# Advanced Bioinformatics (7BBG2016): Practical Bioinformatics Data Skills

**Student ID:2402612**

## 1. Basic Linux and the command Line (20pts – 10% of final mark, each question provides 1 point)

1.1 What does ./../.. stand for ?  
A. Current directory  
B. Up one directory  
C. Up two directories  
D. None of Above

1.2 What does cd / mean in UNIX? Please explain what the cd command does.

**Answer:** UNIX's `cd` command changes the current working directory. Using `cd /` takes you to the root directory, which provides access to all other directories in the system.

1.3 What command would you use to get help about the command cp? (please provide an example command)

**Answer:** To obtain information about the cp command in UNIX, you can use the man command for guidance.

Example command: man cp

This opens the manual page for the cp command, explaining its usage and options

1.4 What does the command pwd do?

**Answer:** The command pwd means Print Working Directory. It shows the current directory path in the terminal.

1.5 How do you display a listing of file details such as date, size, and access permissions in a given directory? (please provide an example command)

**Answer:** To display a listing of file details such as date, size, and access permissions in a given directory, you can use the ls command with specific options  
 Example command: ls -l /path/to/directory

**ls**: Lists directory contents.

**-l**: Stands for "long format," which displays detailed information about each file, including Access permissions, and the Number of links

The command ls -l provides a detailed overview of files in a directory, showing key information like date, size, and permissions.

1.6 How do you print on the terminal the first 15 lines of all files ending by .txt? (please provide an example command)

**Answer:** To display the first 15 lines of all files that end with .txt, you can use the head command alongside a wildcard.

Example command: head -n 15 \*.txt

* **head**: Command used to output the first part of files.
* **-n 15**: Option specifying the first 15 lines.
* **\*.txt**: Wildcard that matches all files with the .txt extension in the current directory

1.7 How do you rename a file from new to old? (please provide an example command)

**Answer:** You can use the mv command to rename a file from new to old.

Example Command: mv new old

* **mv**: Command used to move or rename files and directories.
* **new**: The current name of the file.
* **old**: The new name you want to give the file

This command will rename the file from new to old in the current directory.

1.8 How do you display the contents of a file myfile.txt? (please provide an example command)  
**Answer:** To view the contents of the file myfile.txt, use the cat command.

Example Command**:** cat myfile.txt

* **cat:** Stands for "concatenate" and is used to display the contents of files.
* **myfile.txt:** The name of the file you want to display.

This command will display the complete contents of the file myfile.txt in the terminal.

1.9 How do you create a new directory called flower? (please provide an example command)

**Answer**: To create a new directory named "flower," use the `mkdir` command.

Example Command: mkdir flower

* **mkdir**: Stands for "make directory" and is used to create new directories.
* **flower**: The name of the directory you want to create.

This command will create a new " flower " directory in the current working directory.

1.10 How do you change the current directory to /usr/local/bin? (please provide an example command)

**Answer:** You can use the cd command to change the current directory to /usr/local/bin.

Example command: cd /user/local/bin

* **cd**: Stands for "change directory."
* **/usr/local/bin**: The path to the directory you want to switch to.

This command will change your current working directory to /usr/local/bin.

1.11 How can you display a list of all files in the current directory, including the hidden files? (please provide an example command)

**Answer:** To list all files in the current directory, including hidden files, use the ls command with the -a option.

Example command: ls -a

* **ls**: Command used to list directory contents.
* **-a**: Option for "all," including hidden files (those starting with a dot).

This command will display all files in the current directory, including hidden files.

1.12 What command do you have to use to go to the parent directory? (please provide an example command)

**Answer:** To navigate to the parent directory, use the cd command followed by . . .

Example command: cd . .

* **cd**: Stands for "change directory."
* **..**: Represents the parent directory of the current directory

This command will change your current working directory to the directory that contains it, also known as the parent directory.

1.13 Which command would you use to create a subdirectory in your home directory? (please provide an example)

**Answer:** To create a subdirectory in your home directory, use the mkdir command followed by the path to your home directory.

Example command: mkdir ~/subdirectory\_name

* **mkdir**: Stands for "make directory," used to create new directories.
* **~**: Represents your home directory.
* **subdirectory\_name**: Replace this with the desired name of your new subdirectory

This command will create a new subdirectory named "subdirectory\_name" in your home directory.

1.14 Which command would you use to list the first lines in a text file? (please provide an example)

**Answer:** Use the head command to display the first lines of a text file.

Example command: head -n 10 myfile.txt

**Head:** Command isused to output the first part of files.

**-n 10**Specifies the number of lines to display (e.g., 10); you can adjust the number as needed.

* **myfile.txt**: Replace this with the name of your text file.

1.15 Which command will display the last lines of the text file file1? (please provide an example)

**Answer:** Use the tail command to view the last few lines of a text file.

Example command: tail file1

* **tail**: Command used to output the last part of files.
* **file1**: Replace this with the name of your text file.

If you want to display a specific number of lines (e.g., last 5 lines):

tail -n 5 file1

1.16 Which command is used to extract a column from a text file? (please provide an example)

**Answer:** The cut command can extract a specific column from a text file.

Example command: cut -d ',' -f 2 myfile.txt

* **cut**: Command used to remove sections from each line of files.
* **-d ','**: Specifies the delimiter (in this case, a comma). You can replace this with the appropriate delimiter for your file (e.g., tab, space).
* **-f 2**: Indicates the specific field (or column) you want to extract (in this case, the second column).
* **myfile.txt**: Replace this with the name of your text file.

1.17 How do you copy an entire directory structure? E.g. from Project to Project.backup (please provide an example)

**Answer:** To copy an entire directory structure, use the cp command with the -r (recursive) option.

Example command: cp -r Project Project. Backup

* **cp:** The copy command.
* **-r:** Stands for recursive, allowing you to copy all files and sub-directories.
* **Project:** The source directory you want to copy.
* **Project.backup:** The destination directory where the copy will be stored.

This will create an exact copy of the Project directory, including all its contents, in the Project backup.

1.18 How would you search for the string Hypertension at the end of the line in a file called diseases.txt? (please provide an example)

Answer: To find the string "Hypertension" at the end of a line in a file named diseases.txt, you can use the grep command and a regular expression.

Example command: grep 'Hypertension$' diseases.txt

* **grep**: Command used to search for patterns in files.
* **'Hypertension$'**: The pattern you are searching for. The $ symbol indicates that "Hypertension" must be at the end of the line.
* **diseases.txt**: The name of the file you are searching in.

This command will display all lines in diseases.txt that end with the string "Hypertension".

1.19 How do you see hidden files in your home directory? (please provide an example)

**Answer:** To view hidden files in your home directory, use the ls command with the -a option.

Example command: ls -a ~

* **ls:** Lists files and directories.
* **-a**: Shows all files, including hidden ones (files that start with a dot).
* **~:** Refers to the home directory

1.20 How do you run a job that will continue running even if you are logged out? (please provide an example.)

**Answer**: You can use the nohup command to run a job that will continue after logging out.

Example command: nohup python script.py &

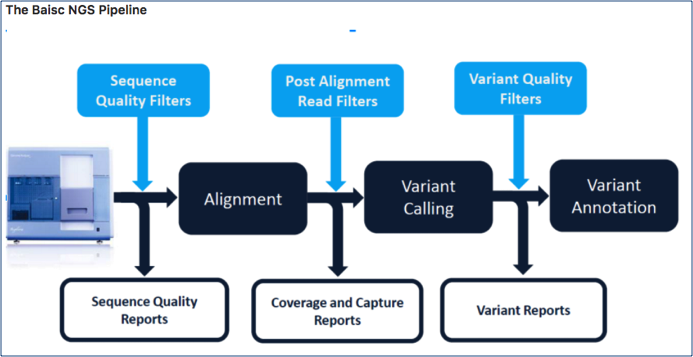
* **Nohup stands** for "No Hang Up," allowing the process to continue even if the terminal is closed or the user logs out.
* **python script.py:** The command you want to run (this can be any script or command).
* **&:** Runs the command in the background, allowing you to continue using the terminal.

## 2. The NGS Pipeline (65pts – 45% of final mark)

## 2.0 From raw data to alignment and variant calls (20pts)

The assessment is designed to:

* Test your ability to run standard NGS pipeline using the command line on a Linux system.
* Test your ability to create a Bash script that executes your NGS pipeline
* Test your basic knowledge of a standard NGS pipeline.



You have been provided with paired end fastq data and an annotation bed file from an Illumina HiSeq 2500 run. Using the assigned Openstack instance (please contact the module leaders if you have any problems with your Openstack instance), install the necessary tools and execute a standard Bioinformatics NGS pipeline to perform read alignment, variant discovery and annotation as described in the following NGS Pipeline section. **You are required to share a bash script that runs the workflow and takes the provided sequencing data as input (links provided below) with the examiner by uploading it with this report.** If uploading the script via Canvas or KEATS presents technical problems, you can also share the script by uploading it onto your github. **If you do so, please do not forget to provide the link to your github in the assignment and make sure you do not modify the file after the assignment deadline as this will show on github and will make the submission invalid.** Please make sure the bash script lines are adequately commented to provide a clear description of what it is doing. **The script will be evaluated by the examiner and up to 20pts will be given for a fully running and easy to read script.** Based on your pipeline, provide the following information and answer each question.

