:

- **nCount_RNA** (total UMI count per cell): very low values indicate empty droplets or debris; very high values often indicate doublets.
- **nFeature_RNA** (number of detected genes): the lowest quantile typically corresponds to broken or empty droplets.
- **percent.mt** (percentage of reads mapping to mitochondrial genes): elevated values indicate apoptotic or stressed cells whose cytoplasmic mRNA has been degraded preferentially.

Thresholds are dataset-specific; fixed cutoffs (`percent.mt < 15` for 10x human samples) work for typical experiments, but adaptive thresholds based on the median absolute deviation (`scater::isOutlier`) generalise better across datasets.

For doublet detection, dedicated tools (`scDblFinder`, `DoubletFinder`) provide more reliable identification than naive UMI-based heuristics.

12.41.4 Assumptions

Mitochondrial percentage is a valid proxy for cell health (fails in some tissue types — cardiomyocytes naturally have very high mitochondrial content); doublets show abnormally high UMI counts on average; the chosen filtering thresholds preserve biological diversity.

12.41.5 R Implementation

```
library(Seurat)

set.seed(2026)
counts <- matrix(rpois(300 * 200, lambda = 2), 200, 300)
rownames(counts) <- c(paste0("MT-", 1:10),
                      paste0("gene_", 11:200))
colnames(counts) <- paste0("cell_", 1:300)
counts[1:10, 1:30] <- counts[1:10, 1:30] + 20   # 30 cells with high MT

so <- CreateSeuratObject(counts)
so[["percent.mt"]] <- PercentageFeatureSet(so, pattern = "^MT-")

VlnPlot(so, c("nFeature_RNA", "nCount_RNA", "percent.mt"),
        ncol = 3, pt.size = 0.1)

so_f <- subset(so,
               subset = nFeature_RNA > 50 &
                        nFeature_RNA < 180 &
                        percent.mt   < 15)
dim(so_f)       # cells x genes after QC
```

12.41.6 Output & Results

`PercentageFeatureSet()` adds the mitochondrial percentage to per-cell metadata; `VlnPlot()` shows the distribution of QC metrics across cells. `subset()` applies thresholds and returns a filtered Seurat object. Inspecting the violin plots before and after filtering confirms that the chosen thresholds remove the tails without truncating the bulk of the distribution.

12.41.7 Interpretation

A reporting sentence: “Per-cell QC removed 4{,}166 of 12{,}400 cells (33.6 %) that fell outside the chosen thresholds (nFeature < 500, nFeature > 6{,}000, percent.mt > 15); doublet detection with `scDblFinder` flagged an additional 312 cells (3.8 %), yielding 7{,}922 high-quality cells for clustering. Thresholds were chosen by inspecting violin plots per sample.” Always state thresholds and the rationale.

12.41.8 Practical Tips

- Inspect QC metrics per sample or batch separately before pooling; differing distributions reflect technical variation that should inform sample-specific thresholds.
- Use median-absolute-deviation-based thresholds (`scater::isOutlier(metric, nmads = 3)`) for reproducibility across datasets; fixed cutoffs are sample-specific.
- Run doublet detection (`scDblFinder` is the modern recommendation, `DoubletFinder` is the older popular alternative) separately from count-based QC; doublets often pass naïve UMI thresholds.
- Remove cells failing on multiple metrics (low UMI *and* low gene count *and* high mitochondrial percentage) rather than aggressively filtering on any single metric; combinations are more reliable than thresholds on individual metrics.

- Document QC parameters in the methods section; results are highly sensitive to filter choice and reproducibility requires explicit thresholds.
- For tissues where mitochondrial percentage is biologically meaningful (heart, skeletal muscle), adjust thresholds accordingly or use alternative QC metrics.

12.41.9 R Packages Used

`Seurat::PercentageFeatureSet()`, `VlnPlot()`, `subset()` for the standard QC workflow; `scater::isOutlier()` for adaptive MAD-based thresholds; `scDblFinder` and `DoubletFinder` for doublet detection; `emptyDrops` (in `DropletUtils`) for distinguishing real cells from empty droplets at the upstream step.

12.41.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.41.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.42 Trajectory Analysis

12.42.1 Introduction

Trajectory inference orders single cells along a continuous developmental or activation path, producing a pseudotime value per cell that represents its progress along the trajectory. Biological processes such as cell differentiation, cell-cycle progression, T-cell activation, and disease progression are continuous in nature; clustering imposes discrete categories that conceal the underlying continuous structure. Trajectory methods recover that continuity by fitting smooth curves or graphs through the data in dimensional-reduced space.

12.42.2 Prerequisites

A working understanding of scRNA-seq normalisation, dimensional reduction (PCA, UMAP), clustering, and the conceptual difference between discrete cell types and continuous developmental trajectories.

12.42.3 Theory

The two dominant trajectory frameworks:

Slingshot constructs a minimum spanning tree on cluster centroids in a reduced-dimensional space (typically PCA), specifies or infers a starting cluster, and fits smooth principal curves through each lineage from the start. Pseudotime is the arc length along each curve, computed per cell.

Monocle3 uses a reverse-graph-embedding approach on UMAP coordinates, fitting a principal graph and ordering cells along it. Monocle3 handles complex branching topologies more naturally than Slingshot.

Both methods require either a specified root cluster or an inferred one (with biological-marker-based validation strongly recommended). Pseudotime values are then used as a continuous covariate in downstream gene-by-pseudotime models (`tradeSeq`, generalised additive models) to identify genes whose expression varies along the trajectory.

12.42.4 Assumptions

Cells lie on a low-dimensional biological manifold; the cluster topology approximates the trajectory topology; the chosen root cluster represents the developmental starting state.

12.42.5 R Implementation

```
library(slingshot); library(SingleCellExperiment)

set.seed(2026)
n_cells ← 200
x ← seq(0, 1, length.out = n_cells)
pc1 ← x + rnorm(n_cells, 0, 0.05)
pc2 ← x^2 + rnorm(n_cells, 0, 0.05)

sce ← SingleCellExperiment(
  assays = list(counts = matrix(rpois(n_cells * 50, 3), 50, n_cells))
)
reducedDim(sce, "PCA") ← cbind(pc1, pc2)
sce$cluster ← factor(cut(x, breaks = 4, labels = 1:4))

sce ← slingshot(sce, clusterLabels = sce$cluster,
                reducedDim = "PCA", start.clus = "1")
head(slingPseudotime(sce))

plot(reducedDim(sce, "PCA"), col = sce$cluster, pch = 16, asp = 1,
     main = "Slingshot lineage")
lines(SlingshotDataSet(sce), lwd = 2, col = "black")
```

12.42.6 Output & Results

`slingshot()` returns the `SingleCellExperiment` object augmented with pseudotime values (one column per lineage). The plot overlays smooth principal curves on the PCA scatter, with cells coloured by cluster, making the inferred lineage structure visible.

12.42.7 Interpretation

A reporting sentence: "Slingshot trajectory inference on the cluster-resolved PCA embedding recovered a single lineage from cluster 1 (root, validated as quiescent stem cells by SOX2 expression) through clusters 2 and 3 to cluster 4 (terminally activated cells); pseudotime correlated strongly with the known maturation marker BMI1 ($\rho = 0.84$, $p < 0.001$)." Always validate trajectory direction with known biology.

12.42.8 Practical Tips

- Specify `start.clus` explicitly when biology supports a clear initial state; automatic root inference is often unreliable for complex datasets.
- Compute trajectories on PCA space, not UMAP; UMAP distorts distances and produces unstable trajectory inference.
- Test genes for differential expression along pseudotime with `tradeSeq` (generalised additive models on count data) for principled identification of trajectory-driven genes.
- Monocle3 is preferred when the trajectory has complex branching structure (multiple lineages diverging); Slingshot is simpler and works well for linear or weakly branching trajectories.
- Always validate the inferred direction against known biological markers (start-state vs end-state); pseudotime is direction-less without a defined root.
- Pair trajectory analysis with cluster annotation; the lineage structure should connect biologically meaningful states, not arbitrary clusters.

