Cereal Biotechnology and Biochemistry
Online ISSN: 2783-5170
Homepage: https://cbb.razi.ac.ir

Principles and Advanced Applications of Next-Generation Sequencing (NGS) Technology in Life Sciences with a Focus on Cereal crops breeding

Omid Mohammadalizadeh[1], Reza Darvishzadeh[2] ✉, Valiollah Mohammadi[3], Somaieh Soufimaleky[4] & Danial Kahrizi[5]

[1] MSc Graduate of Agricultural Biotechnology, Department of Agronomy & Plant Breeding, Faculty of Sciences and Agricultural Engineering, Tehran University, Karaj, Iran.

[2] Professor, Department of Plant Production and Genetics, Faculty of Agriculture, Urmia University, Urmia, Iran.

[3] Associate Professor, Department of Agronomy & Plant Breeding, Faculty of Sciences and Agricultural Engineering, Tehran University, Karaj, Iran.

[4] MSc Graduate, Institut des Sciences du Cerveau de Toulouse, France.

[5] Professor, Department of Agricultural Biotechnology, Faculty of Agriculture, Tarbiat Modares University, Tehran, Iran.

✉ Corresponding author. E-mail: r.darvishzadeh@urmia.ac.ir

ABSTRACT

Introduction: Since the introduction of Next-Generation Sequencing (NGS) in the early 2000s, this technology has emerged as a transformative advancement in the life sciences, significantly propelling genomic, transcriptomic, epigenomic, and other related research fields. The core principles of NGS technology encompass library preparation, sequencing, and the analysis of the resulting data. By enabling the parallel sequencing of millions of DNA fragments with high accuracy, low cost, and rapid turnaround, NGS has effectively replaced older methods like Sanger sequencing. It has revolutionized our understanding of genetic complexities, genome structures, and genetic diversity through the swift and precise sequencing of entire genomes and target regions. Key applications of NGS in the life sciences include the identification and study of genes related to quantitative and qualitative traits, genetic diversity studies, population genetics, the diagnosis of genetic diseases, epidemiology, microbiome analysis, forensic science, phylogenetics, systems biology, genetic engineering, genome editing, and plant and animal breeding. However, the effective use of NGS data necessitates the development of robust computational infrastructure and advanced algorithms, as well as the expansion of researchers' knowledge regarding the bioinformatic applications and challenges associated with NGS data analysis and interpretation.

Materials and methods: The present article is a review paper, conducted through content analysis by searching for keywords related to Next-Generation Sequencing (NGS), types of NGS sequencing, NGS data analysis, and the applications of NGS in relevant articles found in online databases such as PubMed, Web of Science, Google Scholar, and Scopus.

Results: This study aims to provide a comprehensive guide for the efficient and optimal analysis of NGS data by thoroughly reviewing first-, second-, and third-generation sequencing methods, examining NGS data analysis pipelines, and exploring the broad applications of NGS in various fields, including cereal research. The first section reviews first-generation sequencing (Maxam-Gilbert and Sanger), second-generation sequencing (Illumina, ABI/SOLID, Roche/454 pyrosequencing, Ion Torrent), and third-generation sequencing (Heliscope, SMRT, and Oxford Nanopore). The second section introduces various NGS sequencing methods, such as Whole Genome Sequencing (WGS), Whole Exome Sequencing (WES), Bulk RNA-Seq, and others, and examines their analysis pathways. The subsequent discussion elaborates on the application of NGS in diverse areas, including the identification of structural genomic variations (SVs), the study of epigenetic changes, microbial population analysis, and agriculture (with an emphasis on cereal breeding). Finally, the advantages and challenges of NGS are discussed.

Conclusion: As a revolutionary technology in genomics, Next-Generation Sequencing has profoundly impacted life sciences research. The reduction in sequencing costs, coupled with increased accuracy and the development of new methods, has positioned NGS as a critical tool for a deeper understanding of genetics and the development of personalized therapeutic strategies. With ongoing advancements in this field and the integration of NGS with artificial intelligence, the future of NGS in enhancing the precision of genetic data analysis and improving therapeutic processes appears promising.

Keywords: Big Data Analysis؛ Bioinformatics؛ Sequencing Platforms؛ Next-Generation Sequencing (NGS).

Article Type: Review Article

Article history: Received: 30 Jan 2024, **Revised:** 02 Feb 2024, **Accepted:** 19 Feb 2024, **Published online:** 27 Mar 2024

Cite this article: Mohammadalizadeh, O., Darvishzadeh, R., Mohammadi, V., Soufimaleky, S. & Kahrizi, D. (2024). Principles and Advanced Applications of Next-Generation Sequencing (NGS) Technology in Life Sciences with a Focus on Cereal crops breeding. *Cereal Biotechnology and Biochemistry*, *3*(1), 110-231. DOI: 10.22126/cbb.2024.11011.1080

© The Author(s).
10.22126/cbb.2024.11011.1080

Publisher: Razi University

بیوتکنولوژی و بیوشیمی غلات

شاپا الکترونیکی: ۵۱۷۰-۲۷۸۳

Homepage: https://cbb.razi.ac.ir

اصول و کاربردهای فناوری توالی‌یابی نسل جدید (NGS) در علوم زیستی (با رویکرد به‌نژادی غلات)

امید محمدعلیزاده[۱]، رضا درویش‌زاده[۲]✉، ولی‌اله محمدی[۳]، سمیه صوفی ملکی[۴] و دانیال کهریزی[۵]

[۱] دانش آموخته کارشناسی ارشد، گروه زراعت و اصلاح نباتات، دانشکده علوم و مهندسی کشاورزی، دانشکدگان کشاورزی و منابع طبیعی، دانشگاه تهران، کرج، ایران.

[۲] استاد، گروه تولید و ژنتیک گیاهی، دانشکده کشاورزی، دانشگاه ارومیه، ارومیه، ایران.

[۳] دانشیار، گروه زراعت و اصلاح نباتات، دانشکده علوم و مهندسی کشاورزی، دانشکدگان کشاورزی و منابع طبیعی، دانشگاه تهران، کرج، ایران.

[۴] دانش آموخته کارشناسی ارشد، انستیتو علوم اعصاب تولوز، فرانسه.

[۵] استاد، گروه بیوتکنولوژی کشاورزی، دانشکده کشاورزی، دانشگاه تربیت مدرس، تهران، ایران.

✉نویسنده مسئول. رایانامه: r.darvishzadeh@urmia.ac.ir

چکیده

مقدمه: از زمان معرفی توالی‌یابی نسل جدید (NGS)، در اوایل دهه ۲۰۰۰، این فناوری به عنوان یکی از تحولات بنیادین در علوم زیستی، باعث پیشرفت چشمگیر در تحقیقات ژنومیک، ترنسکریپتوم ، اپیژنوم و... شده است. اصول فناوری NGS شامل مباحث مربوط به تهیه کتابخانه، توالی‌یابی و تحلیل داده‌های حاصل از آن است. این فناوری با توالی‌یابی میلیون‌ها قطعه DNA به‌صورت موازی و با دقت بالا و هزینه پایین و همچنین تولید حجم زیادی از داده‌های ژنومی در زمان کوتاه، توانسته است جایگزین روش‌های قدیمی‌تری مانند توالی‌یابی سنگر شود و با توالی‌یابی سریع، دقیق و کامل ژنوم‌ها و نواحی هدف انقلابی بزرگ در درک پیچیدگی‌های ژنتیکی، ساختار ژنوم و تعیین تنوع ژنتیکی ایجاد کند. از مهم‌ترین کاربردهای NGS در علوم زیستی می‌توان به شناسایی و مطالعه ژن‌های مرتبط با صفات کمی و کیفی، مطالعات تنوع ژنتیکی و ژنتیک جمعیت، تشخیص بیماری‌های ژنتیکی، اپیدمیولوژی، میکروبیوم‌شناسی، پزشکی قانونی، فیلوژنتیک، زیست‌شناسی سامانه‌ای، مهندسی ژنتیک و ویرایش ژنوم و اصلاح نبات و دام اشاره کرد. بااین‌حال، استفاده مؤثر از داده‌های NGS مستلزم توسعه زیرساخت‌های محاسباتی قوی و الگوریتم‌های پیشرفته و همچنین گسترش اطلاعات محققان در رابطه با کاربردها و چالش‌های بیوانفورماتیکی مرتبط با داده‌های NGS، برای تحلیل و تفسیر این حجم عظیم از اطلاعات است.

مواد و روش‌ها: مقاله حاضر یک مقاله مروری می‌باشد که به شیوه تحلیل محتوا (Content analysis) با جستجوی کلید واژه‌های توالی‌یابی نسل جدید (NGS)، انواع توالی‌یابی NGS، تجزیه و تحلیل داده‌های NGS، کاربردهای توالی‌یابی NGS، در مقاله‌های مرتبط در پایگاه‌های اینترنتی PubMed، Web of science، Google scholar و Scopus به دست آمده است.

یافته‌ها: این مطالعه قصد دارد با مرور تفصیلی توالی‌یابی‌های نسل اول، دوم و سوم ، بررسی مسیر تجزیه و تحلیل داده‌های NGS و کاربردهای گسترده NGS در زمینه‌های مختلف از جمله تحقیقات غلات، راهنمایی تقریباً کاملی برای تجزیه و تحلیل کارآمد و بهینه داده های حاصل از توالی‌یابی نسل جدید ارائه نماید. به این‌منظور در بخش اول به مرور توالی‌یابی‌های نسل اول (ماکسام-گیلبرت و سنگر)، نسل‌دوم (Illumina، ABI/SOLID، Roche/454 pyrosequencing، Ion Torrent) و نسل سوم (Heliscope، SMRT، Oxford Nanopore) پرداخته شد. سپس در بخش دوم انواع توالی‌یابی‌های NGS مانند: Whole genome sequencing; WGS، Whole exome sequencing; WES، Bulk RNA-seq و سایر روش ها معرفی و مسیر تجزیه و تحلیل آنها بررسی شده‌اند و در ادامه کاربرد توالی‌یابی نسل جدید در حوزه های مختف مانند شناسایی تنوعات ساختاری ژنومی (SVs)، مطالعه تغییرات اپی‌ژنتیکی، تجزیه و تحلیل جمعیت میکروبی، کشاورزی (با تأکید بر بهنژادی غلات) توضیح داده شد. در نهایت مزایا و چالش‌های پیشروی توالی‌یابی نسل جدید بیان گردید.

نتیجه‌گیری: توالی‌یابی نسل جدید به عنوان یک فناوری انقلابی در ژنومیک، تأثیر بسزایی در تحقیقات علوم زیستی داشته است. کاهش هزینه‌ها و افزایش دقت توالی‌یابی به همراه توسعه روش‌های جدید، باعث شده است تا NGS به ابزاری کلیدی برای درک بهتر ژنتیک و توسعه راهبردهای درمانی شخصی‌سازی شده تبدیل شود. با پیشرفت‌های مداوم در این حوزه و ترکیب این فناوری با هوش مصنوعی، آینده‌ی NGS در تحلیل دقیق‌تر داده‌های ژنتیکی و بهبود فرایندهای درمانی بسیار روشن به نظر می‌رسد.

واژه‌های کلیدی: آنالیز داده‌های حجیم؛ بیوانفورماتیک؛ پلتفرم‌های توالی‌یابی؛ توالی‌یابی نسل جدید (NGS).