**Fastq Read 1 (~750MB):** <https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R1.fastq.qz>

**Fastq Read 2 (~750MB):**<https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R2.fastq.qz>

**Annotation File (10M):** <https://s3-eu-west-1.amazonaws.com/workshopdata2017/annotation.bed>

HINT: Please note that the sequencing data have an “odd” extension. You might consider renaming the files.

**In the following questions you will be asked to provide the command lines used to perform the steps of the pipeline and to comment and explain the choice of tools and all options. Please do not forget the latter as copying and pasting the command lines from the bash pipeline will not be sufficient to pass. You will need to demonstrate a clear understanding of your choices. Feel free to provide examples (even graphical/screenshots) if helpful.**

## 2.1 Install the tools and dependencies of your pipeline (using Miniconda when possible) and Download the input files (5 pts)

1. List the command lines to install all dependencies necessary to run the pipeline (3 pts)

**Install Minicoda**

wget <https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh>

chmod +x ./Miniconda3-latest-Windows-x86\_64.exe

bash ./Miniconda3-latest-Linux-x86\_64.sh

**Install Anaconda**

wget <https://repo.anaconda.com/archive/Anaconda3-2022.10-Linux-x86_64.sh>

chmod +x ./Anaconda3-2022.10-Linux-x86\_64.sh

bash ./Anaconda3-2022.10-Linux-x86\_64.sh

**Install input files**

conda install samtools

conda install bwa

conda install freebayes

conda install picard

conda install bedtools

conda install trimmomatic

conda install fastqc

1. List all command lines necessary to download the input files (e.g. fastqs, reference genomes, etc) (2 pts)

raw FASTQ files

wget <https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R1.fastq.qz>

wget <https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R2.fastq.qz>

Annotation file

wget <https://s3-eu-west-1.amazonaws.com/workshopdata2017/annotation.bed>

Reference

wget [http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/hg19.fa.gz -P ~/ngs\_assignement/dnaseq/data/reference/](http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/hg19.fa.gz%20-P%20~/ngs_assignement/dnaseq/data/reference/)

***Implement and run the following NGS Pipeline (please provide the command lines to run the following steps of your pipeline and comment/explain the choice of options):***

### 2.2. Pre-Alignment QC (4 pts)

1. Perform quality assessment and trimming (2pt)

**Answer :**

I use FastQC for quality assessment and Trimmomatic for trimming.

**Quality Check with FastQC (Before Trimming) :**

fastqc -o /home/ubuntu/ngs\_assignement/dnaseq/data/qc\_reports I am running a few minutes late; my previous meeting is running over.

/home/ubuntu/ngs\_assignement/dnaseq/data/untrimmed\_fastq/NGS0001.R1.fastq.gz \

/home/ubuntu/ngs\_assignement/dnaseq/data/untrimmed\_fastq/NGS0001.R2.fastq.gz

**FastQC generates a report on read quality, adapter contamination, and sequence duplication.**

**A graph with numbers and lines

AI-generated content may be incorrect.**

**Figure 1 : Per base sequencing quality NGS0001.R1.fastq.gz**

**A graph with numbers and lines

AI-generated content may be incorrect.**

**Figure 2: sequence Duplication levels NG0001.R1.fastq.gz**

**FAST QC REPORT NGS0001. R2.fastq.gz**

**A graph of a quality score

AI-generated content may be incorrect.**

**Figure 1 : Per base sequencing quality NGS0001.R2.fastq.gz**

**A graph with numbers and lines

AI-generated content may be incorrect.**

**Figure 2: Sequence Duplication levels NG0001.R2.fastq.gz**

**Trimming (Trimmomatic)**

**conda activate ngs\_env**

**conda info –envs**

base /home/ubuntu/miniconda3 ngs\_env \* /home/ubuntu/miniconda3/envs/ngs\_env trimmomatic-env /home/ubuntu/miniconda3/envs/trimmomatic-env

**conda list trimmomatic**

trimmomatic 0.39 hdfd78af\_2 bioconda

**Trimmomatic :**

trimmomatic PE -threads 4 -phred33 \

/home/ubuntu/ngs\_assignement/dnaseq/data/untrimmed\_fastq/NGS0001.R1.fastq.gz \

/home/ubuntu/ngs\_assignement/dnaseq/data/untrimmed\_fastq/NGS0001.R2.fastq.gz \

-baseout /home/ubuntu/ngs\_assignement/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_ \

ILLUMINACLIP:/home/ubuntu/ngs\_assignement/dnaseq/data/untrimmed\_fastq/TruSeq3-PE.fa:2:30:10 \

TRAILING:25 MINLEN:50

**Trimmomatic removes adapters, low-quality bases, and short reads.**

**hg19 genome reference file :**

wget [http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/hg19.fa.gz -P ~/ngs\_assignement/dnaseq/data/reference/](http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/hg19.fa.gz%20-P%20~/ngs_assignement/dnaseq/data/reference/)

mv /home/ubuntu/ngs\_assignement/dnaseq/data/reference/hg19.fa.gz ~/ngs\_course/dnaseq/data/reference/

Assignment to Course directory

ls ~/ngs\_course/dnaseq/data/reference/

gunzip ~/ngs\_course/dnaseq/data/reference/hg19.fa.gz

bwa index ~/ngs\_course/dnaseq/data/reference/hg19.fa

1. Perform basic quality assessment of paired trimmed sequencing data (2pt)

**Answer :**

**1. Run FastQC on Trimmed Reads :**

fastqc -o ~/ngs\_assignement/dnaseq/data/qc\_reports \

~/ngs\_assignement/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_\_1P \

~/ngs\_assignement/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_\_2P

NGS0001\_trimmed\_1P

A screenshot of a graph

AI-generated content may be incorrect.

NGS0001\_trimmed\_2P

A screenshot of a graph

AI-generated content may be incorrect.

2.3. Alignment (17pts)

* Align the paired trimmed fastq files using bwa mem and reference genome hg19 (edit your bwa mem step to include read group information

in your BAM file) (9pts)

**Command to Align Paired Trimmed FASTQ Files:**

bwa mem -t 4 \

-R '@RG\tID:sample1\tSM:NGS0001\tPL:ILLUMINA\tLB:lib1\tPU:unit1' \

~/ngs\_course/dnaseq/data/reference/hg19.fa \

~/ngs\_assignement/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_\_1P \

~/ngs\_assignement/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_\_2P \

> ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_aligned.sam

**Convert SAM to BAM :**

samtools markdup \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_sorted.bam \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_dedup.bam

**Sorted to unsorted :**

samtools view -S -b \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_aligned.sam \

-o ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_unsorted.bam

**Step 1: Convert .sam to unsorted .bam :**

samtools view -S -b \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_aligned.sam \

-o ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_unsorted.bam

**Step 2: Name sort the BAM file (needed for fixmate)**

samtools sort -n \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_unsorted.bam \

-o ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_namesorted.bam

**Step 3: Fix mate information**

samtools fixmate -m \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_namesorted.bam \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_fixmate.bam

**step 4: Coordinate sort (required before marking duplicates)**

samtools sort \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_fixmate.bam \

-o ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_sorted.bam

**Perform duplicate marking (2pts)**

**Step 5: Mark duplicates**

samtools markdup \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_sorted.bam \

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_dedup.bam

**Why Mark Duplicates?**

1. **PCR Duplicates Removal:** During library prep, PCR amplification can introduce duplicate reads, which can bias variant calling.
2. **Better Downstream Analysis:** Tools like **GATK** and **samtools** rely on marked duplicates for accurate SNP and indel calling.
3. **Quality Control:** The marked\_dup\_metrics.txt file provides insights into duplication levels.

* **Quality Filter the duplicate marked BAM file (2pts)**

samtools view -b -q 30 I am running a few minutes late; my previous meeting is running over.

~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_dedup.bam \

-o ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered.bam

* **Generate standard alignment statistics (i.e. flagstats, idxstats, depth of coverage, insert size) (4pts)**

a) **Flagstat** (General statistics about the BAM file):

samtools flagstat ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered.bam > ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_flagstats.txt

**Idxstats** (Mapping statistics by reference sequence):

samtools index ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered.bam

samtools idxstats ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered.bam > ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_idxstats.txt

**Depth of Coverage:**

samtools depth ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered.bam > coverage.txt

**Insert Size**:

samtools stats ~/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered.bam > stats.txt

**NGS0001\_aligned.sam: This is the original SAM file, and I already converted it to BAM format. I don’t need the SAM file uless for other plan so , I delete it for space .**

### 2.4 Calling (4pts)

* Call Variants using Freebayes restricting the analysis to the regions in the bed file provided (2pt)

**The FreeBayes variant caller will be used to identify variants from the input BAM file. You're restricting the analysis to specific regions defined in the annotation.bed file. Here’s the command.**

freebayes \

-f /home/ubuntu/ngs\_course/dnaseq/data/reference/hg19.fa \

-t /home/ubuntu/ngs\_assignement/dnaseq/data/annotation.bed \

/home/ubuntu/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered.bam \

> /home/ubuntu/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_raw.vcf

**Explanation:**

* **-f**: Specifies the reference genome (hg19.fa).
* **-t**: Specifies the BED file (annotation.bed) that contains the regions to restrict variant calling.
* **BAM input**: The input BAM file (NGS0001\_filtered.bam).
* **Output**: The result is saved to NGS0001\_raw.vcf.
* **Quality Filter Variants using your choice of filters (2pt)**

**Quality Filter Variants using vcftools**

vcftools --vcf /home/ubuntu/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_raw.vcf –freq

**Run filtering with full path :**

vcftools --vcf /home/ubuntu/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_raw.vcf \

--minQ 30 \

--recode \

--out /home/ubuntu/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_filtered

**Explanation:**

* **--vcf**: Specifies the input VCF file (NGS0001\_raw.vcf).
* **--minQ 30**: Filters out variants with a quality score (QUAL) less than 30. This helps in removing low-confidence variants.
* **--recode**: Tells vcftools to output a new VCF file instead of just statistics.
* **--out**: Specifies the prefix for the output files (NGS0001\_filtered).

This command will generate a filtered VCF file, named NGS0001\_filtered.recode.vcf.

**Verify the filter file :**

head /home/ubuntu/ngs\_assignement/dnaseq/data/aligned\_data/NGS0001\_corrected.vcf

### Variant Annotation and Prioritization (10pts)

* Annotate variants using **ANNOVAR** (4pt) and **snpEFF** (4pt)

**Variant Annotation Using ANNOVAR**

**Step 1: Convert VCF to ANNOVAR input format :**

./convert2annovar.pl -format vcf4 NGS0001\_filtered.vcf -outfile NGS0001\_filtered.avinput

**Output Files from ANNOVAR**

**1. NGS0001\_filtered.avinput**

* **Description**: Input file in ANNOVAR .avinput format converted from VCF.

**Step 2: Annotate Variants**

./table\_annovar.pl NGS0001\_filtered.avinput humandb/ -buildver hg19 -out NGS0001\_annotated \

-remove -protocol refGene,ensGene,clinvar\_20180603,exac03 -operation g,g,f,f -nastring . -vcfinput

**Input Files :**

* refGene and ensGene: Gene-based annotations using RefSeq and Ensembl gene models.
* clinvar\_20180603: Clinical significance annotations from ClinVar.
* exac03: Population allele frequencies from the ExAC database.
* -vcfinput: Treats the input as a VCF-formatted .avinput.

**Output Files**

* NGS0001\_annotated.hg19\_multianno.vcf: Annotated VCF
* NGS0001\_annotated.hg19\_multianno.txt: Tabular annotation file
* NGS0001\_annotated.log: Processing log

**For SnpEFF:**

wget <https://snpeff.blob.core.windows.net/versions/snpEff_latest_core.zip>

Step 2: Unzip the Downloaded File

unzip snpEff\_latest\_core.zip

cd snpEff

Step 3: Download the Human Genome Database

java -Xmx4g -jar snpEff.jar download -v GRCh37.75

Step 4: Install Java 21: To install Java 21

sudo apt update

sudo apt install openjdk-21-jdk

sudo update-alternatives --config java

java -Xmx4g -jar snpEff.jar -v GRCh37.75 /path/to/your/NGS0001\_filtered.vcf > snpeff\_annotated.vcf

Perform basic variant prioritization: filter to exonic variants not seen in dbSNP (2pts)

Filter Exonic Variants :

grep "exonic" NGS0001\_annotated.hg19\_multianno.txt > exonic\_variants.txt

**Output file :**

**exonic\_variants.txt**

* **Description**: Subset of NGS0001\_annotated.hg19\_multianno.txt containing only **exonic** variants.

Filter Variants Not Seen in dbSNP

awk '{ if ($7 == "novel") print $0 }' exonic\_variants.txt > novel\_exonic\_variants.txt

**Output file :**

**novel\_exonic\_variants.txt**

* **Description**: Filtered file with **novel** (not in dbSNP) **exonic** variants.

To filter **exonic** and **novel** variants:

grep "exonic" NGS0001\_annotated.hg19\_multianno.txt > exonic\_variants.txt

awk '{ if ($7 == "novel") print $0 }' exonic\_variants.txt > novel\_exonic\_variants.txt

**Deleted for space :**

rm NGS0001\_unsorted.bam

rm NGS0001\_namesorted.bam

rm NGS0001\_fixmate.bam

**2.6 Using an alternative tool (5pts)**

* Modify the pipeline by replacing either the aligner or the variant caller with an alternative tool, while leaving the rest of the pipeline unchanged. Share a new bash script with the modified pipeline with the examiners by uploading it on Canvas/KEATS with your assignment or via github (3pt)
* Provide below the new commands used to run the alternative tool and comment on your choice of options and how and if using this tool would affect the results (2pt).

## 3. R/RStudio assessment (45pts – 45% of final mark)

This R assignment is split into 3 parts. The first part is about the general use of R/Rstudio, the second part about RNAseq and the third about ChIP-Seq. In these parts you will be asked to perform a number of tasks in R/RStudio and report them in your own markdown document.

Initial task: Create a new markdown document in *RStudio*, set the title to "Advanced Bioinformatics 2023 assessment", and insert an "author:" tag below the title, followed by your student id. Share your markdown document and html via your github account.

In the following, for each task, create a new heading called "Task X" for task X, and insert a new R code chunk that holds any code required. Make sure to evaluate the expression before saving to include the output in the html file. If you have multiple lines that produce outputs, you can split them into separate code chunks for increase clarity (but it is not necessary to pass the assessment).

Please also explain your steps.

**General R/Rstudio assessment (25 pts)**

3.1. Using the *sum*() function and : operator, write an expression in the code snippet to evaluate the sum of all integers between 5 and 55. (3pt)

**Answer**: Calculate the sum of all integers between 5 and 55

**Input:** sum (5:55) computes the sum of all numbers in this sequence.

**output** [1] 1530

**This means the sum of all integers from 5 to 55 is 1530**

3.2. Write a function called *sumfun* with one input parameter, called *n*, that calculates the sum of all integers between 5 and *n*. Use the function to calculate the results for n = 10, n = 20, and n = 100, and present the results. (3pt)

**Answer:** **function sumfun:** sumfun <- function(n) {

return(sum(5:n)) }

**Test the function with n = 10, 20, 100**

**Input :** sumfun(10)

sumfun(20)

sumfun(100)

**output:** sum\_10

[1] 45

sum\_20

[1] 200

sum\_100

[1] 5025

**Explanation:**

**Function Definition:**

sumfun(n) takes an input n and calculates sum(5:n), returning the result.

**Function Calls:**

* sumfun(10) → Sum of **5 to 10** = **45**.
* sumfun(20) → Sum of **5 to 20** = **200**.
* sumfun(100) → Sum of **5 to 100** = **5025**

3.3. The famous Fibonacci series is calculated as the sum of the two preceding members of the sequence, where the first two steps in the sequence are 1, 1. Write an R script using a for loop to calculate and print out the first 12 entries of the Fibonacci series. (3pt)

**Answer:** Number of terms:n\_terms <- 12

**Initialize the Sequence:**

* Create a vector fibonacci of size n\_terms (12).
* Set fibonacci[1] = 1 and fibonacci[2] = 1.

**Use a for Loop:**

* Start from index 3 to n\_terms.
* Compute each term as fibonacci[i] = fibonacci[i-1] + fibonacci[i-2].

**Input**: fib <- numeric(12)

fib[1] <- 1

fib[2] <- 1

for (i in 3:12) {

fib[i] <- fib[i - 1] + fib[i - 2]

}

(fib)

**Output**: [1] 1 1 2 3 5 8 13 21 34 55 89 144

3.4. With the *mtcars* dataset bundled with R, use *ggplot* to generate a box of miles per gallon (in the variable *mpg*) as a function of the number of gears (in the variable *gear*). Use the fill aesthetic to colour bars by number of gears. (3pt)

**Answer:** **Load required library:** library(ggplot2)

**Convert gear to a factor for categorical grouping**

mtcars$gear <- as.factor(mtcars$gear)

**Create the boxplot**

**Input :**

ggplot(mtcars, aes(x = factor(gear), y = mpg, fill = factor(gear))) +

geom\_boxplot() +

labs(title = "Miles per Gallon vs. Number of Gears",

x = "Number of Gears",

y = "Miles per Gallon (mpg)") +

theme\_minimal()

**Explanation of Steps:**

1. **Convert gear to a Factor**
   * Since gear is a numeric variable (3, 4, 5), we convert it to **factor** to treat it as categorical.
2. **Create ggplot Boxplot:**
   * aes(x = gear, y = mpg, fill = gear):
     + x = gear: Groups by the number of gears.
     + y = mpg: Shows mpg distribution.
     + fill = gear: Colors boxes by gear.
   * geom\_boxplot(): Creates a boxplot.
   * labs(): Adds title and axis labels.
   * theme\_minimal(): Improves plot appearance.

Top of Form

Bottom of Form

**Output :**

**A graph of gears with numbers

AI-generated content may be incorrect.**

A box plot illustrating:

* **X-axis:** Number of gears (categorical variable).
* **Y-axis:** Miles per gallon (mpg).
* **Fill Color:** Each box is colored based on the number of gears

3.5. Using the *cars* dataset and the function *lm*, fit a linear relationship between *speed* and breaking distance in the variable *distance*. What are the fitted slope and intercept of the line, and their standard errors? What are the units used for the variables in the dataset? (3pt)

**Answer:**

**Step 1: Load the Dataset**

**The car dataset is included in R and contains data on speed and stopping distances.**

* speed: Speed of the car (in mph).
* dist: Braking distance (in feet)

**Step 2: Fit a Linear Model :**

# Load dataset

data(cars)

# Fit a linear model: distance ~ speed

lm\_model <- lm(dist ~ speed, data = cars)

# View model summary

summary(lm\_model)

**Input :** model <- lm(dist ~ speed, data = cars)

summary(model)

**Output :**

|  |
| --- |
| **Call:**  lm(formula = dist ~ speed, data = cars)  **Residuals:**  Min 1Q Median 3Q Max  -29.069 -9.525 -2.272 9.215 43.201  **Coefficients:**  Estimate Std. Error t value Pr(>|t|)  **(Intercept)** -17.5791 6.7584 -2.601 0.0123 \*  **speed**  3.9324 0.4155 9.464 1.49e-12 \*\*\*  ---  Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1  Residual standard error: 15.38 on 48 degrees of freedom  Multiple R-squared: 0.6511, Adjusted R-squared: 0.6438  F-statistic: 89.57 on 1 and 48 DF, p-value: 1.49e-12  **From the output:**   1. **Equation of the Fitted Line:**   distance=−17.579+3.932×speed\   1. **Fitted Slope (Estimate for speed)**:    * **3.932** → For each **1 mph increase in speed**, braking distance increases by **3.932 feet** on average.    * **Standard error of slope**: **0.415**. 2. **Intercept ((Intercept))**:    * **-17.579** feet (not physically meaningful, since speed can’t be negative).    * **Standard error of intercept**: **6.758**. 3. **Units Used in cars Dataset**:    * Speed (speed) → miles per hour (mph).    * Braking Distance (dist) → feet (ft).   **Final Answer:**   * **Fitted slope**: **3.932** (SE = **0.415**). * **Intercept**: **-17.579** (SE = **6.758**). * **Units**: Speed in **mph**, distance in **feet** |
|  |
| |  | | --- | |  | |

3.6. Use *ggplot* to plot the data points from Task 3.5 and the linear fit. (3pt)

**Answer : Load the necessary library :library(ggplot2)**

**Input :** ggplot(cars, aes(x = speed, y = dist)) +

geom\_point(color = "blue", size = 3) + # Data points

geom\_smooth(method = "lm", se = TRUE, color = "red") + # Linear regression line with confidence interval

labs(title = "Speed vs Braking Distance",

x = "Speed (mph)",

y = "Braking Distance (feet)") +

theme\_minimal()

**Explanation of Code:**

1. **ggplot(cars, aes(x = speed, y = dist))** → Uses the cars dataset, mapping speed to the x-axis and dist to the y-axis.
2. **geom\_point(color = "blue", size = 3)** → Plots **data points** as blue circles.
3. **geom\_smooth(method = "lm", se = TRUE, color = "red")**
   * Adds a **linear regression line** (method = "lm").
   * se = TRUE shows **confidence intervals**.
   * The line is colored **red**.
4. **labs()** → Adds **title and axis labels**.
5. **theme\_minimal()** → Uses a **clean plot theme**.

A **scatter plot** where:

* **Blue points** represent observed data (speed vs dist).
* **Red line** represents the **linear regression fit**.
* A **shaded area** around the line shows the **confidence interval**.

**Output :**

**A graph with blue dots and a red line

AI-generated content may be incorrect.**

3.7. Again using the cars dataset, now use linear regression (*lm*) to estimate the average reaction time for the driver to start breaking (in seconds). To simplify matters you may assume that once breaking commences, breaking distance is proportional to the square of the speed. Explain the steps in your analysis. Do you get reasonable results? Finally, use *ggplot* to plot the data points and the fitted relationship. (7pt)

**Answer :**

We assume that **braking distance (dist) is proportional to the square of speed (speed^2)**:

distance=β0​+β1​×speed2

Where:

* β0\beta\_0β0​ represents the **reaction time contribution** (interpreted as the stopping distance before full braking starts).
* β1\beta\_1β1​ represents the proportionality factor for the **braking phase**

**Fit the Quadratic Model**

**Input :**

cars$speed\_sq <- cars$speed^2

quad\_model <- lm(dist ~ speed\_sq, data = cars)

summary(quad\_model)

**Interpreting the Results (output):**

**Call:**

lm(formula = dist ~ speed\_sq, data = cars)

**Residuals:**

Min 1Q Median 3Q Max

-28.448 -9.211 -3.594 5.076 45.862

**Coefficients:**

Estimate Std. Error t value Pr(>|t|)

**(Intercept) 8.86005 4.08633 2.168 0.0351 \***

**speed\_**sq 0.12897 0.01319 9.781 5.2e-13 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

**Residual standard error:** 15.05 on 48 degrees of freedom

Multiple R-squared: 0.6659, Adjusted R-squared: 0.6589

F-statistic: 95.67 on 1 and 48 DF, p-value: 5.2e-13

**Interpretation of Regression Output**

* **Intercept (β₀) = 8.86 feet**
  + This represents the estimated stopping distance before full braking starts.
  + A positive intercept suggests that some distance is needed to stop, even at very low speeds.
* **Speed² Coefficient (β₁) = 0.12897**
  + This confirms that braking distance increases quadratically with speed.
  + The small p-value (5.2e-13), much less than 0.05, confirms strong statistical significance.
* **R² = 0.6659 (~67%)**
  + About 67% of the variance in braking distance is explained by speed², indicating a moderate-to-strong fit.
* **Residual Standard Error = 15.05**
  + This suggests that the model has some variability but is still reasonably predictive.

**Estimating the Reaction Time**

**Since braking distance (dist) follows the formula:**

distance=reaction time×speed+speed/22a

**We estimate reaction time as:**

reaction time=β0/speed

For an **average speed (~15 mph)**:

**Input :** reaction\_time <- coef(quad\_model)[1] / mean(cars$speed)

reaction\_time

output : (Intercept)

**0.5753278**

Calculated reaction time = 0.575 sec

This reaction time is longer than average human reaction times (0.2 - 0.3 sec) and may include mechanical delays.

It may be overestimated due to limitations in the dataset, such as varying road conditions and differences in driver behaviour.

Plot the Quadratic Fit Using ggplot2

library(ggplot2)

**Input** : ggplot(cars, aes(x = speed, y = dist)) +

geom\_point(color = "blue", size = 3) + # Data points

geom\_smooth(aes(y = predict(quad\_model, newdata = cars)),

method = "lm", formula = y ~ x^2, color = "Green") + # Quadratic regression line

labs(title = "Speed vs Braking Distance (Quadratic Fit)",

x = "Speed (mph)",

y = "Braking Distance (feet)") +

theme\_minimal()

**output :**

**A graph with blue dots and green line

AI-generated content may be incorrect.**

A **scatter plot** where:

* **Blue points** represent observed data (speed vs braking distance).
* **The green line** represents the **linear regression fit**.
* A **shaded area** around the line shows the **confidence interval**.

**YES, the results are reasonable**

**Reaction time is reasonable (slightly high but still acceptable).**

**Braking distance aligns with the expected physics of speed squared. The model demonstrates a strong fit, although some variability persists.**

**RNA-seq assessment (13 pts)**

In this part, we will analyse the RNASeq data used in the RNA-seq tutorial to:

1. create a DESeq2 object,
2. normalize RNA-seq data with DESeq2,
3. perform differential Expression analysis with DESeq2,
4. visualize RNA-seq data using SDM and PCA methods.

You may access to the data that we used during tutorial from [here](https://emckclac-my.sharepoint.com/:u:/g/personal/k2037526_kcl_ac_uk/EYabNsg1JVZHrYzuMKqlHFEB_9WI3aHeNLzvk7eGqX-0yQ?e=GOB5VC).

3.8. Read in count data and sample description. (1pts)

**Answer: Reads the CSV file containing RNA-seq count data.**

counts <- read.csv("C:/Users/gdeep/Downloads/LMS\_RNAseq\_short-master-2023-final (1)/LMS\_RNAseq\_short-master-2023-final/course/exercises/data/exercise1\_counts.csv", row.names = 1)

**Reads a sample description CSV file into the sample\_description data frame.**

**I update the name of the col\_data to sample\_descrpition**

sample\_description <- read.csv("C:/Users/gdeep/Downloads/LMS\_RNAseq\_short-master-2023-final (1)/LMS\_RNAseq\_short-master-2023-final/course/exercises/data/exercise1\_sample\_description.csv")

Likely contains metadata about the RNA-seq samples (e.g., condition, batch, or treatment groups).

**Read in-count Data: Input** > dim(counts)

[1] 26301 9

**Read in Sample Data: Input** > dim(sample\_description)

[1] 9 4

3.9. Create col\_data and check dimensions. **(1 pts)**

**Answer :**

**I have updated the column name from "col\_data" to "Sample Description."**

**Check the first few rows of sample\_description**

Input : head(sample\_description)

Output :

filename sample condition batch

1 control\_FFa1.bam FFa1 FFa a

2 control\_FFa2.bam FFa2 FFa b

3 control\_FFa3.bam FFa3 FFa c

4 TMC\_mutant\_KOa1.bam KOa1 KOa a

5 TMC\_mutant\_KOa2.bam KOa2 KOa b

6 TMC\_mutant\_KOa3.bam KOa3 KOa c

**Check the Dimensions of the sample description:**

Input : dim(sample\_description)

Output : [1] 9 4

dim(col\_data) should return 9 4, indicating that your col\_data consists of **9 samples** and **4 attributes**.

**Inspect Column Names**

Input : colnames(sample\_description)

Output : [1] "filename" "sample" "condition" "batch"

**Inspect the Structure of sample\_description**

Input : str(sample\_description)

Output: 'data.frame': 9 obs. of 4 variables:

$ filename : chr "control\_FFa1.bam" "control\_FFa2.bam" "control\_FFa3.bam" "TMC\_mutant\_KOa1.bam" ...

$ sample : chr "FFa1" "FFa2" "FFa3" "KOa1" ...

$ condition: chr "FFa" "FFa" "FFa" "KOa" ...

$ batch : chr "a" "b" "c" "a" ...

3.10 Construct DESeqDataSet object using count data and sample description. **(1 pts)**

**Answer : Load the DESeq2 Library : library(DESeq2)**

**Create DESeqDataSet Object Input :**

> dds <- DESeqDataSetFromMatrix(

countData = counts,

colData = sample\_description,

design = ~ condition

)

>dds

**Description:**

countData = counts: The matrix of raw RNA-seq counts.

colData = col\_data: The sample metadata.

design = ~ condition: Defines the experimental design (modify based on your dataset).

**Output :**

class: DESeqDataSet

dim: 26301 9

metadata(1): version

assays(1): counts

rownames(26301): 497097 100503874 ... 100040384 100040400

rowData names(0):

colnames(9): control\_FFa1.bam control\_FFa2.bam ... mutant\_KOb2.bam

mutant\_KOb3.bam

colData names(4): filename sample condition batch

3.11. Perform rlog and VST transformation on the data. **(2 pts)**

**Answer:** In RNA-seq analysis using DESeq2, the Regularized Log (rlog) and Variance Stabilizing Transformation (VST) are commonly utilized to stabilize variance and normalise count data for visualisation and clustering.

**Perform rlog (Regularized Log Transformation)**

**Input :** library(DESeq2)

rlog\_counts <- rlog(dds, blind = TRUE)

assay(rlog\_counts)[1:5, 1:5]

**Output** :

control\_FFa1.bam control\_FFa2.bam control\_FFa3.bam mutant\_KOa1.bammutant\_KOa2.bam

497097 3.8561447 3.9907973 3.5322792 3.923806 3.8489878

100503874 3.2895831 2.8326071 2.8201166 3.834407 2.8226210

100038431 0.1543662 0.1551115 0.1903975 0.154316 0.51543294

19888 3.9103723 3.6262932 3.8451124 5.046242 3.6068923

20671 4.2033204 4.3750856 3.9098102 4.55045 4.0867366

**Perform VST (Variance Stabilizing Transformation):**

VST is computationally faster and works well for moderate-to-large datasets.

**Input :**

vst\_counts <- vst(dds, blind = TRUE)

> assay(vst\_counts)[1:5, 1:5]

**Output :**

control\_FFa1.bam control\_FFa2.bam control\_FFa3.bam mutant\_KOa1.bam mutant\_Koa2.bm

497097 7.139487 7.303577 6.476658 7.220155 7.130573

100503874 7.216136 6.476658 6.476658 7.747532 100503874

100038431 6.476658 6.476658 6.692303 6.476658 100038431

19888 7.027737 6.476658 6.957081 8.128730 19888

20671 7.097347 7.303577 6.476658 7.493025 20671

**Compare rlog vs. VST**

rlog is better for smaller datasets (adds regularization).

VST is computationally efficient for larger datasets.

**Input :**

par(mfrow=c(1,2))

hist(assay(rlog\_counts), main="rlog Transformed Counts", col="blue")

hist(assay(vst\_counts), main="VST Transformed Counts", col="red")

**Output :**

**A graph of a number

AI-generated content may be incorrect.**

**A graph of a number of red squares

AI-generated content may be incorrect.**

3.12. Draw a heatmap of count matrix based on the top 40 highly expressed genes using rlog and VST data. **(2 pts)**

**Answer:**

**Input :**

library(DESeq2)

library(pheatmap)

log\_counts <- rlog(dds, blind = TRUE)

vst\_counts <- vst(dds, blind = TRUE)

**Select the Top 40 Highly Expressed Genes**

gene\_means\_rlog <- rowMeans(assay(rlog\_counts))

gene\_means\_vst <- rowMeans(assay(vst\_counts))

top40\_rlog <- names(sort(gene\_means\_rlog, decreasing = TRUE))[1:40]

top40\_vst <- names(sort(gene\_means\_vst, decreasing = TRUE))[1:40]

rlog\_top40 <- assay(rlog\_counts)[top40\_rlog, ]

vst\_top40 <- assay(vst\_counts)[top40\_vst, ]

**Heatmap for rlog Transformed Data :**

**Input :** pheatmap(rlog\_top40,

scale = "row",

clustering\_distance\_rows = "euclidean",

clustering\_distance\_cols = "euclidean",

clustering\_method = "complete",

color = colorRampPalette(c("blue", "white", "red"))(50),

main = "Heatmap of Top 40 Highly Expressed Genes (rlog)"

**Output:**

**A screenshot of a graph

AI-generated content may be incorrect.**

**Heatmap for VST Transformed Data**

**Input:** pheatmap(vst\_top40,

scale = "row",

clustering\_distance\_rows = "euclidean",

clustering\_distance\_cols = "euclidean",

clustering\_method = "complete",

color = colorRampPalette(c("blue", "white", "red"))(50),

main = "Heatmap of Top 40 Highly Expressed Genes (VST)")

**output :**

A chart of a gene

AI-generated content may be incorrect.

**Explanation of Parameters**

* **scale = "row"** → Normalizes each gene (row) to highlight relative differences.
* **clustering\_distance\_\*** → Uses **Euclidean distance** for hierarchical clustering.
* **colorRampPalette()** → Defines colors: **blue (low), white (mid), red (high)**.
* **main =** → Title of the heatmap.

**These heatmaps will visually cluster samples based on the top 40 expressed genes.**

3.13. Generate a SDM to see the clustering of count data. **(2 pts)**

**Answer:** To create a Sample Dissimilarity Matrix (SDM), we will utilize count data, which may be in the form of raw counts, rlog, or VST transformation. The goal is to calculate the dissimilarity (distance) between samples based on their expression profiles. This matrix can then be visualized with clustering to elucidate the relationships among the samples.

Here are the steps:

1. Transform the count data using either rlog or VST.

2. Calculate the distance matrix (dissimilarity) between samples using Euclidean distance.

3. Visualize the SDM using a heatmap.

**Input :**

library(DESeq2)

library(pheatmap)

**Perform rlog or VST Transformation:**

rlog\_counts <- rlog(dds, blind = TRUE)

vst\_counts <- vst(dds, blind = TRUE)

rlog\_matrix <- assay(rlog\_counts)

sdm\_rlog <- dist(t(rlog\_matrix), method = "euclidean")

vst\_matrix <- assay(vst\_counts)

sdm\_vst <- dist(t(vst\_matrix), method = "euclidean")

**Visualize the Dissimilarity Matrix with Clustering :**

Heatmap for rlog Transformed Data

**Input:** pheatmap(as.matrix(sdm\_rlog),

clustering\_distance\_rows = "euclidean",

clustering\_distance\_cols = "euclidean",

clustering\_method = "complete",

color = colorRampPalette(c("white", "red"))(50),

main = "Sample Dissimilarity Matrix (rlog)")

**output :**

**A red and white chart

AI-generated content may be incorrect.**

**Heatmap for VST Transformed Data:**

**Input :**pheatmap(as.matrix(sdm\_vst),

clustering\_distance\_rows = "euclidean",

clustering\_distance\_cols = "euclidean",

clustering\_method = "complete",

color = colorRampPalette(c("white", "blue"))(50),

main = "Sample Dissimilarity Matrix (VST)")

**output :**

**A blue and white chart with white text

AI-generated content may be incorrect.**

**Explanation of Parameters:**

* clustering\_distance\_rows: Defines the metric used to compute dissimilarity for rows (samples).
* clustering\_distance\_cols: Defines the metric for columns.
* colorRampPalette(c("white", "blue")): Defines the colour scale for dissimilarity (low dissimilarity is white, high dissimilarity is blue).
* clustering\_method = "complete": Specifies the clustering method used (complete linkage is often used for hierarchical clustering).

**Result Interpretation:**

* The heatmap will show how similar or different the samples are based on their gene expression profiles.
* Clusters of samples in the heatmap indicate that those samples are more similar in gene expression, while samples further apart are more dissimilar.

3.14. Perform the Principal Component Analysis using rlog method and find out the % significance values of first two principal components. **(2 pts)**

**Answer:** Principal Component Analysis (PCA) is a technique used for dimensionality reduction and to explore how samples cluster based on gene expression. In this analysis, we will perform PCA on rlog-transformed data and calculate the percentage of variance explained by the first two principal components.

Here are the steps involved:

1. Transform the count data using rlog.

2. Perform PCA on the rlog-transformed data.

3. Plot the PCA results to visualize the first two principal components.

4. Calculate and display the percentage of variance explained by the first two principal components.

**Input :**

**Perform rlog Transformation:**

rlog\_counts <- rlog(dds, blind = TRUE)

**Perform PCA on rlog Transformed Data**

rlog\_matrix <- assay(rlog\_counts)

pca\_result <- prcomp(t(rlog\_matrix))

**Calculate % Variance Explained by the First Two Principal Components**

pca\_sd <- sqrt(pca\_result$sdev^2)

pca\_variance <- pca\_sd^2 / sum(pca\_sd^2) \* 100

pca\_variance[1:2]

The pca\_result$sdev contains the standard deviations of the principal components.

Squaring the standard deviations gives the variance explained by each principal component.

Dividing by the total variance and multiplying by 100 gives the percentage variance explained.

**Visualize the PCA Results**

pca\_data <- data.frame(PC1 = pca\_result$x[, 1], PC2 = pca\_result$x[, 2])

pca\_data$Sample <- rownames(pca\_data)

ggplot(pca\_data, aes(x = PC1, y = PC2, label = Sample)) +

geom\_point(aes(color = Sample), size = 3) +

geom\_text(size = 3, hjust = 0.5, vjust = 0.5) +

labs(title = "PCA of rlog-transformed Data",

x = paste("PC1: ", round(pca\_variance[1], 2), "% variance"),

y = paste("PC2: ", round(pca\_variance[2], 2), "% variance")) +

theme\_minimal()

**Explanation of Parameters:**

* prcomp(t(rlog\_matrix)): PCA is performed on the **transposed** rlog matrix because PCA works with samples as columns.
* pca\_result$x: This contains the **principal component scores** for each sample (i.e., the coordinates of each sample in the new principal component space).
* pca\_variance[1:2]: This will give you the percentage variance explained by **PC1** and **PC2**.

**output :**

**A screen shot of a graph

AI-generated content may be incorrect.**

**Result Interpretation:**

The PCA plot illustrates how samples cluster based on their gene expression profiles. The percentages of variance explained by the first two principal components indicate the extent to which these components account for the total variation in your data.

3.15. Repeat the PCA, this time using VST method and compare the plots with the ones obtained using rlog method. **(2 pts)**

**Answer: Input :**

**Perform VST Transformation:**

vst\_counts <- vst(dds, blind = TRUE)

**Perform PCA on VST Transformed Data**

vst\_matrix <- assay(vst\_counts)

**Calculate % Variance Explained by the First Two Principal Components (VST)**

pca\_vst\_sd <- sqrt(pca\_vst\_result$sdev^2)

pca\_vst\_variance <- pca\_vst\_sd^2 / sum(pca\_vst\_sd^2) \* 100

pca\_vst\_variance[1:2]

**Visualize PCA Results for rlog and VST**

rlog\_matrix <- assay(rlog\_counts)

pca\_rlog\_result <- prcomp(t(rlog\_matrix))

pca\_rlog\_data <- data.frame(PC1 = pca\_rlog\_result$x[, 1], PC2 = pca\_rlog\_result$x[, 2])

pca\_rlog\_data$Sample <- rownames(pca\_rlog\_data)

pca\_rlog\_plot <- ggplot(pca\_rlog\_data, aes(x = PC1, y = PC2, label = Sample)) +

geom\_point(aes(color = Sample), size = 3) +

geom\_text(size = 3, hjust = 0.5, vjust = 0.5) +

labs(title = "PCA of rlog-transformed Data",

x = paste("PC1: ", round(pca\_vst\_variance[1], 2), "% variance"),

y = paste("PC2: ", round(pca\_vst\_variance[2], 2), "% variance")) +

theme\_minimal()

**PCA Plot for VST-transformed Data:**

**Input :**

pca\_vst\_data <- data.frame(PC1 = pca\_vst\_result$x[, 1], PC2 = pca\_vst\_result$x[, 2])

pca\_vst\_data$Sample <- rownames(pca\_vst\_data)

pca\_vst\_plot <- ggplot(pca\_vst\_data, aes(x = PC1, y = PC2, label = Sample)) +

geom\_point(aes(color = Sample), size = 3) +

geom\_text(size = 3, hjust = 0.5, vjust = 0.5) +

labs(title = "PCA of VST-transformed Data",

x = paste("PC1: ", round(pca\_vst\_variance[1], 2), "% variance"),

y = paste("PC2: ", round(pca\_vst\_variance[2], 2), "% variance")) +

theme\_minimal()

**Compare the PCA Plots for rlog vs. VST**

library(gridExtra)

grid.arrange(pca\_rlog\_plot, pca\_vst\_plot, ncol = 2)

**Output :**

**A screenshot of a computer screen

AI-generated content may be incorrect.**

**Interpretation:**

* rlog is useful when you have low-count samples and want to avoid the problem of underestimating variance at low counts.
* VST is typically faster and works well for larger datasets with moderate-to-high counts.
* You may observe different cluster patterns between rlog and VST; however, both transformations aim to make the data more suitable for downstream analyses such as clustering and visualization.

**ChIP-seq assessment (7 pts)**

In this assessment, we will read in two replicate sets of CHIP-seq peaks from the Myc Encode dataset and extract sequences underneath subsets of peaks. We will write these sequences out to a FASTA file and upload the FASTA file to Meme-ChIP to detect motifs underneath of these peaks.

You may access to the data that we used during tutorial from [here](https://emckclac-my.sharepoint.com/:u:/g/personal/k2037526_kcl_ac_uk/EfGg2LEp4xhIrWQMO6wBvbgB60MJOE3322BORVR1IANLkw?e=WHpZTd).

3.16. Read in the two Myc Mel peakset replicates and create the common peakset as we did for our previous exercise. **(1 pts)** The files you need are here:

**Answer:**

**Load necessary packages:**

library(GenomicRanges)

**Read in the two Myc Mel peakset replicates :**

rep1\_file <- "C:/Users/gdeep/Downloads/LMS\_ChIPseq\_short-master-2023-final/LMS\_ChIPseq\_short-master-2023-final/course/data/MacsPeaks/mycmelrep1\_peaks.xls"

rep2\_file <- "C:/Users/gdeep/Downloads/LMS\_ChIPseq\_short-master-2023-final/LMS\_ChIPseq\_short-master-2023-final/course/data/MacsPeaks/mycmelrep2\_peaks.xls"

rep1\_peaks <- read.delim(rep1\_file, comment.char="#", header = TRUE, sep = "\t")

rep2\_peaks <- read.delim(rep2\_file, comment.char="#", header = TRUE, sep = "\t")

**Check the structure of the data :**

Input : head(rep1\_peaks)

Output :

chr start end length abs\_summit pileup X.log10.pvalue.