12.42.9 R Packages Used

`slingshot` for principal-curve-based trajectory inference; `monocle3` for graph-based trajectory inference with branching support; `tradeSeq` for differential gene expression along pseudotime; `condiments` for trajectory testing under multiple conditions; `dyno` for unified access to dozens of trajectory inference methods.

12.42.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.42.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.43 Sequence Alignment: Overview

12.43.1 Introduction

Sequence alignment is the foundational operation of bioinformatics: comparing DNA, RNA, or protein sequences to find similarities, detect mutations, identify conserved regions, or annotate functional domains. The two canonical algorithms are Needleman-Wunsch (global, full-length alignment) and Smith-Waterman (local, best-matching subregion). All modern aligners — BWA, Bowtie, BLAST, minimap2 — are heuristic approximations to these dynamic-programming gold standards, optimised for speed at the cost of theoretical optimality.

12.43.2 Prerequisites

A working understanding of DNA and protein sequences, the FASTA format, and basic dynamic programming as the algorithmic technique behind alignment.

12.43.3 Theory

Global alignment (Needleman-Wunsch) finds the optimal alignment across the entire length of both sequences. Appropriate when the sequences are believed to be homologous over their full length, e.g. comparing two complete protein sequences from related species.

Local alignment (Smith-Waterman) finds the optimal alignment of a substring of each sequence. Appropriate when only a region is conserved, e.g. searching for a domain match within a longer protein.

Scoring schemes combine match rewards, mismatch penalties, and gap penalties. For proteins, substitution matrices encode biological similarity:

- **BLOSUM62**: derived from blocks of distantly related proteins; the BLAST default.
- **PAM250**: derived from closely related proteins; appropriate for close evolutionary distances.

Affine gap penalties (gap-opening + gap-extension) better model real biological gaps than constant penalties.

12.43.4 Assumptions

Sequences are biologically comparable (same molecule type, similar functional category); the chosen scoring scheme reflects the evolutionary distance involved.

12.43.5 R Implementation

```
library(Biostrings)

s1 ← DNAString("ATCGATCGTACGTACG")
s2 ← DNAString("ATCAATCATACGTGG")

# Global alignment
ga ← pairwiseAlignment(s1, s2, type = "global", substitutionMatrix =
nucleotideSubstitutionMatrix(),
                       gapOpening = 10, gapExtension = 4)
ga

# Local alignment
la ← pairwiseAlignment(s1, s2, type = "local",
                       substitutionMatrix = nucleotideSubstitutionMatrix(),
                       gapOpening = 10, gapExtension = 4)
la

# Protein alignment with BLOSUM62
p1 ← AAString("ACDEFGHI")
p2 ← AAString("ACDFGHI")
data(BLOSUM62)
pairwiseAlignment(p1, p2, substitutionMatrix = BLOSUM62, type = "global",
                  gapOpening = 10, gapExtension = 4)
```

12.43.6 Output & Results

`pairwiseAlignment()` returns an object with the aligned sequences (with gaps inserted), the alignment score, and percent identity. Subset accessors (`pattern()`, `subject()`, `score()`, `pid()`) extract specific components.

12.43.7 Interpretation

A reporting sentence: "Needleman-Wunsch global alignment of two 16-bp DNA sequences scored 6 (13/16 matches, two mismatches and a single-base gap); Smith-Waterman local alignment returned the best-matching 13-character substring with no gaps. The protein alignment with BLOSUM62 highlighted the conserved core (8/9 identity) despite the single deletion." Always state the scoring scheme and gap penalties.

12.43.8 Practical Tips

- Choose the scoring scheme to match the question: BLOSUM for distant protein relatives, PAM for close ones; for nucleotide alignment, simple match/mismatch is usually adequate.
- For short-read alignment (NGS), use BWA, Bowtie2, or minimap2 — they are heuristic but vastly faster than Needleman-Wunsch on millions of reads.
- For multiple sequence alignment (MSA), use the `msa` package which wraps Clustal Omega, ClustalW, and MUSCLE; or external tools (MAFFT, T-Coffee) for very large alignments.
- Affine gap penalties (gap-open and gap-extension) better model biological gaps than constant penalties; the BLAST default is gap-open 10, gap-extension 4 for nucleotides.

- For long reads (PacBio, Nanopore), use minimap2 — it is designed for the higher error rates and longer alignment lengths.
- For sequence search rather than pairwise comparison, BLAST and DIAMOND scale to large databases via heuristic seeding.

12.43.9 R Packages Used

`Biostrings` for `pairwiseAlignment()`, BLOSUM/PAM matrices, and sequence-handling classes; `msa` for multiple sequence alignment; `Rsamtools` and `GenomicAlignments` for BAM-aware alignment workflows; `Rsubread` for short-read alignment integrated with the broader RNA-seq Bioconductor workflow.

12.43.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.43.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.44 Single-Cell with Seurat

12.44.1 Introduction

Seurat is the most widely used R framework for single-cell RNA-seq (scRNA-seq) analysis. A standard Seurat workflow loads a count matrix from Cell Ranger or a similar source, performs quality control on per-cell metrics, normalises expression, selects variable features, scales, runs dimensionality reduction (PCA + UMAP/t-SNE), clusters cells, and identifies marker genes. The framework's strength is its end-to-end coverage and large user community; the alternative Bioconductor stack (`SingleCellExperiment` + `scran`) offers comparable functionality with tighter integration into the broader Bioconductor ecosystem.

12.44.2 Prerequisites

A working understanding of single-cell RNA-seq experimental design, count matrices from droplet-based protocols (10x Chromium), and basic clustering and dimensionality-reduction concepts.

12.44.3 Theory

Seurat stores cells × genes counts and derived matrices in a `Seurat` object with structured slots for raw counts, normalised data, variable features, scaled data, dimensional reductions, and per-cell metadata. The standard pipeline:

1. **QC**: filter cells by minimum gene count, maximum percent mitochondrial reads, library size thresholds.
2. **Normalisation**: log-normalisation (`NormalizeData`) or regularised regression (`SCTransform`).
3. **Variable-feature selection**: identify genes with biologically interesting variability.
4. **Scaling**: centre and unit-variance scale variable features.
5. **PCA**: linear dimensionality reduction on scaled variable features.
6. **Graph-based clustering**: shared-nearest-neighbour graph + Louvain or Leiden community detection.
7. **UMAP / t-SNE**: non-linear visualisation only — not used for clustering decisions.

12.44.4 Assumptions

Each droplet is mostly single-cell (low doublet rate, typically below 5 %); per-cell library size correlates with mRNA content; variable genes capture biological signal rather than technical noise.

12.44.5 R Implementation

```
library(Seurat)

set.seed(2026)
# Minimal simulated count matrix (300 cells x 200 genes)
counts ← matrix(rpois(300 * 200, lambda = 2), 200, 300)
rownames(counts) ← paste0("gene_", 1:200)
colnames(counts) ← paste0("cell_", 1:300)

so ← CreateSeuratObject(counts = counts, min.cells = 3, min.features = 10)
so ← NormalizeData(so, verbose = FALSE)
so ← FindVariableFeatures(so, nfeatures = 100, verbose = FALSE)
so ← ScaleData(so, verbose = FALSE)
so ← RunPCA(so, npcs = 10, verbose = FALSE)
so ← FindNeighbors(so, dims = 1:10, verbose = FALSE)
so ← FindClusters(so, resolution = 0.5, verbose = FALSE)
so ← RunUMAP(so, dims = 1:10, verbose = FALSE)

table(Idents(so))
DimPlot(so, reduction = "umap", label = TRUE)
```

12.44.6 Output & Results

A `Seurat` object containing per-cell cluster assignments, dimensional-reduction coordinates (PCA, UMAP), and gene-level statistics. `DimPlot()` produces the canonical UMAP visualisation with cluster boundaries; `FeaturePlot()` overlays gene expression on the UMAP.

12.44.7 Interpretation

A reporting sentence: "Standard Seurat processing of the 300-cell dataset (`NormalizeData`, `FindVariableFeatures`, `ScaleData`, `RunPCA`, Louvain clustering at resolution 0.5) produced 5 clusters; cluster 3 uniquely expressed CD8A and CD8B, consistent with cytotoxic T cells, and cluster 1 expressed CD79A and MS4A1, consistent with B cells." Always describe the QC thresholds, normalisation method, number of PCs, and clustering resolution.

12.44.8 Practical Tips

- Run QC before normalisation; low-quality cells dominate variable-feature selection and contaminate downstream clustering.
- Use `SCTransform` instead of `NormalizeData + FindVariableFeatures + ScaleData` for a more principled variance-stabilising approach; SCTransform is the modern default for new analyses.
- The clustering `resolution` controls cluster granularity; explore values from 0.2 to 1.2 and pick based on biological interpretability rather than a single statistical criterion.
- Document Seurat version with `sessionInfo()`; the API has changed substantially between major versions (v3, v4, v5), and old code may not run on current versions.
- For very large datasets ($> 100{,}000$ cells), use Seurat v5's sketch-based subsampling or switch to the Bioconductor stack (`SingleCellExperiment`, `scran`) for memory efficiency.
- Pair clustering results with marker-gene analysis (`FindAllMarkers`) and biological annotation (SingleR, scType) before reporting cluster identities.

12.44.9 R Packages Used

`Seurat` for the canonical end-to-end scRNA-seq workflow; `SingleCellExperiment` for the Bioconductor data structure; `scran` and `scater` for Bioconductor-style normalisation and clustering; `harmony` for batch correction; `SingleR` for automatic cell-type annotation; `DoubletFinder` for doublet detection.

12.44.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.44.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.45 Spatial Transcriptomics

12.45.1 Introduction

Spatial transcriptomics assays (10x Visium, Slide-seq, Stereo-seq, MERFISH) measure gene expression with spatial coordinates. They bridge histology and scRNA-seq: the tissue architecture is preserved, so analyses like spatial domain detection, neighbourhood effects, and tumour-immune interface mapping become possible.

12.45.2 Prerequisites

scRNA-seq analysis basics; familiarity with tissue histology.

12.45.3 Theory

Visium spots contain ~10-100 cells each, giving spot-resolution (not single-cell) expression. Analysis proceeds like scRNA-seq but with two additions: (i) spatial variable-gene detection (Moran's I), and (ii) spatial domain inference (BayesSpace, Giotto, or spatial-aware clustering).

Deconvolution (cell2location, RCTD, SpatialExperiment) decomposes each spot into cell-type proportions using a matched scRNA-seq reference.

12.45.4 Assumptions

Spatial coordinates are correctly registered to H&E; spot-level expression is a linear combination of underlying cell types; tissue processing preserves mRNA integrity.

12.45.5 R Implementation

```r
library(Seurat)

set.seed(2026)
# Minimal spatial-like object: spots on a grid
n_spots <- 100
coords <- expand.grid(x = 1:10, y = 1:10)
counts <- matrix(rpois(200 * n_spots, lambda = 3), 200, n_spots)
rownames(counts) <- paste0("g", 1:200)
colnames(counts) <- paste0("spot_", 1:n_spots)

# Inject spatial pattern: top-left vs bottom-right
region <- ifelse(coords$x + coords$y < 10, "A", "B")
counts[1:20, region == "A"] <- counts[1:20, region == "A"] + 8

so <- CreateSeuratObject(counts)
so$x <- coords$x; so$y <- coords$y; so$region <- region

so <- NormalizeData(so, verbose = FALSE)
so <- FindVariableFeatures(so, nfeatures = 100, verbose = FALSE)
so <- ScaleData(so, verbose = FALSE)
so <- RunPCA(so, npcs = 10, verbose = FALSE)
so <- FindNeighbors(so, dims = 1:10, verbose = FALSE)
so <- FindClusters(so, resolution = 0.5, verbose = FALSE)

plot(coords, col = factor(Idents(so)), pch = 15, cex = 2,
     main = "Spot clusters on tissue coordinates",
     xlab = "x", ylab = "y")
```

12.45.6 Output & Results

A clustered spatial object; the 2-D scatter shows clusters aligning with spatial regions, recapitulating tissue architecture.

12.45.7 Interpretation

“Spatial clustering of the Visium section identified six domains corresponding to tumour core, tumour edge, stroma, and three immune niches; interface enrichment analysis revealed T-cell accumulation at the tumour-stroma boundary.”

12.45.8 Practical Tips

- Always overlay clusters on H&E to validate biological interpretation.
- Moran’s I (`SpatialExperiment::spatialCorrelation` or `Seurat::FindSpatiallyVariableFeatures`) finds spatially variable genes.
- Deconvolution is essential when Visium spots cover multiple cell types; use a matched scRNA-seq reference.
- BayesSpace and SpaGCN integrate spatial structure into clustering, giving smoother domain boundaries.
- Stereo-seq and MERFISH offer sub-cellular resolution; analysis pipelines differ slightly from spot-based Visium.

12.45.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.45.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.46 STAR: Spliced Alignment

12.46.1 Introduction

STAR (Spliced Transcripts Alignment to a Reference) is the de facto standard short-read RNA-seq aligner. It is designed specifically for the splice-aware alignment problem: reads from mature mRNA cross exon-exon junctions, and a naive aligner that requires contiguous matches misses these split alignments and misclassifies them as multi-mappers or unaligned. STAR's seed-extend algorithm splits reads at candidate junctions, aligns the segments to the reference genome, and assembles them into spliced alignments — including across novel (unannotated) junctions.

12.46.2 Prerequisites

A working understanding of RNA-seq experimental design, the difference between mRNA and DNA sequencing, and basic familiarity with sequence alignment concepts.

12.46.3 Theory

STAR's two-pass mapping is the standard mode: the first pass aligns each read independently and accumulates a list of candidate splice junctions; the second pass adds those junctions to the reference index and re-aligns, producing more accurate spliced alignments especially across novel junctions. The output is a BAM file plus a tab-separated splice-junction summary (`SJ.out.tab`).

STAR also supports chimeric alignment for fusion-gene detection (reads spanning two distinct genomic loci), expectation-maximisation-based multi-mapper resolution, and stranded alignment when library protocols preserve transcript orientation.

12.46.4 Assumptions

Short RNA-seq reads (typically 50–150 bp Illumina); a reference genome and (preferably) a gene annotation in GTF format; sufficient memory — STAR's index for human genome takes about 30 GB of RAM.

12.46.5 R Implementation

```
library(Rsubread)

# Rsubread::align is Subjunc (splice-aware) or align
# index ← "subread_index"
# buildindex(basename = index, reference = "genome.fa")

# Splice-aware alignment
# align(index = index, readfile1 = "reads.fastq.gz",
#       output_file = "aligned.bam",
#       type = "rna", useAnnotation = TRUE,
#       annot.inbuilt = "hg38")
```

12.46.6 Output & Results

The primary output is a BAM file containing spliced alignments and a `SJ.out.tab` file summarising splice junctions detected (including their support count and annotation status). For typical RNA-seq experiments, 90–95 % of reads align uniquely; a much lower rate signals contamination, library-prep problems, or annotation mismatch.

12.46.7 Interpretation

A reporting sentence: "STAR (v2.7.10a) two-pass alignment to GRCh38 with the GENCODE v44 annotation aligned 94 % of reads uniquely; 2.3 million splice junctions were detected, of which 98 % matched annotated junctions, suggesting limited novel splicing in the experimental conditions." Always report the genome assembly, annotation version, and unique-mapping rate.

12.46.8 Practical Tips

- STAR needs about 30 GB of RAM for the human genome index; for smaller genomes (yeast, bacteria), it runs comfortably on a laptop.
- Two-pass mode (`twopassMode = "Basic"`) is standard for novel-junction detection; the first-pass-only mode is faster but less accurate.
- For transcript-level quantification, pair STAR alignments with RSEM, Salmon, or Kallisto; the BAM is the input to abundance estimation.
- HISAT2 is a faster, lower-memory alternative aligner with similar accuracy; use it when memory is constrained.
- Pseudo-aligners (Salmon, Kallisto) skip the BAM-producing alignment entirely; faster but less suitable when isoform-level alignments are needed (variant calling, fusion detection).
- Use a reference GTF annotation to improve junction accuracy; novel-junction detection works without it but is more error-prone.

12.46.9 R Packages Used

`Rsubread` provides splice-aware alignment via `subjunc` and feature counting via `featureCounts`; `Rsamtools` and `GenomicAlignments` for downstream BAM manipulation; `tximport` for importing transcript-level abundance estimates from Salmon, Kallisto, or RSEM into Bioconductor; STAR itself is a command-line tool but its output integrates with all standard R/Bioconductor RNA-seq workflows.

12.46.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.46.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.47 Structural Variant Detection

12.47.1 Introduction

Structural variants (SVs) – deletions, duplications, inversions, translocations, and complex rearrangements – are genomic events $\geq$ 50 bp. They often have larger phenotypic impact than single-nucleotide variants but are harder to call because they involve breakpoints across long distances. Short-read callers like Manta and Delly, plus long-read callers (Sniffles, pbsv), are standard.

12.47.2 Prerequisites

BAM files from high-coverage WGS; reference genome.

12.47.3 Theory

Short-read SV callers combine three signals: - **Read pairs** with abnormal insert size or orientation. - **Split reads** whose two halves map to distant loci. - **Read-depth** anomalies (CNV overlap).

Long-read callers directly observe breakpoint-spanning reads, dramatically improving resolution of repeat-rich SVs.

12.47.4 Assumptions

Sequencing depth is adequate ($\geq$ 30x WGS); insert-size distribution is approximately normal; reference matches the sample ancestry.

12.47.5 R Implementation

Manta/Delly run on the command line producing VCF; R ingests and annotates:

```
library(VariantAnnotation)
library(StructuralVariantAnnotation)

# Use a VariantAnnotation-shipped VCF as a proxy
vcf_file ← system.file("extdata", "chr22.vcf.gz", package = "VariantAnnotation")
vcf ← readVcf(vcf_file, "hg19")
vcf_sv ← vcf[1:10, ]   # illustrative subset; treat as SV records

# breakpointRanges builds a GRanges of breakpoints from SV VCF INFO fields
gr ← breakpointRanges(vcf_sv)
length(gr)
head(as.data.frame(gr))
```

Example command-line calls:

```
manta configManta.py --bam sample.bam --reference ref.fa --runDir manta_out
delly call -g ref.fa -o sample.bcf sample.bam
```

12.47.6 Output & Results

A VCF of SV calls with type (DEL, DUP, INV, INS, BND), coordinates, and supporting-read counts; annotation against genes yields per-gene SV impact.

12.47.7 Interpretation

“WGS SV calling identified 4,120 events (2,800 deletions, 1,100 duplications, 180 inversions, 40 translocations) per sample. The germline 17q12 deletion spans RCAD and was concordant with the clinical suspicion.”

12.47.8 Practical Tips

- Use **two** callers and intersect (high-precision) or union (high-sensitivity); any single caller misses events.
- Filter by breakpoint support; single-read evidence is almost always false.
- Long-read WGS resolves complex rearrangements that short reads cannot; use for structural-disease phenotyping.
- Tumour SV calling requires matched normal to remove germline events; somatic callers: Manta + DELLY (cancer mode), GRIDSS.
- Annotate breakpoints with `StructuralVariantAnnotation` to identify gene fusions and breakpoint-disrupted regions.

12.47.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.47.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.48 Surrogate Variable Analysis (SVA)

12.48.1 Introduction

Surrogate variable analysis (SVA) estimates latent factors that capture unmodelled heterogeneity in high-dimensional expression data – unknown batch effects, sample-processing artefacts, or unmeasured biology correlated with technical variables. Including surrogate variables as covariates in downstream regression restores power and controls false positives where ComBat cannot apply (because the batch variable is itself unknown).

12.48.2 Prerequisites

Linear models on expression data, batch effects, ComBat.

12.48.3 Theory

SVA decomposes the residual expression matrix (after regressing out the primary model) into latent factors using a bootstrap-based singular value decomposition. The number of surrogate variables k is chosen by a permutation test (`num.sv`). Each surrogate is then added as a covariate in the downstream limma/edgeR model.

For RNA-seq count data, use `svaseq`, which applies SVA on a log-transformed scale suitable for negative-binomial outcomes.

12.48.4 Assumptions

The primary variable of interest is modelled correctly; surrogate variables are orthogonal to it; residual variation reflects technical or unmodelled factors.

12.48.5 R Implementation

```r
library(sva); library(limma)

set.seed(2026)
n_genes <- 3000; n_samples <- 20
group   <- factor(rep(c("ctrl", "trt"), each = 10))
batch   <- factor(rep(c("A", "B"), times = 10))           # hidden

y <- matrix(rnorm(n_genes * n_samples), n_genes, n_samples)
y <- y + matrix(rep(as.numeric(batch) * 0.6, each = n_genes),
                n_genes, n_samples)

mod  <- model.matrix(~ group)
mod0 <- model.matrix(~ 1, data = data.frame(group))

n_sv <- num.sv(y, mod, method = "be")
svobj <- sva(y, mod, mod0, n.sv = n_sv)

mod_sv <- cbind(mod, svobj$sv)
fit <- lmFit(y, mod_sv); fit <- eBayes(fit)
topTable(fit, coef = 2, n = 5)
```

12.48.6 Output & Results

`svobj$sv` contains $n_s v$ surrogate variables per sample. Including them typically recovers differential expression that batch confounding otherwise hides.

12.48.7 Interpretation

"We estimated k surrogate variables using SVA and included them in the limma model, adjusting for unmodelled heterogeneity without needing to specify a batch variable."

12.48.8 Practical Tips

- Use `svaseq` for RNA-seq count data.
- The `num.sv(..., method = "be")` Buja-Eyuboglu estimator is conservative; `"leek"` is more aggressive.
- Surrogates should not correlate with the variable of interest; `sva` orthogonalises them by construction.
- Always inspect what surrogates correlate with (library size, processing date) for interpretation.
- If the batch is known, prefer ComBat-seq or direct covariate inclusion; SVA is for hidden factors.

12.48.9 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.48.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.49 Transcript-to-Gene Summarisation

12.49.1 Introduction

Pseudo-alignment tools (Salmon, Kallisto) and EM-based quantifiers (RSEM) produce transcript-level (isoform-level) counts. Most differential-expression workflows operate at the gene level, so transcript counts must be aggregated to genes via a transcript-to-gene mapping derived from the annotation. The `tximport` package handles this aggregation cleanly and produces output ready for DESeq2, edgeR, or limma-voom; the related `tximeta` extends this with automatic metadata and reference-genome tracking.

12.49.2 Prerequisites

A working understanding of transcript-level vs gene-level expression quantification, RNA-seq annotation files (GTF), and the difference between counts, TPM, and effective length.

12.49.3 Theory

The simplest transcript-to-gene aggregation sums transcript counts within each gene: $\text{gene count} = \sum_{\text{transcripts} \in \text{gene}} \text{transcript count}$. `tximport` adds two important refinements:

- **Length-scaled TPM** (`countsFromAbundance = "lengthScaledTPM"`): combines transcript abundance and length information to produce gene-level counts that respect both isoform-resolution and gene-level interpretation.
- **Effective-length offsets**: passes effective-length information to downstream DE tools (DESeq2 with `tximeta::tximeta()` integration) for accurate normalisation.

Direct transcript-level DE (without aggregation) is supported by tools like swish (`fishpond`) and DRIMSeq for differential transcript usage; aggregation is the right choice when the question is about gene-level expression.

12.49.4 Assumptions

The transcript-to-gene map matches the annotation used for transcript quantification; sample-file ordering is consistent with metadata.

12.49.5 R Implementation

```
library(tximport); library(tximeta)

# Example: salmon outputs
# files ← list.files("salmon_out", pattern = "quant.sf", recursive = TRUE, full.names
= TRUE)
# tx2gene ← read.table("tx2gene.tsv", header = TRUE)
#
# txi ← tximport(files, type = "salmon", tx2gene = tx2gene,
#                countsFromAbundance = "lengthScaledTPM")
#
# txi$counts: gene x sample matrix
```

12.49.6 Output & Results

`tximport()` returns a list with `counts` (gene-by-sample matrix), `abundance` (gene-level TPM), and `length` (effective lengths used for downstream normalisation). The matrix is ready as input to `DESeq2::DESeqDataSetFromTximport()` or as a starting point for edgeR / limma workflows.

12.49.7 Interpretation

A reporting sentence: “tximport aggregated 190{,}000 transcript-level Salmon estimates to 22{,}000 gene-level counts using a GENCODE v44 transcript-to-gene map; `countsFromAbundance = 'lengthScaledTPM'` produced gene counts that DESeq2 then analysed with effective-length offsets passed via the `txi$length` matrix.” Always document the aggregation method and tx-to-gene source.

12.49.8 Practical Tips

- `countsFromAbundance = "lengthScaledTPM"` is the standard for DE analysis; it scales raw counts by transcript length while preserving the count interpretation.
- Use `tximeta` instead of raw `tximport` when possible — it automatically retrieves the transcript-to-gene map from the Salmon index’s metadata, eliminating annotation-mismatch errors.
- For differential transcript usage (DTU), use DRIMSeq or DEXSeq on the transcript-level matrix; aggregation collapses isoform-level information.
- For differential transcript expression (DTE), use swish (`fishpond::swish`) on the transcript-level matrix with bootstrap inferential replicates from Salmon.
- Verify that sample order in `files` matches the rows of your metadata table; off-by-one errors here corrupt the entire downstream analysis silently.
- For very large studies, `tximport` reads files sequentially; parallelise with `BiocParallel` for substantial speedups.

12.49.9 R Packages Used

`tximport` for the canonical aggregation; `tximeta` for metadata-aware aggregation that handles tx-to-gene mapping automatically; `fishpond` for swish-based DTE with bootstrap replicates; `DRIMSeq` and `DEXSeq` for differential transcript usage at the transcript level.

12.49.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.49.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.50 Variant Calling with GATK

12.50.1 Introduction

Variant calling turns aligned reads into a list of single-nucleotide variants (SNVs) and small insertions/deletions (indels). GATK's HaplotypeCaller is the de facto standard: it locally reassembles regions with evidence of variation and emits probabilistic genotype calls in VCF format.

12.50.2 Prerequisites

Sequence alignment, BAM format, reference genome indexing.

12.50.3 Theory

HaplotypeCaller proceeds in four stages: define **active regions** (loci with potential variation), perform **local de novo assembly** of haplotypes, **align reads to haplotypes** via pair-HMM, and **genotype** using likelihoods. For cohorts, each sample is first emitted as a per-sample GVCF, then joint-genotyped with `GenotypeGVCFs` – this preserves reference evidence and scales linearly with samples.

Downstream filtering is done with variant-quality score recalibration (VQSR) for whole-genome data or hard filters for smaller studies.

12.50.4 Assumptions

Reads are duplicate-marked, base-quality-recalibrated (BQSR), and aligned to a standard reference; ploidy is specified.

12.50.5 R Implementation

The GATK workflow runs on the command line (Java-based); R interacts with VCF output via `VariantAnnotation`.

```r
library(VariantAnnotation)

# Sample VCF from the package
vcf_file <- system.file("extdata", "chr22.vcf.gz", package = "VariantAnnotation")
vcf <- readVcf(vcf_file, "hg19")

vcf                                     # summary (SNPs, indels, samples)
head(rowRanges(vcf))                    # variant coordinates
head(info(vcf)[, c("AF", "DP")])        # population AF, depth
geno(vcf)$GT[1:5, 1:3]                  # genotype calls (sample x variant)
```

A representative GATK command (not run here):

```
gatk HaplotypeCaller -R ref.fa -I sample.bam -O sample.g.vcf.gz -ERC GVCF
gatk GenotypeGVCFs   -R ref.fa -V cohort.g.vcf.gz -O cohort.vcf.gz
```

12.50.6 Output & Results

A VCF / GVCF with per-variant coordinates, alleles, annotations, and per-sample genotypes, depths, and likelihoods.

12.50.7 Interpretation

“Joint genotyping with GATK HaplotypeCaller called 4.2M SNVs and 0.8M indels across the cohort; after VQSR with 99.9% sensitivity we retained 3.9M high-confidence SNVs.”

12.50.8 Practical Tips

- Always BQSR before calling; skipping it inflates false positives, especially for low-frequency variants.
- Use GVCF workflow for multi-sample studies; `HaplotypeCaller` in direct VCF mode is only for single-sample work.
- `DeepVariant` is a competitive deep-learning alternative, producing similar quality without assembly.
- For somatic calls, use `Mutect2`, not HaplotypeCaller.
- Filter by depth, strand bias, and mapping quality (see `VariantFiltration`); many library artefacts look like rare variants.

12.50.9 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.50.10 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.51 VCF Manipulation

12.51.1 Introduction

VCF (Variant Call Format) is the standard tab-delimited format for representing genotype calls at genomic positions. Whether the data come from whole-genome sequencing, exome sequencing, genotyping arrays, or imputation, downstream analysis nearly always begins by reading, filtering, and subsetting a VCF. The `VariantAnnotation` Bioconductor package provides programmatic access in R; `bcftools` is the command-line workhorse for cohort-scale operations.

12.51.2 Prerequisites

A working understanding of VCF format basics (CHROM, POS, REF, ALT, FILTER, INFO, FORMAT, per-sample GT), the difference between phased and unphased genotypes, and the typical variant-level QC metrics (QUAL, depth, missingness, allele frequency).

12.51.3 Theory

A VCF row represents a variant at one genomic position; columns include INFO (variant-level annotations: allele frequency, depth, functional consequence) and per-sample FORMAT fields (genotype GT, depth DP, genotype quality GQ). Standard filtering steps:

- **PASS filter**: keep only variants flagged `PASS` by the calling pipeline; non-PASS variants are typically calibration artefacts.
- **Allele frequency**: minimum minor-allele frequency (MAF) threshold to retain common variants for association testing.
- **Depth and missingness**: minimum per-sample depth and maximum site-level missingness.
- **Hardy-Weinberg**: large deviations in unrelated controls indicate genotyping errors.
- **Multi-allelic normalisation**: split multi-allelic sites into bi-allelic rows and left-align indels (`bcftools norm`).

12.51.4 Assumptions

VCF is bgzipped (`.vcf.gz`) and tabix-indexed for random access; reference build matches the calling pipeline; sample identifiers are consistent across cohorts when merging.

12.51.5 R Implementation

```
library(VariantAnnotation)

vcf_file ← system.file("extdata", "chr22.vcf.gz", package = "VariantAnnotation")

# Targeted reading (region + fields only; faster than full read)
rng ← GRanges("22", IRanges(50301422, width = 50000))
param ← ScanVcfParam(which = rng,
                     info = c("AF", "DP"),
                     geno = "GT")
vcf_sub ← readVcf(vcf_file, "hg19", param = param)
vcf_sub

# Filter by MAF
af ← unlist(info(vcf_sub)$AF)
af ← pmin(af, 1 - af)
vcf_common ← vcf_sub[af > 0.05, ]
length(vcf_common)

# Extract and reshape genotypes
gt ← geno(vcf_common)$GT
head(gt)
```

12.51.6 Output & Results

`readVcf()` returns a `VCF` Bioconductor object with structured slots for variant ranges, INFO fields, FORMAT fields, and sample information. Accessor functions (`info()`, `geno()`, `rowRanges()`) extract specific components, and indexing (`vcf[af > 0.05, ]`) filters by row.

12.51.7 Interpretation

A reporting sentence: "After filtering to PASS variants with MAF > 5 %, missingness < 5 %, and Hardy-Weinberg $p > 10^{-6}$ in 1{,}532 unrelated controls, we retained 412{,}380 high-quality common autosomal variants for association testing." Always describe the filtering pipeline in detail; cohort-comparison studies require precisely matched filtering.

12.51.8 Practical Tips

- For cohort-scale operations (millions of variants, thousands of samples), use `bcftools` at the command line; it is orders of magnitude faster than R-based reading and writing.
- Use `ScanVcfParam()` to read only the regions and fields you need; full WGS VCFs can be tens of GB and reading them entirely is wasteful.
- Filter by `FILTER == "PASS"` early; non-PASS variants accumulate in downstream analyses if not removed.
- Normalise variants with `bcftools norm -m -any -f reference.fa` before merging cohorts; multi-allelic representations differ across callers and produce silent merge errors.
- For association studies, export to PLINK or BGEN formats via `snpStats` or `plink2` for efficient large-scale regression.

- For functional annotation (consequence prediction, conservation scores), use `VariantAnnotation::predictCoding()`, `Ensembl VEP`, or `SnpEff`.

12.51.9 R Packages Used

`VariantAnnotation` for reading, writing, and accessor-based VCF manipulation; `Rsamtools` for tabix index handling; `snpStats` for PLINK-format export and SNP-level statistics; `bcftoolsR` and standalone `bcftools` for command-line operations; `vcfR` for an alternative interface focused on population genetics.

12.51.10 For Reviewers

Use the shared reviewer checklist, together with this method's assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript's application of the method.

12.51.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.52 Volcano Plots

12.52.1 Introduction

A volcano plot summarises a differential-expression result in a single scatter: log fold change on the x-axis, $-\log_{10}$ adjusted p-value on the y-axis. Genes in the upper-left and upper-right corners — large effect *and* high precision — are the targets of interest; genes near the bottom of the plot fail the significance criterion regardless of effect size, and genes near zero on the x-axis fail the biological-relevance criterion regardless of significance. The combined visual makes the standard two-criterion ranking immediately apparent and is the canonical figure of any differential-expression report.

12.52.2 Prerequisites

A working understanding of differential-expression analysis output (DESeq2, edgeR, limma-voom), log fold change, and FDR-adjusted p-values.

12.52.3 Theory

The volcano plot's signature shape arises from the relationship between effect-size estimates and their standard errors: extreme fold changes are typically reported by genes with low expression (and therefore noisier estimates with worse p-values), so the "wings"

of high fold change at low significance are sparse. The dense central column near zero fold change, with p-values uniformly distributed from very high to moderately significant, represents the bulk of unchanged genes.

Standard cutoffs combine fold-change and significance: $|\log_2 \mathrm{FC}| > 1$ (two-fold change) and adjusted $p < 0.05$ jointly. Either criterion alone is misleading — large effects with poor precision (e.g. low-expression genes) and small effects at extreme significance (e.g. very high-expression genes with tiny but reproducible changes) both produce different scientific conclusions.

12.52.4 Assumptions

The DE analysis was correctly specified; p-values are calibrated (DE methods generally produce well-calibrated p-values under their assumptions); fold-change sign is interpretable as a directional effect.

12.52.5 R Implementation

```
library(ggplot2)

set.seed(2026)
n ← 5000
logFC ← c(rnorm(n - 100), rnorm(100, 2))
pval  ← c(runif(n - 100), 10^(-runif(100, 2, 6)))
padj  ← p.adjust(pval, "BH")

df ← data.frame(logFC = logFC, padj = padj)
df$cat ← with(df, ifelse(padj < 0.05 & abs(logFC) > 1,
                         ifelse(logFC > 0, "up", "down"), "ns"))

ggplot(df, aes(logFC, -log10(padj), colour = cat)) +
  geom_point(alpha = 0.5, size = 1) +
  geom_hline(yintercept = -log10(0.05), linetype = 2) +
  geom_vline(xintercept = c(-1, 1), linetype = 2) +
  scale_colour_manual(values = c(up = "#E76F51",
                                 down = "#2A9D8F",
                                 ns = "grey70")) +
  labs(x = "log2 fold change", y = "-log10 adjusted p",
       title = "Volcano plot (DE results)") +
  theme_minimal()
```

12.52.6 Output & Results

A scatter with up- and down-regulated genes highlighted in distinct colours and dashed lines at the chosen FC and FDR thresholds. The visual immediately conveys how many genes pass both criteria, the symmetry of up vs down regulation, and the distribution of effect sizes among significant hits.

12.52.7 Interpretation

A reporting sentence (figure caption): “Volcano plot of differential expression between conditions A and B (n = 6 per group); 100 genes passed FDR < 0.05 and $|\log_2 \mathrm{FC}| > 1$ (45 up, 55 down); the five most significant genes are labelled. Dashed lines mark the FDR and

fold-change thresholds.” Always state the cutoffs and sample sizes.

12.52.8 Practical Tips

- Use `EnhancedVolcano::EnhancedVolcano()` for publication-ready volcano plots with automatic gene labelling, customisable thresholds, and `ggrepel`-based label placement.
- Truncate extreme $-\log_{10} p$ values (e.g. cap at 50) for readability; very small p-values otherwise stretch the y-axis and compress the bulk of the plot.
- Report both raw and FDR-adjusted p-value thresholds transparently in the figure caption.
- Add gene labels only for the top-ranked hits or biologically-curated genes of interest; many labels obscure the plot.
- Volcano plots do not show absolute expression magnitude; pair with MA plots when expression-dependent effects are suspected.
- For comparing multiple contrasts, use facets or different colour schemes; overlaying multiple contrasts on a single volcano usually fails.

12.52.9 R Packages Used

`ggplot2` for hand-built volcano plots; `EnhancedVolcano` for publication-ready volcano plots with extensive customisation; `ggrepel` for non-overlapping gene labels; `Glimma::glimmaVolcano()` for interactive volcano plots embeddable in HTML reports.

12.52.10 For Reviewers

Use the shared reviewer checklist, together with this method’s assumptions and practical tips.

Method-specific reviewer notes have not yet been developed for this entry. The links above support a structured review; they do not certify the example or a manuscript’s application of the method.

12.52.11 See also — labs in this chapter

- RNA-seq with DESeq2 / edgeR
- Enrichment analysis (GSEA / ORA)
- scRNA-seq with Seurat

12.53 Week 4, Session 2 — Enrichment analysis (GSEA / ORA)

Workflow labs use the variant template: Goal → Approach → Execution → Check → Report.

12.53.1 Learning objectives

- Perform an over-representation analysis (ORA) by the hypergeometric test on a simulated example.
- Describe the GSEA running-sum statistic and when it is preferred over ORA.
- Recognise the hazards of gene-set overlap and of using different background universes.

12.53.2 Prerequisites

Multiple testing; bulk RNA-seq DE results.

12.53.3 Background

Pathway or gene-set enrichment analyses take a list of candidate genes — from a differential-expression analysis, say — and ask whether any annotated set contains more of them than expected by chance. Over- representation analysis (ORA) is the simplest version: given a universe of N genes, k of which are in a pathway, and a shortlist of n genes of which m are in that pathway, the hypergeometric distribution returns a p-value for the overlap.

Gene-set enrichment analysis (GSEA) uses the full ranked list of genes and tests whether members of a pathway are concentrated at the top or bottom of the ranking via a running-sum statistic. Unlike ORA, it does not require a threshold, and it retains information from genes that fall just short of significance.

Both methods are sensitive to the background universe. If the universe is “all human genes” but the measurement technology only detects half of them, the p-values are wrong. The correct universe is the set of genes actually tested in the upstream analysis.

12.53.4 Setup

```
library(tidyverse)
set.seed(42)
theme_set(theme_minimal(base_size = 12))
```

12.53.5 1. Goal

Demonstrate a hypergeometric ORA by hand, then sketch the `fgsea` call pattern for a running-sum GSEA.

12.53.6 2. Approach

Simulate a universe of 5 000 genes; a pathway of 80 members is enriched for differentially expressed (DE) genes.

```
N ← 5000
pathway ← sample(seq_len(N), 80)
true_de ← unique(c(sample(pathway, 40), sample(setdiff(seq_len(N), pathway), 60)))

tibble(in_pathway = seq_len(N) %in% pathway,
       de = seq_len(N) %in% true_de) ▷
  count(in_pathway, de) ▷
  ggplot(aes(in_pathway, n, fill = de)) +
  geom_col(position = "dodge") +
  labs(x = "gene in pathway?", y = "count")
```

12.53.7 3. Execution

Hypergeometric test by hand.

```
n_de ← length(true_de)
k_in ← sum(true_de %in% pathway)
p_hyper ← phyper(k_in - 1, 80, N - 80, n_de, lower.tail = FALSE)
c(overlap = k_in, expected = n_de * 80 / N, p = p_hyper)
```

Make a ranked gene list (as if from DE) and sketch `fgsea`.

```
stat ← rnorm(N)
stat[pathway] ← stat[pathway] + 1.2    # shift pathway genes up
names(stat) ← paste0("g", seq_len(N))
stat ← sort(stat, decreasing = TRUE)

# Running-sum demonstration by hand.
in_set ← names(stat) %in% paste0("g", pathway)
p_hit ← abs(stat) / sum(abs(stat[in_set]))
p_miss ← 1 / sum(!in_set)
run_sum ← cumsum(ifelse(in_set, p_hit, -p_miss))
ES ← run_sum[which.max(abs(run_sum))]
tibble(i = seq_along(run_sum), rs = run_sum) ▷
  ggplot(aes(i, rs)) + geom_line() +
  geom_hline(yintercept = 0, colour = "grey50") +
  labs(x = "rank", y = "running enrichment score")
```

`fgsea` sketch.

```
library(fgsea)
gene_sets ← list(pathway = paste0("g", pathway))
fgsea_res ← fgsea(pathways = gene_sets, stats = stat,
                  minSize = 15, maxSize = 500)
```

12.53.8 4. Check

Compare the ORA result for different shortlist sizes.

```
sweep_ora ← sapply(c(50, 100, 200, 500), function(top) {
  short ← names(stat)[seq_len(top)]
  k ← sum(short %in% paste0("g", pathway))
  phyper(k - 1, 80, N - 80, top, lower.tail = FALSE)
})
tibble(top = c(50, 100, 200, 500), p = sweep_ora)
```

12.53.9 5. Report

> In a simulated universe of 5 000 genes with a pathway of 80 members enriched for DE, the hypergeometric ORA for the full DE shortlist gave $p =$ `signif(p_hyper, 2)`. A running-sum GSEA on the full ranked list produced an enrichment score of `round(ES, 2)` with the running sum peaking near the top.

ORA is a quick, interpretable first pass; GSEA is better when the effect is diffuse. The two answer different questions and sometimes disagree; both are worth showing when the result matters.

12.53.10 Common pitfalls

- Using the full genome as the ORA universe when only a subset was assayed.
- Double-counting overlapping pathways as independent discoveries.
- Reporting a GSEA *p*-value without the normalised enrichment score.

12.53.11 Further reading

- Subramanian A et al. (2005), *Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles*.
- Huang DW, Sherman BT, Lempicki RA (2009), *Bioinformatics enrichment tools: paths toward the comprehensive functional analysis of large gene lists*.

12.53.12 Session info

```
sessionInfo()
```

12.53.13 See also — chapter index

- Methods in this chapter
- Find your method (Chapter 0)

12.54 Week 4, Session 1 — RNA-seq with DESeq2 / edgeR

Workflow labs use the variant template: Goal → Approach → Execution → Check → Report.

12.54.1 Learning objectives

- Describe the count-model assumptions behind DESeq2 and edgeR and explain the role of dispersion estimation.
- Set up a design matrix for a two-group comparison and for a more complex experiment.
- Recognise when a log-transformed `limma-voom` workflow is preferable.

12.54.2 Prerequisites

GLMs; multiple testing.

12.54.3 Background

Bulk RNA-sequencing yields integer counts per gene per sample. Raw counts are heteroscedastic — variance grows with the mean — and the distribution has a long tail. DESeq2 and edgeR model counts as negative-binomial, with a gene-specific dispersion shrunk toward a trend estimated across the dataset. This shrinkage is the statistical heart of modern differential-expression analysis in low-replicate studies.

`limma-voom` takes a different route: model the log-counts with precision weights that reflect the count-level mean–variance trend. The output is a familiar linear-model workflow with all of its tools (contrasts, interactions, F-tests). For large or complex designs, `limma-voom` is often the most flexible.

DESeq2 and edgeR are Bioconductor packages, not CRAN. We show their call pattern with `#| eval: false` and walk through a conceptually parallel simulation with base R GLMs so the ideas run end-to-end in any environment.

12.54.4 Setup

```
library(tidyverse)
library(MASS)
set.seed(42)
theme_set(theme_minimal(base_size = 12))
```

12.54.5 1. Goal

Simulate a small RNA-seq count matrix with two conditions and demonstrate the DESeq2/edgeR pipeline pattern alongside a base-R negative-binomial GLM that actually runs.

12.54.6 2. Approach

Simulate 500 genes × 10 samples (5 per group). Ten genes are truly differentially expressed.

```
n_gene ← 500; n_samp ← 10
group ← rep(c("A", "B"), each = 5)
mu ← 2 * runif(n_gene, 1, 10)   # baseline mean per gene
lfc ← c(rep(log(3), 10), rep(0, n_gene - 10))
counts ← sapply(seq_len(n_samp), function(j) {
  rate ← mu * exp(ifelse(group[j] == "B", lfc, 0))
  rnbinom(n_gene, mu = rate, size = 5)
})
rownames(counts) ← paste0("g", seq_len(n_gene))
colnames(counts) ← paste0("s", seq_len(n_samp))

tibble(mean = rowMeans(counts), var = apply(counts, 1, var)) ▷
  ggplot(aes(mean, var)) + geom_point(alpha = 0.3) +
  scale_x_log10() + scale_y_log10() +
  geom_abline(slope = 1, intercept = 0, colour = "grey50") +
  labs(title = "mean-variance (log-log)")
```

12.54.7 3. Execution

A DESeq2 call pattern (not run).

```
library(DESeq2)
dds ← DESeqDataSetFromMatrix(
  countData = counts,
  colData   = data.frame(group = group),
  design    = ~ group
)
dds ← DESeq(dds)
res ← results(dds, contrast = c("group", "B", "A"))
head(res[order(res$padj), ])
```

An edgeR call pattern (not run).

```
library(edgeR)
dge ← DGEList(counts = counts, group = group)
dge ← calcNormFactors(dge)
design ← model.matrix(~ group)
dge ← estimateDisp(dge, design)
fit ← glmQLFit(dge, design)
qlf ← glmQLFTest(fit, coef = 2)
topTags(qlf)
```

A per-gene negative-binomial GLM that runs.

```
pvals ← numeric(n_gene)
logfcs ← numeric(n_gene)
for (g in seq_len(n_gene)) {
  y ← counts[g, ]
  fit ← try(glm.nb(y ~ group), silent = TRUE)
  if (inherits(fit, "try-error")) { pvals[g] ← NA; next }
  s ← summary(fit)
  pvals[g]  ← s$coefficients[2, 4]
  logfcs[g] ← s$coefficients[2, 1]
}
padj ← p.adjust(pvals, method = "BH")
```

12.54.8 4. Check

Volcano plot with the truly DE genes flagged.

```
truth ← c(rep("DE", 10), rep("null", n_gene - 10))
tibble(lfc = logfcs, padj = padj, truth = truth) ▷
  drop_na() ▷
  ggplot(aes(lfc, -log10(padj), colour = truth)) +
  geom_point(alpha = 0.6) +
  labs(x = "log fold change (B vs A)", y = "-log10(padj)")
```

```
mean(padj[1:10] < 0.05, na.rm = TRUE)
mean(padj[11:n_gene] < 0.05, na.rm = TRUE)
```

12.54.9 5. Report

> On a simulated 500-gene, 10-sample RNA-seq dataset with 10 truly differentially expressed genes (log fold change ≈ 1.1), per-gene negative-binomial GLMs followed by Benjamini–Hochberg adjustment recovered `round(mean(padj[1:10] < 0.05, na.rm = TRUE) * 100, 0)` % of the true DE genes at FDR < 0.05 and yielded a false-discovery rate among null genes of `round(mean(padj[11:n_gene] < 0.05, na.rm = TRUE) * 100, 1)` %.

DESeq2 and edgeR would improve power by shrinking gene-wise dispersion across the experiment; the per-gene fit shown here is a transparent but low-power version of the same idea.

12.54.10 Common pitfalls

- Running a *t*-test on raw counts — the variance structure is wrong.
- Filtering genes after seeing the results (data-driven filters are fine only if applied before testing).
- Forgetting to include batch or library-size terms in the design.

12.54.11 Further reading

- Love MI, Huber W, Anders S (2014), *Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2*.
- Law CW et al. (2014), *voom: precision weights unlock linear model analysis tools for RNA-seq read counts*.

12.54.12 Session info

```
sessionInfo()
```

12.54.13 See also — chapter index

- Methods in this chapter

- Find your method (Chapter 0)

12.55 Week 4, Session 3 — scRNA-seq with Seurat

Workflow labs use the variant template: Goal → Approach → Execution → Check → Report.

12.55.1 Learning objectives

- Walk through the Seurat pipeline: QC, normalisation, feature selection, PCA, neighbour graph, clustering, UMAP.
- Describe why scRNA-seq counts need different handling from bulk counts.
- Produce a small analogous pipeline in base R with simulated cells and the `uwot` UMAP.

12.55.2 Prerequisites

UMAP; bulk RNA-seq differential expression.

12.55.3 Background

Single-cell RNA-seq measures gene expression in thousands of cells simultaneously, producing a sparse, noisy, zero-inflated matrix. Clustering and cell-type annotation are exploratory steps that require a chain of careful preprocessing: drop empty droplets, normalise library size, select variable genes, reduce to principal components, build a nearest-neighbour graph, cluster on the graph, embed with UMAP.

Seurat is the dominant R toolkit for this pipeline. Every step has alternatives — `SCTransform` vs `NormalizeData`, Louvain vs Leiden clustering, Harmony for batch correction — and each choice changes the resulting partition. The good reflex is to run the same pipeline twice with a perturbed parameter and check that the biology survives.

Seurat is not on CRAN and its installation is non-trivial in a rendering environment. We sketch the pipeline with `#| eval: false` and run an analogous, tiny base-R + `uwot` pipeline on a simulated cell × gene matrix so the reasoning is visible in every render.

12.55.4 Setup

```
library(tidyverse)
library(uwot)
set.seed(42)
theme_set(theme_minimal(base_size = 12))
```

12.55.5 1. Goal

Walk through an scRNA-seq analysis conceptually with a small simulated cell × gene matrix, ending at a 2-D embedding with cluster labels.

12.55.6 2. Approach

300 cells, 500 genes, three cell types with partly overlapping expression programmes.

```
n_cell ← 300; n_gene ← 500
cell_type ← rep(c("T", "B", "myeloid"), each = n_cell / 3)
mu_base ← exp(rnorm(n_gene, 1, 1))
X ← matrix(0, n_cell, n_gene)
for (i in seq_len(n_cell)) {
  program ← rep(1, n_gene)
  if (cell_type[i] == "T")       program[1:20]  ← 5
  if (cell_type[i] == "B")       program[21:40] ← 5
  if (cell_type[i] == "myeloid") program[41:60] ← 5
  X[i, ] ← rnbinom(n_gene, mu = mu_base * program, size = 2)
}
tibble(lib = rowSums(X), ct = cell_type) ▷
  ggplot(aes(ct, lib)) + geom_boxplot() +
  labs(x = NULL, y = "total counts per cell")
```

12.55.7 3. Execution

Log-normalise, select variable genes, PCA, UMAP, cluster.

```
logX ← log1p(sweep(X, 1, rowSums(X), FUN = "/") * 1e4)
gene_var ← apply(logX, 2, var)
keep ← order(gene_var, decreasing = TRUE)[1:100]
pcs ← prcomp(logX[, keep])$x[, 1:10]
emb ← umap(pcs, n_neighbors = 20, min_dist = 0.1)

# A simple k-means cluster on the PCs as a stand-in.
km ← kmeans(pcs, centers = 3, nstart = 10)
tibble(x = emb[, 1], y = emb[, 2],
       cluster = factor(km$cluster), truth = cell_type) ▷
  ggplot(aes(x, y, colour = cluster, shape = truth)) +
  geom_point(alpha = 0.8, size = 1) +
  labs(x = "UMAP1", y = "UMAP2")
```

A Seurat call pattern (not run).

```
library(Seurat)
obj ← CreateSeuratObject(counts = t(X))
obj ← NormalizeData(obj)
obj ← FindVariableFeatures(obj, nfeatures = 100)
obj ← ScaleData(obj)
obj ← RunPCA(obj, npcs = 10)
obj ← FindNeighbors(obj, dims = 1:10)
obj ← FindClusters(obj, resolution = 0.5)
obj ← RunUMAP(obj, dims = 1:10)
DimPlot(obj, label = TRUE)
```

12.55.8 4. Check

Contingency of cluster vs true cell type.

```
table(cluster = km$cluster, truth = cell_type)
```

12.55.9 5. Report

> On a simulated 300-cell × 500-gene dataset with three cell types, a small pipeline (log normalisation, top-100 variable genes, 10 PCs, UMAP, k-means) recovered the three types with high agreement. The same reasoning — preprocess, reduce, graph, cluster, embed — underlies the Seurat pipeline sketched in the accompanying code.

On a real dataset the critical choices are the normalisation (`SCTransform` vs `NormalizeData`), the number of PCs (elbow or `JackStraw`), and the clustering resolution. Perturb each and confirm that the biological conclusions do not.

12.55.10 Common pitfalls

- Clustering before QC; apoptotic cells and empty droplets cluster together.
- Forgetting that UMAP distances are not biological distances.
- Using marker genes from a tutorial on a different tissue for cell-type annotation.

12.55.11 Further reading

- Stuart T et al. (2019), *Comprehensive integration of single-cell data*.
- Luecken MD, Theis FJ (2019), *Current best practices in single-cell RNA-seq analysis: a tutorial*.

12.55.12 Session info

```
sessionInfo()
```

12.55.13 See also — chapter index

- Methods in this chapter
- Find your method (Chapter 0)