نوع مقاله: مقاله مروری

نوع مقاله: دریافت: ۱۴۰۲/۱۱/۳۰ **اصلاح:** ۱۴۰۲/۱۱/۱۳ **پذیرش:** ۱۴۰۲/۱۱/۳۰، **انتشار آنلاین:** ۱۴۰۳/۰۱/۰۸

استناد: محمدعلیزاده، ا.، درویش‌زاده، ر.، محمدی، و.، صوفی ملکی، س. و کهریزی، د. (۱۴۰۳). اصول و کاربردهای فناوری توالی‌یابی نسل جدید (NGS) در علوم زیستی (با رویکرد به¬نژادی غلات). *بیوتکنولوژی و بیوشیمی غلات*، ۲(۳)، ۱۱۰-۲۳۱. DOI: 10.22126/cbb.2024.11011.1080

ناشر: دانشگاه رازی

© نویسندگان.

CC BY NC

فایل تکمیلی

این فایل تکمیلی شامل اسکریپت‌های تجزیه داده برای طیف گسترده‌ای از تکنیک‌های توالی‌یابی نسل بعدی (NGS) از جمله Transcriptomics by bulk RNA-، Whole exome sequencing; WES، Whole genome sequencing; WGS *de novo* Genome ،Small RNA sequencing ،Transcriptomics by Single-cell RNA-seq; scRNAseq ،seq ،Chromatin immunoprecipitation sequencing; ChIP-seq ،*de novo* Transcriptome assembly ،assembly Whole و Targeted sequencing; TGS ،Epigenomics by DNA methylation sequencing; Methyl-seq metagenome sequencing; WMS می‌باشد. اسکریپت‌ها برای اجرا در سیستم عامل لینوکس و محیط برنامه‌نویسی R طراحی شده‌اند و شامل مراحل کنترل کیفیت، پیش‌پردازش، آنالیز و تجسم داده‌ها می‌باشند. هر بخش شامل دستورات و توضیحات مرحله به مرحله است. توجه داشته باشید که این اسکریپت‌ها ممکن است نیاز به تنظیم برای داده‌های خاص داشته باشند و اطمینان حاصل کنید که تمام نرم‌افزارهای مورد نیاز قبل از اجرا نصب شده‌اند.

Whole genome sequencing (WGS):

1. Quality Control of Raw Reads
Using a tool like FastQC to assess the quality of raw sequencing reads.

 # Run FastQC on raw reads
```
fastqc raw_reads.fastq -o qc_reports/
```
raw_reads.fastq: Input file containing raw sequencing reads.
-o qc_reports/: Output directory for quality control reports.

2. Adapter Trimming and Quality Filtering
Using a tool like Trimmomatic to remove adapters and low-quality bases.
Run Trimmomatic for adapter trimming and quality filtering
```
trimmomatic PE raw_reads_1.fastq raw_reads_2.fastq
trimmed_1.fastq trimmed_2.fastq unpaired_1.fastq
unpaired_2.fastq ILLUMINACLIP:adapters.fa:2:30:10 LEADING:3
TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```
PE: Paired-end mode.
raw_reads_1.fastq raw_reads_2.fastq: Input paired-end read files.
trimmed_1.fastq trimmed_2.fastq: Output files for trimmed paired-end reads.
unpaired_1.fastq unpaired_2.fastq: Output files for unpaired reads.
ILLUMINACLIP:adapters.fa:2:30:10: Adapter trimming parameters.

LEADING:3 TRAILING:3: Remove low-quality bases from the start and end.
SLIDINGWINDOW:4:15: Perform sliding window trimming.
MINLEN:36: Minimum length of reads to keep.

3. Reference Genome Indexing

Using **BWA** for indexing.

Index the reference genome

```
bwa index reference_genome.fasta
```

reference_genome.fasta: Input reference genome file.

4. Alignment to Reference Genome

Using **BWA** for alignment.

Align reads to the reference genome

```
bwa mem reference_genome.fasta trimmed_1.fastq
trimmed_2.fastq >  aligned_reads.sam
```

bwa mem: BWA command for alignment.
reference_genome.fasta: Reference genome file.
trimmed_1.fastq trimmed_2.fastq: Input trimmed reads.
> aligned_reads.sam: Output SAM file containing aligned reads.

5. Duplicate Marking

Using **Picard** for marking duplicates.

Mark duplicates using Picard

```
java -jar picard.jar MarkDuplicates I=aligned_reads.sam
O=dedup_reads.bam M=metrics.txt
```

I=aligned_reads.sam: Input SAM file.
O=dedup_reads.bam: Output BAM file with duplicates marked.
M=metrics.txt: Output file for duplication metrics.

6. Variant Calling (SNPs, Indels, SNVs)

Using **GATK** for variant calling.

Call variants using GATK

```
gatk HaplotypeCaller -R reference_genome.fasta -I
dedup_reads.bam -O raw_variants.vcf
```

-R reference_genome.fasta: Reference genome file.
-I dedup_reads.bam: Input BAM file.
-O raw_variants.vcf: Output VCF file containing raw variants.

CNVnator for Copy Number Variants

```
cnvnator -root cnv.root -tree sorted_aligned.bam
cnvnator -root cnv.root -his 100
cnvnator -root cnv.root -stat 100
cnvnator -root cnv.root -partition 100
```

```
cnvnator -root cnv.root -call 100 > cnv_calls.txt
```
Comment: Different tools are used to identify various types of variants:
#GATK HaplotypeCaller for SNPs and small indels
#CNVnator for copy number variants
#Structural variants are often detected separately

7. Variant Quality Score Recalibration (VQSR)
Using **GATK** for VQSR.
```
# Perform VQSR using GATK
gatk ApplyVQSR -R reference_genome.fasta -V raw_variants.vcf -O
recalibrated_variants.vcf \
    --truth-sensitivity-filter-level 99.0 --tranches-file
output.tranches --recal-file output.recal
```
-R reference_genome.fasta: Reference genome file.
-V raw_variants.vcf: Input VCF file with raw variants.
-O recalibrated_variants.vcf: Output VCF file with recalibrated variants.
--truth-sensitivity-filter-level 99.0: Desired level of truth sensitivity.
--tranches-file output.tranches: Tranches file from VariantRecalibrator.
--recal-file output.recal: Recalibration file from VariantRecalibrator.

8. Structural Variant (SV) Detection
Using **Delly** for SV detection.
```
   # Detect structural variants using Delly
delly call -g reference_genome.fasta -o sv_calls.bcf
dedup_reads.bam
```
-g reference_genome.fasta: Reference genome file.
-o sv_calls.bcf: Output BCF file with structural variant calls.
dedup_reads.bam: Input BAM file with deduplicated reads.

9. Variant Annotation
Using **SnpEff** for variant annotation.
```
# Annotate variants using SnpEff
java -jar snpEff.jar -v GRCh38.86 recalibrated_variants.vcf >
annotated_variants.vcf
```
-v: Verbose output.
GRCh38.86: Human genome version (adjust as needed).
recalibrated_variants.vcf: Input VCF file with recalibrated variants.
> annotated_variants.vcf: Output VCF file with annotated variants.

10. Variant Filtering
Using **GATK** for variant filtering.

Filter variants using GATK
```
gatk VariantFiltration -R reference_genome.fasta -V
annotated_variants.vcf \
    -O filtered_variants.vcf \
    --filter-expression "QD < 2.0 || FS > 60.0 || MQ < 40.0" \
    --filter-name "my_filter"
```
-R reference_genome.fasta: Reference genome file.
-V annotated_variants.vcf: Input VCF file with annotated variants.
-O filtered_variants.vcf: Output VCF file with filtered variants.
--filter-expression: Filtering criteria (adjust as needed).
--filter-name: Name for the filter applied.

11. Functional Analysis

For this step, we'll use R to perform functional enrichment analysis using the **clusterProfiler** package.

Load required libraries
```
library(clusterProfiler)
library(org.Hs.eg.db)

# Read in the filtered variants (assuming we have a list of genes)
genes <- read.table("filtered_genes.txt", header = TRUE,
stringsAsFactors = FALSE)

# Perform GO enrichment analysis
go_enrichment <- enrichGO(gene = genes$SYMBOL,
                          OrgDb = org.Hs.eg.db,
                          keyType = "SYMBOL",
                          ont = "BP",
                          pAdjustMethod = "BH",
                          pvalueCutoff = 0.05,
                          qvalueCutoff = 0.2)
```
Save results
```
write.csv(as.data.frame(go_enrichment), file =
"go_enrichment_results.csv", row.names = FALSE)
```
Visualize results
```
pdf("go_enrichment_plot.pdf", width = 10, height = 8)
dotplot(go_enrichment, showCategory = 20)
dev.off()
```

enrichGO(): Performs GO enrichment analysis.
gene: List of gene symbols.
OrgDb: Organism database (human in this case).

keyType: Type of gene identifiers used.
ont: Ontology to use (BP: Biological Process).
pAdjustMethod: Method for p-value adjustment.
pvalueCutoff and qvalueCutoff: Significance thresholds.
write.csv(): Saves enrichment results to a CSV file.
dotplot(): Creates a dot plot of enrichment results.

12. Reporting and Visualization

We'll use R to create some visualizations of our results.

```
# Load required libraries
library(ggplot2)
library(dplyr)

# Read in coverage data (assuming we have a bedGraph file)
coverage <- read.table("coverage.bedGraph", header = FALSE,
                       col.names = c("chromosome", "start",
"end", "coverage"))

# Calculate average coverage per chromosome
avg_coverage <- coverage %>%
  group_by(chromosome) %>%
  summarize(avg_coverage = mean(coverage))

# Create a bar plot of average coverage per chromosome
pdf("average_coverage_plot.pdf", width = 10, height = 6)
ggplot(avg_coverage, aes(x = chromosome, y = avg_coverage)) +
  geom_bar(stat = "identity", fill = "steelblue") +
  theme_minimal() +
  labs(title = "Average Coverage per Chromosome",
       x = "Chromosome", y = "Average Coverage")
dev.off()

# Read in variant data
variants <- read.table("filtered_variants.vcf", header = TRUE,
comment.char = "#")

# Create a histogram of variant quality scores
pdf("variant_quality_histogram.pdf", width = 10, height = 6)
ggplot(variants, aes(x = QUAL)) +
  geom_histogram(binwidth = 10, fill = "darkgreen", color =
"black") +
  theme_minimal() +
```

```
  labs(title = "Distribution of Variant Quality Scores",
       x = "Quality Score", y = "Count")
dev.off()
```

read.table(): Reads in data files.
group_by() and summarize(): Calculate average coverage per chromosome.
ggplot(): Creates visualizations.
geom_bar(): Creates a bar plot for average coverage.
geom_histogram(): Creates a histogram for variant quality scores.
pdf()and dev.off(): Opens and closes PDF devices for saving plots.

References:

Fuentes-Pardo, A. P., & Ruzzante, D. E. 2017. Whole-genome sequencing approaches for conservation biology: Advantages, limitations and practical recommendations. Molecular Ecology, 26(20), 5369–5406. Portico. https://doi.org/10.1111/mec.14264

Ugur Sezerman, O., Ulgen, E., Seymen, N., & Melis Durasi, I. 2019. Bioinformatics Workflows for Genomic Variant Discovery, Interpretation and Prioritization. Bioinformatics Tools for Detection and Clinical Interpretation of Genomic Variations. https://doi.org/10.5772/intechopen.85524

Van der Auwera, G. A., Carneiro, M. O., Hartl, C., Poplin, R., Del Angel, G., Levy-Moonshine, A., ... & DePristo, M. A. 2013. From FastQ data to high-confidence variant calls: the genome analysis toolkit best practices pipeline. Current Protocols in Bioinformatics, 43(1), 11-10. https://doi.org/10.1002/0471250953.bi1110s43

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., ... & DePristo, M. A. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Research, 20(9), 1297-1303. https://doi.org/10.1101/gr.107524.110

Li, H., & Durbin, R. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics, 25(14), 1754-1760. https://doi.org/10.1093/bioinformatics/btp324

Cingolani, P., Platts, A., Wang, L. L., Coon, M., Nguyen, T., Wang, L., ... & Ruden, D. M. 2012. A program for annotating and predicting the effects of single nucleotide polymorphisms, SnpEff: SNPs in the genome of Drosophila melanogaster strain w1118; iso-2; iso-3. Fly, 6(2), 80-92. https://doi.org/10.4161/fly.19695

DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V., Maguire, J. R., Hartl, C., ... & Daly, M. J. 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nature Genetics, 43(5), 491-498. https://doi.org/10.1038/ng.806

Robinson, J. T., Thorvaldsdóttir, H., Winckler, W., Guttman, M., Lander, E. S., Getz, G., & Mesirov, J. P. 2011. Integrative genomics viewer. Nature Biotechnology, 29(1), 24-26. https://doi.org/10.1038/nbt.1754

Rausch, T., Zichner, T., Schlattl, A., Stütz, A. M., Benes, V., & Korbel, J. O. 2012. DELLY: Structural variant discovery by integrated paired-end and split-read analysis. Bioinformatics, 28(18), i333-i339. https://doi.org/10.1093/bioinformatics/bts378

Whole exome sequencing (WES):

1. Quality Control
Tool: FastQC
Run FastQC on raw sequencing data
```
fastqc sample_R1.fastq.gz sample_R2.fastq.gz –o fastqc_output
```
fastqc: Command to run FastQC.
sample_R1.fastq.gz sample_R2.fastq.gz: Input FASTQ files (paired-end).
-o fastqc_output: Output directory for FastQC reports.

2. Read Trimming
Tool: Trimmomatic
Trim adapters and low-quality bases
```
trimmomatic PE –phred33 sample_R1.fastq.gz sample_R2.fastq.gz \
sample_R1_paired.fastq.gz sample_R1_unpaired.fastq.gz \
sample_R2_paired.fastq.gz sample_R2_unpaired.fastq.gz \
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10 LEADING:3 TRAILING:3
SLIDINGWINDOW:4:15 MINLEN:36
```
trimmomatic PE: Run Trimmomatic in paired-end mode.
-phred33: Quality encoding format.
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10: Adapter trimming parameters.
LEADING:3 TRAILING:3: Trim low-quality bases from the start and end.
SLIDINGWINDOW:4:15: Trim when average quality in a 4-base window falls below 15.
MINLEN:36: Discard reads shorter than 36 bases.

3. Alignment
Tool: BWA
Index the reference genome
bwa index reference.fasta

Align reads to the reference genome
Index the reference genome
```
bwa index reference.fasta
```
Align reads to the reference genome
```
bwa mem reference.fasta sample_R1_paired.fastq.gz
sample_R2_paired.fastq.gz > aligned.sam
```
bwa index: Index the reference genome.
bwa mem: Align reads using the BWA-MEM algorithm.

reference.fasta: Reference genome file.
sample_R1_paired.fastq.gz sample_R2_paired.fastq.gz: Input paired-end reads.
> aligned.sam: Output SAM file.

4. Post-alignment Processing
Tools: SAMtools, Picard
Convert SAM to BAM, sort, and index
```
samtools view -bS aligned.sam | samtools sort -o sorted.bam
```
samtools index sorted.bam
Mark duplicates
```
picard MarkDuplicates I=sorted.bam O=dedup.bam M=metrics.txt
```
samtools view -bS: Convert SAM to BAM.
samtools sort: Sort BAM file.
samtools index: Index BAM file.
picard MarkDuplicates: Mark duplicate reads.
I=sorted.bam
O=dedup.bam: Input and output files.
M=metrics.txt: Metrics file for duplicates.

5.Base Quality Score Recalibration (BQSR)
We'll use GATK for BQSR.
Generate recalibration table
```
gatk BaseRecalibrator \
    -I sample1_dedup.bam \
    -R reference_genome.fasta \
    --known-sites known_variants.vcf \
    -O recal_data.table
```
Apply BQSR
```
gatk ApplyBQSR \
    -R reference_genome.fasta \
    -I sample1_dedup.bam \
    --bqsr-recal-file recal_data.table \
    -O sample1_recal.bam
```
-I: Input BAM file
-R: Reference genome
--known-sites: Known variants file
-O: Output recalibration table or recalibrated BAM

6. Variant Calling
Tool: GATK
Create a sequence dictionary for the reference

```
gatk CreateSequenceDictionary -R reference.fasta
```
Call variants
```
gatk HaplotypeCaller -R reference.fasta -I dedup.bam -O
raw_variants.vcf
```
gatk CreateSequenceDictionary: Create a sequence dictionary.
gatk HaplotypeCaller: Call variants.
-R reference.fasta: Reference genome.
-I dedup.bam: Input BAM file.
-O raw_variants.vcf: Output VCF file.

7. Variant Annotation
Tool: ANNOVAR
Convert VCF to ANNOVAR input format
```
convert2annovar.pl -format vcf4 raw_variants.vcf >
variants.avinput
```
Annotate variants
```
table_annovar.pl variants.avinput humandb/ -buildver hg19 -out
annotated -remove -protocol refGene -operation g -nastring .
```
convert2annovar.pl: Convert VCF to ANNOVAR format.
table_annovar.pl: Annotate variants.
humandb/: Directory with ANNOVAR databases.
-buildver hg19: Genome build version.
-out annotated: Output file prefix.
-remove: Remove intermediate files.
-protocol refGene: Annotation protocol.
-operation g: Gene-based operation.
-nastring .: String for missing values.

8. Variant Filtering
Tool: BCFtools
Filter variants based on quality and frequency
```
bcftools filter -i 'QUAL>30 && DP>10 && ExAC_ALL<0.01'
annotated.vcf > filtered.vcf
```
bcftools filter: Filter variants.
-i 'QUAL>30 && DP>10 && ExAC_ALL<0.01': Inclusion criteria.
QUAL>30: Quality score greater than 30.
DP>10: Depth greater than 10.
ExAC_ALL<0.01: ExAC allele frequency less than 1%.

9. Statistical Analysis and Visualization in R
Tool: R

```
# Load necessary libraries
library(ggplot2)
library(VariantAnnotation)
# Read VCF file
vcf <- readVcf("filtered.vcf", "hg19")
# Summary statistics
summary(vcf)
# Plot variant quality
qual <- info(vcf)$QUAL
ggplot(data.frame(Quality=qual), aes(x=Quality)) +
  geom_histogram(binwidth=5) +
  theme_minimal() +
  labs(title="Variant Quality Distribution", x="Quality Score",
y="Frequency")
```

library(ggplot2): Load ggplot2 for plotting.
library(VariantAnnotation): Load VariantAnnotation for VCF handling.
readVcf: Read VCF file.
summary(vcf): Summary statistics of VCF.
ggplot: Create a plot.
geom_histogram: Plot histogram of quality scores.

References:

Bao, R., Huang, L., Andrade, J., Tan, W., Kibbe, W. A., Jiang, H., & Feng, G. 2014. Review of Current Methods, Applications, and Data Management for the Bioinformatics Analysis of Whole Exome Sequencing. Cancer Informatics, 13s2, CIN.S13779. https://doi.org/10.4137/cin.s13779

Koboldt, D. C. 2020. Best practices for variant calling in clinical sequencing. Genome Medicine, 12(1). https://doi.org/10.1186/s13073-020-00791-w

Pereira, R., Oliveira, J., & Sousa, M. 2020. Bioinformatics and Computational Tools for Next-Generation Sequencing Analysis in Clinical Genetics. Journal of Clinical Medicine, 9(1), 132. https://doi.org/10.3390/jcm9010132

Hanssen, F., Garcia, M. U., Folkersen, L., Pedersen, A. S., Lescai, F., Jodoin, S., Miller, E., Seybold, M., Wacker, O., Smith, N., Gabernet, G., & Nahnsen, S. 2024. Scalable and efficient DNA sequencing analysis on different compute infrastructures aiding variant discovery. NAR Genomics and Bioinformatics, 6(2). https://doi.org/10.1093/nargab/lqae031

DePristo, M. A., Banks, E., Poplin, R., Garimella, K. V., Maguire, J. R., Hartl, C., ... & Daly, M. J. 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nature Genetics, 43(5), 491-498. https://doi.org/10.1038/ng.806

Van der Auwera, G. A., Carneiro, M. O., Hartl, C., Poplin, R., Del Angel, G., Levy-Moonshine, A., ... & DePristo, M. A. 2013. From FastQ data to high-confidence variant calls: the Genome

Analysis Toolkit best practices pipeline. Current Protocols in Bioinformatics, 43(1), 11-10. https://doi.org/10.1002/0471250953.bi1110s43
Li, H., & Durbin, R. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics, 25(14), 1754-1760. https://doi.org/10.1093/bioinformatics/btp324
McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., ... & DePristo, M. A. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing next-generation DNA sequencing data. Genome Research, 20(9), 1297-1303. https://doi.org/10.1101/gr.107524.110
Wang, K., Li, M., & Hakonarson, H. 2010. ANNOVAR: functional annotation of genetic variants from high-throughput sequencing data. Nucleic Acids Research, 38(16), e164-e164. https://doi.org/10.1093/nar/gkq603

Bulk RNA-seq:

1. Quality Control of Raw Reads

First, we'll use FastQC to check the quality of our raw sequencing reads.

```
# Run FastQC on raw reads
fastqc sample1_R1.fastq.gz sample1_R2.fastq.gz -o fastqc_results
```

sample1_R1.fastq.gz sample1_R2.fastq.gz: Input FASTQ files (forward and reverse reads)
-o fastqc_results: Output directory for FastQC results

2. Adapter Trimming and Quality Filtering

We'll use Trimmomatic to remove adapter sequences and low-quality bases.

```
# Trim adapters and filter low-quality reads
java -jar trimmomatic-0.39.jar PE -threads 4 \
    sample1_R1.fastq.gz sample1_R2.fastq.gz \
    sample1_R1_trimmed.fastq.gz sample1_R1_unpaired.fastq.gz \
    sample1_R2_trimmed.fastq.gz sample1_R2_unpaired.fastq.gz \
    ILLUMINACLIP:TruSeq3-PE-2.fa:2:30:10 LEADING:3 TRAILING:3
SLIDINGWINDOW:4:15 MINLEN:36
```

PE: Paired-end mode
-threads 4: Use 4 CPU threads
ILLUMINACLIP:TruSeq3-PE-2.fa:2:30:10: Adapter trimming parameters
LEADING:3 TRAILING:3: Remove low-quality bases from the start and end
SLIDINGWINDOW:4:15: Perform sliding window trimming
MINLEN:36: Minimum length of reads to keep

3. Alignment to Reference Genome

We'll use HISAT2 to align the reads to a reference genome.

```
# Align reads to reference genome
hisat2 -p 4 -x reference_genome -1 sample1_R1_trimmed.fastq.gz -2 sample1_R2_trimmed.fastq.gz -S sample1_aligned.sam
```

-p 4: Use 4 CPU threads
-x reference_genome: Reference genome index
-1 sample1_R1_trimmed.fastq.gz -2 sample1_R2_trimmed.fastq.gz: Input paired-end reads
-S sample1_aligned.sam: Output SAM file

4. Convert SAM to BAM and Sort

We'll use SAMtools to convert the SAM file to BAM format and sort it.

```
# Convert SAM to BAM and sort
samtools view -bS sample1_aligned.sam | samtools sort -o sample1_sorted.bam
```

view -bS: Convert SAM to BAM
sort -o sample1_sorted.bam: Sort BAM file and output

5. Count Reads per Gene

We'll use featureCounts to count the number of reads mapped to each gene.

```
# Count reads per gene
featureCounts -T 4 -a annotation.gtf -o counts.txt sample1_sorted.bam
```

-T 4: Use 4 CPU threads
-a annotation.gtf: Gene annotation file
-o counts.txt: Output file for counts

6. Differential Expression Analysis in R

We'll use the DESeq2 package in R for differential expression analysis.

```
 Load DESeq2 library
library(DESeq2)
# Read in count data
countData <- read.table("counts.txt", header=TRUE, row.names=1)
colData <- data.frame(condition=factor(c("control", "treated")))
# Create DESeq2 dataset
dds <- DESeqDataSetFromMatrix(countData=countData, colData=colData, design=~condition)
# Run DESeq2
dds <- DESeq(dds)
# Get results
res <- results(dds)
# Plot results
```

```
plotMA(res, main="DESeq2", ylim=c(-2,2))
```
DESeqDataSetFromMatrix: Creates a DESeq2 dataset
design=~condition: Design formula for the experiment
DESeq: Runs the DESeq2 analysis
results: Extracts results from the analysis
plotMA: Plots the results

References:

Jiang, G., Zheng, J.-Y., Ren, S.-N., Yin, W., Xia, X., Li, Y., & Wang, H.-L. 2024. A comprehensive workflow for optimizing RNA-seq data analysis. BMC Genomics, 25(1). https://doi.org/10.1186/s12864-024-10414-y

Conesa, A., Madrigal, P., Tarazona, S., Gomez-Cabrero, D., Cervera, A., McPherson, A., Szcześniak, M. W., Gaffney, D. J., Elo, L. L., Zhang, X., & Mortazavi, A. 2016. A survey of best practices for RNA-seq data analysis. Genome Biology, 17(1), 13. https://doi.org/10.1186/s13059-016-0881-8

Love, M. I., Huber, W., & Anders, S. 2014. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8

Pertea, M., Kim, D., Pertea, G. M., Leek, J. T., & Salzberg, S. L. 2016. Transcript-level expression analysis of RNA-seq experiments with HISAT, StringTie and Ballgown. Nature Protocols, 11(9), 1650-1667. https://doi.org/10.1038/nprot.2016.095

Single cell RNA-seq (scRNAseq) :

1. Quality Control of Raw Reads

First, we'll use FastQC to check the quality of our raw sequencing reads.

```
# Run FastQC on raw reads
fastqc sample1_R1.fastq.gz sample1_R2.fastq.gz -o fastqc_results
```

sample1_R1.fastq.gz sample1_R2.fastq.gz : Input FASTQ files (forward and reverse reads)
-o fastqc_results : Output directory for FastQC results

2. Adapter Trimming and Quality Filtering

We'll use Trimmomatic to remove adapter sequences and low-quality bases.

```
# Trim adapters and filter low-quality reads
java -jar trimmomatic-0.39.jar PE -threads 4 \
    sample1_R1.fastq.gz sample1_R2.fastq.gz \
    sample1_R1_trimmed.fastq.gz sample1_R1_unpaired.fastq.gz \
    sample1_R2_trimmed.fastq.gz sample1_R2_unpaired.fastq.gz \
    ILLUMINACLIP:TruSeq3-PE-2.fa:2:30:10 LEADING:3 TRAILING:3
SLIDINGWINDOW:4:15 MINLEN:36
```

PE: Paired-end mode
-threads 4: Use 4 CPU threads
ILLUMINACLIP:TruSeq3-PE-2.fa:2:30:10: Adapter clipping parameters
LEADING:3 TRAILING:3: Remove low-quality bases from the start and end
SLIDINGWINDOW:4:15: Perform sliding window trimming
MINLEN:36: Minimum read length to keep

3. Alignment to Reference Genome

We'll use STAR for aligning reads to the reference genome.

Align reads to reference genome

```
STAR --runThreadN 4 --genomeDir genome_index --readFilesIn
sample1_R1_trimmed.fastq.gz sample1_R2_trimmed.fastq.gz --
readFilesCommand zcat --outFileNamePrefix sample1_ --outSAMtype
BAM SortedByCoordinate
```

--runThreadN 4 : Use 4 CPU threads
--genomeDir genome_index : Directory containing the genome index
--readFilesIn : Input trimmed FASTQ files
--readFilesCommand zcat : Decompress gzipped files
--outFileNamePrefix sample1_ : Prefix for output files
--outSAMtype BAM SortedByCoordinate : Output sorted BAM file

4. Quality Control of Aligned Reads

We'll use samtools to assess the quality of the aligned reads.

Index the BAM file

samtools index sample1_Aligned.sortedByCoord.out.bam

Generate alignment statistics

```
samtools flagstat sample1_Aligned.sortedByCoord.out.bam >
alignment_stats.txt
```

- index: Index the BAM file for fast access
- flagstat: Generate alignment statistics

5. Quantification of Gene Expression

We'll use featureCounts to quantify gene expression.

Quantify gene expression

```
featureCounts -T 4 -a annotation.gtf -o counts.txt
sample1_Aligned.sortedByCoord.out.bam
-T 4: Use 4 CPU threads
```

-a annotation.gtf : Annotation file in GTF format
-o counts.txt: Output file for gene counts

6. Clustering Analysis

We'll use the Seurat package in R for Clustering analysis.

```
# Load necessary libraries
library(Seurat)
library(dplyr)
library(ggplot2)
# Load the count data
counts <- Read10X(data.dir = "filtered_feature_bc_matrix/")
# Create a Seurat object
seurat_obj <- CreateSeuratObject(counts = counts, project =
"scRNAseq", min.cells = 3, min.features = 200)
# Normalize the data
seurat_obj <- NormalizeData(seurat_obj)
# Identify highly variable features
seurat_obj <- FindVariableFeatures(seurat_obj, selection.method
= "vst", nfeatures = 2000)
# Scale the data
seurat_obj <- ScaleData(seurat_obj)
# Perform linear dimensional reduction
seurat_obj <- RunPCA(seurat_obj, features =
VariableFeatures(object = seurat_obj))
# Cluster the cells
seurat_obj <- FindNeighbors(seurat_obj, dims = 1:10)
seurat_obj <- FindClusters(seurat_obj, resolution = 0.5)
# Run non-linear dimensional reduction (UMAP/tSNE)
seurat_obj <- RunUMAP(seurat_obj, dims = 1:10)
# Visualize the clusters
DimPlot(seurat_obj, reduction = "umap")
```

Read10X: Reads 10X Genomics data
CreateSeuratObject : Creates a Seurat object
NormalizeData : Normalizes the data
FindVariableFeatures : Identifies highly variable features
ScaleData : Scales the data
RunPCA : Performs PCA
FindNeighbors : Finds neighboring cells
FindClusters : Clusters the cells
RunUMAP : Runs UMAP for visualization
DimPlot: Plots the clusters

7. Differential Expression Analysis

```
# Find markers for each cluster
```

```
all_markers <- FindAllMarkers(seurat_obj, only.pos = TRUE,
min.pct = 0.25, logfc.threshold = 0.25)
# View the top markers for each cluster
top_markers <- all_markers %>%
    group_by(cluster) %>%
    top_n(n = 10, wt = avg_log2FC)

# Visualize marker expression
FeaturePlot(seurat_obj, features = top_markers$gene[1:9],
min.cutoff = "q9")
# Heatmap of top markers
DoHeatmap(seurat_obj, features = top_markers$gene) + NoLegend()
```

8. Trajectory Analysis

```
# For this example, we'll use Monocle 3 for trajectory analysis
# Load necessary libraries
library(monocle3)
# Convert Seurat object to cell_data_set object
cds <- as.cell_data_set(seurat_obj)
cds <- cluster_cells(cds)
# Learn the trajectory graph
cds <- learn_graph(cds)
# Order cells along the trajectory
cds <- order_cells(cds)
# Plot the trajectory
plot_cells(cds,
           color_cells_by = "pseudotime",
           label_cell_groups = FALSE,
           label_leaves = FALSE,
           label_branch_points = FALSE)
# Visualize genes changing as a function of pseudotime
plot_genes_in_pseudotime(cds[c("Gene1", "Gene2", "Gene3")],
color_cells_by = "cluster")
# Find genes that change as a function of pseudotime
pr_test_res <- graph_test(cds, neighbor_graph =
"principal_graph", cores = 4)
pr_deg_ids <- row.names(subset(pr_test_res, q_value < 0.05))
# Plot the top genes
plot_genes_in_pseudotime(cds[pr_deg_ids[1:10]], color_cells_by =
"cluster")
```

References:

Stuart, T., Butler, A., Hoffman, P., Hafemeister, C., Papalexi, E., Mauck III, W. M., ... & Satija, R. 2019. Comprehensive integration of single-cell data. Cell, 177(7), 1888-1902. https://doi.org/10.1016/j.cell.2019.05.031

Luecken, M. D., & Theis, F. J. 2019. Current best practices in single-cell RNA-seq analysis: a tutorial. Molecular Systems Biology, 15(6), e8746. https://doi.org/10.15252/msb.20188746

Hwang, B., Lee, J. H., & Bang, D. 2018. Single-cell RNA sequencing technologies and bioinformatics pipelines. Experimental & Molecular Medicine, 50(8), 1-14. https://doi.org/10.1038/s12276-018-0071-8

Small RNA Sequencing:

1. Quality Control of Raw Reads

First, we'll use FastQC to check the quality of our raw sequencing reads.

Run FastQC on raw reads

```
fastqc forward.fastq.gz reverse.fastq.gz -o fastqc_results
```

sample1_R1.fastq.gz sample1_R2.fastq.gz: Input FASTQ files (forward and reverse reads)

-o fastqc_results: Output directory for FastQC results

2. Adapter Trimming

Next, we'll trim adapters and low-quality bases using Trimmomatic.

Install Trimmomatic if not already installed

Run Trimmomatic on a paired-end sample

```
java -jar /path/to/trimmomatic.jar PE -threads 4 \
    input_forward.fastq.gz input_reverse.fastq.gz \
    output_forward_paired.fastq.gz
output_forward_unpaired.fastq.gz \
    output_reverse_paired.fastq.gz
output_reverse_unpaired.fastq.gz \
    ILLUMINACLIP:/path/to/adapters/TruSeq3-PE.fa:2:30:10 \
    LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

java -jar /path/to/trimmomatic.jar: Runs Trimmomatic using Java.

PE: Indicates paired-end sequencing data.

-threads 4: Uses 4 CPU threads for faster processing.

input_forward.fastq.gz input_reverse.fastq.gz: Input files for forward and reverse reads.

output_forward_paired.fastq.gz output_forward_unpaired.fastq.gz

output_reverse_paired.fastq.gz output_reverse_unpaired.fastq.gz: Output files for paired and unpaired reads after trimming.

ILLUMINACLIP:/path/to/adapters/TruSeq3-PE.fa:2:30:10: Trims Illumina adapters.

LEADING:3 TRAILING:3: Trims low-quality bases from the start and end of reads.

SLIDINGWINDOW:4:15: Trims bases in a sliding window if the average quality drops below 15.
MINLEN:36: Discards reads shorter than 36 bases after trimming.

3. Alignment to Reference Genome

We'll use Bowtie to align the trimmed reads to a reference genome.

```
# Install Bowtie if not already installed
# Build Bowtie index for the reference genome
bowtie-build reference_genome.fa reference_genome_index
# Align reads to the reference genome
bowtie -p 4 -v 2 -m 1 --best --strata \
    reference_genome_index \
    -1 output_forward_paired.fastq.gz \
    -2 output_reverse_paired.fastq.gz \
    -S aligned_reads.sam
```

bowtie-build reference_genome.fa reference_genome_index: Builds a Bowtie index for the reference genome.
bowtie: Runs the Bowtie aligner.
-p 4: Uses 4 CPU threads.
-v 2: Allows up to 2 mismatches.
-m 1: Reports only uniquely mapped reads.
--best --strata: Reports the best alignment for each read.
reference_genome_index: The prefix of the Bowtie index files.
-1 output_forward_paired.fastq.gz -2 output_reverse_paired.fastq.gz: Input files for forward and reverse reads.
-S aligned_reads.sam: Output file in SAM format.

4. Convert SAM to BAM and Sort

We'll convert the SAM file to BAM format and sort it using SAMtools.

```
# Convert SAM to BAM and sort
samtools view -bS aligned_reads.sam | samtools sort -o
sorted_aligned_reads.bam
```

samtools view -bS aligned_reads.sam: Converts SAM to BAM format.
|: Pipes the output to the next command.
samtools sort -o sorted_aligned_reads.bam: Sorts the BAM file and saves it as sorted_aligned_reads.bam.
Now, let's switch to R Studio for the remaining analysis steps.

5. Read Count Quantification

We'll use the Rsubread package to quantify read counts.

```
# Install and load required packages
```

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("Rsubread")
library(Rsubread)

# Quantify read counts
fc <- featureCounts ("sorted_aligned_reads.bam",
                     annot.ext = "annotation.gtf",
                     isGTFAnnotationFile = TRUE,
                     GTF.featureType = "exon",
                     GTF.attrType = "gene_id",
                     useMetaFeatures = TRUE,
                     allowMultiOverlap = FALSE,
                     nthreads = 4)
# Save count data
write.csv(fc$counts, "gene_counts.csv")
```

featureCounts(): Function from Rsubread to count reads.
"sorted_aligned_reads.bam": Input BAM file.
annot.ext = "annotation.gtf": GTF annotation file.
isGTFAnnotationFile = TRUE: Specifies that we're using a GTF file.
GTF.featureType = "exon": Counts reads aligning to exons.
GTF.attrType = "gene_id": Uses gene_id as the identifier.
useMetaFeatures = TRUE: Summarizes counts at the gene level.
allowMultiOverlap = FALSE: Doesn't count reads overlapping multiple features.
nthreads = 4: Uses 4 CPU threads.

6. Differential Expression Analysis

We'll use the DESeq2 package for differential expression analysis.

```
# Install and load required packages
BiocManager::install("DESeq2")
library(DESeq2)
# Load count data
count_data <- read.csv("gene_counts.csv", row.names = 1)
# Create sample information
sample_info <- data.frame(
  condition = factor(c("control", "control", "treatment",
"treatment")),
  row.names = colnames(count_data)
)
# Create DESeqDataSet object
dds <- DESeqDataSetFromMatrix(countData = count_data,
                              colData = sample_info,
```

```
                                design = ~ condition)
```
Run DESeq2 analysis
```
dds <- DESeq(dds)
```
Get results
```
res <- results(dds)
```
Order results by adjusted p-value
```
res_ordered <- res[order(res$padj), ]
```
Save results
```
write.csv(as.data.frame(res_ordered),
"differential_expression_results.csv")
```
Explanation:

DESeqDataSetFromMatrix(): Creates a DESeqDataSet object.

countData = count_data: Input count data.

colData = sample_info: Sample information.

design = ~ condition: Specifies the experimental design.

DESeq(dds): Performs differential expression analysis.

results(dds): Extracts results from the DESeq analysis.

res[order(res$padj),]: Orders results by adjusted p-value.

7.Visualization

Finally, we'll create some visualizations using ggplot2.

Install and load required packages
```
install.packages("ggplot2")
library(ggplot2)
```
MA plot
```
ggplot(as.data.frame(res), aes(x = baseMean, y =
log2FoldChange)) +
  geom_point(aes(color = padj < 0.05)) +
  scale_x_log10() +
  ggtitle("MA Plot") +
  xlab("Mean of Normalized Counts") +
  ylab("Log2 Fold Change")
ggsave("ma_plot.png")
```
Volcano plot
```
ggplot(as.data.frame(res), aes(x = log2FoldChange, y = -
log10(padj))) +
  geom_point(aes(color = padj < 0.05)) +
  ggtitle("Volcano Plot") +
  xlab("Log2 Fold Change") +
  ylab("-Log10 Adjusted P-value")
ggsave("volcano_plot.png")
```
Explanation:

ggplot(): Creates a ggplot object.
aes(): Defines aesthetics for the plot.
geom_point(): Adds points to the plot.
scale_x_log10(): Applies log10 scale to x-axis.
ggtitle(), xlab(), ylab(): Add title and axis labels.
ggsave(): Saves the plot as an image file.

References:

Tam, S., Tsao, M. S., & McPherson, J. D. 2015. Optimization of miRNA-seq data preprocessing. Briefings in Bioinformatics, 16(6), 950-963. https://doi.org/10.1093/bib/bbv019

Friedländer, M. R., Mackowiak, S. D., Li, N., Chen, W., & Rajewsky, N. 2012. miRDeep2 accurately identifies known and hundreds of novel microRNA genes in seven animal clades. Nucleic Acids Research, 40(1), 37-52. https://doi.org/10.1093/nar/gkr688

Love, M. I., Huber, W., & Anders, S. 2014. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8

de novo Genome Assembly:

1. Quality Control of Raw Reads

```
fastqc sample1_R1.fastq.gz sample1_R2.fastq.gz -o fastqc_results
```

sample1_R1.fastq.gz sample1_R2.fastq.gz: Input FASTQ files (forward and reverse reads)
-o fastqc_results: Output directory for FastQC results

2. Adapter Trimming and Quality Filtering

Next, we'll use Trimmomatic to remove adapter sequences and low-quality bases from the reads.

```
# Trim adapters and filter low-quality reads
java -jar trimmomatic-0.39.jar PE -threads 4 \
    sample1_R1.fastq.gz sample1_R2.fastq.gz \
    sample1_R1_trimmed.fastq.gz sample1_R1_unpaired.fastq.gz \
    sample1_R2_trimmed.fastq.gz sample1_R2_unpaired.fastq.gz \
    ILLUMINACLIP:TruSeq3-PE.fa:2:30:10 LEADING:3 TRAILING:3
SLIDINGWINDOW:4:15 MINLEN:36
```

PE: Paired-end mode
-threads 4: Use 4 CPU threads
sample1_R1.fastq.gz sample1_R2.fastq.gz: Input FASTQ files
sample1_R1_trimmed.fastq.gz sample1_R1_unpaired.fastq.gz: Output files for trimmed and unpaired reads

ILLUMINACLIP:TruSeq3-PE.fa:2:30:10: Adapter clipping parameters
LEADING:3 TRAILING:3: Remove low-quality bases from the start and end
SLIDINGWINDOW:4:15: Perform a sliding window trimming
MINLEN:36: Minimum length of reads to keep

3. Genome Assembly

We'll use SPAdes for the de novo assembly of the genome.

Run SPAdes for genome assembly

```
spades.py -1 sample1_R1_trimmed.fastq.gz -2
sample1_R2_trimmed.fastq.gz -o spades_output
```

-1 sample1_R1_trimmed.fastq.gz: Forward reads
-2 sample1_R2_trimmed.fastq.gz: Reverse reads
-o spades_output: Output directory for assembly results

4. Assembly Quality Assessment

Finally, we'll use QUAST to assess the quality of the assembled genome.

Run QUAST for assembly quality assessment

```
quast.py spades_output/contigs.fasta -o quast_results
```

spades_output/contigs.fasta: Assembled contigs file
-o quast_results: Output directory for QUAST results

References:

Sohn, J. I., & Nam, J. W. 2018. The present and future of de novo whole-genome assembly. Briefings in Bioinformatics, 19(1), 23-40. https://doi.org/10.1093/bib/bbw096

Ekblom, R., & Wolf, J. B. 2014. A field guide to whole-genome sequencing, assembly and annotation. Evolutionary Applications, 7(9), 1026-1042. https://doi.org/10.1111/eva.12178

Badouin, H., Gouzy, J., Grassa, C. J., Murat, F., Staton, S. E., Cottret, L., ... & Langlade, N. B. 2017. The sunflower genome provides insights into oil metabolism, flowering and Asterid evolution. Nature, 546(7656), 148-152. https://doi.org/10.1038/nature22380

de novo transcriptome assembly – RNA-seq:

1. Quality Control of Raw Reads

First, we'll use FastQC to check the quality of our raw sequencing reads.

FastQC command

```
fastqc sample1_R1.fastq.gz sample1_R2.fastq.gz -o fastqc_results
```

sample1_R1.fastq.gz sample1_R2.fastq.gz: Input FASTQ files (forward and reverse reads)
-o fastqc_results: Output directory for FastQC results

This command runs FastQC on the input FASTQ files and generates quality reports in the specified output directory.

2. Adapter Trimming and Quality Filtering

Next, we'll use Trimmomatic to remove adapter sequences and low-quality bases from the reads.

```
# Trim adapters and filter low-quality reads
java -jar trimmomatic-0.39.jar PE -threads 4 \
    sample1_R1.fastq.gz sample1_R2.fastq.gz \
    sample1_R1_trimmed.fastq.gz sample1_R1_unpaired.fastq.gz \
    sample1_R2_trimmed.fastq.gz sample1_R2_unpaired.fastq.gz \
    ILLUMINACLIP:TruSeq3-PE.fa:2:30:10 LEADING:3 TRAILING:3
SLIDINGWINDOW:4:15 MINLEN:36
```

PE: Paired-end mode
-threads 4: Use 4 CPU threads
sample1_R1.fastq.gz sample1_R2.fastq.gz: Input FASTQ files
sample1_R1_trimmed.fastq.gz sample1_R1_unpaired.fastq.gz: Output files for trimmed and unpaired reads
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10: Adapter trimming parameters
LEADING:3 TRAILING:3: Remove low-quality bases from the start and end
SLIDINGWINDOW:4:15: Perform sliding window trimming
MINLEN:36: Discard reads shorter than 36 bases

3. *de novo* Transcriptome Assembly

We'll use Trinity for the de novo assembly of the transcriptome.

```
# Run Trinity for transcriptome assembly
Trinity --seqType fq --max_memory 50G --CPU 8 \
    --left sample1_R1_trimmed.fastq.gz --right
sample1_R2_trimmed.fastq.gz \
    --output trinity_output
```

--seqType fq: Input file type is FASTQ
--max_memory 50G: Maximum memory to use
--CPU 8: Number of CPU threads to use
--left sample1_R1_trimmed.fastq.gz --right sample1_R2_trimmed.fastq.gz: Input trimmed reads
--output trinity_output: Output directory for assembly results

4. Assembly Quality Assessment

Finally, we'll use TransRate to assess the quality of the assembled transcriptome.

```
# Run TransRate for assembly quality assessment
```

```
transrate --assembly trinity_output/Trinity.fasta --left
sample1_R1_trimmed.fastq.gz --right sample1_R2_trimmed.fastq.gz
--output transrate_results
```
--assembly trinity_output/Trinity.fasta: Assembled transcriptome file
--left sample1_R1_trimmed.fastq.gz --right sample1_R2_trimmed.fastq.gz: Input trimmed reads
--output transrate_results: Output directory for TransRate results

5. Indexing was performed using the below command:
```
$ bowtie2-build Trinity.out/Trinity.fasta
bowtie2build.out/I1.reference
```
The first and second arguments in the command line show the input and output
path/name, respectively.
Since RSEM (used in the next step of the analysis pipeline) currently does not handle indel, local and discordant alignments, the parameters in Bowtie2 aligner were set in a way to avoid those alignments. This rate can be set by option '--bowtie2-mismatch-rate'. If reads are paired-end, we additionally use options '--no-mixed' and '--no-discordant'.(Default: off)

6. Alignment:
```
$ bowtie2 -q --phred33 --sensitive --dpad 0 --gbar 99999999 --mp
1,1 --np 1 --scoremin L,0,-0.1 -I 1 -X 1000 --no-mixed --no-
discordant -p 16 -k 200 -x I1.reference
-1 validfile_1.fq.gz -2 validfile_2.fq.gz | samtools view -S -b
-o validfile.bam
```

-q shows that reads (specified with -1 and -2) are FASTQ files,
-- phred33 shows that input quality equals to the Phred plus 33 encoding used by the very latest Illumina
pipelines,
--dpad “Pads” dynamic programming problems by 0 columns on either side to allow gaps,
--gbar disallowed gaps within 99999999 positions of the beginning or end of the read,
--mp MX,MN sets the maximum (MX) and minimum (MN) mismatch penalties
both of which are integers,
--np sets the penalty for positions containing ambiguous character (such as N) in the read,reference, or both,
--score-min sets a function governing the minimum alignment score needed for an alignment to be considered valid enough to
report. Specifying L,0,-0.1 sets the minimum-score function f to f(x) = 0 + -0.1 * x, where x is the read length and the last parameter '-0.1', is the negative of maximum mismatch rate,
-I and -X are paired-end options which respectively exhibit the minimum and maximum fragment length for valid paired-end alignments. These two make the “outer distance” which is

expected range of inter-mates distances measured from the furthest
extremes,
--no-mixed this option disables the program to find alignments for the
individual mates when bowtie2 cannot find a concordant or discordant alignment for a pair,
--no-discordant disallowed Bowtie2 for discordant alignments when it could not find any concordant alignments; A discordant alignment is an alignment where both mates align uniquely but that does not satisfy the paired-end constraints,
-p uses a packed (2-bits-per-nucleotide) representation for transcript strings.
This saves memory but makes indexing 2-3 times slower, -k Bowtie2 searches for at most k number of distinct, valid alignments for each read, All alignments found were reported in descending order by
alignment score,
-x specifies the basename of the reference index by bowtie2-built, -1
includes the file (or files as a comma-separated list) containing mate(s) 1. The sequence(s) specified with this argument must correspond file-for-file and read-for-read with that
(those) specified in the mate 2 argument of "-2"

The second part of the command contained the conversion of the output to BAM format using Samtools. Ultimately, the resulted BAM files were sorted by name using samtools:

```
$ samtools sort –n validfile.bam –o validfile–namesorted.bam
```

7. Expression calculation

RSEM is a software package for expression estimation from RNA-Seq data at gene and isoform levels, generating BAM files in transcript-coordinate which can be visualized by IGV. A basic feature unique to RSEM is that reference genome is not a prerequisite . As the typical RSEM analysis, our run consisted of two steps:

1) The reference transcript set was preprocessed for use by later RSEM analysis using the script "rsem-prepare-reference

The reference transcript set was preprocessed for use by later RSEM analysis using
the script "rsem-prepare-reference".

```
$ rsem–prepare–reference –p 8 ––transcript–to–gene–map map.txt –
––bowtie2 I1.reference.fasta I1.reference
```

-p 8 tells RSEM to use 8 threads,
--transcript-to-gene-map option: when gene-level quantification is desired,
RSEM via a Perl script provides map file in which the transcripts of the same gene are specified,
--bowtie2 tells RSEM to align reads using Bowtie 2,
I1.reference.fasta is introduced as the input to yield the output files with I1.reference basename.

2) The reference-aligned reads resulted from the previous section was used to estimate abundances and their credibility intervals using "rsem-calculate-expression". The rsemcalculate-expression script handles both the read alignment against reference and abundance calculation.

Alternatively, in this study, Bowtie2 was the user-provided read aligner through which the alignments in SAM format were manually provided as the entry to RSEM.

The name-sorted BAM files of the reads were converted to RSEM appropriate ones using the command below:

```
$ convert-sam-for-rsem validfile-namesorted.bam validfile-namesorted-converted
```

And finally, the outcome was used to quantify the expression by typing as bellow:

```
$ rsem-calculate-expression --alignments --paired-end validfile-sorted-converted.bam I1.reference validfile-sorted-converted-expressed
```

--alignments and --paired-end show that the entries are as alignments and paired-end reads, then the command continued with input-, reference- and sample-names,

8. Differential Expression Analysis

DESeq is an efficient package in R software environment which tests for differential expression of count data from highthroughput sequencing assays . It expects count data in the form of a table including integer values, as an input and uses negative binomial generalized linear models. Before running the DESeq2 functions, a pre-filtering was performed to remove low count genes and keep the rows that have at least 10 reads total. Then the differential genes were estimated using 1% false discovery rate (FDR) by applying padj < 0.01. FDR is a general statement about the probability of false discoveries given a certain number of simultaneous tests and their p-value distribution. The padj, also known as q-value is the p-value adjusted for multiple testing with controlling the number of false positives across all significant statistical tests.DEseq was operated by typing the code below:

```
directory <- "H:/RSEM.counts/RSEM1vsRSEM2"
sampleFiles <- grep(".counts",list.files(directory),value=TRUE)
sampleFiles
treatment <- c("RSEM1", " RSEM1", " RSEM2", " RSEM2")
treatment
sampleTable <- data.frame(sampleName = sampleFiles,
fileName = sampleFiles,
treatment = treatment)
sampleTable
library("DESeq2")
dds <- DESeqDataSetFromMatrix(sampleTable = sampleTable,
directory = directory,
design= ~ treatment)
dds <- DESeq(dds)
```

```
ddsnorm <- estimateSizeFactors(dds)
ddsnorm
keep <- rowSums(counts(ddsnorm, normalized=TRUE) >= 10) >= 1
filtered.counts <- ddsnorm[keep,]
filtered.counts
dds<- filtered.counts
dds
dds$group <- factor(paste0(dds$treatment))
dds$group
design(dds) <- ~ group
dds <- DESeq(dds)
48
resultsNames(dds)
#DE-genes between RSEM1 and RSEM2.
RSEM1vsRSEM2 <- results(dds, contrast=c('group' , 'RSEM1' , 'RSEM1'))
head(RSEM1vsRSEM2)
sum(RSEM1vsRSEM2$padj < 0.01, na.rm=TRUE)
resAOrdered<- RSEM1vsRSEM2[order(RSEM1vsRSEM2$padj),]
resASig <- subset(resAOrdered, padj<0.01)
resASig
write.table(resASig, file='H:/RSEM.counts/ RSEM1vsRSEM2.txt' ,sep="\t",quote=F)
```

RSEM1, RSEM2, … are the renamed form of RSEM counts for different treatment and time conditions (the name was originally expressed as validfile-sorted-convertedexpressed in the previous section).

Using the Excel software the sequence of the DE-genes (as the longest isoform) was extracted from the trinity file and were prepared for the next step. The ratio of FPKM values was used to make direct comparisons between the samples.

References:

Haas, B. J., Papanicolaou, A., Yassour, M., Grabherr, M., Blood, P. D., Bowden, J., ... & Regev, A. 2013. De novo transcript sequence reconstruction from RNA-seq using the Trinity platform for reference generation and analysis. Nature Protocols, 8(8), 1494-1512. https://doi.org/10.1038/nprot.2013.084

Conesa, A., Madrigal, P., Tarazona, S., Gomez-Cabrero, D., Cervera, A., McPherson, A., ... & Mortazavi, A. 2016. A survey of best practices for RNA-seq data analysis. Genome Biology, 17(1), 13. https://doi.org/10.1186/s13059-016-0881-8

Love, M. I., Huber, W., & Anders, S. 2014. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8

Grabherr, M. G., Haas, B. J., Yassour, M., Levin, J. Z., Thompson, D. A., Amit, I., ... & Regev, A. 2011. Full-length transcriptome assembly from RNA-Seq data without a reference genome. Nature Biotechnology, 29(7), 644-652. https://doi.org/10.1038/nbt.1883

Martin, M. 2011. Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet Journal, 17(1), 10-12. https://doi.org/10.14806/ej.17.1.200

Targeted sequencing (TGS):

1. Quality Control

Tool: FastQC

Run FastQC on raw sequencing data

```
fastqc sample_R1.fastq.gz sample_R2.fastq.gz –o fastqc_output
```

fastqc: Command to run FastQC.
sample_R1.fastq.gz sample_R2.fastq.gz: Input FASTQ files (paired-end).
-o fastqc_output: Output directory for FastQC reports.

2. Read Trimming

Tool: Trimmomatic

Trim adapters and low-quality bases

```
trimmomatic PE –phred33 sample_R1.fastq.gz sample_R2.fastq.gz \
sample_R1_paired.fastq.gz sample_R1_unpaired.fastq.gz \
sample_R2_paired.fastq.gz sample_R2_unpaired.fastq.gz \
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10      LEADING:3      TRAILING:3
SLIDINGWINDOW:4:15 MINLEN:36
```

trimmomatic PE: Run Trimmomatic in paired-end mode.
-phred33: Quality encoding format.
ILLUMINACLIP:TruSeq3-PE.fa:2:30:10: Adapter trimming parameters.
LEADING:3 TRAILING:3: Trim low-quality bases from the start and end.
SLIDINGWINDOW:4:15: Trim when average quality in a 4-base window falls below 15.
MINLEN:36: Discard reads shorter than 36 bases.

3. Alignment

Tool: BWA

Index the reference genome

```
bwa index reference.fasta
```

Align reads to the reference genome

Index the reference genome

```
bwa index reference.fasta
```
Align reads to the reference genome
```
bwa mem reference.fasta sample_R1_paired.fastq.gz
sample_R2_paired.fastq.gz > aligned.sam
```
bwa index: Index the reference genome.
bwa mem: Align reads using the BWA-MEM algorithm.
reference.fasta: Reference genome file.
sample_R1_paired.fastq.gz sample_R2_paired.fastq.gz: Input paired-end reads.
> aligned.sam: Output SAM file.

4. Convert SAM to BAM and Sort

Tool: SAMtools
Convert SAM to BAM
```
samtools view -S -b aligned_reads.sam > aligned_reads.bam
```
Sort BAM file
```
samtools sort aligned_reads.bam -o sorted_reads.bam
```
samtools view: Converts SAM to BAM format.
-S -b: Options to specify input is SAM and output is BAM.
samtools sort: Sorts the BAM file.
-o sorted_reads.bam: Output sorted BAM file.

5. Post-alignment Processing
Tools: Picard
Mark duplicates
```
picard MarkDuplicates I= sorted_reads.bam O=dedup.bam
M=metrics.txt
```
picard MarkDuplicates: Mark duplicate reads.
I=sorted.bam
O=dedup.bam: Input and output files.
M=metrics.txt: Metrics file for duplicates.

6. Base Quality Score Recalibration (BQSR)
We'll use GATK for BQSR.
Generate recalibration table
```
gatk BaseRecalibrator \
    -I sample1_dedup.bam \
    -R reference_genome.fasta \
    --known-sites known_variants.vcf \
    -O recal_data.table
```
Apply BQSR
```
gatk ApplyBQSR \
```

```
    -R reference_genome.fasta \
    -I sample1_dedup.bam \
    --bqsr-recal-file recal_data.table \
    -O sample1_recal.bam
```

-I: Input BAM file
-R: Reference genome
--known-sites: Known variants file
-O: Output recalibration table or recalibrated BAM

7. Variant Calling

Tool: GATK (Genome Analysis Toolkit)

```
# Call variants
gatk HaplotypeCaller -R reference_genome.fasta -I dedup.bam -O raw_variants.vcf
```

gatk HaplotypeCaller: Command to call variants.
-R reference_genome.fasta: Reference genome file.
-I sorted_reads.bam: Input sorted BAM file.
-O raw_variants.vcf: Output file with raw variants in VCF format.

8. Annotation

Tool: ANNOVAR

```
# Annotate variants
table_annovar.pl raw_variants.vcf humandb/ -buildver hg19 -out annotated_variants -remove -protocol refGene -operation g -nastring . -vcfinput
```

table_annovar.pl: Command to run ANNOVAR.
raw_variants.vcf: Input VCF file with raw variants.
humandb/: Directory with annotation databases.
-buildver hg19: Genome build version.
-out annotated_variants: Output prefix for annotated variants.
-remove: Remove intermediate files.
-protocol refGene: Annotation protocol.
-operation g: Operation type (gene-based).
-nastring .: String for missing values.
-vcfinput: Input is a VCF file.

9. Filtering

Tool:VCFtools

```
# Filter variants
```

```
vcftools --vcf annotated_variants.vcf --remove-indels --minQ 30
--recode --out filtered_variants
```

vcftools: Command to run VCFtools.
--vcf annotated_variants.vcf: Input VCF file with annotated variants.
--remove-indels: Remove indels from the VCF.
--minQ 30: Minimum quality score of 30.
--recode: Recode the VCF file.
--out filtered_variants: Output prefix for filtered variants.

10. Visualization and Analysis in R

```
# Load necessary libraries
library(VariantAnnotation)
library(ggplot2)
# Read VCF file
vcf <- readVcf("filtered_variants.recode.vcf", "hg19")
# Summary of variants
summary(vcf)
# Plot variant quality
qual <- qual(vcf)
ggplot(data.frame(Quality=qual), aes(x=Quality)) +
  geom_histogram(binwidth=1) +
  theme_minimal() +
  labs(title="Variant Quality Distribution", x="Quality Score",
y="Frequency")
```

library(VariantAnnotation): Load VariantAnnotation package for handling VCF files.
library(ggplot2): Load ggplot2 for visualization.
readVcf: Function to read VCF files.
summary(vcf): Provides a summary of the variants.
qual(vcf): Extracts quality scores from the VCF.
ggplot: Function to create plots.
geom_histogram: Creates a histogram of quality scores.

References:

DePristo, M.A., Banks, E., Poplin, R., Garimella, K.V., Maguire, J.R., Hartl, C., Philippakis, A.A., Del Angel, G., Rivas, M.A., Hanna, M. and McKenna, A., 2011. A framework for variation discovery and genotyping using next-generation DNA sequencing data. Nature genetics, 43(5), pp.491-498. https://doi.org/10.1038/ng.806

Li, H., & Durbin, R. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. Bioinformatics, 25(14), 1754-1760. https://doi.org/10.1093/bioinformatics/btp324

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., ... & DePristo, M. A. 2010. The Genome Analysis Toolkit: a MapReduce framework for analyzing

next-generation DNA sequencing data. Genome Research, 20(9), 1297-1303. https://doi.org/10.1101/gr.107524.110
Cock, P. J., Fields, C. J., Goto, N., Heuer, M. L., & Rice, P. M. 2010. The Sanger FASTQ file format for sequences with quality scores, and the Solexa/Illumina FASTQ variants. Nucleic Acids Research, 38(6), 1767-1771. https://doi.org/10.1093/nar/gkp1137

Methylation Sequencing (Methyl-seq):

1. Quality Control of Raw Reads
First, we'll use FastQC to check the quality of our raw sequencing reads.
FastQC command
```
fastqc sample_methylation.fastq.gz –o fastqc_results
```
sample_methylation.fastq.gz: Input FASTQ file for methylation sequencing
-o fastqc_results: Output directory for FastQC results
This command runs FastQC on the input FASTQ file and generates quality reports in the specified output directory.

2. Trimming Adapters and Low-Quality Bases
Next, we'll use Trim Galore! to remove adapter sequences and low-quality bases from the reads.
Trim adapters and filter low-quality reads
```
trim_galore --fastqc sample_methylation.fastq.gz –o
trimmed_results
```
--fastqc: Run FastQC after trimming
sample_methylation.fastq.gz: Input FASTQ file
-o trimmed_results: Output directory for trimmed reads

3. Alignment to Reference Genome
We'll use Bismark to align the trimmed reads to a reference genome.
Align reads to reference genome using Bismark
```
bismark --genome /path/to/genome/ –o bismark_results
trimmed_results/sample_methylation_trimmed.fq.gz
```

--genome /path/to/genome/: Path to the reference genome directory
-o bismark_results: Output directory for Bismark results
trimmed_results/sample_methylation_trimmed.fq.gz: Input trimmed FASTQ file

4. Methylation Extraction

After alignment, we'll extract methylation information using Bismark's methylation extractor.

Extract methylation information

```
bismark_methylation_extractor --bedGraph --cytosine_report --
genome_folder /path/to/genome/
bismark_results/sample_methylation_trimmed_bismark_bt2.bam
```

--bedGraph: Output in bedGraph format

--cytosine_report: Generate a cytosine report

--genome_folder /path/to/genome/: Path to the reference genome directory

bismark_results/sample_methylation_trimmed_bismark_bt2.bam: Input BAM file from Bismark alignment

5. Visualization and Analysis in R

Finally, we'll use R to visualize and analyze the methylation data.

Load necessary libraries

library(methylKit)

Read methylation data

```
methylation_data <-
read.bismark("bismark_results/CpG_context_sample_methylation.txt
", sample.id="Sample1", assembly="hg19")
```

Filter and normalize data

```
filtered_data <- filterByCoverage(methylation_data, lo.count=10,
hi.perc=99.9)
normalized_data <- normalizeCoverage(filtered_data)
```

Plot methylation profile

```
plotMethylationProfile(normalized_data)
```

read.bismark: Read Bismark methylation data

filterByCoverage: Filter data by coverage

normalizeCoverage: Normalize coverage across samples

plotMethylationProfile: Plot methylation profile

References:

Krueger, F., & Andrews, S. R. 2011. Bismark: a flexible aligner and methylation caller for Bisulfite-Seq applications. Bioinformatics, 27(11), 1571-1572. https://doi.org/10.1093/bioinformatics/btr167

Akalin, A., Kormaksson, M., Li, S., Garrett-Bakelman, F. E., Figueroa, M. E., Melnick, A., & Mason, C. E. 2012. methylKit: a comprehensive R package for the analysis of genome-wide DNA methylation profiles. Genome Biology, 13(10), R87. https://doi.org/10.1186/gb-2012-13-10-r87

Andrews, S. 2010. FastQC: a quality control tool for high throughput sequence data. Available online at: https://www.bioinformatics.babraham.ac.uk/projects/fastqc/

Martin, M. 2011. Cutadapt removes adapter sequences from high-throughput sequencing reads. EMBnet.Journal, 17(1), 10-12. https://doi.org/10.14806/ej.17.1.200

Feng, H., Conneely, K. N., & Wu, H. 2014. A Bayesian hierarchical model to detect differentially methylated loci from single nucleotide resolution sequencing data. Nucleic Acids Research, 42(8), e69-e69. https://doi.org/10.1093/nar/gku154

Assenov, Y., Müller, F., Lutsik, P., Walter, J., Lengauer, T., & Bock, C. 2014. Comprehensive analysis of DNA methylation data with RnBeads. Nature Methods, 11(11), 1138-1140. https://doi.org/10.1038/nmeth.3115

Chromatin immunoprecipitation sequencing (ChIP-seq):

1. Quality Control of Raw Reads

First, we'll use FastQC to check the quality of our raw sequencing reads.

FastQC command

```
fastqc sample_chip.fastq.gz sample_input.fastq.gz -o
fastqc_results
```

sample_chip.fastq.gz sample_input.fastq.gz: Input FASTQ files (ChIP and input control)

-o fastqc_results: Output directory for FastQC results

This command runs FastQC on the input FASTQ files and generates quality reports in the specified output directory.

2. Trimming Adapters and Low-Quality Bases

Next, we'll use Trimmomatic to remove adapter sequences and low-quality bases from the reads.

Trim adapters and filter low-quality reads

```
java -jar trimmomatic-0.39.jar SE -threads 4 \
    sample_chip.fastq.gz sample_chip_trimmed.fastq.gz \
    ILLUMINACLIP:TruSeq3-SE.fa:2:30:10 LEADING:3 TRAILING:3
SLIDINGWINDOW:4:15 MINLEN:36
```

SE: Single-end mode

-threads 4: Use 4 CPU threads

sample_chip.fastq.gz: Input FASTQ file

sample_chip_trimmed.fastq.gz: Output file for trimmed reads

ILLUMINACLIP:TruSeq3-SE.fa:2:30:10: Adapter trimming parameters

LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36: Quality filtering parameters

3. Aligning Reads to the Reference Genome

We'll use Bowtie2 to align the trimmed reads to the reference genome.

Align reads to the reference genome

```
bowtie2 -x reference_genome -U sample_chip_trimmed.fastq.gz -S
sample_chip.sam
```
-x reference_genome: Reference genome index
-U sample_chip_trimmed.fastq.gz: Input trimmed reads
-S sample_chip.sam: Output SAM file

4. Converting SAM to BAM and Sorting
We'll use SAMtools to convert the SAM file to BAM format and sort it.
Convert SAM to BAM and sort
```
samtools view -bS sample_chip.sam | samtools sort -o
sample_chip_sorted.bam
```
view -bS: Convert SAM to BAM
sort -o sample_chip_sorted.bam: Sort BAM file and output

5. Post-alignment Processing
Tools: Picard
Mark duplicates
```
picard MarkDuplicates I= sample_chip_sorted.bam
O=dedup_sample_chip_sorted.bam M=metrics.txt
```
picard MarkDuplicates: Mark duplicate reads.
I=sorted.bam
O=dedup.bam: Input and output files.
M=metrics.txt: Metrics file for duplicates.

6. Peak Calling
We'll use MACS2 to call peaks from the aligned reads.
Call peaks with MACS2
```
macs2 callpeak -t dedup_sample_chip_sorted.bam -c
sample_input_sorted.bam -f BAM -g hs -n sample_chip_peaks -B --
outdir macs2_results
```
-t sample_chip_sorted.bam: ChIP sample BAM file
-c sample_input_sorted.bam: Input control BAM file
-f BAM: Format of input files
-g hs: Genome size (e.g., human)
-n sample_chip: Name for output files
-B: Generate bedGraph files
--outdir macs2_results: Output directory for MACS2 results

7. Motif Discovery
Motif Discovery using MEME-ChIP

Convert narrowPeak to BED format (if necessary)
```
awk '{print $1 "\t" $2 "\t" $3}'
macs2_results/sample_chip_peaks.narrowPeak >
sample_chip_peaks.bed
```
Extract sequences from peak regions
```
bedtools getfasta -fi /path/to/genome.fa -bed
sample_chip_peaks.bed -fo sample_chip_peaks.fasta
```
Run MEME-ChIP
```
meme-chip -oc meme_chip_results -db
/path/to/motif_databases/JASPAR2022_CORE_vertebrates_redundant_p
fms_meme.txt -meme-minw 6 -meme-maxw 30 -meme-nmotifs 10 -dreme-
e 0.05 -centrimo-score 5.0 -centrimo-ethresh 10.0
sample_chip_peaks.fasta
```

8. Peak Annotation

In R Studio, we'll use the ChIPseeker package to annotate the peaks.

Load necessary libraries
```
library(ChIPseeker)
library(TxDb.Hsapiens.UCSC.hg19.knownGene)
library(clusterProfiler)
```
Annotate peaks
```
peak <-
readPeakFile("macs2_results/sample_chip_peaks.narrowPeak")
txdb <- TxDb.Hsapiens.UCSC.hg19.knownGene
peakAnno <- annotatePeak(peak, tssRegion=c(-3000, 3000),
TxDb=txdb, annoDb="org.Hs.eg.db")
```
Visualize annotation
```
plotAnnoPie(peakAnno)
```
readPeakFile: Read the peak file generated by MACS2

TxDb.Hsapiens.UCSC.hg19.knownGene: Transcript database for human genome

annotatePeak: Annotate peaks with gene information

plotAnnoPie: Visualize peak annotation as a pie chart

References:

Andrews, S. 2010. FastQC: A quality control tool for high throughput sequence data. Available online at: http://www.bioinformatics.babraham.ac.uk/projects/fastqc

Bolger, A. M., Lohse, M., & Usadel, B. 2014. Trimmomatic: a flexible trimmer for Illumina sequence data. Bioinformatics, 30(15), 2114-2120. https://doi.org/10.1093/bioinformatics/btu170

Langmead, B., & Salzberg, S. L. 2012. Fast gapped-read alignment with Bowtie 2. Nature Methods, 9(4), 357-359. https://doi.org/10.1038/nmeth.1923

Li, H., Handsaker, B., Wysoker, A., Fennell, T., Ruan, J., Homer, N., ... & Durbin, R. 2009. The sequence alignment/map format and SAMtools. Bioinformatics, 25(16), 2078-2079. https://doi.org/10.1093/bioinformatics/btp352

Broad Institute. n.d. Picard Tools. http://broadinstitute.github.io/picard/

Zhang, Y., Liu, T., Meyer, C. A., Eeckhoute, J., Johnson, D. S., Bernstein, B. E., ... & Liu, X. S. 2008. Model-based analysis of ChIP-Seq (MACS). Genome Biology, 9(9), R137. https://doi.org/10.1186/gb-2008-9-9-r137

Machanick, P., & Bailey, T. L. 2011. MEME-ChIP: motif analysis of large DNA datasets. Bioinformatics, 27(12), 1696-1697. https://doi.org/10.1093/bioinformatics/btr189

Yu, G., Wang, L. G., & He, Q. Y. 2015. ChIPseeker: an R/Bioconductor package for ChIP peak annotation, comparison and visualization. Bioinformatics, 31(14), 2382-2383. https://doi.org/10.1093/bioinformatics/btv145

Whole metagenome sequencing (WMS):

1. Quality Control of Raw Reads

First, we'll use FastQC to check the quality of our raw sequencing reads.

FastQC command

```
fastqc sample_metagenome.fastq.gz -o fastqc_results
```

sample_metagenome.fastq.gz: Input FASTQ file for metagenome sequencing

-o fastqc_results: Output directory for FastQC results

This command runs FastQC on the input FASTQ file and generates quality reports in the specified output directory.

2. Trimming Adapters and Low-Quality Bases

Next, we'll use Trim Galore! to remove adapter sequences and low-quality bases from the reads.

Trim adapters and filter low-quality reads

```
trim_galore --fastqc sample_metagenome.fastq.gz -o trimmed_reads
```

--fastqc: Run FastQC after trimming

sample_metagenome.fastq.gz: Input FASTQ file

-o trimmed_reads: Output directory for trimmed reads

This command trims adapters and low-quality bases from the reads and outputs the trimmed reads to the specified directory.

3. Metagenomic Assembly

We'll use MEGAHIT for metagenomic assembly.

Metagenomic assembly with MEGAHIT

```
megahit -r trimmed_reads/sample_metagenome_trimmed.fq.gz -o
megahit_output
```
-r: Input file for single-end reads
trimmed_reads/sample_metagenome_trimmed.fq.gz: Trimmed reads file
-o megahit_output: Output directory for assembly results
This command assembles the metagenomic reads into contigs.

4. Taxonomic Profiling
We'll use MetaPhlAn for taxonomic profiling.
```
# Taxonomic profiling with MetaPhlAn
metaphlan megahit_output/final.contigs.fa --input_type fasta -o
metaphlan_output.txt
```
megahit_output/final.contigs.fa: Input contigs file
--input_type fasta: Specify input file type
-o metaphlan_output.txt: Output file for taxonomic profile
This command generates a taxonomic profile of the metagenome.

5. Functional Profiling
We'll use HUMAnN for functional profiling.
```
# Functional profiling with HUMAnN
humann --input megahit_output/final.contigs.fa --output
humann_output
```
--input: Input contigs file
megahit_output/final.contigs.fa: Input contigs file
--output humann_output: Output directory for functional profile
This command generates a functional profile of the metagenome.

6. Visualization in R Studio
```
library(phyloseq)
library(ggplot2)
library(DESeq2)
library(vegan)
```
```
# Read taxonomic data from MetaPhlAn output
tax_data <- read.table("metaphlan_output.txt", header=TRUE,
row.names=1, sep="\t")
```
```
# Create phyloseq object
OTU <- otu_table(as.matrix(tax_data), taxa_are_rows=TRUE)
TAX <-
tax_table(as.matrix(tax_data[,c("Kingdom","Phylum","Class","Orde
r","Family","Genus","Species")]))
```
```
# Create sample data (adjust this according to your experimental design)
```

```
sample_data <- data.frame(
    sample = colnames(OTU),
    group = c("A","A","B","B")  # Modify this to match your
experimental groups
)
rownames(sample_data) <- sample_data$sample
sampledata <- sample_data(sample_data)
# Create complete phyloseq object
physeq <- phyloseq(OTU, TAX, sampledata)
# Display alpha diversity
plot_richness(physeq, measures=c("Shannon", "Simpson"),
x="group") +
  theme_bw() +
  ggtitle("Alpha Diversity")
# Display taxonomic composition at genus level
plot_bar(physeq, fill="Genus") +
  theme_bw() +
  ggtitle("Taxonomic Composition at Genus Level")
# Beta diversity analysis
# Calculate beta distances
beta_div <- phyloseq::distance(physeq, method="bray")
# Perform PCoA
pcoa_result <- ordinate(physeq, method="PCoA",
distance=beta_div)
# Plot PCoA
plot_ordination(physeq, pcoa_result, color="group") +
  theme_bw() +
  ggtitle("PCoA of Bray-Curtis distances")
# Perform NMDS
nmds_result <- ordinate(physeq, method="NMDS",
distance=beta_div)
# Plot NMDS
plot_ordination(physeq, nmds_result, color="group") +
  theme_bw() +
  ggtitle("NMDS of Bray-Curtis distances")
# Perform PERMANOVA test
permanova_result <- adonis2(beta_div ~ group, data =
sample_data)
print(permanova_result)
# Differential abundance analysis using DESeq2
physeq_deseq <- phyloseq_to_deseq2(physeq, ~ group)
deseq_results <- DESeq(physeq_deseq)
```

```
res <- results(deseq_results)
# Display significantly different taxa
signif_taxa <- res[which(res$padj < 0.05), ]
print(signif_taxa)
# Plot volcano plot for differentially abundant taxa
plot(res$log2FoldChange, -log10(res$padj),
     col = ifelse(res$padj < 0.05, "red", "black"),
     xlab = "Log2 Fold Change", ylab = "-Log10 Adjusted P-
value",
     main = "Volcano Plot of Differentially Abundant Taxa")
```

References:

Andrews, S. 2010. FastQC: A quality control tool for high throughput sequence data. Available online at: http://www.bioinformatics.babraham.ac.uk/projects/fastqc

Krueger, F. 2015. Trim Galore! Available online at: https://www.bioinformatics.babraham.ac.uk/projects/trim_galore/

Li, D., Liu, C. M., Luo, R., Sadakane, K., & Lam, T. W. 2015. MEGAHIT: an ultra-fast single-node solution for large and complex metagenomics assembly via succinct de Bruijn graph. Bioinformatics, 31(10), 1674-1676. https://doi.org/10.1093/bioinformatics/btv033

Segata, N., Waldron, L., Ballarini, A., Narasimhan, V., Jousson, O., & Huttenhower, C. 2012. Metagenomic microbial community profiling using unique clade-specific marker genes. Nature Methods, 9(8), 811-814. https://doi.org/10.1038/nmeth.2066

Franzosa, E. A., McIver, L. J., Rahnavard, G., Thompson, L. R., Schirmer, M., Weingart, G., ... & Huttenhower, C. 2018. Species-level functional profiling of metagenomes and metatranscriptomes. Nature Methods, 15(11), 962-968. https://doi.org/10.1038/s41592-018-0176-y

McMurdie, P. J., & Holmes, S. 2013. phyloseq: An R package for reproducible interactive analysis and graphics of microbiome census data. PloS One, 8(4), e61217. https://doi.org/10.1371/journal.pone.0061217

Love, M. I., Huber, W., & Anders, S. 2014. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. Genome Biology, 15(12), 550. https://doi.org/10.1186/s13059-014-0550-8

Oksanen, J., Blanchet, F. G., Friendly, M., Kindt, R., Legendre, P., McGlinn, D., ... & Wagner, H. 2019. vegan: Community Ecology Package. R package version 2.5-6. https://CRAN.R-project.org/package=vegan