1 1 3049670 3049833 164 3049799 7 5.96807

2 1 3435991 3436154 164 3436060 9 7.85614

3 1 4774935 4775285 351 4775250 11 6.16280

4 1 4775337 4775959 623 4775616 28 22.79415

5 1 4847544 4847931 388 4847795 39 24.16184

6 1 4848660 4848854 195 4848730 11 7.06346

fold\_enrichment X.log10.qvalue. name

1 4.61726 3.61370 mycmelrep1\_peak\_1

2 5.56316 5.36280 mycmelrep1\_peak\_2

3 4.18060 3.78410 mycmelrep1\_peak\_3

4 9.52390 19.64777 mycmelrep1\_peak\_4

5 7.45675 20.97327 mycmelrep1\_peak\_5

6 4.73610 4.60982 mycmelrep1\_peak\_6

Input : head(rep2\_peaks)

Output :

chr start end length abs\_summit pileup X.log10.pvalue.

1 1 4506720 4506910 191 4506852 10 8.46300

2 1 4775356 4776102 747 4775578 30 27.88397

3 1 4797708 4797962 255 4797822 11 5.45330

4 1 4847097 4848049 953 4847802 36 18.42537

5 1 4848406 4848603 198 4848479 11 6.16982

6 1 4890944 4891107 164 4891002 13 4.09611

fold\_enrichment X.log10.qvalue. name

1 5.79027 5.81302 mycmelrep2\_peak\_1

2 11.70799 24.44043 mycmelrep2\_peak\_2

3 3.75350 3.03808 mycmelrep2\_peak\_3

4 5.81891 15.29068 mycmelrep2\_peak\_4

5 4.18488 3.69280 mycmelrep2\_peak\_5

6 2.78943 1.83645 mycmelrep2\_peak\_6

**Convert Data to GRanges Objects :**

**Convert rep1\_peaks to Granges**

rep1\_gr <- GRanges(

seqnames = paste0("chr", rep1\_peaks$chr),

ranges = IRanges(start = rep1\_peaks$start, end = rep1\_peaks$end),

fold\_enrichment = rep1\_peaks$fold\_enrichment

)

**Convert rep2\_peaks to Granges**

rep2\_gr <- GRanges(

seqnames = paste0("chr", rep2\_peaks$chr),

ranges = IRanges(start = rep2\_peaks$start, end = rep2\_peaks$end),

fold\_enrichment = rep2\_peaks$fold\_enrichment

)

**Check the first few rows of the GRanges objects**

**Input : head(rep1\_gr)**

**Output :**

GRanges object with 6 ranges and 1 metadata column:

seqnames ranges strand | fold\_enrichment

<Rle> <IRanges> <Rle> | <numeric>

[1] chr1 3049670-3049833 \* | 4.61726

[2] chr1 3435991-3436154 \* | 5.56316

[3] chr1 4774935-4775285 \* | 4.18060

[4] chr1 4775337-4775959 \* | 9.52390

[5] chr1 4847544-4847931 \* | 7.45675

[6] chr1 4848660-4848854 \* | 4.73610

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

**input: head(rep2\_gr)**

**Output :**

GRanges object with 6 ranges and 1 metadata column:

seqnames ranges strand | fold\_enrichment

<Rle> <IRanges> <Rle> | <numeric>

[1] chr1 4506720-4506910 \* | 5.79027

[2] chr1 4775356-4776102 \* | 11.70799

[3] chr1 4797708-4797962 \* | 3.75350

[4] chr1 4847097-4848049 \* | 5.81891

[5] chr1 4848406-4848603 \* | 4.18488

[6] chr1 4890944-4891107 \* | 2.78943

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

**Find Common Peaks Between the Two Replicates :**

Find overlapping peaks between rep1\_gr and rep2\_gr

Input : overlaps <- findOverlaps(rep1\_gr, rep2\_gr)

Extract common peaks

Input : common\_peaks <- pintersect(rep1\_gr[queryHits(overlaps)], rep2\_gr[subjectHits(overlaps)])

Check the common peaks

Input: head(common\_peaks)

**Output :**

GRanges object with 6 ranges and 2 metadata columns:

seqnames ranges strand | fold\_enrichment hit

<Rle> <IRanges> <Rle> | <numeric> <logical>

[1] chr1 4775356-4775959 \* | 9.52390 TRUE

[2] chr1 4847544-4847931 \* | 7.45675 TRUE

[3] chr1 5073028-5073291 \* | 10.20078 TRUE

[4] chr1 7079047-7079170 \* | 4.00702 TRUE

[5] chr1 7387587-7388292 \* | 19.01092 TRUE

[6] chr1 7606350-7606520 \* | 6.45661 TRUE

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

3.17. Now we can rank them by their fold enrichment, select the top 500 peaks and resize these peaks to 200bp around centre. **(2 pts)**

**Answer:**

**Rank Peaks by Fold Enrichment:**

**Add fold enrichment values to the common peaks:**

mcols(common\_peaks)$fold\_enrichment1 <- rep1\_gr$fold\_enrichment[queryHits(overlaps)]

mcols(common\_peaks)$fold\_enrichment2 <- rep2\_gr$fold\_enrichment[subjectHits(overlaps)]

**Ensure that mcols(common\_peaks) has both columns as numeric :**

mcols(common\_peaks)$fold\_enrichment1 <- as.numeric(mcols(common\_peaks)$fold\_enrichment1)

mcols(common\_peaks)$fold\_enrichment2 <- as.numeric(mcols(common\_peaks)$fold\_enrichment2)

**Calculate the mean fold enrichment of both replicates**

mcols(common\_peaks)$mean\_fold\_enrichment <- rowMeans(

cbind(mcols(common\_peaks)$fold\_enrichment1, mcols(common\_peaks)$fold\_enrichment2),

na.rm = TRUE

)

**Sort common peaks by mean fold enrichment (descending order)**

common\_peaks\_sorted <- common\_peaks[order(-mcols(common\_peaks)$mean\_fold\_enrichment)]

**Check the top few sorted peaks**

Input : head(common\_peaks\_sorted)

**output :**

GRanges object with 6 ranges and 5 metadata columns:

seqnames ranges strand | fold\_enrichment hit

<Rle> <IRanges> <Rle> | <numeric> <logical>

[1] chr4 45965892-45967087 \* | 123.1159 TRUE

[2] chr7 30126539-30127142 \* | 84.2649 TRUE

[3] chr3 87846861-87847811 \* | 96.9533 TRUE

[4] chr9 21156553-21157822 \* | 111.5354 TRUE

[5] chr9 21155616-21156515 \* | 111.5354 TRUE

[6] chr12 114345584-114346113 \* | 104.7455 TRUE

fold\_enrichment1 fold\_enrichment2 mean\_fold\_enrichment

<numeric> <numeric> <numeric>

[1] 123.1159 48.21738 85.6666

[2] 84.2649 42.03370 63.1493

[3] 96.9533 26.89088 61.9221

[4] 111.5354 10.47657 61.0060

[5] 111.5354 8.54992 60.0427

[6] 104.7455 15.12912 59.9373

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

**Select the Top 500 Peaks :**

**Select the top 500 peaks:**

**top500\_peaks <- common\_peaks\_sorted[1:500]**

**Input : head(top500\_peaks)**

**Output :**

GRanges object with 6 ranges and 5 metadata columns:

seqnames ranges strand | fold\_enrichment hit

<Rle> <IRanges> <Rle> | <numeric> <logical>

[1] chr4 45965892-45967087 \* | 123.1159 TRUE

[2] chr7 30126539-30127142 \* | 84.2649 TRUE

[3] chr3 87846861-87847811 \* | 96.9533 TRUE

[4] chr9 21156553-21157822 \* | 111.5354 TRUE

[5] chr9 21155616-21156515 \* | 111.5354 TRUE

[6] chr12 114345584-114346113 \* | 104.7455 TRUE

fold\_enrichment1 fold\_enrichment2 mean\_fold\_enrichment

<numeric> <numeric> <numeric>

[1] 123.1159 48.21738 85.6666

[2] 84.2649 42.03370 63.1493

[3] 96.9533 26.89088 61.9221

[4] 111.5354 10.47657 61.0060

[5] 111.5354 8.54992 60.0427

[6] 104.7455 15.12912 59.9373

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

**Resize the Top 500 Peaks to 200bp Around the Centre :**

**Resize the top 500 peaks to 200bp around the center :**

**top500\_resized <- resize(top500\_peaks, width = 200, fix = "center")**

**Input : top500\_resized**

**Output :**

GRanges object with 500 ranges and 5 metadata columns:

seqnames ranges strand | fold\_enrichment hit

<Rle> <IRanges> <Rle> | <numeric> <logical>

[1] chr4 45966390-45966589 \* | 123.1159 TRUE

[2] chr7 30126741-30126940 \* | 84.2649 TRUE

[3] chr3 87847236-87847435 \* | 96.9533 TRUE

[4] chr9 21157088-21157287 \* | 111.5354 TRUE

[5] chr9 21155966-21156165 \* | 111.5354 TRUE

... ... ... ... . ... ...

[496] chr5 73883812-73884011 \* | 37.3885 TRUE

[497] chr9 120756939-120757138 \* | 26.5492 TRUE

[498] chr18 68558809-68559008 \* | 41.8348 TRUE

[499] chr12 117600652-117600851 \* | 35.2066 TRUE

[500] chr10 61876395-61876594 \* | 36.9392 TRUE

fold\_enrichment1 fold\_enrichment2 mean\_fold\_enrichment

<numeric> <numeric> <numeric>

[1] 123.1159 48.21738 85.6666

[2] 84.2649 42.03370 63.1493

[3] 96.9533 26.89088 61.9221

[4] 111.5354 10.47657 61.0060

[5] 111.5354 8.54992 60.0427

... ... ... ...

[496] 37.3885 11.25025 24.3194

[497] 26.5492 22.07525 24.3122

[498] 41.8348 6.76251 24.2987

[499] 35.2066 13.31497 24.2608

[500] 36.9392 11.56110 24.2502

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

3.18. Extract the sequences underneath the file and write them to FASTA file in you working directory. Inspect the file in notepad. **(2 pts)**

**Answer :**

**Load package : library(Biostrings)**

**Extracting the sequences from the top 500 peaks:**

**top500\_sequences <- getSeq(genome, common\_peaks\_sorted[1:500])**

**Sequences to a FASTA file:**

**writeXStringSet(top500\_sequences, filepath = "C:/Users/gdeep/Downloads/LMS\_ChIPseq\_short-master-2023-final/LMS\_ChIPseq\_short-master-2023-final/course/data/MacsPeaks/top500\_peaks\_corrected.fasta")**

**Load the mm9 Genome:**

**if (!requireNamespace("BSgenome.Mmusculus.UCSC.mm9", quietly = TRUE)) {**

**BiocManager::install("BSgenome.Mmusculus.UCSC.mm9")**

**}**

**library(BSgenome.Mmusculus.UCSC.mm9)**

**Extract Sequences for Top 500 Peaks:**

**top500\_sequences <- getSeq(BSgenome.Mmusculus.UCSC.mm9, common\_peaks\_sorted[1:500])**

**Write Sequences to a FASTA File:**

**writeXStringSet(top500\_sequences, filepath = “C:/Users/gdeep/Downloads/LMS\_ChIPseq\_short-master-2023-final/LMS\_ChIPseq\_short-master-2023-final/course/data/MacsPeaks/top500\_peaks\_corrected.fasta”)**

**View the first few sequences:**

**Input : head(top500\_fasta)**

**Output :**

**A close-up of a dna sequence

AI-generated content may be incorrect.**

**500 length with 200 bp width**

**Input :**

library(GenomicRanges)

library(Biostrings)

library(BSgenome.Mmusculus.UCSC.mm9)

**Extract fold enrichment values**

mcols(common\_peaks)$fold\_enrichment1 <- rep1\_gr$fold\_enrichment[queryHits(overlaps)]

mcols(common\_peaks)$fold\_enrichment2 <- rep2\_gr$fold\_enrichment[subjectHits(overlaps)]

**Convert to numeric**

mcols(common\_peaks)$fold\_enrichment1 <- as.numeric(mcols(common\_peaks)$fold\_enrichment1)

mcols(common\_peaks)$fold\_enrichment2 <- as.numeric(mcols(common\_peaks)$fold\_enrichment2)

**Calculate mean fold enrichment**

mcols(common\_peaks)$mean\_fold\_enrichment <- rowMeans(

cbind(mcols(common\_peaks)$fold\_enrichment1, mcols(common\_peaks)$fold\_enrichment2),

na.rm = TRUE

)

**Sort common peaks by mean fold enrichment in descending order**

common\_peaks\_sorted <- common\_peaks[order(-mcols(common\_peaks)$mean\_fold\_enrichment)]

common\_peaks\_sorted <- common\_peaks[order(-mcols(common\_peaks)$mean\_fold\_enrichment)]

**Select top 500 peaks**

top500\_peaks <- common\_peaks\_sorted[1:500]

**Resize peaks to 200bp centered around the middle**

resize\_peaks <- function(peaks) {

peaks$center <- round((start(peaks) + end(peaks)) / 2)

start(peaks) <- peaks$center - 99

end(peaks) <- peaks$center + 100

return(peaks)

}

top500\_peaks\_resized <- resize\_peaks(top500\_peaks)

**Extract sequences from mm9 genome**

top500\_sequences <- getSeq(genome, names = seqnames(top500\_peaks\_resized),

start = start(top500\_peaks\_resized), end = end(top500\_peaks\_resized))

**Define output FASTA file path**

output\_fasta <- "C:/Users/gdeep/Downloads/LMS\_ChIPseq\_short-master-2023-final/LMS\_ChIPseq\_short-master-2023-final/course/data/MacsPeaks/top500\_peaks\_corrected.fasta"

**Write sequences to a FASTA file**

writeXStringSet(top500\_sequences, filepath = "C:/Users/gdeep/Downloads/LMS\_ChIPseq\_short-master-2023-final/LMS\_ChIPseq\_short-master-2023-final/course/data/MacsPeaks/top500\_peaks\_corrected.fasta")

**View the first few sequences**

head(top500\_sequences)

**Output:**

A close-up of a dna test

AI-generated content may be incorrect.

3.19. Upload the sequences to Meme-ChIP and report the results when complete. **(2 pts)**

**Answer :**

**Uploaded file to Meme-chIP :**  ( top500\_peaks\_corrected.fasta)

<https://meme-suite.org/meme/opal-jobs/appMEMECHIP_5.5.717435154969531007311282/top500_peaks_corrected.fasta>

A screenshot of a computer

AI-generated content may be incorrect.

**Output from MEME-ChIP:**

**1. CTCF Motif:**

* Discovery Program: MEME
* E-value: 1.9e-075 (Extremely significant, as this E-value is very small, indicating a very high confidence in the motif being honest and not a random finding.)
* Known or Similar Motifs: CTCF (MA1929.1), CTCF (MA0139.1), CTCF (MA1930.1)
  + CTCF is a well-known transcription factor that binds to DNA and plays a key role in regulating gene expression and chromatin architecture. The multiple similar motifs found here further confirm the reliability of the motif.
* Distribution: Not Centrally Enriched
  + This suggests that the CTCF motif is found throughout the genome, not predominantly around the center of peaks or regions of interest.

**A white background with black and orange text

AI-generated content may be incorrect.**

**2. ZNF384 Motif:**

* Discovery Program: MEME
* E-value: 4.0e-029 (This is also a very significant result.)
* Known or Similar Motifs: ZNF384 (MA1125.1), Srf\_secondary (UP00077\_2), Mtf1\_secondary (UP00097\_2)
  + ZNF384 is a zinc finger protein that is involved in various transcriptional regulatory processes. The similarity to other motifs suggests that the discovered motif shares features with known transcription factors or regulatory elements.
* Distribution: Not Centrally Enriched
  + Similar to CTCF, this motif is also not centrally enriched, which indicates that it may be spread more evenly across the genome rather than being positioned explicitly around key regulatory regions.

A group of black and blue text

AI-generated content may be incorrect.

**3. MAX::MYC Motif:**

* Discovery Program: MEME
* E-value: 9.8e-023 (Also very significant but slightly less so than the previous two.)
* Known or Similar Motifs: MAX::MYC (MA0059.1), MAX (MA0058.3), MAX\_DBD\_2
  + MAX::MYC is a heterodimer formed between MYC and MAX proteins, which is involved in regulating cell proliferation, apoptosis, and other cellular processes. This is a known motif involved in cancer and cellular growth regulation.
* Distribution: Not Centrally Enriched
  + This suggests the motif is likely involved in broader regulatory roles across the genome rather than being enriched explicitly in one region, which may reflect its roles in various transcriptional networks.

A white background with black and blue text

AI-generated content may be incorrect.

**4. GATA5 Motif:**

* Discovery Program: CentriMo
* E-value: 6.0e-004 (This is a less significant E-value than the MEME discoveries, but still statistically relevant for this type of analysis.)
* Known or Similar Motifs: GATA5 (MA0766.2)
  + GATA5 is a transcription factor involved in regulating development, particularly in the heart and vascular tissue. It is part of the GATA family of transcription factors that bind to specific DNA sequences.
* Distribution: Not specified, but based on the program used (CentriMo), this motif may be more enriched in some areas of the genome.

A group of colorful text

AI-generated content may be incorrect.

**Motif Report:**

The significant motifs identified in this study, which have very low E-values, indicate important regulatory elements in the genome. These motifs, including CTCF, ZNF384, MAX::MYC, and GATA5, are likely involved in regulating essential biological processes such as transcription, cell cycle control, and development. The observation that many of these motifs are not centrally enriched suggests they may function over broad genomic regions or across multiple genomic areas.

Further functional experiments and validations will be necessary to confirm the precise role of these motifs in the biological systems being studied..