Project 2: RNA-Seq analysis

Fereshteh Noroozi

Spring 2023

Objective

My selected codes in this project are SRR17172481, SRR17172486.

To download these files:

```
prefetch SRR17172481
prefetch SRR17172486
```

In the next step, I went to NCBI SRA database search page (https://www.ncbi.nlm.nih.gov/sra/), and checked both files.

Therefore, both files are single-end sequencing data.

Data preparation

Each student is supposed to preprocess paired data of normal and covid19 lung tissue samples. To do this project, you need to download fastq files from SRA using sratoolkit using the *fastq-dump* (Hint: use *fastq-dump [options] file.fastq.gz*)

Answer:

Due to both files are single-end sequencing to convert the SRA files to fastq format, I used the fastq-dump command from the SRA toolkit

```
fastq-dump /home/Fereshteh/Project3-algorithim/SRR17172481.sra
fastq-dump /home/Fereshteh/Project3-algorithim/SRR17172486.sra
```

Part a- Quality control and trimming

In this step, first, assess the read qualities using the *FastQC* software. Then use the *Trimmomatic* software to improve the read qualities through the read trimming. Recheck the read qualities to make sure the problems are solved. (Hint: you may use *fastqc* and *TrimmomaticPE*)

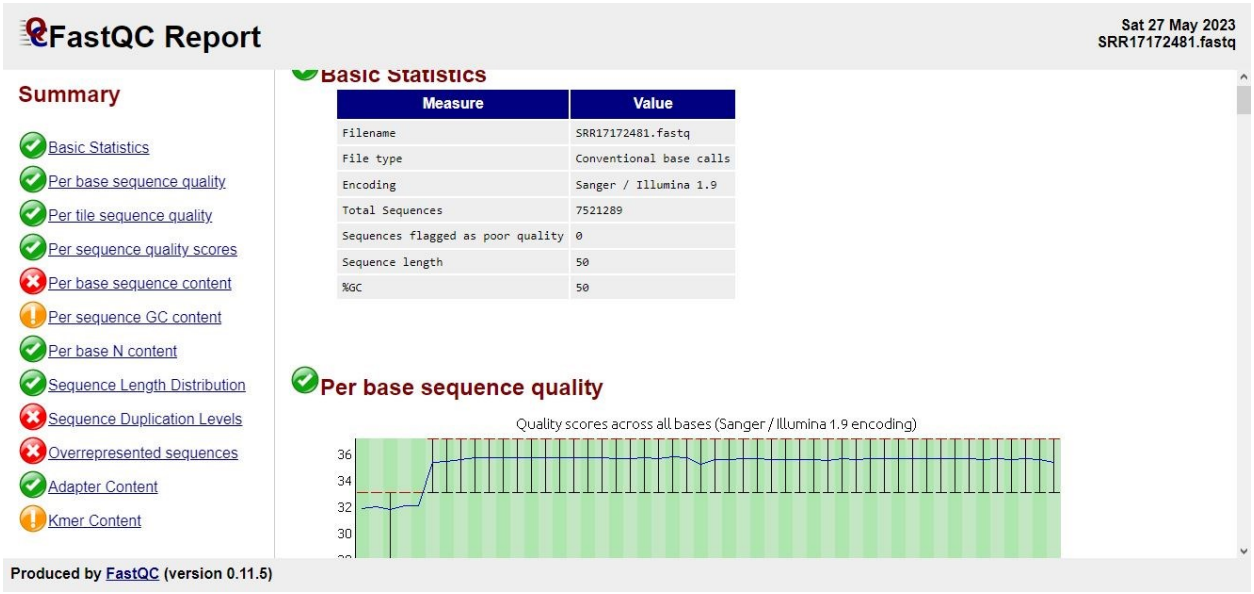
Answer:

SRR17172481

To run FastQC on the FASTQ files for SRR17172481:

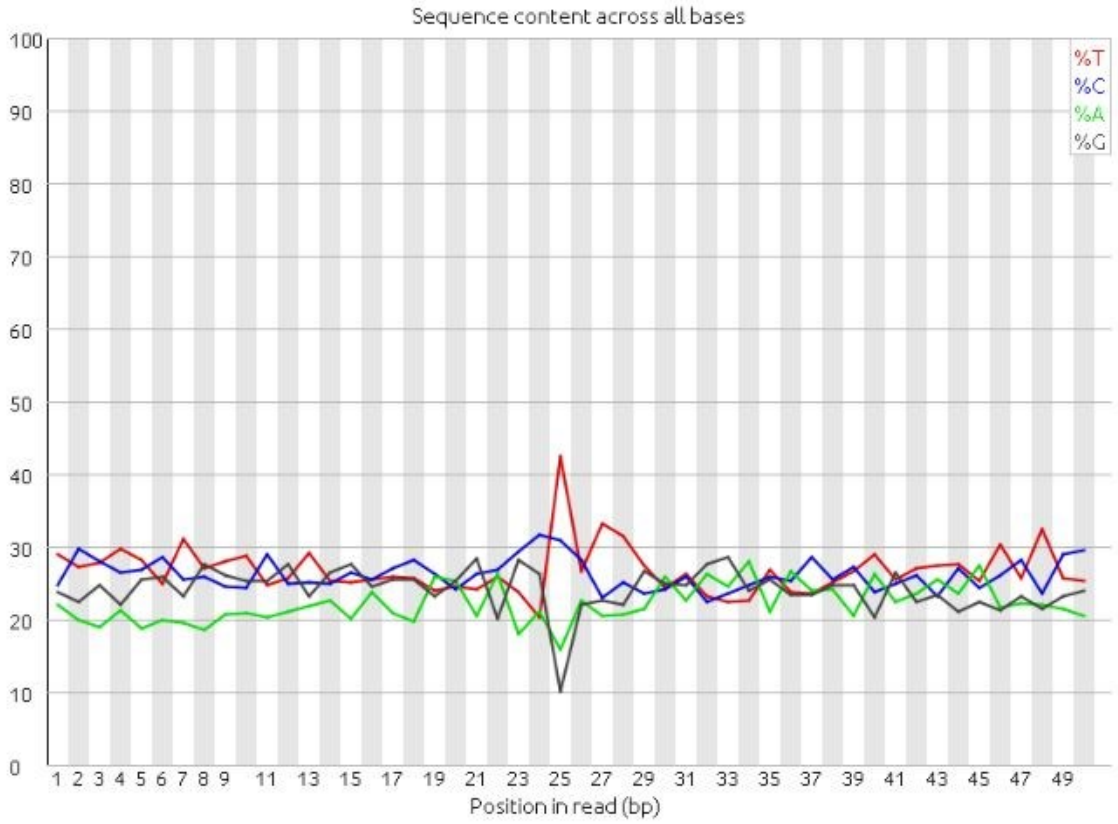
```
fastqc /home/Fereshteh/Project3-algorithim/SRR17172481.fastq -o /ho
me/Fereshteh/Project3-algorithm
```

The result:

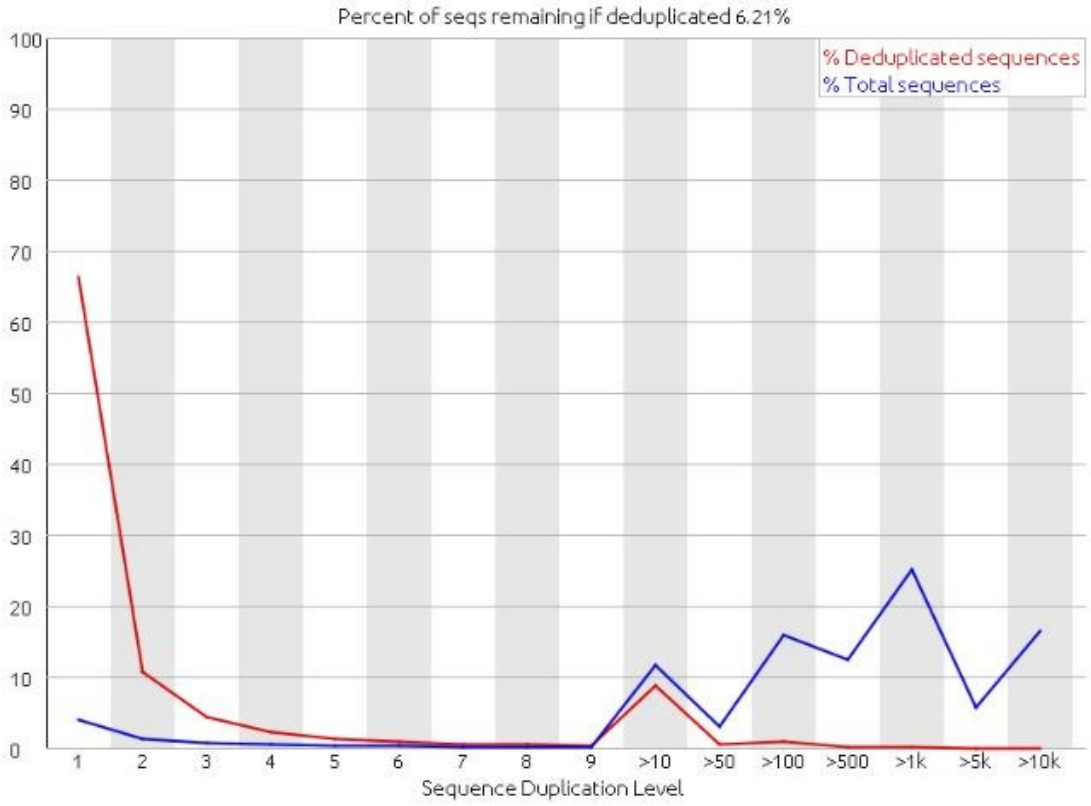


As it is clear in summary part, there are red icons for "Per base sequence content", "Sequence Duplication Levels", and "Overrepresented sequences", it suggests that the sequencing data may have some quality issues that need to be addressed before downstream analysis. Before set trimming, options must look at each errors carefully.

Per base sequence content



Sequence Duplication Levels



Overrepresented sequences

Sequence	Count	Percentage	Possible Source
GTTCCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATCT	150335	1.9987930260358298	No Hit
TCCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATCTCG	130928	1.7407654459228996	No Hit
ACTTTTGTGGAATCTGCCAGGAGGTATAGACGAGGCAGCAGAGCTCTTTG	72381	0.962348342152522	No Hit
GACAGCTGAACCCTCGTGGAGCCATTCATACAAGTCCCTATTTAAGGAAC	63387	0.8427677755767662	No Hit
ACCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATCTCG	46736	0.6213828507321019	No Hit
CTCTTGCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATAT	46011	0.6117435455544921	No Hit
CTCACGCCTTCCAGAGCCACATCATCGCGGTCGAAATAGAAGCCCAGAGA	40833	0.5428989631963351	No Hit
TTGTTGCTATCAGTCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCA	34824	0.46300574276563494	No Hit
GTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATCTCGTATGC	33738	0.44856672838924283	No Hit
TCATTTTGGTAGGATCTTCGTGGCTGTCTTGGTAGCAGCTTTTTAGGTGA	28490	0.378791454496696	No Hit
CCAGGATGCCATCGAACTTGGCTGCGATGAAGGTGATGCCTGGCTGCTTG	26826	0.3566675871649128	No Hit
CCTTGCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATC	26705	0.3550588203697531	No Hit
TATTTTCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATAT	25667	0.3412579944740855	No Hit
TGGTGCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATC	24828	0.33010299165475493	No Hit
GCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATCTCGT	21318	0.2834354590017748 6	No Hit
AGTGCTTCATAGTAGAGGGTCACATTGACAAGCGGTGCATTGGACTTCTT	21279	0.2829169308611862	No Hit
CCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATCTCGTA	19670	0.26152432116356655	No Hit
ACCTAGAGCAGTTACAGGACAATCTCCCATGTGCTGCTCAGTCCCAGTGC	19656	0.26133818285668853	No Hit
GTGAGCACAGTTACCTCTGGAGGTACATTGGTGATCGGAGTATAGTTGGA	19307	0.25669802077808734	No Hit
CGGCCATCACTTGAAGCAAACACTCCTGAGGGAGCTGGTTGAGGAAGGGC	19219	0.2555280085634258	No Hit
TCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACTCAGTACATATCTCGT	19167	0.25483663770930753	No Hit
AAGCCAGATTCGGGCGCTCCCATCTTGCGCAAGTTGGTCACGTGGTCACC	17971	0.23893510806458843	No Hit
AGGCCCAGGAGCAGGAGCAACATCAGTACTGGGATGCTGCCCCATGGGAT	17960	0.23878885653775572	No Hit
GGTTTGAAGCAAATGCTCAGAGGGCCTGGTAGGCGGATGGGCGTCTCCGG	17419	0.23159594053625648	No Hit
CCATGATGTCCTTCTCGCTATTGGAGACGCTGTAGGAATTGTATCCGTAG	17006	0.22610486048335598	No Hit

Therefore, I used trimming on windows:

```
cd C:\Users\Fereshteh\Downloads\Trimmomatic-0.39
```

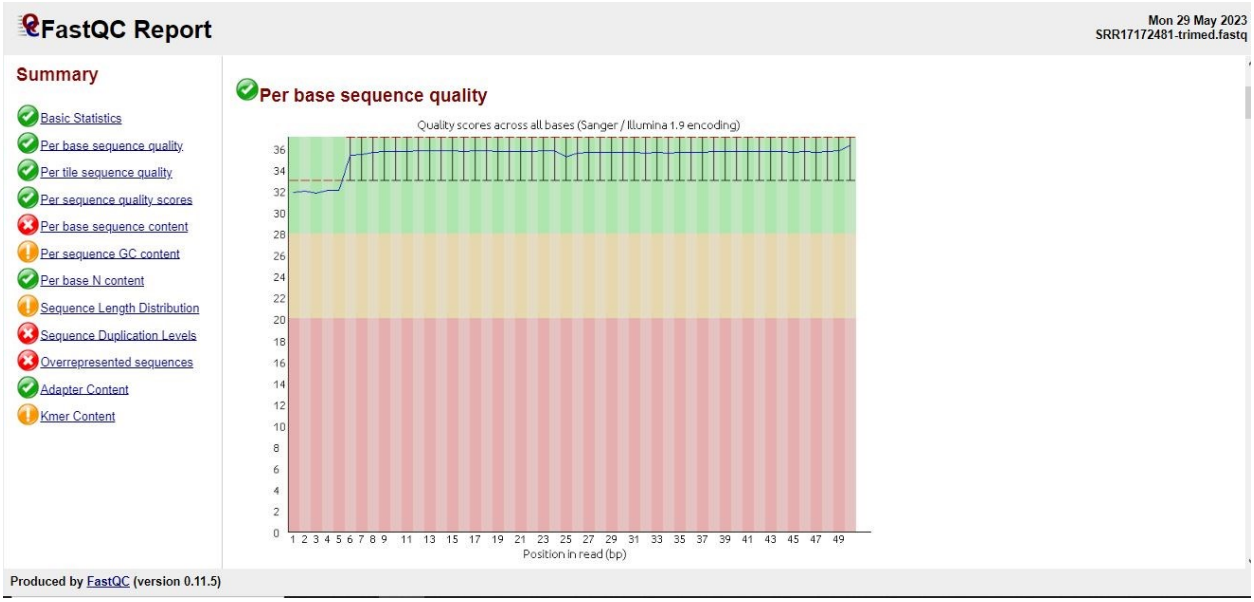
```
java -jar C:\Users\Fereshteh\Downloads\Trimmomatic-0.39\trimmomatic
-0.39.jar SE -phred33 E:\PhD-CLASSES\algorithm\project3\part-A\SRR1
7172481.fastq E:\PhD-CLASSES\algorithm\project3\part-A\SRR17172481-
trimmed.fastq  LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

"SE" specifies that the input file contain single-end reads. "-phred33" specifies the quality encoding used in the input file. "LEADING: 3" and "TRAILING: 3" specify that any bases with a quality score lower than three at the beginning or end of a read should be trimmed. "SLIDINGWINDOW: 4:15" specifies that a sliding window should be used to trim bases with an average quality score of less than 15 over a window of four bases. "MINLEN: 36" specifies that reads shorter than 36 bases after trimming should be discarded.

FastQC after trimming:

```
fastqc /home/Fereshteh/Project3-algorithim/SRR17172481-trimed.fastq
-o /home/Fereshteh/Project3-algorithim
```

The result:



Overall quality has improved but there may still be specific regions of the data that flagged as problematic.

Note:

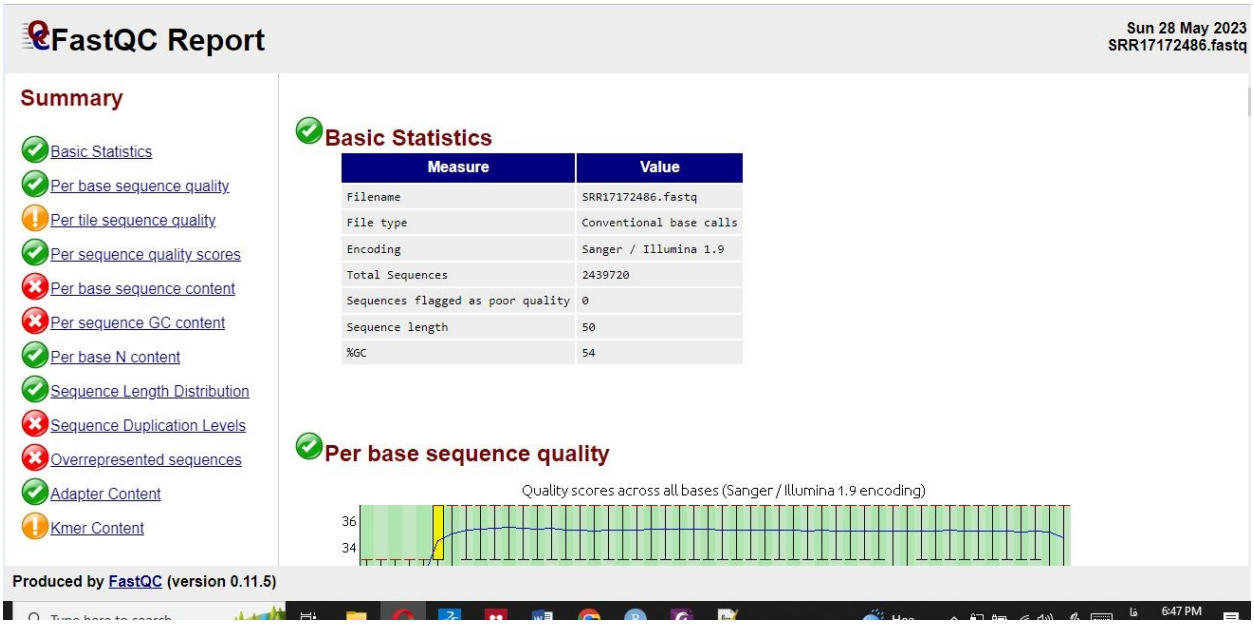
A T percentage of 42% at position 26 suggests that there may be a bias towards T's at that position. This bias could be due to several factors, such as PCR amplification bias, sequencing errors, or a specific genomic feature that is enriched in T's. Also, If the level of sequence duplication is high (e.g., 70%), this can indicate that there are technical artifacts in the data, such as PCR amplification bias or sequencing errors. The "Overrepresented sequences" error in FastQC indicates that one or more sequences in your data are significantly overrepresented compared to the rest of the data. This can be a sign of adapter contamination, PCR artifacts, or other technical issues.

Therefore, in the next part I will decide to continue project with which file.

SRR17172486

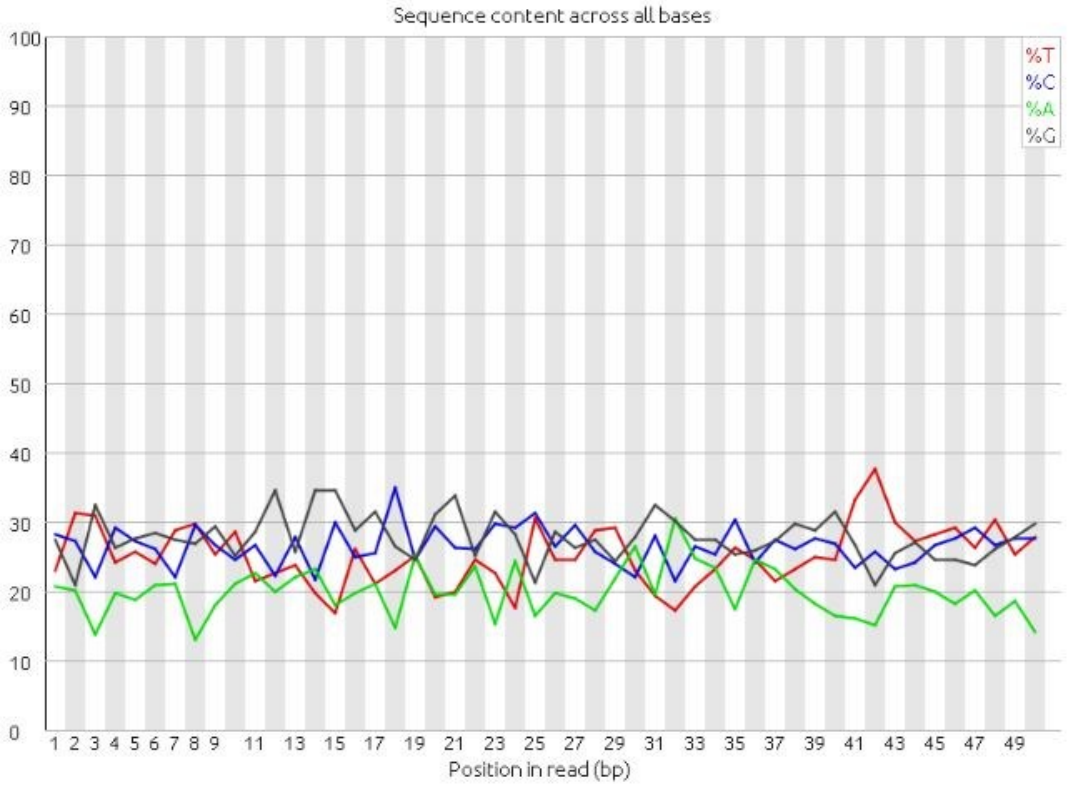
To run FastQC on the FASTQ files for SRR17172481:

```
fastqc /home/Fereshteh/Project3-algorithim/SRR17172486.fastq -o /ho
me/Fereshteh/Project3-algorithim
```

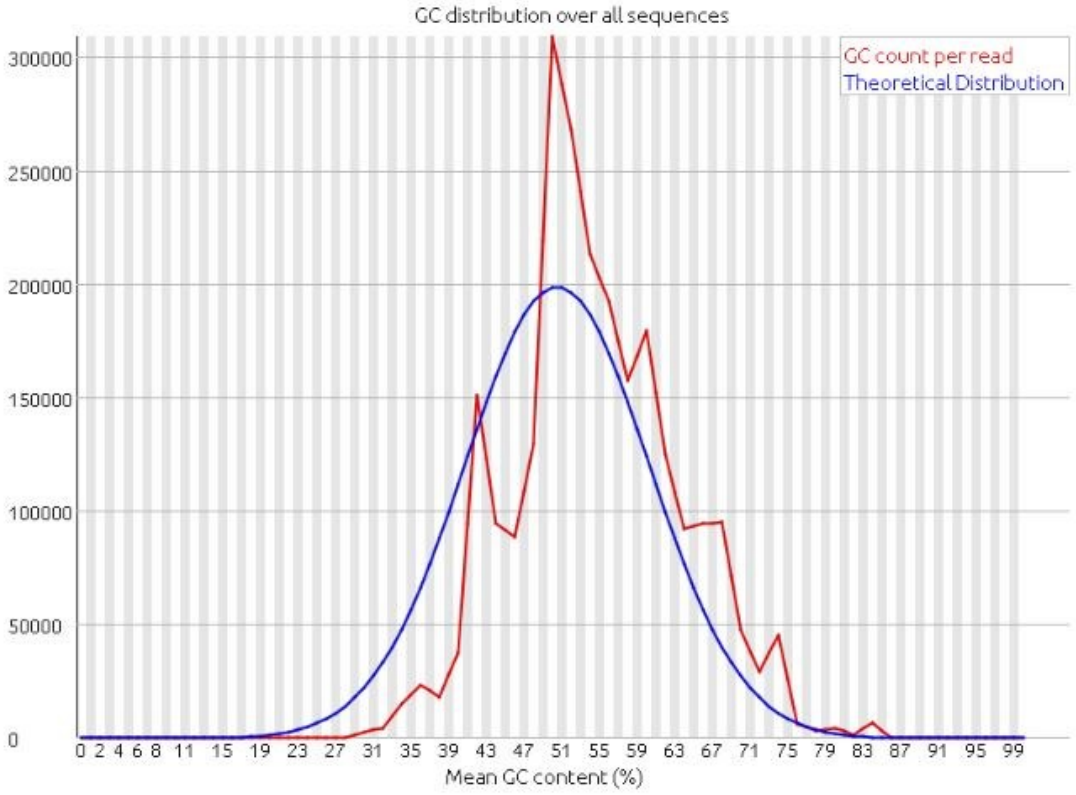


As it is clear in summary part, there are red icons for "Per base sequence content", "Per base GC sequence content", "Sequence Duplication Levels", and "Overrepresented sequences", it suggests that the sequencing data may have some quality issues that need to be addressed before downstream analysis. Before set trimming, options must look at each errors carefully.

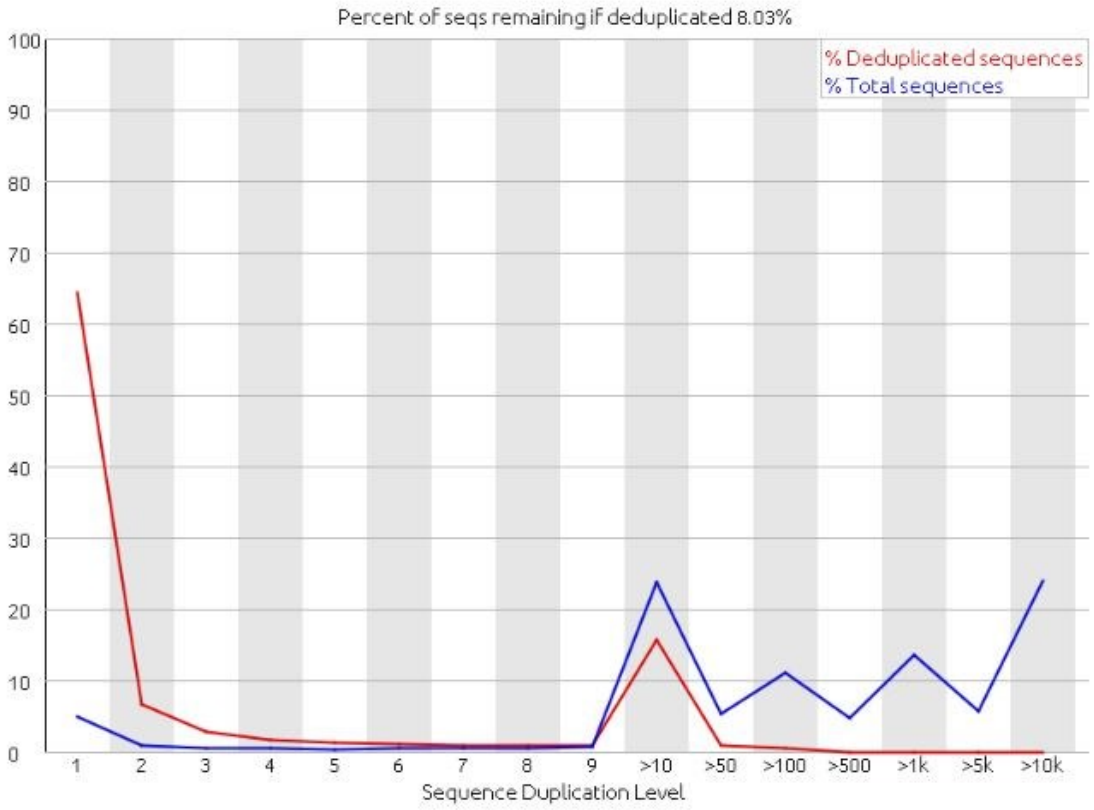
Per base sequence content



Per sequence GC content



Sequence Duplication Levels



Overrepresented sequences

Sequence	Count	Percentage	Possible Source
GTTCCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACGGGTTGTTAATCT	69157	2.8346285639335664	No Hit
CTGAAAATGTAACAGGACTGAGTCCTGTTACATTTTCAGCTTGTCCTGTG	53845	2.2070155591625267	No Hit
AAGGCTGCCAAGAGAGCCACGGCCAGCTTGGAAGTCATGTTTACACACAG	50275	2.060687291984326	No Hit
GCGCGGGCCTGGAGGAGTAGTAGAGGCCCAGGCCCCTGTGGTCGTAGCAT	48584	1.991376059547817	No Hit
TCCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACGGGTTGTTAATCTCG	45270	1.8555407997639075	No Hit
CTGAAAATGTAACAGGACTCGAGTCCTGTTACATTTTCAGCTTGTCCTGT	44993	1.8441870378568033	No Hit
AGTGTGATGAGGAGCTGGGCAGCATGCTGCCCAGCTCCTCTTCAATTCTG	43012	1.7629891954814485	No Hit
CTGTGTGTTCTCTTATGTTGGTTGCTAGCACGTGGAACCCAATAGGCACC	39070	1.6014132769334186	No Hit
GCAGCATCCCAGCCGCCGCCGCCATAGGTGGACGCGGTCGTCATGGCGGC	24948	1.0225763612217795	No Hit
CGTCGTCCTGCAGGCTGCGGGTGAGACCTTTGGTCTCACCCCAGCCTGCA	20181	0.8271850868132409	No Hit
GTGGTCGTAGCATCAGCTAGTAGGACAGACGGGTTGTTAATCTCGTATGC	19157	0.7852130572360764	No Hit
GAGTGGCTTGTTGACTGCTGGCTGCCTGCAGAGCCAGGACCTGTCTGTGT	18900	0.7746790615316512	No Hit
CTCTTGCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACGGGTTGTTAAT	18839	0.7721787746134803	No Hit
TTGTTGCTATCAGTCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACGGG	15999	0.6557719738330628	No Hit
ACCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACGGGTTGTTAATCTCG	14893	0.6104389028249143	No Hit
TATTTTCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACGGGTTGTTAAT	12053	0.4940321020444969	No Hit
GACAGCTGAACCCTCGTGGAGCCATTCATACAAGTCCCTATTTAAGGAAC	11722	0.48046497139015953	No Hit
ACTGTTGAGCTTCTGTTCTGCTGGCGCCAGCAGAACAAAGACAGCGACAA	11672	0.478415555883462	No Hit
TGGTGCCTGTGGTCGTAGCATCAGCTAGTAGGACAGACGGGTTGTTAATC	11417	0.4679635367993048	No Hit
GATTTTTGAAATTACCCTGACAGGGTAATTTCAAAAATCTTAGGCCTGTG	10435	0.42771301624776614	No Hit
CATGGGTCATCTTTGTCAGGCTGAGTTGATGATGCTATACACGTAGTCTC	9682	0.3968488187169019	No Hit
TTAGGGTACCACGGCCGTTAAACATGTCACTGGGCAGGCGGTGCCTCTCC	9365	0.3838555244044398	No Hit

Therefore, I used trimming on windows:

```
cd C:\Users\Fereshteh\Downloads\Trimmomatic-0.39
```

```
java -jar C:\Users\Fereshteh\Downloads\Trimmomatic-0.39\trimmomatic
-0.39.jar SE -phred33 E:\PhD-CLASSES\algorithm\project3\part-A\SRR1
7172486.fastq E:\PhD-CLASSES\algorithm\project3\part-A\SRR17172486-
trimed.fastq LEADING:3 TRAILING:3 SLIDINGWINDOW:4:15 MINLEN:36
```

"SE" specifies that the input file contain single-end reads. "-phred33" specifies the quality encoding used in the input file. "LEADING: 3" and "TRAILING: 3" specify that any bases with a quality score lower than three at the beginning or end of a read should be trimmed. "SLIDINGWINDOW: 4:15" specifies that a sliding window should be used to trim bases with an average quality score of less than 15 over a window of four bases. "MINLEN: 36" specifies that reads shorter than 36 bases after trimming should be discarded.

FastQC after trimming:

```
fastqc /home/Fereshteh/Project3-algorithim/SRR17172486-trimed.fastq
-o /home/Fereshteh/Project3-algorithim
```

The result:

Overall quality has improved but there may still be specific regions of the data that flagged as problematic.

A T percentage of 38% at position 41 suggests that there may be a bias towards T's at that position. This bias could be due to several factors, such as PCR amplification bias, sequencing errors, or a specific genomic feature that is enriched in T's. If the mean GC content is still 51% in the GC distribution plot generated by FastQC even after trimming the sequences, it suggests that the GC content of the sequences is relatively consistent across the entire length of the reads. It's worth noting that the GC content can sometimes be biased due to the presence of adapter sequences or other technical artifacts in the reads. Trimming these sequences can help to reduce this bias and produce a more accurate estimate of the true GC content. However, even after trimming, there may still be some residual bias in the GC content due to other factors, such as PCR amplification or sequencing errors. Also, If the level of sequence duplication is high (e.g., 70%), this can indicate that there are technical artifacts in the data, such as PCR amplification bias or sequencing errors. The "Overrepresented sequences" error in FastQC indicates that one or more sequences in your data are significantly overrepresented compared to the rest of the data. This can be a sign of adapter contamination, PCR artifacts, or other technical issues.

Therefore, in the next part I will decide to continue project with which file.

Please answer the following questions.

1. What is the average number of reads across samples before and after the read trimming?

Answer:

SRR17172481

Basic Statistics

Measure	Value
Filename	SRR17172481.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	7521289
Sequences flagged as poor quality	0
Sequence length	50
%GC	50

Basic Statistics

Measure	Value
Filename	SRR17172481-trimed.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	7427994
Sequences flagged as poor quality	0
Sequence length	36-50
%GC	50

To answer this part, mean of total sequences of the two, fastQC files must calculate:

(7521289+7427994)/2 = 7454641.5

SRR17172486

Basic Statistics

Measure	Value
Filename	SRR17172486.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	2439720
Sequences flagged as poor quality	0
Sequence length	50
%GC	54

Basic Statistics

Measure	Value
Filename	SRR17172486-trimed.fastq
File type	Conventional base calls
Encoding	Sanger / Illumina 1.9
Total Sequences	2399484
Sequences flagged as poor quality	0
Sequence length	36-50
%GC	54

To answer this part, mean of total sequences of the two, fastQC files must calculate:

(2439720+2399484)/2 = 2419602

2. Compare the read length averages in different samples before and after the read trimming?

Answer:

SRR17172481

```
# Replace "your_fastq_file.fastq" with the path to your fastq file
fastq_file <- file("SRR17172481.fastq", open = "r")

# Initialize variables for counting bases and reads
bases <- 0
reads <- 0

# Loop through the fastq file and count the number of bases and reads
while (length(header <- readLines(fastq_file, n = 1)) > 0) {
  sequence <- readLines(fastq_file, n = 1)
  readLines(fastq_file, n = 1)
  quality <- readLines(fastq_file, n = 1)
  bases <- bases + nchar(sequence)
  reads <- reads + 1
}

# Calculate the average read length
avg_read_length <- bases/reads

# Print the average read length
cat("Average read length:", avg_read_length, "\n")
```

```
## Average read length: 50
```

```
# Close the fastq file
close(fastq_file)
```

```r
# Replace "your_fastq_file.fastq" with the path to your fastq file
fastq_file <- file("SRR17172481-trimed.fastq", open = "r")

# Initialize variables for counting bases and reads
bases <- 0
reads <- 0

# Loop through the fastq file and count the number of bases and reads
while (length(header <- readLines(fastq_file, n = 1)) > 0) {
  sequence <- readLines(fastq_file, n = 1)
  readLines(fastq_file, n = 1)
  quality <- readLines(fastq_file, n = 1)
  bases <- bases + nchar(sequence)
  reads <- reads + 1
}

# Calculate the average read length
avg_read_length <- bases/reads

# Print the average read length
cat("Average read length:", avg_read_length, "\n")
```

```
## Average read length: 49.91517
```

```r
# Close the fastq file
close(fastq_file)
```

SRR17172486

```r
# Replace "your_fastq_file.fastq" with the path to your fastq file
fastq_file <- file("SRR17172486.fastq", open = "r")

# Initialize variables for counting bases and reads
bases <- 0
reads <- 0

# Loop through the fastq file and count the number of bases and reads
while (length(header <- readLines(fastq_file, n = 1)) > 0) {
  sequence <- readLines(fastq_file, n = 1)
  readLines(fastq_file, n = 1)
  quality <- readLines(fastq_file, n = 1)
  bases <- bases + nchar(sequence)
  reads <- reads + 1
}

# Calculate the average read length
avg_read_length <- bases/reads

# Print the average read length
cat("Average read length:", avg_read_length, "\n")
```

```
## Average read length: 50
```

```r
# Close the fastq file
close(fastq_file)
```

```r
# Replace "your_fastq_file.fastq" with the path to your fastq file
fastq_file <- file("SRR17172486-trimed.fastq", open = "r")

# Initialize variables for counting bases and reads
bases <- 0
reads <- 0

# Loop through the fastq file and count the number of bases and reads
while (length(header <- readLines(fastq_file, n = 1)) > 0) {
  sequence <- readLines(fastq_file, n = 1)
  readLines(fastq_file, n = 1)
  quality <- readLines(fastq_file, n = 1)
  bases <- bases + nchar(sequence)
  reads <- reads + 1
}

# Calculate the average read length
avg_read_length <- bases/reads

# Print the average read length
cat("Average read length:", avg_read_length, "\n")
```

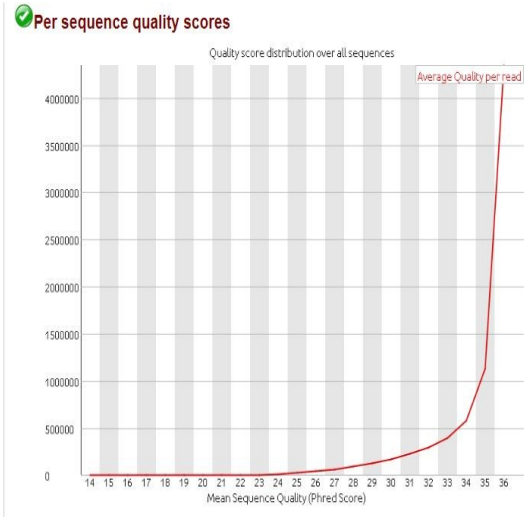
```
## Average read length: 49.88462
```

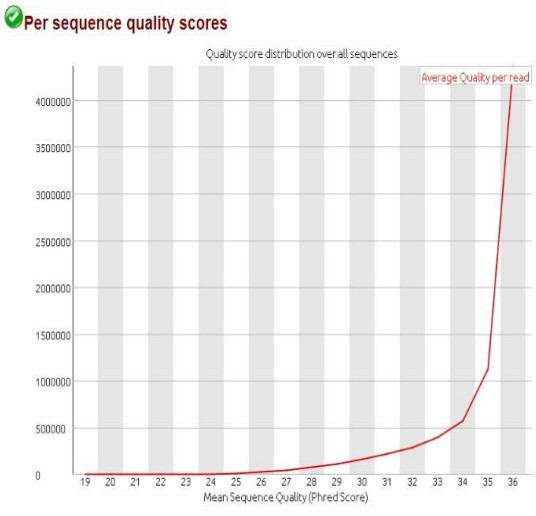
```r
# Close the fastq file
close(fastq_file)
```

3. Compare the read quality distributions over all sequences before and after the read trimming.

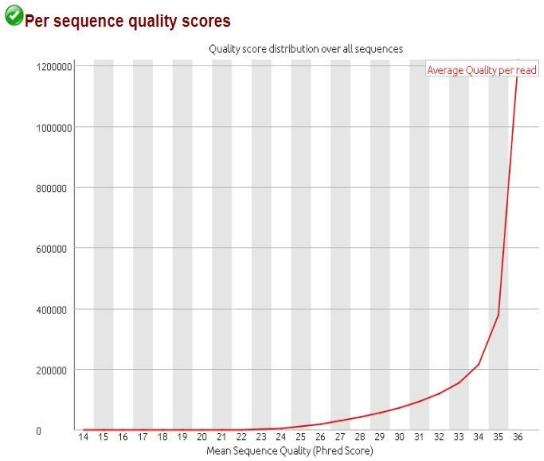
Answer:

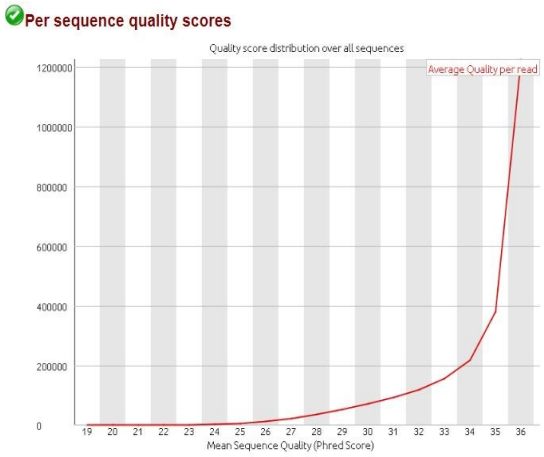
SRR17172481





SRR17172486





The fact that the quality distribution per read during the Phred score is increasing slowly at first and then has a significant increase after a Phred score of 35 suggests that the quality of the reads in my datas are generally good in all fastq files (before and after trimming).

4. What does the Adaptor Content warning indicate?

Answer:

Adaptor content typically refers to a type of low-quality sequencing data that results from the presence of adaptor sequences in the sequenced reads. Adaptors are short, artificial DNA sequences that are added to the ends of the DNA fragments during the library preparation stage of sequencing. These adaptors are used to attach the DNA fragments to the sequencing platform and allow for efficient sequencing. However, if the adaptor sequences are not removed, during the sequencing process, they can result in low-quality sequencing data that can affect downstream analysis. Therefore, one important step in quality control and trimming of sequencing data is to remove these adaptor sequences using specialized software tools. There are various tools available for adaptor trimming, such as Trimmomatic, Cutadapt, and BBDuk. These tools can automatically detect and remove adaptor sequences from the sequencing reads, thereby improving the overall quality of the data and reducing the risk of false positives or other errors in downstream analysis.

5. Why do we first remove the Adapter sequences for the reads and then the low-quality bases?

Answer:

When processing sequencing data, it is typically recommended to first remove adapter sequences from the reads and then trim low-quality bases. This is because adapter sequences can interfere with the alignment and assembly of the reads, and can lead to overestimation of the number of unique sequences. By removing adapter sequences first, we can ensure that the reads are of high quality before proceeding with the next step of trimming low-quality bases. This can help to reduce the number of errors and artifacts in the data, which can improve the accuracy and sensitivity of downstream analysis. Additionally, adapter sequences are typically located at the ends of the reads, while low-quality bases can be present throughout the length of the reads. By removing adapter sequences first, we can ensure that the low-quality bases are not mistakenly removed from the reads due to their proximity to the adapters.

6. What does the quality of bases mean, and how is it obtained?

Answer:

In the context of DNA sequencing, the quality of bases refers to the confidence level or probability that a given base call is correct. This information is typically encoded in the form of a Phred score, which is a logarithmic scale that ranges from 0 to 40. A higher Phred score indicates a higher quality base call, with a Phred score of 10 corresponding to a base call accuracy of 90% and a Phred score of 20 corresponding to a base call accuracy of 99%. The quality of bases is obtained during the sequencing process, where it is determined by various factors such as the intensity of the fluorescent signal, the level of background noise, and the accuracy of base calling algorithms. These factors can affect the accuracy and reliability of the base calls, and can lead to errors or artifacts in the sequencing data. To assess the quality of bases in sequencing data, specialized software tools are used to generate quality scores for each base call. The most commonly used tool for this purpose is FastQC, which calculates various quality metrics such as the distribution of Phred scores, the percentage of bases with low quality scores, and the presence of overrepresented sequences or adapter contamination. Based on the quality scores obtained from these tools, researchers can make informed decisions about how to preprocess the sequencing data, such as by trimming low-quality bases, filtering out low-quality reads, or adjusting the parameters of downstream analysis tools. By ensuring that the sequencing data is of high quality, researchers can maximize the accuracy and reliability of their analysis and draw more robust conclusions from their results.

Part b- Read mapping

In the second step, map the reads to the reference genome using the HISAT2 software. To map reads to the reference, the HISAT2 software uses a graph-based alignment and a variety

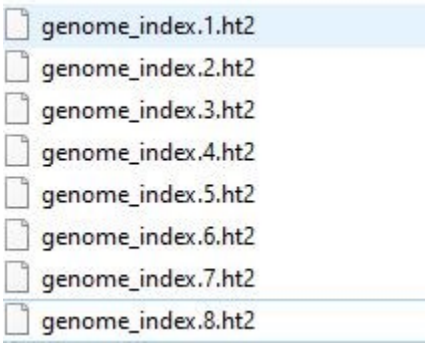
of metrics. The output of this step will be a SAM file for each sample. (Hint: you may use hisat2 [options]* -x <ht2-idx> {-1 <m1> -2 <m2> } [-S <sam>]). Use the Samtools software to convert the HISAT2 SAM files to BAM files (Hint: you may use samtools view [options] <in.sam>). Make sure you delete the SAM files afterwards.

Answer:

At first, I downloaded reference genome from:
(https://hgdownload.soe.ucsc.edu/goldenPath/hs1/bigZips/)

In the next step, created an index for reference genome using HISAT2

```
hisat2-build /home/Fereshteh/Project3-algorithim/hs1.fa genome_inde
x
```



Specifically, the HISAT2-build command creates 8 index files with different extensions, all of which are needed for the HISAT2 alignment process. Each of these files is used by HISAT2 to quickly align reads to the reference genome. Together, these files make up the HISAT2 index and enable fast and accurate alignment of short reads to the reference genome.

Then mapped reads to the reference genome using HISAT2 (I applied this step for both fastq files, before and after trimming):

```
hisat2 -x genome_index -1 /home/Fereshteh/Project3-algorithim/SRR17
172481.fastq -S hisat2_output/SRR17172481.sam
```

```
hisat2 -x genome_index -1 /home/Fereshteh/Project3-algorithim/SRR17
172481.fastq -S hisat2_output/SRR17172481-trim.sam
```

```
hisat2 -x genome_index -1 /home/Fereshteh/Project3-algorithim/SRR17
172481.fastq -S hisat2_output/SRR17172486.sam
```

```
hisat2 -x genome_index -1 /home/Fereshteh/Project3-algorithim/SRR17
172481.fastq -S hisat2_output/SRR17172486-trim.sam
```

Note: I will discuss about this part results later

Then, converted SAM files to BAM files using Samtools (I applied this step for both fastq files, before and after trimming):

```
samtools view -bS hisat2_output/SRR17172481.sam > bam_files/SRR1717
2481.bam
```

```
samtools view -bS hisat2_output/SRR17172481-trim.sam > bam_files/SR
R17172481-trim.bam
```

```
samtools view -bS hisat2_output/SRR17172486.sam > bam_files/SRR1717
2486.bam
```

```
samtools view -bS hisat2_output/SRR17172486-trim.sam > bam_files/SR
R17172486-trim.bam
```

1- What is the difference between SAM and BAM files?

SAM (Sequence Alignment/Map) and BAM (Binary Alignment/Map) files are both file formats used to represent the alignment of sequencing reads to a reference genome. The main difference between the two formats is that SAM files are in plain text, while BAM files are binary-encoded, which makes them more compact and faster to read and process. Here are some key differences between SAM and BAM files:

SAM files are plain text files, while BAM files are binary-encoded. This means that BAM files are more compact and faster to read and process, but cannot be easily read or edited by humans. SAM files are larger than their corresponding BAM files because they contain each alignment in a human-readable text format, while BAM files are compressed and stored in a binary format. BAM files can be indexed, which allows faster access to specific regions of the alignment. SAM files can also be indexed, but this requires an additional step. BAM files can be used to store additional information about each alignment, such as the quality scores of the reads, while SAM files only store the essential information needed to represent the alignment.

2- What is the purpose of indexing the genome?

Indexing the genome is an important step in bioinformatics analysis, especially when dealing with large genomes and large sequencing datasets. The purpose of indexing the genome is to create an efficient data structure that allows for fast retrieval of specific regions of the genome, without the need to read the entire genome sequence. Here are some common use cases for indexing the genome:

Alignment: When aligning sequencing reads to a reference genome, an index allows for fast lookup of the portions of the genome that match the reads, which can be important when dealing with large sequencing datasets. Variant calling:

In variant calling, an index allows for efficient retrieval of the portions of the genome that are likely to contain variants, which can speed up the analysis and reduce the computational resources needed. Annotation: When annotating genomic features, an index allows for fast lookup of the regions of the genome that correspond to specific genes, regulatory elements, or other functional elements. There are different types of genome indexing methods, such as the Burrows-Wheeler Transform (BWT), FM-index, and hash-based indexing. The choice of indexing method depends on the specific use case and the characteristics of the genome and the sequencing data.

3- Report mapping percentages of all samples in a table. Please explain why a low percentage of reads cannot be mapped.

Sample	Total reads	Unaligned (%)	Aligned once (%)	Aligned >1 times (%)	Overall alignment rate (%)
SRR17172481	7,521,289	21.31	74.56	4.13	78.69
SRR17172481-trimmed	7,427,994	21.21	74.63	4.15	78.79
SRR17172486	2,439,720	72.24	25.69	2.08	27.76
SRR17172481-trimmed	2,399,484	72.07	25.83	2.10	27.93

As you can see, SRR17172481 and SRR17172481-trimmed have very similar alignment statistics, with an overall alignment rate of around 78%. SRR17172486 and SRR17172486-trimmed, on the

other hand, have much lower overall alignment rates, at around 28%. So normal mapped with reference genome better in comparison with COVID19 data. These differences in alignment statistics could be due to a number of factors, including differences in the quality of the sequencing runs, differences in the samples themselves, or issues with the alignment process. The differences in the alignment statistics between the normal human bronchoalveolar lavage cells and the COVID-19 patient bronchoalveolar lavage cells could be due to a number of factors. For example:RNA degradation: FFPE samples are often subject to RNA degradation, which can cause low quality reads and make it difficult to align reads to a reference genome. The COVID-19 patient samples may have more degradation due to the disease state.Differences in gene expression: The COVID-19 patient samples may have different gene expression profiles compared to the normal human samples. This could make it more difficult to align reads to a reference genome, particularly if the reference genome used for alignment is based on a different species or if the reference genome does not accurately represent the sample being sequenced. Differences in viral load: The COVID-19 patient samples may contain a higher proportion of reads originating from the SARS-CoV-2 virus, which can make it more difficult to align reads to the human reference genome. Technical variation: There may be technical differences between the sequencing runs for the different samples, which can affect the quality of the sequencing reads and the ability to align reads to a reference genome.

A low percentage of reads that cannot be mapped to a reference genome can be caused by several factors, including low quality reads, contamination, poor reference genome, and variability in the sample. Low quality reads can result from degraded samples, poor quality DNA or RNA, and errors introduced during library preparation or sequencing. Contamination can originate from DNA or RNA from a source other than the sample of interest during sample collection, library preparation, or sequencing. Poor reference genome can occur if the reference genome used for alignment is of low quality or does not accurately represent the sample being sequenced. Variability in the sample can also make it difficult to map reads to a reference genome, particularly for samples from organisms with high levels of genetic diversity or tissues with high levels of gene expression variability. It is essential to consider these factors when interpreting alignment statistics and to account for them during quality control analysis of sequencing data.

Part c- Building gene expression matrix

In the third step, run htseq-count on count aligned reads for differential expression analysis. You can use the HTSeq documentation for further explanation. In this step, you need a gene/transcript annotation file that you can download from this link. Merge results files into a single matrix for use in the *edgeR* package.

Answer:

*Note: From this step based on alignment results, I preferred to continue analysis on fastq files before trimming.

Before run htseq-count, I sorted my Bam files:

```
samtools sort /home/Fereshteh/Project3-algorithim/SRR17172481.bam -
o /home/Fereshteh/Project3-algorithim/SRR17172481_SORTED.bam
```

```
samtools sort /home/Fereshteh/Project3-algorithim/SRR17172486.bam -
o /home/Fereshteh/Project3-algorithim/SRR17172486_SORTED.bam
```

Then I indexed my Bam files:

```
samtools index /home/Fereshteh/Project3-algorithim/SRR17172481_SORT
ED.bam
```

```
samtools index /home/Fereshteh/Project3-algorithim/SRR17172486_SORT
ED.bam
```

In the following, I downloaded gene/transcript annotation file (catLiftOffGenesV1.gtf), from the link:

(https://hgdownload.soe.ucsc.edu/goldenPath/hs1/bigZips/)

```
htseq-count -f bam -r name -s no -i gene_name /home/Fereshteh/Proje
ct3-algorithim/SRR17172481_SORTED.bam catLiftOffGenesV1.gtf > count
_SRR17172481.txt
```

```
htseq-count -f bam -r name -s no -i gene_name /home/Fereshteh/Proje
ct3-algorithim/SRR17172486_SORTED.bam catLiftOffGenesV1.gtf > count
_SRR17172486.txt
```

This command uses the htseq-count tool to count the number of reads mapping to each gene in the genome annotation file catLiftOffGenesV1.gtf, using the sorted BAM file.

The output:

count_SRR17172481.txt

```
5S_rRNA 0
5_8S_rRNA   0
7SK 0
A1BG    0
A1BG-AS1    0
A1CF    0
A2M 530
A2M-AS1 0
A2ML1   0
A2ML1-AS1   0
A2ML1-AS2   0
A2MP1   0
A3GALT2 0
A4GALT  207
A4GNT   20
AA06    0
AAAS    72
AACS    121
AACSP1  0
AADAC   0
AADACL2 0
AADACL2-AS1 0
AADACL3 0
AADACL4 0
AADACP1 0
AADAT   0
AAGAB   695
AAK1    0
AAMDC   0
AAMP    384
```

count_SRR17172486.txt

```
5S_rRNA 0
5_8S_rRNA   0
7SK 0
A1BG    0
A1BG-AS1    0
A1CF    19
A2M 22
A2M-AS1 0
A2ML1   0
A2ML1-AS1   0
A2ML1-AS2   0
A2MP1   0
A3GALT2 0
A4GALT  58
A4GNT   0
AA06    0
AAAS    0
AACS    43
AACSP1  0
AADAC   0
AADACL2 0
AADACL2-AS1 0
AADACL3 0
AADACL4 0
AADACP1 0
AADAT   0
AAGAB   0
AAK1    0
AAMDC   0
AAMP    49
```

Then, to merge results files into a single matrix:

```
# Load the first count matrix
counts_SRR17172481 <- read.table("count_SRR17172481.txt", header = TRUE, row.names = 1)

# Load the second count matrix
counts_SRR17172486 <- read.table("count_SRR17172486.txt", header = TRUE, row.names = 1)

# Merge the two matrices using the intersect of row names
counts_merged <- merge(counts_SRR17172481, counts_SRR17172486, by = "row.names")
rownames(counts_merged) <- counts_merged[,1]
counts_merged <- counts_merged[,-1]

# Write the merged count matrix to file
write.table(counts_merged, file = "Merge-count-matrix_SRR17172481-SRR17172486.txt", sep = "\t", quote = FALSE, row.names = T
RUE)
```

1.How many genes are not expressed in control and covid samples? Explain the results.

Answer:

```
# Read the CSV file
counts <- read.csv("Merge-count-matrix_SRR17172481-SRR17172486.csv", header = TRUE, row.names = 1)

# Count the number of genes with zero counts in the control sample
zero_counts_control <- sum(counts[, "Counts.81"] == 0)

# Print the number of genes with zero counts in the control sample
cat("Number of genes with zero counts in the control sample:", zero_counts_control, "\n")
```

```
## Number of genes with zero counts in the control sample: 50771
```

```
# Count the number of genes with zero counts in the COVID sample
zero_counts_covid <- sum(counts[, "Counts.86"] == 0)

# Print the number of genes with zero counts in the COVID sample
cat("Number of genes with zero counts in the COVID sample:", zero_counts_covid, "\n")
```

```
## Number of genes with zero counts in the COVID sample: 53708
```

Number of genes with zero counts in the control sample: 50771
Number of genes with zero counts in the COVID sample: 53708

If the number of genes that were not expressed in a normal human bronchoalveolar lavage (BAL) cells sample is less than in the COVID samples, it could suggest that COVID infection may be associated with changes in gene expression patterns in the BAL cells. One possible explanation is that COVID infection may induce the expression of genes that are not normally expressed in the BAL cells. For example, COVID infection may activate genes involved in the immune response or inflammation, which may not be expressed in the normal cells. Another possible explanation is that COVID infection may cause changes in the expression levels of genes that are normally

expressed in the BAL cells. For example, COVID infection may upregulate or downregulate the expression of genes involved in lung function or metabolism, which could affect the gene expression patterns. However, it is important to note that the interpretation of the results depends on the specific experimental design and the context of the study. It is also important to consider additional factors such as sample size, statistical power, and potential confounding variables that may affect the results. In addition, we observe a large number of genes with zero counts or no detectable expression in gene expression data, including in samples of normal human bronchoalveolar lavage (BAL) cells and COVID-infected BAL cells. There are several possible reasons for the large number of non-expressed genes: Technical limitations: Gene expression measurements are subject to technical variability and limitations, such as sequencing depth, sample quality, and data processing. These factors can affect the sensitivity and accuracy of the gene expression measurements, and may contribute to the large number of non-expressed genes. Biological complexity: Gene expression is a complex process that is regulated by multiple factors, including genetic, epigenetic, and environmental factors. The expression of many genes may be tissue-specific, cell-type-specific, or context-specific, and may depend on the developmental stage, disease status, or other factors. The large number of non-expressed genes may reflect the diversity and complexity of gene expression in the samples. Evolutionary constraints: Not all genes are expressed in all cells or under all conditions, and some genes may have evolved to be expressed only in specific tissues or under specific conditions. The large number of non-expressed genes may reflect the evolutionary constraints on gene expression.

2. Compare the matrix obtained at this stage with the corresponding gene expression submatrix of the main study. Discuss the differences.

Answer:

At first, I downloaded main matrix, from:

(https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE190496)

The COUNTS column is the common data field that exists in both CSV files, and it is the basis on which the two files are merged. After that, I merged all columns in one .csv file:

```r
# Load the dplyr library
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.2.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# Read in the CB_data CSV file and assign it to the CB_data data frame
CB_data = read.csv('GSE190496_Gene_counts_CBdata.csv')

# Rename the first five columns of CB_data using the values in the first row,
# and the remaining columns using their existing names
colnames(CB_data) = c(CB_data[1, 1:5], colnames(CB_data)[6:13])

# Remove the first row of CB_data, which contains column names, and the 14th column
CB_data = CB_data[-1, ]
CB_data = CB_data[, -14]

# Read in the NB_data CSV file and assign it to the NB_data data frame
NB_data = read.csv('GSE190496_Gene_counts_NBdata.csv')

# Merge the NB_data and CB_data data frames by the 'COUNTS' column and assign the
# resulting merged data frame to merged_matrix
merged_matrix = merge(NB_data, CB_data, by='COUNTS')

# Write the merged_matrix data frame to a CSV file named "merged_all_main_matrix__NB_CB.csv"
write.csv(merged_matrix, "merged_all_main_matrix__NB_CB.csv")
```

The main matrix has 22339 genes but Count files of SRR17172481 and SRR17172486 has 60836 genes. Therefore, this result shows that our data has problems. It's difficult to determine what the problem might be without more information, but one possibility is that the count files for SRR17172481 and SRR17172486 were generated using a different reference genome or annotation file than the one used to generate the main matrix.

2. What are other software available to do this step? Name two other software and discuss their advantages and disadvantages.

Answer:

Two popular options are featureCounts and Salmon. FeatureCounts is a tool that counts reads aligned to annotated genomic features such as genes, and is known for its speed and ability to process large datasets quickly. However, it may not perform as well on low-quality or highly variable datasets. Salmon, on the other hand, uses a quasi-mapping algorithm to estimate transcript-level abundance and can be more accurate and less biased than alignment-based

methods, particularly for highly variable or complex datasets. However, it may not be as suitable for analyzing differential gene expression. The choice of which tool to use will depend on the specific research question and dataset being analyzed, and it is important to carefully evaluate the performance of each tool before selecting the one that best suits the needs of the analysis.

FeatureCounts has several advantages, including fast processing speed, annotation flexibility, the ability to handle multi-mapping reads, accurate gene-level quantification, and compatibility with downstream analysis tools. One disadvantage of FeatureCounts is that it may not perform as well on low-quality or highly variable datasets. Additionally, FeatureCounts may not be as well-suited for analyzing alternative splicing events compared to other tools that use transcript-level quantification. Another potential disadvantage of FeatureCounts is that it requires aligned reads as input, which can be a time-consuming and computationally intensive step. This means that the quality of the alignment can affect the accuracy of the gene expression quantification.

For Salmon, advantages, including fast processing speed, accurate quantification using a quasi-mapping algorithm, flexibility, compatibility with downstream analysis tools, and reproducibility. However, it may require more computational resources and may not perform as well on datasets with low complexity or low read depth. Overall, Salmon is a powerful tool for gene expression analysis, but researchers should evaluate its performance on their specific dataset before using it for downstream analyses.

Part d- Differential gene expression analysis

Use edgeR to perform differential gene expression analysis. The edgeR is an open-source Bioconductor package designed for differential expression analysis based on count-based RNA-seq data. Use the expression matrix of the main study (all samples) to answer the following questions.

1. How many genes are given to edgeR? How many of them are differentially expressed in covid versus normal samples? How do you define statistical significance in this context?

Answer:

The COUNTS column is the common data field that exists in both CSV files, and it is the basis on which the two files are merged.

```r
# Load the edgeR library
library(edgeR)
```

```
## Loading required package: limma
```

```r
# Read in the count data matrix from the CSV file
counts <- read.csv("merged_all_main_matrix__NB_CB.csv", header = TRUE, row.names = 1)

# Determine the number of genes in the count matrix
n_genes <- nrow(counts)

# Print the number of genes in the count matrix to the console
cat("Number of genes in the count matrix:", n_genes, "\n")
```

```
## Number of genes in the count matrix: 22339
```

```r
# Load the required library
library(edgeR)
```

```
## Loading required package: limma
```

```r
# Read in the count matrix from the CSV file
counts <- read.csv("merged_all_main_matrix__NB_CB.csv", header = TRUE, row.names = 1)

# Replace any NA values with 0
counts[is.na(counts)] <- 0

# Write the modified count matrix to a new CSV file
write.csv(counts, file = "modify-merge-main-matrix.csv", row.names = TRUE)

# Print a message indicating that the file has been saved
cat("Modified count matrix saved to modify-merge-main-matrix.csv\n")
```

```
## Modified count matrix saved to modify-merge-main-matrix.csv
```

```r
# Identify any non-numeric values in each column of the data frame
non_numeric_cols <- sapply(counts, function(x) any(grepl("[^0-9.]", x)))

# Exclude the non-numeric columns from the data frame
gene_expression_numeric <- counts[, !non_numeric_cols]

# