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

**Student ID: 2407303 (SGUL) or k24121590 (KCL)**

## 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.

The **cd** command stands for “change directory”, it changes the current directory to the one specified after the command.

The **cd /** changes the current working directory to the root directory (**/**), the top-level directory in the file system hierarchy.

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

Answer: To get help about the **cp** command, I can use the ***man*** (manual) command.

Example command: **man cp**, this will display the manual page for the **cp** command, showing its usage, options and examples.

(Alternatively, I can also use the ***- -help*** option with the **cp** command, to get a summary of available options. For example, by typing **cp - -help**, this will display a list of options and a brief description of how to use the cp command, without opening the full manual)

1.4 What does the command pwd do?

The **pwd** command stands for "print working directory". It displays the current directory I’m in within the file system.

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: I can use the **ls** command with the **-l** (long listing) option

Example command: **ls –l**

a) **ls** is the command for listing files and directories.

b) The **-l** option (long format) shows detailed information for each file or directory

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: I can use the **head** command in combination with a wildcard **\*.txt.**

Example command: **head -n 15 \*.txt**

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

Answer: I can use the **mv** (move) command. The mv command is used both for renaming files and moving them.

Example command: **mv new old**

1.8 How do you display the contents of a file myfile.txt? (please provide an example command)

Answer: The **less** command lets you view the file page by page with navigation controls. Best for large files.

Example command: **less myfile.txt**.

Answer: The **cat** command is commonly used to display the entire contents of the file. Best for small files. (Unlike **less**, you have no control on how you view that text or what you do with it)

Example command: **cat myfile.txt**

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

Answer: I can use the **mkdir** (make directory) command.

Example command: **mkdir flower**

1.10 How do you change the current directory to /usr/local/bin? (please provide an example command)  
Answer: I can use the **cd** (change directory) command.

Example command: **cd /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: I can use the **ls** command with the **-a** (all) option.

Example command: **ls -a**

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

Answer: I can use the **cd** (change directory) command with the **..** notation.

Example command: **cd ..**

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

Answer: I can use the **mkdir** (make directory) command, followed by the **~** command (this is a shortcut that represent my home directory).

Example command: if for example, I want to create a sub-directory named *assessment* in my home directory, the command is: **mkdir ~/assessment**

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

Answer: To list the first few lines of a text file (called *txtfile*), I can use the **head** command.

Example command: **head txtfile**

(The command **head** shows the first 10 lines of the specified file by default)

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

Answer: To display the last few lines of a text file (called *file1*), I can use the **tail** command.

Example command: **tail file1**

(The **tail** command shows the last 10 lines of the specified file by default)

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

Answer: I can use the **cut** command.

Example command: if I have a file called *file.txt* where columns are separated by commas (,) and I want to extract the first column, I use the command: **cut -d ',' -f 1 file.txt** (where **–d** specifies the delimiter that separates columns in the text file, e.g., comma, and **-f 1** specifies the first column)

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

Answer: I can use the **cp** command with the **-r** option. The -r option ensures that the entire directory structure is copied, not just individual files.

Example command: **cp -r Project 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: I can use the **grep** command with a regular expression that specifies the end of a line (**$**).

Example command: **grep 'Hypertension$' diseases.txt**

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

Answer: I can use the **ls** command with the **-a** option. (Hidden files in Unix/Linux start with a dot (.), and the -a option makes sure those files are included in the listing).

Example command: **ls -a ~**

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

Answer: I can use the **nohup** command.

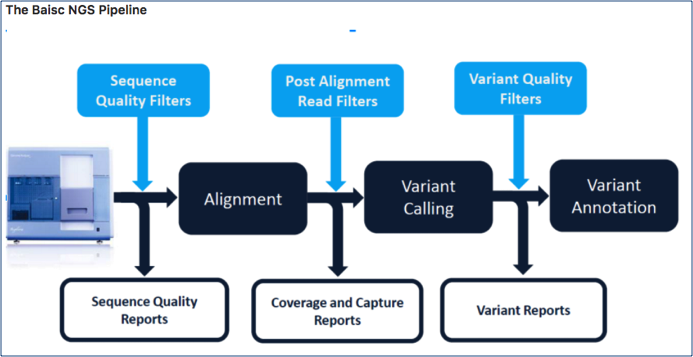
Example command: if the job I want to run is a file or script called *data*, the command is **nohup ./data &**

## 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.**

Step 1: I created a new repositories on Github, called Advanced Bioinformatics Assessment

Step 2: I uploaded it on my Linux

Command: git clone <https://github.com/RobertaCo1/Advanced-Bioinformatics-Assessment>

## 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)

Step 1: I installed Miniconda

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

Command: chmod +x ./Miniconda3-latest-Linux-x86\_64.sh

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

Miniconda will now be installed in /home/ubuntu/miniconda3n

Step 2: I installed the Necessary Bioinformatics Tools

Command: conda install samtools

Command: conda install bwa

Command: conda install freebayes

Command: conda install picard

Command: conda install bedtools

Command: conda install trimmomatic

Command: conda install fastqc

Command: sudo apt install libvcflib-tools

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

Step 1: I created my assessment directories

Command: mkdir assessment

Command: mkdir assessment/dnaseq

Command: cd assessment/dnaseq

Command: mkdir data meta results logs

Step 2: I created two directories within the data directory, one folder for untrimmed reads and another for our trimmed reads

Command: cd ~/assessment/dnaseq/data

Command: mkdir untrimmed\_fastq

Command: mkdir trimmed\_fastq

Step 2: I downloaded the raw\_fastq data and the bed file

Command: wget [*https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R1.fastq.qz*](https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R1.fastq.qz)

Command: wget [*https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R2.fastq.qz*](https://s3-eu-west-1.amazonaws.com/workshopdata2017/NGS0001.R2.fastq.qz)

As result, I got 2 zipped files:

NGS0001.R1.fastq.qz

NGS0001.R2.fastq.qz

Then I renamed the two files above (by right clicking on each file and selecting “rename”):

NGS0001.R1.fastq.gz

NGS0001.R2.fastq.gz

Command: wget [*https://s3-eu-west-1.amazonaws.com/workshopdata2017/annotation.bed*](https://s3-eu-west-1.amazonaws.com/workshopdata2017/annotation.bed)

Step 3: I copied the raw fastq files to our *untrimmed\_fastq* directory and the bed file in the *data* directory

Command: mv \*fastq.gz ~/assessment/dnaseq/data/untrimmed\_fastq

mv annotation.bed ~/assessment/dnaseq/data

Step 4: I downloaded the reference genome Hg19 and I copy into data

Command: wget http://hgdownload.cse.ucsc.edu/goldenPath/hg19/bigZips/hg19.fa.gz

Command: mv hg19.fa.gz ~/assessment/dnaseq/data/

***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)

QUALITY ASSESSMENT

Step 1: I changeed directory to untrimmed\_fastq

Command: cd ~/assessment/dnaseq/data/untrimmed\_fastq

FastQC is the command for running the **FastQC** tool, which performs a quality control check on sequencing data in **FASTQ format**.

FastQC will accept multiple file names as input, so we can use the **\*.fastq.gz** wildcard. This is a **wildcard** pattern that selects all files in the current directory that have the .fastq.gz extension (which is the compressed version of FASTQ files).

The \* means "any file that ends with .fastq.gz", so it will match all such files in the directory.

Command: fastqc \*.fastq.gz

I can use the multi-threading functionality of FastQC to run 4 jobs at once. The **-t 4** option specifies the number of threads (or processors) to use for running FastQC. The **-t 4** tells FastQC to use 4 CPU threads. This option speeds up the process, especially if I’m running FastQC on multiple large files, by processing multiple files simultaneously. More threads can be used if the system has more cores, improving the processing time.

Command: fastqc -t 4 \*.fastq.gz

Then I created a home for my results and move them there

Command: mkdir ~/assessment/dnaseq/results/fastqc\_untrimmed\_reads

Command: mv \*fastqc\* ~/assessment/dnaseq/results/fastqc\_untrimmed\_reads/

The other output of FastQC is a .zip file. These .zip files need to be unpacked, by using a simple shell for loop to iterate through the list of files in \*.zip.

Command: cd ~/assessment/dnaseq/results/fastqc\_untrimmed\_reads/

Command: for zip in \*.zip

> do

> unzip $zip

> done

It will run unzip once for each file (whose name is stored in the $zip variable). The contents of each file will be unpacked into a separate directory by the unzip program.

~~The commands~~

~~head NGS0001.R1.fastq.gz \_fastqc/summary.txt~~

~~and~~

~~head NGS0001.R2.fastq.gz\_fastqc/summary.txt will display the~~ **~~first 10 lines~~** ~~of the file summary.txt located in the both fastqc/ directory.~~

~~Then I used the cat command for saving all fastqc summary.txt files into one full\_report.txt under the logs directory~~

~~cat \*/summary.txt > ~/assessment/dnaseq/logs/fastqc\_summaries.txt~~

TRIMMING

Step 1: I changed the directories to the untrimmed fastq data location:

Command: cd ~/assessment/dnaseq/data/untrimmed\_fastq

Step 3: Then I decompressed the files:

Command:

gzip -d /home/ubuntu/assessment/dnaseq/data/untrimmed\_fastq/NGS0001.R1.fastq.gz

and

Command:

gzip -d /home/ubuntu/assessment/dnaseq/data/untrimmed\_fastq/NGS0001.R2.fastq.gz

Step 4: Then I did the trimming

Command: trimmomatic PE -threads 4 -phred33 /home/ubuntu/assessment/dnaseq/data/untrimmed\_fastq/NGS0001.R1.fastq /home/ubuntu/assessment/dnaseq/data/untrimmed\_fastq/NGS0001.R2.fastq -baseout /home/ubuntu/assessment/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_R ILLUMINACLIP:/home/ubuntu/anaconda3/pkgs/trimmomatic-0.39-hdfd78af\_2/share/trimmomatic-0.39-2/adapters/NexteraPE-PE.fa:2:30:10 TRAILING:25 MINLEN:50

-**trimmomatic PE**: This tells Trimmomatic you're working with paired-end data.

**-threads 4**: Specifies 4 threads to use, which should help speed up the process.

**-phred33**: Specifies that your quality scores are in Phred+33 format.

**Input files**:

* /home/ubuntu/assessment/dnaseq/data/untrimmed\_fastq/NGS0001.R1.fastq.qz:

Forward read (R1) file.

* /home/ubuntu/assessment/dnaseq/data/untrimmed\_fastq/NGS0001.R2.fastq.qz:

Reverse read (R2) file.

**-baseout**: Specifies the base filename for the trimmed output files. The trimmed R1 and R2 files will be written to /home/ubuntu/assessment/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_R.

**ILLUMINACLIP**: This step trims adapter sequences. I’m using the Nextera PE adapter file (NexteraPE-PE.fa) with the parameters:

* 2: Max mismatch count allowed for an adapter match.
* 30: The seed length for the adapter search.
* 10: The palindrome clip threshold.

**TRAILING:25**: Removes trailing bases with a quality score below 25.

**MINLEN:50**: Removes reads that are shorter than 50 bases after trimming.

**Output Files:**

* **NGS0001\_trimmed\_R\_1P**: Forward reads (R1) that passed the quality trimming (paired reads).
* **NGS0001\_trimmed\_R\_1U**: Forward reads (R1) that did **not** pass the quality trimming (unpaired reads).
* **NGS0001\_trimmed\_R\_2P**: Reverse reads (R2) that passed the quality trimming (paired reads).
* **NGS0001\_trimmed\_R\_2U**: Reverse reads (R2) that did **not** pass the quality trimming (unpaired reads).

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

Command: cd ~/assessment/dnaseq/data/trimmed\_fastq

Command: fastqc /home/ubuntu/assessment/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_R\_1P /home/ubuntu/assessment/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_R\_2P

### 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)

I will use BWA (a software package for mapping DNA sequences against a large reference genome, such as the human genome) and its algorithm MEM (designed for longer sequences ranged from 70bp to a few megabases)

Command: bwa mem

First I created a folder for the references and its index files. Then I moved the

hg19.fa.gz from data to reference

Command: mkdir -p ~/assessment/dnaseq/data/reference

Command: mv ~/assessment/dnaseq/data/hg19.fa.gz ~/assessment/dnaseq/data/reference/

Command: bwa index ~/assessment/dnaseq/data/reference/hg19.fa.gz

First I created a new folder aligned data

Command: mkdir ~/assessment/dnaseq/data/aligned\_data

Then I run the BWA MEM with the RG information for NGS0001 to generate the NGS0001 sam file. I used the following read group infos for the alignment (the same used for Galaxy)

Read group identifier (ID): 11V6WR1:111:D1375ACXX:1:NGS0001

Read group identifier (SM): NGS0001

Read group identifier (PL): ILLUMINA

Read group identifier (LB): Nextera-NGS0001-blood

Read group identifier (PU): 11V6WR1

Read group identifier (DT): 2025-03-17

Command: bwa mem -t 4 -v 1 -R '@RG\tID:11V6WR1:111:D1375ACXX:1:NGS0001\tSM:NGS0001\tPL:ILLUMINA\tLB:nextera-NGS0001-blood\tDT:2025-03-17\tPU:11V6WR1 -I 250,50 ~/assessment/dnaseq/data/reference/hg19.fa.gz ~/assessment/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_R\_1P ~/assessment/dnaseq/data/trimmed\_fastq/NGS0001\_trimmed\_R\_2P > ~/assessment/dnaseq/data/aligned\_data/NGS0001.sam

**-t 4**: This flag specifies the number of threads to use for the alignment. In this case, I’m using 4 threads for parallel processing.

**-v 1**: This specifies the verbosity of the output. -v 1 is appropriate for basic logging.

**-I 250,50:** These are the gap penalties for insertion and deletion. This is perfectly fine for Illumina paired-end data. The parameters 250,50 represent:

250 for the gap opening penalty.

50 for the gap extension penalty.

Then I changed directories into the aligned\_data folder.

Command: cd ~/assessment/dnaseq/data/aligned\_data

and converted the sam file into bam format, sort it and generate an index using samtools.

Command: samtools view -h -b NGS0001.sam > NGS0001.bam

Command: samtools sort NGS0001.bam > NGS0001\_sorted.bam

Command: samtools index NGS0001\_sorted.bam

* Perform duplicate marking (2pts)

I will use Picard to mark duplicated reads

Command: picard MarkDuplicates I=NGS0001\_sorted.bam O=NGS0001\_sorted\_marked.bam M=marked\_dup\_metrics.txt

Command: samtools index NGS0001\_sorted\_marked.bam

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

The **Samtools view** command is used to view, convert, or filter BAM/SAM files. In this case, it's being used for filtering BAM.

Command: samtools view -F 1796 -q 20 -o NGS0001\_sorted\_filtered.bam NGS0001\_sorted\_marked.bam

**-F 1796**: This option filters out alignments that are marked with any of the bits in the given flag value 1796. The -F flag works by excluding reads with the specific flags set.

**-q 20**: This filters the reads by mapping quality. Only reads with a mapping quality of **20 or higher** will be kept. Mapping quality scores are used to indicate how confidently a read is aligned to the reference genome. Setting -q 20 ensures that only high-confidence alignments are retained in the output BAM file.

**-o NGS0001\_sorted\_filtered.bam**: This option specifies the **output file**. The filtered BAM file will be written to NGS0001\_sorted\_filtered.bam.

**NGS0001\_sorted\_marked.bam**: This is the **input BAM file**. It contains the reads that have already been sorted and marked

Command: samtools index NGS0001\_sorted\_filtered.bam

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

1. **FLAGSTATS**

samtools flagstat provides a summary of the alignment flags for the BAM file, such as the number of mapped/unmapped reads, properly paired reads, duplicate reads, and other useful information.

Command: samtools flagstat NGS0001\_sorted\_filtered.bam > NGS0001\_sorted\_filtered\_flagstat.txt

This will output the flag statistics to a text file (NGS0001\_sorted\_filtered\_flagstat.txt).

1. **IDXSTATS**

samtools idxstats provides information about the reference genome sequences, including the number of mapped reads and the length of each reference sequence.

Command: samtools idxstats NGS0001\_sorted\_filtered.bam > NGS0001\_sorted\_filtered\_idxstats.txt

This will give me the alignment statistics per reference sequence (e.g., chromosome) in a text file (NGS0001\_sorted\_filtered\_idxstats.txt).

1. **DEPTH of COVERAGE**

To assess the depth of coverage across my aligned reads, I will use samtools depth. This will give me per-base coverage across the reference genome.

Command: samtools depth NGS0001\_sorted\_filtered.bam > NGS0001\_sorted\_filtered\_depth.txt

This command will produce a file with three columns: reference name, position, and coverage at that position.

1. **INSERT SIZE**

To get insert size distribution (the size of the DNA fragment between paired reads), I will use samtools stats. This will also give me other useful statistics such as mapping quality, insert size, and read length distributions

Command: samtools stats NGS0001\_sorted\_filtered.bam > NGS0001\_sorted\_filtered\_stats.txt

This will generate a file (NGS0001\_sorted\_filtered\_stats.txt) with various alignment statistics, including insert size statistics.

### 2.4. Variant Calling (4pts)

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

Command: zcat ~/assessment/dnaseq/data/reference/hg19.fa.gz > ~/assessment/dnaseq/data/reference/hg19.fa

This command decompresses a **gzipped** FASTA file (hg19.fa.gz) and saves the output as a regular FASTA file (hg19.fa).

Command: samtools faidx ~/assessment/dnaseq/data/reference/hg19.fa

This creates an index file *reference/hg19.fa.fai*, which stores information about sequence names, lengths, and file offsets.

Command: freebayes --bam ~/assessment/dnaseq/data/aligned\_data/NGS0001\_sorted\_filtered.bam --fasta-reference ~/assessment/dnaseq/data/reference/hg19.fa --vcf ~/assessment/dnaseq/results/NGS0001.vcf

Command: bgzip ~/assessment/dnaseq/results/NGS0001.vcf

Command: tabix -p vcf ~/assessment/dnaseq/results/NGS0001.vcf.gz

* Quality Filter Variants using your choice of filters (2pt)

Command: sudo apt install libvcflib-tools

This is to install vcflib

Command: zcat ~/assessment/dnaseq/results/NGS0001.vcf.gz | vcffilter -f "QUAL > 1 & QUAL / AO > 10 & SAF > 0 & SAR > 0 & RPR > 1 & RPL > 1" > ~/assessment/dnaseq/results/NGS0001\_filtered.vcf

This will create the NGS0001 filtered vcf

Using bedtools we can filter the vcf file for the regions in the annotation.bed:

Command: bedtools intersect -header -wa -a ~/assessment/dnaseq/results/NGS0001\_filtered.vcf -b ~/assessment/dnaseq/data/annotation.bed \

> ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.vcf

Command: bgzip ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.vcf

Command: tabix -p vcf ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.vcf.gz

### 2.5. Variant Annotation and Prioritization (10pts)

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

**ANNOVAR**

I downloaded ANNOVAR from the following website:

<https://www.openbioinformatics.org/annovar/annovar_download_form.php>

Then I uploaded it into MobaXterm

Then I unzipped the file

Command: tar -zxvf annovar.latestversion.tar.gz

and downloaded the databases that ANNOVAR uses for the annotation

Command: cd annovar

./annotate\_variation.pl -buildver hg19 -downdb -webfrom annovar knownGene humandb/

./annotate\_variation.pl -buildver hg19 -downdb -webfrom annovar refGene humandb/

./annotate\_variation.pl -buildver hg19 -downdb -webfrom annovar ensGene humandb/

./annotate\_variation.pl -buildver hg19 -downdb -webfrom annovar clinvar\_20180603 humandb/

./annotate\_variation.pl -buildver hg19 -downdb -webfrom annovar exac03 humandb/

./annotate\_variation.pl -buildver hg19 -downdb -webfrom annovar dbnsfp31a\_interpro humandb/

Then VCF to ANNOVAR input format

Command: ./convert2annovar.pl -format vcf4

~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.vcf.gz > ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.avinput

I got PERMISSION DENIED, so I run the convert2annovar

Command: perl /home/ubuntu/annovar/convert2annovar.pl -format vcf4 ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.vcf.gz > ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.avinput

I first downloaded the humandb/hg19\_refGen

Command: perl /home/ubuntu/annovar/annotate\_variation.pl -buildver hg19 -downdb -webfrom annovar refGene /home/ubuntu/annovar/humandb

And then I run ANNOVAR table function

Command: perl /home/ubuntu/annovar/table\_annovar.pl ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.avinput /home/ubuntu/annovar/humandb -buildver hg19 -out ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation -remove -protocol refGene,ensGene,clinvar\_20180603,exac03,dbnsfp31a\_interpro -operation g,g,f,f,f -otherinfo -nastring . -csvout

And I got the attached cvs file



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

The avsnp150 is used to exclude known dbSNP variants

Command: cd ~/annovar/humandb

Considering the **hg19\_avsnp150.txt.gz** is zipped and I don’t have more space, first I created a smaller text file to check if ANNOVAR recognizes it.

Command: ~/annovar/humandb$ zcat /home/ubuntu/annovar/humandb/hg19\_avsnp150.txt.gz | head -n 100000 > /home/ubuntu/annovar/humandb/hg19\_avsnp150.txt

To extract the annovar cvs with exonic (refGene) and dbSNP (avsnp150) from the NGS0001\_filtered annotation.avinput, I will use the following command

Command: perl /home/ubuntu/annovar/table\_annovar.pl \

~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.avinput \

/home/ubuntu/annovar/humandb/ \

-buildver hg19 \

-out ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation\_dbsnp \

-remove \

-protocol refGene,avsnp150 \

-operation g,f \

-otherinfo \

-nastring . \

-csvout

This generates the following excel sheet with avsnp150 column



Then I used column 6 and column 11 to filter to exonic variants not seen in dbSNP.

The **"."** will filter out variants that **are NOT in dbSNP**

- The Func.refGene (Column 6) determines where the variant is located (e.g., exonic, intronic, UTR, intergenic). I filtered for rows where this column contains "exonic".

- The avsnp150 (Column 11) is used to exclude known dbSNP variants (i.e., filter rows where avsnp150 == ".").

I won’t use the following other columns because the exonicFunc.refGene

describes the effect of the variant in exonic regions (e.g., synonymous, missense, stop-gain).

Useful for further classification after filtering exonic variants. And also I won’t use the Func.ensGene because is the same as Func.refGene but based on Ensembl gene annotations.

Command: awk -F, 'NR==1 || ($6 ~ /exonic/ && $11 == ".")' ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation\_dbsnp.hg19\_multianno.csv > ~/assessment/dnaseq/results/exonic\_variants\_not\_in\_dbSNP.csv

I got the following cvs as result:



**snpEFF**

I downloaded snpEFF

Command: cd /home/ubuntu/

wget https://snpeff.blob.core.windows.net/versions/snpEff\_latest\_core.zip

Command: unzip snpEff\_latest\_core.zip

For snpEFF, I would need to use Java. So I installed Java 21 by using the following command

Command: sudo apt install openjdk-21-jdk -y

Then I run the snpEFF annotation:

I can use the ANNOVAR cvs file as input

Command: java -Xmx4G -jar /home/ubuntu/snpEff/snpEff.jar \

hg19 \

~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.hg19\_multianno.csv \

> ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation\_snpeff.vcf

Or I can use the NGS0001\_filtered\_annotation.vcf.gz file as input.

Command: java -Xmx4G -jar /home/ubuntu/snpEff/snpEff.jar hg19 \

~/assessment/dnaseq/results/NGS0001\_filtered\_annotation.vcf.gz > \

~/assessment/dnaseq/results/NGS0001\_filtered\_annotation\_snpeff.vcf

This command generates a snpeff annotated vcf file as output.

Then I can use bcftools to convert the vcf file to Table

Command: bcftools query -f '%CHROM\t%POS\t%REF\t%ALT\t%INFO/ANN\n' ~/assessment/dnaseq/results/NGS0001\_filtered\_annotation\_snpeff.vcf > NGS0001\_snpeff.tsv

**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).

**BASH SCRIPT**

Uploaded into GitHub [RobertaCo1/Advanced-Bioinformatics-Assessment](https://github.com/RobertaCo1/Advanced-Bioinformatics-Assessment)

(please note: I run out of space and I had to delete lots of files on my pipeline, including sam file, bam file, sorted bam file, hg19.fa.gz.sa, hg19.fa.pac etc…)

Command: cd ~/assessment

Command: vim bscript.sh

Then I wrote all commands as per attached bash script:



TO RUN the bash script, I changed directory to assessment

Command: cd ~/assessment/

Then I made it executable

Command: chmod +x bscript.sh

And finally I run it by using the following command.

Command: bash bscript.sh

## 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.

[RobertaCo1/Advanced-Bioinformatics-Assessment](https://github.com/RobertaCo1/Advanced-Bioinformatics-Assessment)



Under the **File** menu in RStudio, I selected **New File** > **R Markdown**

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 (33 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. (4pt)

Command: sum(5:55)

Result: [1] 1530

This code generates the sum of all integers between 5 and 55

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 do the calculation for *n* = 10, *n* = 20, and *n* = 100 and present the results. (4pt)

The function uses sum(5:n) to calculate the sum of all integers between 5 and n.

# Define the sumfun function

Command: sumfun <- function(n) {

sum(5:n)

}

Then I test the function with n = 10, n = 20, and n = 100.

# Test the function with different values of n

Command: result\_10 <- sumfun(10)

result\_20 <- sumfun(20)

result\_100 <- sumfun(100)

# Print the results

Command: cat("Sum of integers between 5 and 10:", result\_10, "\n")

cat("Sum of integers between 5 and 20:", result\_20, "\n")

cat("Sum of integers between 5 and 100:", result\_100, "\n")

Result:

Sum of integers between 5 and 10: 45

Sum of integers between 5 and 20: 180

Sum of integers between 5 and 100: 4950

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. (4pt)

I create a numeric vector fib with a length of 12 to store the Fibonacci numbers. The first two Fibonacci numbers are manually set as 1 and 1.

# Initialize the first two Fibonacci numbers

Command:fib <- numeric(12) # Create an empty vector to store the Fibonacci series

fib[1] <- 1 # First Fibonacci number

fib[2] <- 1 # Second Fibonacci number

The loop starts from index 3 (the third Fibonacci number) and calculates each Fibonacci number as the sum of the previous two numbers (fib[i - 1] + fib[i - 2])

# Use a for loop to calculate the rest of the Fibonacci numbers

Command:for (i in 3:12)

{fib[i] <- fib[i - 1] + fib[i - 2] # Each Fibonacci number is the sum of the

previous two}

After the loop completes, the first 12 Fibonacci numbers are printed.

# Print the first 12 Fibonacci numbers

Command: print(fib)

Result:

[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. (4pt)

# Load ggplot2 package

Command: library(ggplot2)

# Create a boxplot using ggplot2

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

geom\_boxplot() +

labs(title = "Boxplot of Miles per Gallon (mpg) by Number of Gears",

x = "Number of Gears",

y = "Miles per Gallon (mpg)",

fill = "Number of Gears") + theme\_minimal()

**aes(x = factor(gear), y = mpg, fill = factor(gear))**:

* x = factor(gear): Treats the gear variable as a categorical variable (factor) for the x-axis.
* y = mpg: Maps the mpg variable to the y-axis.
* fill = factor(gear): Colors the boxes by the number of gears, which is also treated as a categorical variable.

**geom\_boxplot()**: This function creates the boxplot.

**labs()**: This adds a title and labels to the axes, as well as a label for the fill legend.

**theme\_minimal()**: This applies a minimal theme to the plot for a cleaner look

This code will generate a boxplot showing the distribution of miles per gallon (mpg) for each number of gears (gear), with each box filled in a different color 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? (4pt)

# Load the cars dataset

Command: data(cars)

# Fit a linear model using lm() function

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

**lm(distance ~ speed, data = cars)**: This fits a linear model where distance (the breaking distance) is modelled as a function of speed (the car's speed).

# Display the summary of the model to get the fitted slope, intercept, and standard errors

Command: summary(model)

The **summary(model)** function provides detailed information about the fitted model, including the slope, intercept and their respective standard errors.

Result:

This code will generate the result below:

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

3.6. Use *ggplot* to plot the data points from Task 6 and the linear fit. (4pt)

# Load ggplot2 package

Command: library(ggplot2)

# Load the cars dataset

Command: data(cars)

# Create a scatter plot of speed vs. distance, with a linear regression line

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

geom\_point() + # Add data points as a scatter plot

geom\_smooth(method = "lm", color = "blue", se = FALSE) + # Add linear regression line

labs(title = "Speed vs. Breaking Distance with Linear Fit",

x = "Speed (mph)",

y = "Breaking Distance (ft)") +

theme\_minimal()

**ggplot(cars, aes(x = speed, y = dist))**: This creates a ggplot object using the cars dataset. The speed variable is mapped to the x-axis, and the dist (breaking distance) variable is mapped to the y-axis.

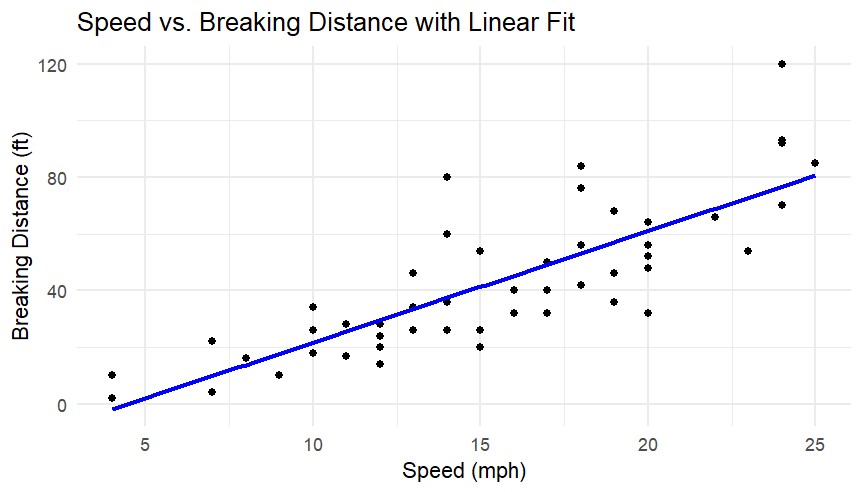
**geom\_point()**: This function adds the data points to the plot, making it a scatter plot.

**geom\_smooth(method = "lm", color = "blue", se = FALSE)**: This adds a linear regression line to the plot using the **lm** method. The color = "blue" argument specifies the color of the regression line. se = FALSE disables the confidence interval shading around the regression line.

**labs()**: This adds the title and axis labels.

**theme\_minimal()**: This applies a minimal theme for a clean look.

This code will generate a scatter plot of **speed** versus **breaking distance** and overlay a linear regression line based on the fitted model. The regression line will visually show how the breaking distance increases with speed.



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. (9pt)

# Load necessary libraries

Command: library(ggplot2)

# Load the cars dataset

Command: data(cars)

# Create a new variable for speed squared (since braking distance is proportional to speed^2)

Command: cars$speed\_squared <- cars$speed^2

I created a new variable speed\_squared that stores the square of the speed (speed^2), as braking distance is assumed to be proportional to speed^2.

# Fit a linear model to the data, where dist = reaction\_time \* speed + k \* speed^2

Command: model <- lm(dist ~ speed + speed\_squared, data = cars)

I fitted a linear model dist ~ speed + speed\_squared, where dist (the braking distance) is modelled as a function of speed and speed\_squared.

This model assumes that the total braking distance (dist) consists of two components: a linear component (reaction\_time \* speed) and a quadratic component (k \* speed^2).

# Display the summary of the model

Command: summary(model)

Result:

Call:

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

Residuals:

Min 1Q Median 3Q Max

-28.720 -9.184 -3.188 4.628 45.152

Coefficients:

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

(Intercept) 2.47014 14.81716 0.167 0.868

speed 0.91329 2.03422 0.449 0.656

speed\_squared 0.09996 0.06597 1.515 0.136

Residual standard error: 15.18 on 47 degrees of freedom

Multiple R-squared: 0.6673, Adjusted R-squared: 0.6532

F-statistic: 47.14 on 2 and 47 DF, p-value: 5.852e-12

# Extract the estimated reaction\_time (coefficient of 'speed') and k (coefficient of 'speed\_squared')

Command: reaction\_time <- coef(model)["speed"]

k <- coef(model)["speed\_squared"]

The **reaction time** is the coefficient of speed in the model.

The **k** value is the coefficient of speed\_squared.

# Print the estimated reaction time and k

Command: cat("Estimated Reaction Time: ", reaction\_time, " seconds\n")

Result: Estimated reaction time: 0.9132876 seconds

Command: cat("Estimated k (braking distance proportionality constant): ", k, "\n")

# Now, let's plot the data points and the fitted relationship

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

geom\_point() + # Scatter plot of the data points

geom\_smooth(method = "lm", formula = y ~ x + I(x^2), se = FALSE, color = "blue") + #

Fitted line

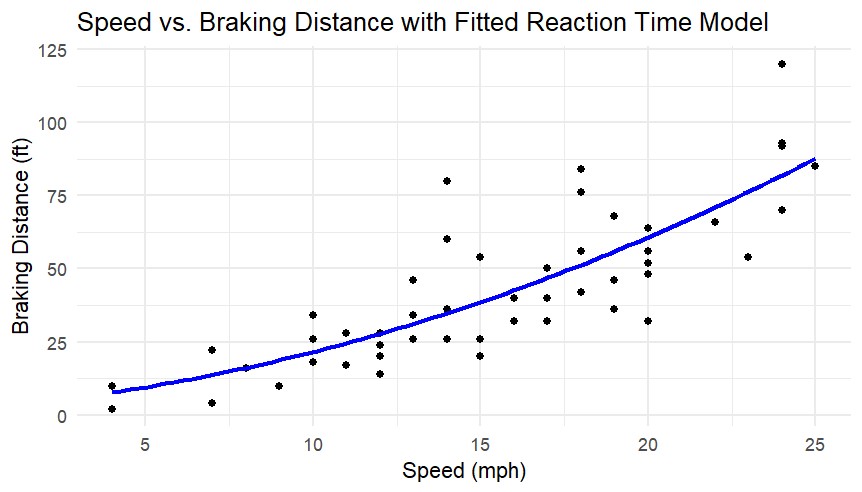
labs(title = "Speed vs. Braking Distance with Fitted Reaction Time Model",

x = "Speed (mph)",

y = "Braking Distance (ft)") +

theme\_minimal()

I used **geom\_point()** to plot the data points and **geom\_smooth()** to plot the fitted line, which is based on the model we just built.



**RNA-seq assessment (8 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)**

* LMS\_RNAseq\_short-master-2023-final/course/exercises/data/exercise1\_counts.csv

Command: count\_data <- read.csv("C:/Users/ContinoR/Downloads/LMS\_RNAseq\_short-

master-2023-final/LMS\_RNAseq\_short-master-2023-

final/course/exercises/data/exercise1\_counts.csv", row.names = 1)

Command: head(count\_data)

Result:

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

497097 16 16 0 21

100503874 20 0 0 64

100038431 0 0 2 0

19888 11 0 10 113

20671 14 16 0 40

27395 465 193 596 436

mutant\_KOa2.bam mutant\_KOa3.bam mutant\_KOb1.bam mutant\_KOb2.bam

497097 16 27 20 0

100503874 0 4 5 0

100038431 0 8 0 0

19888 0 26 14 6

20671 8 33 33 12

27395 686 572 1378 1901

mutant\_KOb3.bam

497097 2

100503874 2

100038431 0

19888 16

20671 24

27395 1553

* LMS\_RNAseq\_short-master-2023-final/course/exercises/data /exercise1\_sample\_description.info

I changed the sample\_description.info file into excel (csv)

Command: sample\_description <-

read.csv("C:/Users/ContinoR/Downloads/LMS\_RNAseq\_short-master-2023-

final/LMS\_RNAseq\_short-master-2023-

final/course/exercises/data/exercise1\_sample\_description.csv")

Command: head(sample\_description)

Result:

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

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

# Create col\_data by selecting relevant columns from sample\_description

Command: col\_data <- sample\_description[c("filename", "sample", "condition", "batch")]

# Check the dimensions of col\_data

Command: dim(col\_data)

Result: [1] 9 4

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

# Check if count\_data is a matrix or data frame with genes as rows and samples as columns

Command: dim(count\_data)

Result: [1] 26301 9

# Check that col\_data has the correct columns and row names match the samples in count\_data

Command: dim(col\_data)

Result: [1] 9 4

Command: head(col\_data)

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

Command: library(DESeq2)

# Construct DESeqDataSet object

Command: dds <- DESeqDataSetFromMatrix(countData = count\_data,

colData = col\_data,

design = ~ condition)

# Check the DESeqDataSet object

Command: dds

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

# Check the dimensions of dds

Command: dim(dds)

Result: [1] 26301 9

3.11. Perform rlog and VST transformation on the data. **(1 pts)**

# Load DESeq2 package

Command: library(DESeq2)

# Perform rlog transformation

Command: rlog\_data <- rlog(dds, blind = TRUE)

# Check the transformed data (rlog transformed counts)

Command: head(assay(rlog\_data))

Result:

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

497097 3.8561447 3.9907973 3.5322792 3.923806

100503874 3.2895831 2.8326071 2.8201166 3.834407

100038431 0.1543662 0.1551115 0.1903975 0.154316

19888 3.9103723 3.6262932 3.8451124 5.046242

20671 4.2033204 4.3750856 3.9098102 4.550450

27395 9.2090107 8.7918710 9.2758269 9.106490

mutant\_KOa2.bam mutant\_KOa3.bam mutant\_KOb1.bam mutant\_KOb2.bam

497097 3.8489878 3.9980385 3.7603473 3.5253500

100503874 2.8226210 2.9321205 2.8973928 2.8131612

100038431 0.1543294 0.3042526 0.1537077 0.1536189

19888 3.6068923 4.1745338 3.8179714 3.6830152

20671 4.0867366 4.4566664 4.2622207 4.0354695

27395 9.5797251 9.3512412 9.6722305 9.8827325

mutant\_KOb3.bam

497097 3.5590907

100503874 2.8554608

100038431 0.1538331

19888 3.8816181

20671 4.2166975

27395 9.9615722

# View the dimensions of the rlog-transformed data

Command: dim(rlog\_data)

Result: [1] 26301 9

# Perform VST transformation

Command: vst\_data <- vst(dds, blind = TRUE)

# Check the transformed data (VST transformed counts)

Command: head(assay(vst\_data))

Result:

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

497097 7.139487 7.303577 6.476658 7.220155

100503874 7.216136 6.476658 6.476658 7.747532

100038431 6.476658 6.476658 6.692303 6.476658

19888 7.027737 6.476658 6.957081 8.128730

20671 7.097347 7.303577 6.476658 7.493025

27395 9.498159 9.038154 9.573177 9.384314

mutant\_KOa2.bam mutant\_KOa3.bam mutant\_KOb1.bam mutant\_KOb2.bam

497097 7.130573 7.305229 7.017147 6.476658

100503874 6.476658 6.799304 6.748085 6.476658

100038431 6.476658 6.932007 6.476658 6.476658

19888 6.476658 7.290141 6.929650 6.756214

20671 6.941007 7.389990 7.168352 6.871396

27395 9.918705 9.658047 10.024914 10.269679

mutant\_KOb3.bam

497097 6.662121

100503874 6.662121

100038431 6.476658

19888 6.998734

20671 7.114359

27395 10.362465

# View the dimensions of the VST-transformed data

Command: dim(vst\_data)

Result: [1] 26301 9

3.12. Draw a heatmap of count matrix based on the top 40 highly expressed genes using rlog and VST data. **(1 pts)**

# Load necessary libraries

Command: library(DESeq2)

Command: library(pheatmap)

# Perform rlog or VST transformation. I’m going to choose rlog transformation for this assessment

rlog transformation

Command: rlog\_data <- rlog(dds, blind = TRUE)

or

VST transformation (it is possible to switch between rlog and VST)

Command: vst\_data <- vst(dds, blind = TRUE)

# Extract the rlog or VST transformed count matrix

Command: rlog\_matrix <- assay(rlog\_data) # If using rlog

or

Command: vst\_matrix <- assay(vst\_data) # If using VST

#Select the top 40 genes with the highest variance

Command: gene\_variances <- apply(rlog\_matrix, 1, var) # Variance for each gene (rows)

Command: top40\_genes <- order(gene\_variances, decreasing = TRUE)[1:40] # Select top 40 genes

# Extract the data for the top 40 genes

Command: top40\_data <- rlog\_matrix[top40\_genes, ] # If using rlog

or

Command: top40\_data <- vst\_matrix[top40\_genes, ] # If using VST

# Draw a heatmap

Command: pheatmap(top40\_data,

cluster\_rows = TRUE,

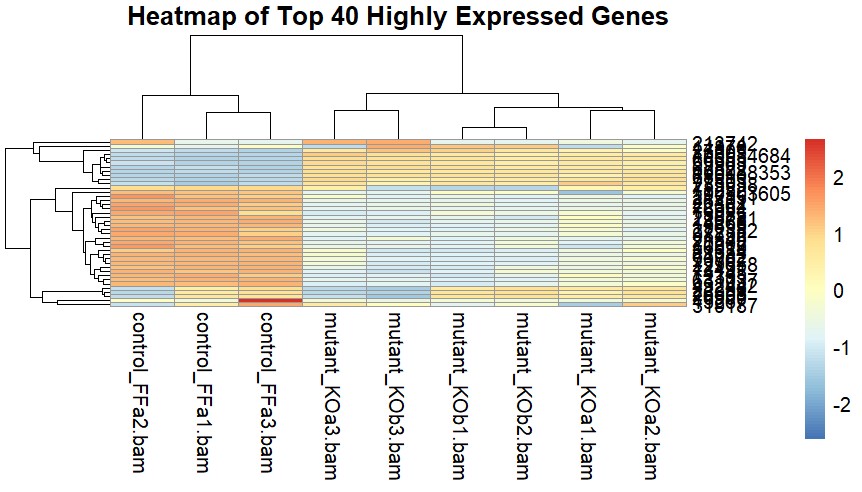
cluster\_cols = TRUE,

scale = "row", # Scale by rows to make gene expression comparable

show\_rownames = TRUE,

show\_colnames = TRUE,

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



3.13. Generate a (Sample Distance Matrix) SDM to see the clustering of count data. **(1 pts)**

Command: install.packages("DESeq2")

Command: install.packages("pheatmap")

Command: library(DESeq2)

Command: library(kohonen)

Command: library(pheatmap)

# Perform rlog transformation

Command: rlog\_data <- rlog(dds, blind = TRUE)

# Extract the rlog transformed count matrix

Command: rlog\_matrix <- assay(rlog\_data) # If using rlog

# Calculate the Euclidean distance between samples (columns)

Command: sample\_distance\_matrix <- dist(t(rlog\_matrix), method = "euclidean") #

`t()` transposes the matrix to make samples the columns

# Visualize the Sample Distance Matrix using a heatmap

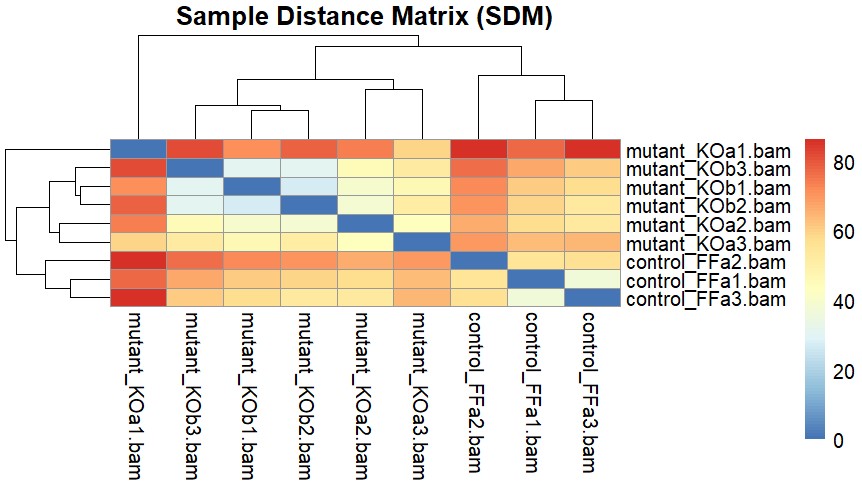
Command: pheatmap(as.matrix(sample\_distance\_matrix),

clustering\_distance\_rows = "euclidean",

clustering\_distance\_cols = "euclidean",

clustering\_method = "complete",

main = "Sample Distance Matrix (SDM)")



3.14. Perform the Principal Component Analysis using rlog method and find out the % significance values of first two principal components. **(1 pts)**

Command: install.packages("DESeq2")

Command: install.packages("ggplot2")

Command: library(DESeq2)

Command: library(ggplot2)

# Perform rlog transformation

Command: rlog\_data <- rlog(dds, blind = TRUE)

# Extract the transformed count matrix

Command: rlog\_matrix <- assay(rlog\_data)

# Perform PCA on the rlog-transformed count matrix (using the transposed matrix)

Command: pca\_result <- prcomp(t(rlog\_matrix)) # Transpose to make samples as rows

# Calculate the proportion of variance explained by each principal component

Command: pca\_variance <- summary(pca\_result)$importance[2,] \* 100

# The second row gives the proportion of variance

# Print the percentage of variance explained by each component

Command: pca\_variance

Result:

PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9

34.982 31.378 12.025 7.832 5.184 3.683 2.784 2.131 0.000

# Create a data frame with PCA results for plotting

Command: pca\_data <- data.frame(PC1 = pca\_result$x[, 1],

PC2 = pca\_result$x[, 2],

sample = colnames(rlog\_matrix)) # Sample labels

# Plot the PCA results using ggplot2

Command: ggplot(pca\_data, aes(x = PC1, y = PC2, label = sample)) +

geom\_point() +

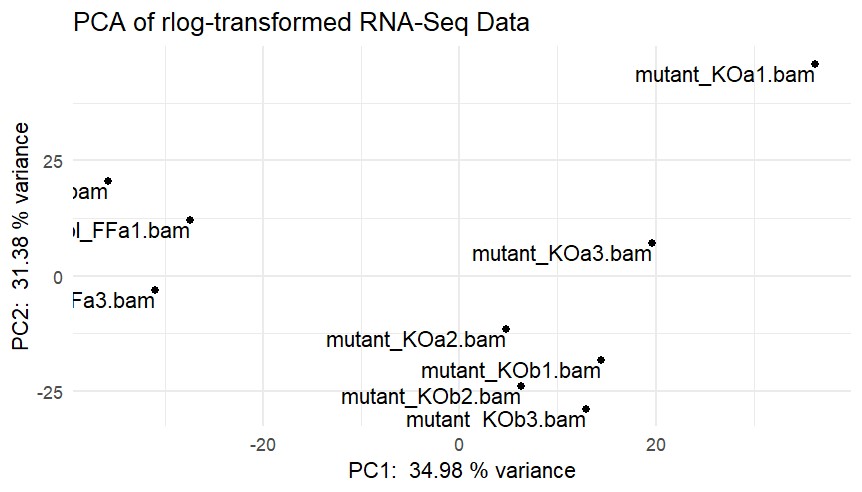
geom\_text(aes(label = sample), vjust = 1, hjust = 1) +

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

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

ggtitle("PCA of rlog-transformed RNA-Seq Data") +

theme\_minimal()



3.15. Repeat the PCA, this time using VST method and compare the plots with the ones obtained using rlog method. **(1 pts)**

Command: install.packages("DESeq2")

Command: install.packages("ggplot2")

Command: library(DESeq2)

Command: library(ggplot2)

# Perform rlog transformation (if not already done)

Command: rlog\_data <- rlog(dds, blind = TRUE)

# Extract the rlog transformed count matrix

Command: rlog\_matrix <- assay(rlog\_data)

# Perform VST transformation

Command: vst\_data <- vst(dds, blind = TRUE)

# Extract the VST transformed count matrix

Command: vst\_matrix <- assay(vst\_data)

# Perform PCA on the rlog-transformed count matrix (transposed matrix to make samples rows)

Command: pca\_rlog <- prcomp(t(rlog\_matrix)) # Transpose to make samples as rows

# Perform PCA on the VST-transformed count matrix (transposed matrix to make samples rows)

Command: pca\_vst <- prcomp(t(vst\_matrix)) # Transpose to make samples as rows

# Calculate variance explained for rlog data

Command: pca\_rlog\_variance <- summary(pca\_rlog)$importance[2,] \* 100

# Percentage of variance explained

# Calculate variance explained for VST data

Command: pca\_vst\_variance <- summary(pca\_vst)$importance[2,] \* 100

# Percentage of variance explained

# Print the variance explained

Command: pca\_rlog\_variance

Result:

PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9

34.982 31.378 12.025 7.832 5.184 3.683 2.784 2.131 0.000

Command: pca\_vst\_variance

Result:

PC1 PC2 PC3 PC4 PC5 PC6 PC7 PC8 PC9

33.718 28.027 12.945 8.587 5.989 4.516 3.440 2.777 0.000

# PCA data for rlog

Command: pca\_rlog\_data <- data.frame(PC1 = pca\_rlog$x[, 1],

PC2 = pca\_rlog$x[, 2],

sample = colnames(rlog\_matrix))

# PCA data for VST

Command: pca\_vst\_data <- data.frame(PC1 = pca\_vst$x[, 1],

PC2 = pca\_vst$x[, 2],

sample = colnames(vst\_matrix))

# Plot the PCA for rlog

Command: ggplot(pca\_rlog\_data, aes(x = PC1, y = PC2, label = sample)) +

geom\_point() +

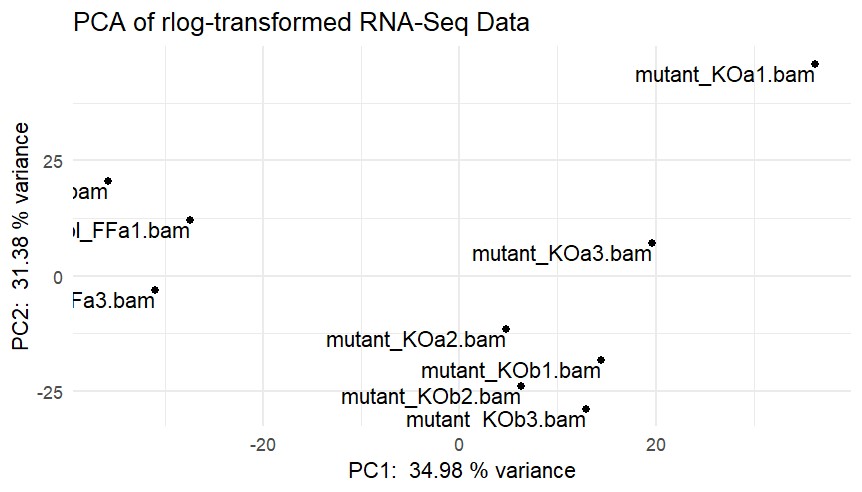
geom\_text(aes(label = sample), vjust = 1, hjust = 1) +

xlab(paste("PC1: ", round(pca\_rlog\_variance[1], 2), "% variance")) +

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

ggtitle("PCA of rlog-transformed RNA-Seq Data") +

theme\_minimal()



# Plot the PCA for VST

Command: ggplot(pca\_vst\_data, aes(x = PC1, y = PC2, label = sample)) +

geom\_point() +

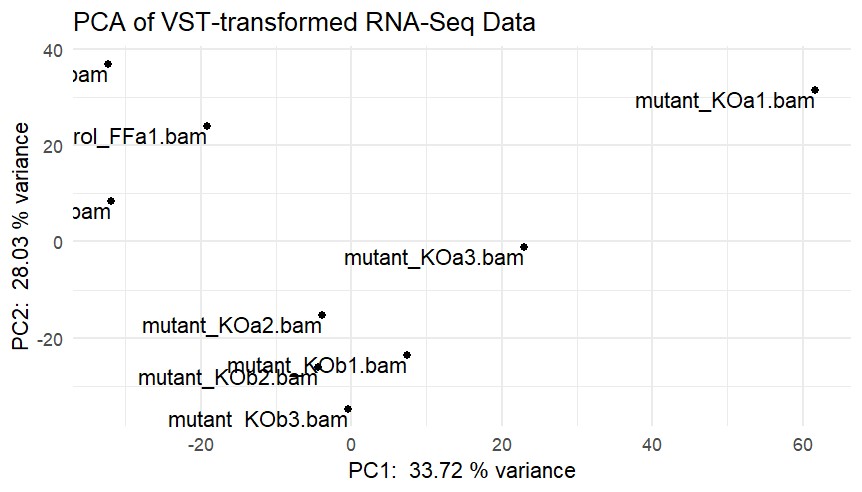
geom\_text(aes(label = sample), vjust = 1, hjust = 1) +

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

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

ggtitle("PCA of VST-transformed RNA-Seq Data") +

theme\_minimal()



**ChIP-seq assessment (4 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).

I selected \*\*Session -> Set Working Directory -> Choose Directory\*\*

Command: setwd("C:/Users/ContinoR/Downloads/LMS\_ChIPseq\_short-master-2023-final")

Then I installed “GenomicRanges” and loaded the required libraries

Command: BiocManager::install("GenomicRanges")

Command: library(GenomicRanges)

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:

* + LMS\_ChIPseq\_short-master-2023-final/course/data /MacsPeaks/mycmelrep1\_peaks.xls
  + LMS\_ChIPseq\_short-master-2023-final/course/data /MacsPeaks/mycmelrep2\_peaks.xls

# Define the file paths

Command: rep1 <- "C:/Users/ContinoR/Downloads/LMS\_ChIPseq\_short-master-2023-

final/LMS\_ChIPseq\_short-master-2023-

final/course/data/MacsPeaks/mycmelrep1\_peaks.xls"

Command: file.exists(rep1)

Command: rep2 <- "C:/Users/ContinoR/Downloads/LMS\_ChIPseq\_unzipped/LMS\_ChIPseq\_short-

master-2023-final/course/data/MacsPeaks/mycmelrep2\_peaks.xls"

Command: file.exists(rep2)

# Read in the data from the .xls files

Command: macsPeaks\_DF <- read.delim(rep1,comment.char="#")

Command: macsPeaks\_DF[1:2,]

Result:

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

fold\_enrichment X.log10.qvalue. name

1 4.61726 3.6137 mycmelrep1\_peak\_1

2 5.56316 5.3628 mycmelrep1\_peak\_2

Command: macsPeaks\_DF <- read.delim(rep2,comment.char="#")

Command: macsPeaks\_DF[1:2,]

Result:

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

fold\_enrichment X.log10.qvalue. name

1 5.79027 5.81302 mycmelrep2\_peak\_1

2 11.70799 24.44043 mycmelrep2\_peak\_2

Now that I have the information in a table, I can create a GRanges object.

Command: library(GenomicRanges)

Command: macsPeaks\_GR1 <- GRanges(

seqnames=macsPeaks\_DF[,"chr"],

IRanges(macsPeaks\_DF[,"start"],

macsPeaks\_DF[,"end"]

)

)

Command: macsPeaks\_GR1

GRanges object with 52933 ranges and 0 metadata columns:

seqnames ranges strand

<Rle> <IRanges> <Rle>

[1] 1 3049670-3049833 \*

[2] 1 3435991-3436154 \*

[3] 1 4774935-4775285 \*

[4] 1 4775337-4775959 \*

[5] 1 4847544-4847931 \*

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

[52929] Y 1913013-1913289 \*

[52930] Y 1934470-1934640 \*

[52931] Y 1964602-1964765 \*

[52932] Y 2555745-2555908 \*

[52933] Y 2890951-2891338 \*

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

Command: macsPeaks\_GR2 <- GRanges(

seqnames=macsPeaks\_DF[,"chr"],

IRanges(macsPeaks\_DF[,"start"],

macsPeaks\_DF[,"end"]

)

)

Command:macsPeaks\_GR2

Result:

GRanges object with 41443 ranges and 0 metadata columns:

seqnames ranges strand

<Rle> <IRanges> <Rle>

[1] 1 4506720-4506910 \*

[2] 1 4775356-4776102 \*

[3] 1 4797708-4797962 \*

[4] 1 4847097-4848049 \*

[5] 1 4848406-4848603 \*

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

[41439] Y 581955-582358 \*

[41440] Y 622263-622682 \*

[41441] Y 622802-623337 \*

[41442] Y 1903412-1903639 \*

[41443] Y 1905483-1905649 \*

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

# Find the overlaps between the two GRanges objects

Command:overlaps <- findOverlaps(macsPeaks\_GR1, macsPeaks\_GR2)

# Create a common peakset by extracting the ranges that overlap

Command:common\_peaks <- macsPeaks\_GR1[queryHits(overlaps)]

# View the common peaks (GRanges object)

Command:common\_peaks

Result

GRanges object with 32484 ranges and 0 metadata columns:

seqnames ranges strand

<Rle> <IRanges> <Rle>

[1] chr4 45966312-45966511 \*

[2] chr9 21156572-21156771 \*

[3] chr9 21156572-21156771 \*

[4] chr12 114345800-114345999 \*

[5] chr3 87847243-87847442 \*

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

[32480] chr2 98502752-98502951 \*

[32481] chr2 98506840-98507039 \*

[32482] chr5 115372314-115372513 \*

[32483] chr9 35112889-35113088 \*

[32484] chrM 16016-16215 \*

-------

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. **(1 pts)**

Command: library(GenomicRanges)

# Add fold enrichment from both replicates

Command:mcols(common\_peaks)$fold\_enrichment1 <-

macsPeaks\_DF1$fold\_enrichment[queryHits(overlaps)]

Command:mcols(common\_peaks)$fold\_enrichment2 <-

macsPeaks\_DF2$fold\_enrichment[subjectHits(overlaps)]

# View updated common\_peaks

Command:common\_peaks

Result:

GRanges object with 32484 ranges and 2 metadata columns:

seqnames ranges strand | fold\_enrichment1

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

[1] chr4 45966312-45966511 \* | 9.52390

[2] chr9 21156572-21156771 \* | 7.45675

[3] chr9 21156572-21156771 \* | 10.20078

[4] chr12 114345800-114345999 \* | 4.00702

[5] chr3 87847243-87847442 \* | 19.01092

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

[32480] chr2 98502752-98502951 \* | 6.23072

[32481] chr2 98506840-98507039 \* | 17.31472

[32482] chr5 115372314-115372513 \* | 18.62359

[32483] chr9 35112889-35113088 \* | 10.23618

[32484] chrM 16016-16215 \* | 14.59007

fold\_enrichment2

<numeric>

[1] 11.70799

[2] 5.81891

[3] 5.97823

[4] 4.70157

[5] 9.72956

... ...

[32480] 8.15374

[32481] 5.64884

[32482] 8.34254

[32483] 4.57042

[32484] 5.85560

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

Option 1:

#Rank by the First Replicate (fold\_enrichment1)

Command:common\_peaks\_sorted <- common\_peaks[order(-common\_peaks$fold\_enrichment1)]

#Select \*\*top 500 peaks\*\*

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

#Resize peaks to \*\*200 bp centered around the summit\*\*

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

#Print top peaks

Command:head(top500\_resized\_peaks)

Option 2:

#Rank by the Second Replicate (fold\_enrichment2)

Command:common\_peaks\_sorted <- common\_peaks[order(-common\_peaks$fold\_enrichment2)]

#Select \*\*top 500 peaks\*\*

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

#Resize peaks to \*\*200 bp centered around the summit\*\*

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

#Print top peaks

Command:head(top500\_resized\_peaks)

Result:

GRanges object with 6 ranges and 2 metadata columns:

seqnames ranges strand | fold\_enrichment1 fold\_enrichment2

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

[1] chr14 19164192-19164391 \* | 123.1159 48.21738

[2] chr9 90021195-90021394 \* | 111.5354 8.54992

[3] chr9 108244869-108245068 \* | 111.5354 10.47657

[4] chr7 52716228-52716427 \* | 104.7455 15.12912

[5] chr6 118087171-118087370 \* | 96.9533 26.89088

[6] chr10 93393085-93393284 \* | 96.8103 22.48929

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

I will use the rep1/fold enrichment1 for task 3.18, therefore I will remove fold enrichment2 column

#Remove fold enrichment2 from common\_peaks

Command: mcols(common\_peaks)$fold\_enrichment2 <- NULL

#Rank by the First Replicate (fold\_enrichment1)

Command: common\_peaks\_sorted <- common\_peaks[order(-common\_peaks$fold\_enrichment1)]

#Select \*\*top 500 peaks\*\*

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

#Resize peaks to \*\*200 bp centered around the summit\*\*

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

#Print top peaks

Command: head(top500\_resized\_peaks)

Result:

GRanges object with 6 ranges and 1 metadata column:

seqnames ranges strand | fold\_enrichment1

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

[1] chr14 19164192-19164391 \* | 123.1159

[2] chr9 90021195-90021394 \* | 111.5354

[3] chr9 108244869-108245068 \* | 111.5354

[4] chr7 52716228-52716427 \* | 104.7455

[5] chr6 118087171-118087370 \* | 96.9533

[6] chr10 93393085-93393284 \* | 96.8103

-------

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. **(1 pts)**

Command:BiocManager::install("Biostrings")

Command:BiocManager::install("BSgenome")

Command:library(GenomicRanges)

Command:library(BSgenome)

Command:library(BSgenome.Mmusculus.UCSC.mm9)

Command:library(Biostrings)

#Define output FASTA file path

Command:output\_fasta <- "C:/Users/ContinoR/Downloads/top500\_peaks\_sequences.fasta"

# Load mm9 genome

Command:genome <- BSgenome.Mmusculus.UCSC.mm9

Command:seqlevelsStyle(common\_peaks\_sorted) <- "UCSC"

#Provide a peak set resized to 200bp

Command:commonPeaks <- resize(common\_peaks\_sorted,200,fix="center")

Command:commonPeaks[1:4,]

Result:

GRanges object with 4 ranges and 1 metadata column:

seqnames ranges strand | fold\_enrichment1

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

[1] chr14 19164192-19164391 \* | 123.116

[2] chr9 90021195-90021394 \* | 111.535

[3] chr9 108244869-108245068 \* | 111.535

[4] chr7 52716228-52716427 \* | 104.745

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

Below the result if I use both fold enrichments

GRanges object with 4 ranges and 2 metadata columns:

seqnames ranges strand | fold\_enrichment1 fold\_enrichment2

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

[1] chr14 19164192-19164391 \* | 123.116 48.21738

[2] chr9 90021195-90021394 \* | 111.535 8.54992

[3] chr9 108244869-108245068 \* | 111.535 10.47657

[4] chr7 52716228-52716427 \* | 104.745 15.12912

-------

seqinfo: 22 sequences from an unspecified genome; no seqlengths

#Extract sequences from the common peaks

Command:commonPeaksSequences <- getSeq(genome,GRanges(commonPeaks))

Command:names(commonPeaksSequences) <- paste0("peak\_",seqnames(commonPeaks),"\_",

start(commonPeaks),

"-",

end(commonPeaks))

Command:commonPeaksSequences[1:2,]

Result:

DNAStringSet object of length 2:

width seq names

[1] 200 CCCCCGCACCTCTCTTTGTGTC...CCTCTCCGCCCCGCCCCGCGCC peak\_chr14\_191641...

[2] 200 AGTCATTTCTGTCCTTACAACC...ACCCTGGGATTAAGAGTCCCAT peak\_chr9\_9002119...

# Rename Sequences with (ID1), (ID2)...

Command:names(sequences) <- paste0("(ID", seq\_along(sequences), ")")

# Write sequences to FASTA format

Command:writeXStringSet(sequences, filepath = output\_fasta, format = "fasta")

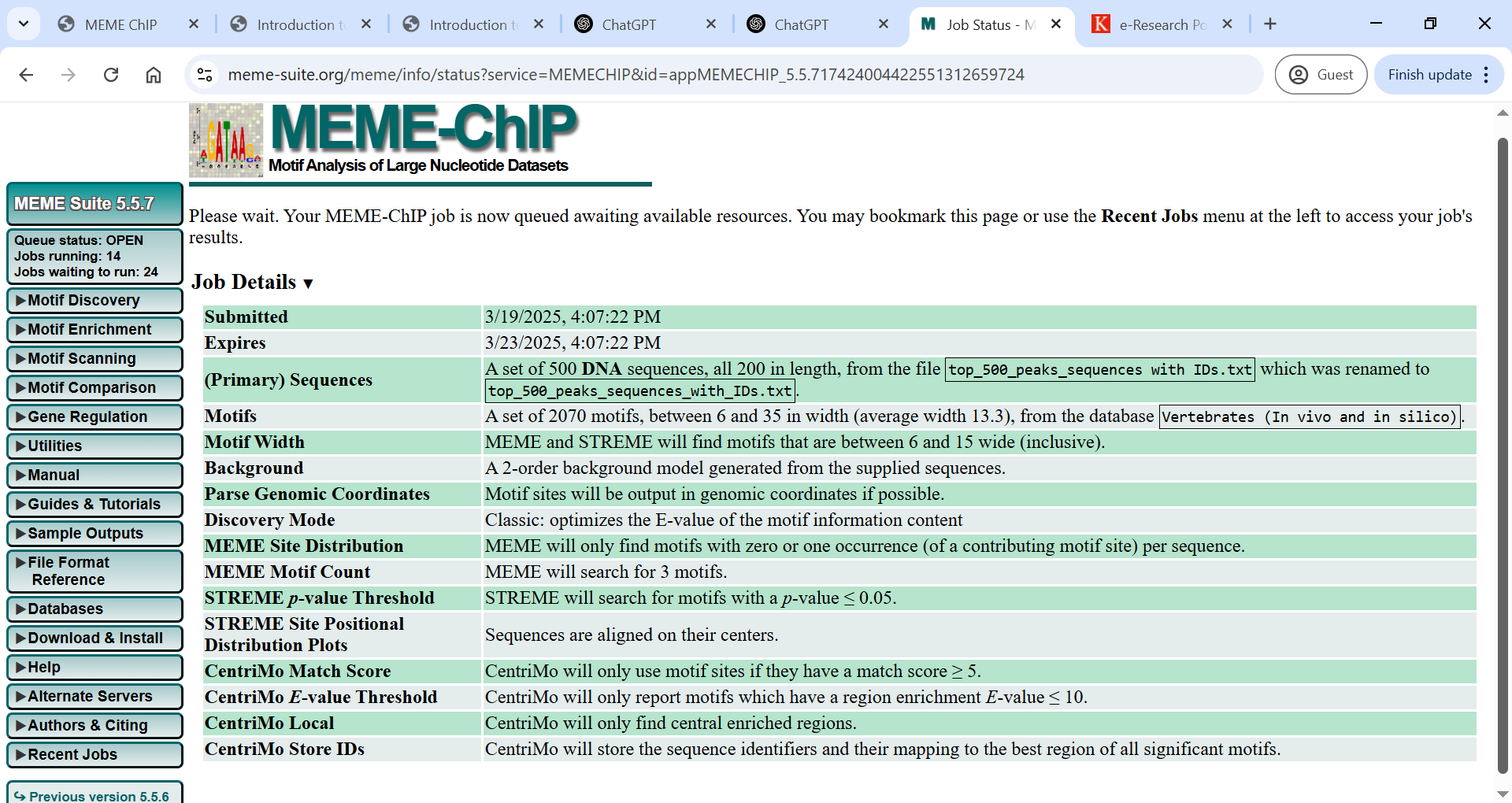


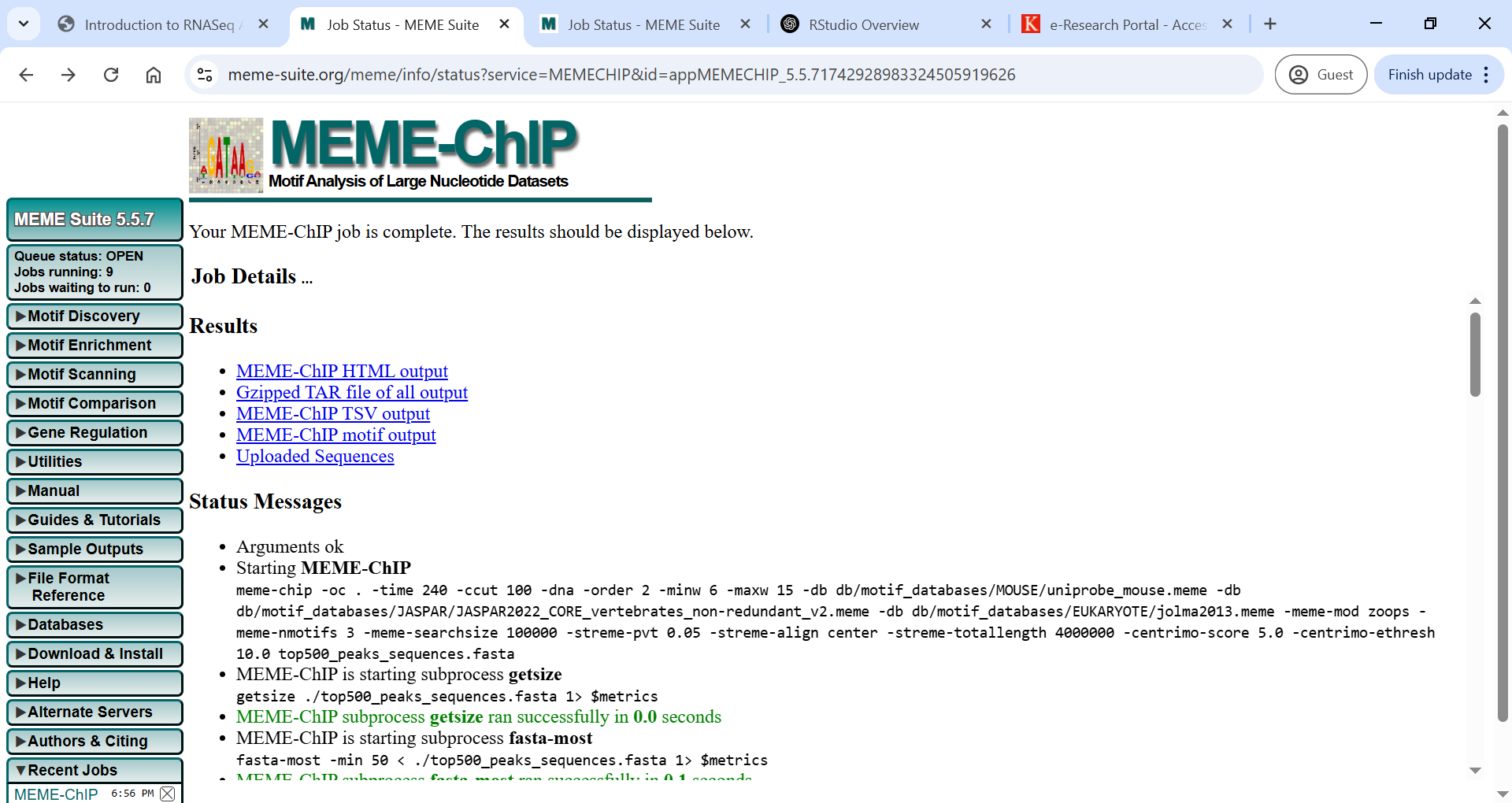
3.19. Upload the sequences to Meme-ChIP and report the results when complete. **(1 pts)**

I uploaded the fasta file with the IDs in the following website:

<https://meme-suite.org/meme/tools/meme-chip>

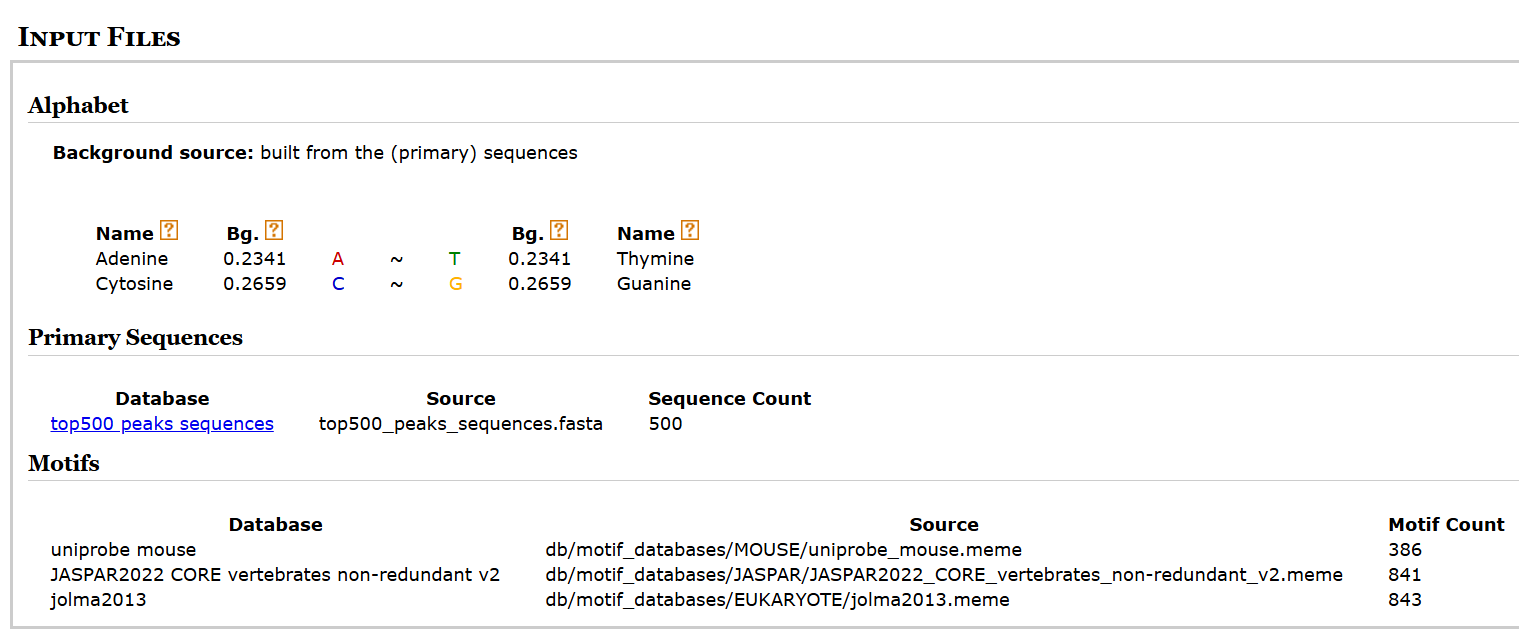
Result:





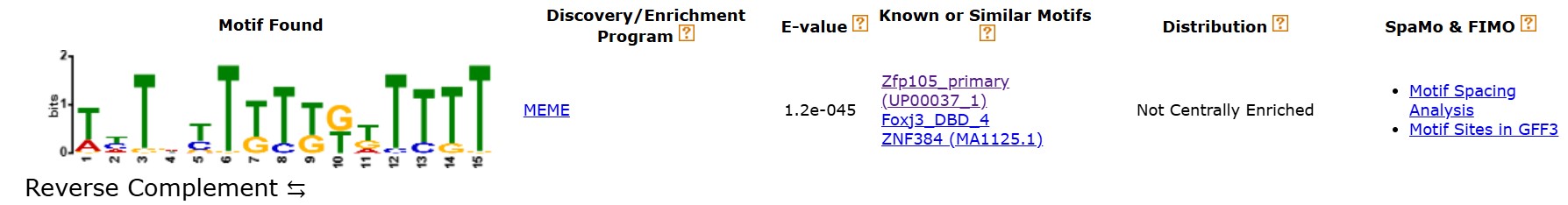
* [MEME-ChIP HTML output](https://meme-suite.org/meme/opal-jobs/appMEMECHIP_5.5.71742928983324505919626/meme-chip.html)

<https://meme-suite.org/meme/opal-jobs/appMEMECHIP_5.5.71742928983324505919626/meme-chip.html>



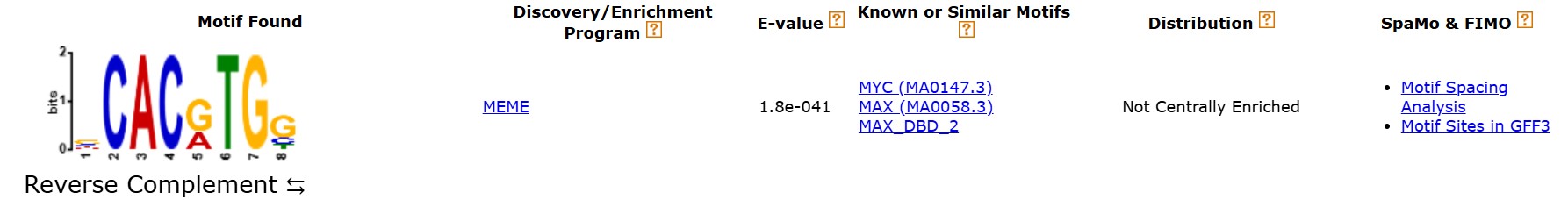
**Motif 1:**

Using MEME-ChIP, a 15 base pairs motif was discovered with high confidence (E-value: 1.2e-45). The motif is not centrally enriched, indicating it may be distributed along the full length of peak sequences. Comparison with known motifs identified potential similarities with transcription factors such as Zfp105, Foxj3, and ZNF384, suggesting possible regulatory roles for these factors in the experimental context.



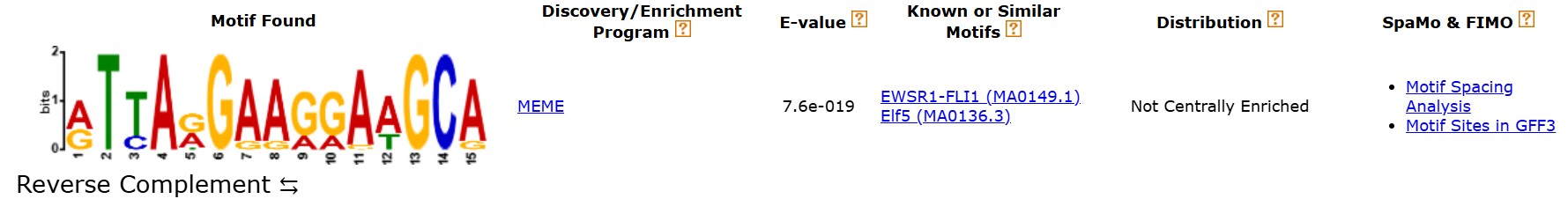
**Motif 2:**

A second motif identified by MEME-ChIP revealed a 8 bp consensus sequence CACGTGG, a classic E-box motif. This sequence is significantly associated with transcription factors such as MYC, MAX, and MAX\_DBD\_2 (E-value: 1.8e-41). These proteins are known regulators of gene expression related to cellular proliferation and differentiation. Though the motif is not centrally enriched in the peak sequences, its recurrence may suggest widespread regulatory influence.



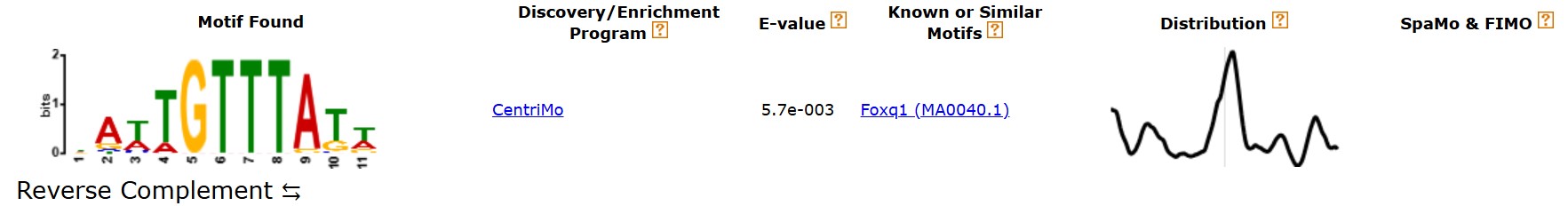
**Motif 3:**

MEME analysis also identified a third motif with a core consensus of GAAGGAA, strongly resembling EWSR1-FLI1 and Elf5 ETS-family transcription factor binding motifs (E-value: 7.6e-019). Though not centrally enriched, the GGAA core is a hallmark of ETS binding sites, often implicated in transcriptional regulation in developmental, immune, and oncogenic pathways. The match to EWSR1-FLI1 may suggest potential roles in cancer-related regulatory networks depending on the experimental context.



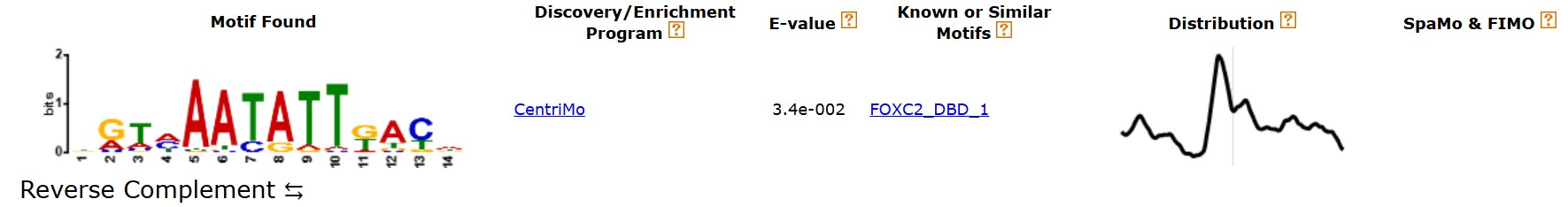
**Motif 4:**

Using CentriMo, a centrally enriched motif has been identified matching the Forkhead transcription factor Foxq1 (MA0040.1), with an E-value of 5.7e-3. The core sequence TGTTTAT is characteristic of Forkhead binding domains. The central localization of this motif within peak regions suggests a potential direct binding event, highlighting Foxq1 as a potential regulator of the genomic regions analysed.



**Motif 5:**

CentriMo identified a centrally enriched motif with the core sequence AATATT, corresponding to the FOXC2\_DBD\_1 motif. The E-value of 3.4e-2 suggests modest statistical support, while central enrichment supports a biologically relevant interaction. FOXC2 is a Forkhead transcription factor involved in vascular and developmental regulation, implicating this TF in the control of genomic loci represented by the ChIP-seq peaks.



* [Gzipped TAR file of all output](https://meme-suite.org/meme/opal-jobs/appMEMECHIP_5.5.71742928983324505919626/appMEMECHIP_5.5.71742928983324505919626.tar.gz)
* [MEME-ChIP TSV output](https://meme-suite.org/meme/opal-jobs/appMEMECHIP_5.5.71742928983324505919626/summary.tsv)
* [MEME-ChIP motif output](https://meme-suite.org/meme/opal-jobs/appMEMECHIP_5.5.71742928983324505919626/combined.meme)

<https://meme-suite.org/meme/opal-jobs/appMEMECHIP_5.5.71742928983324505919626/combined.meme>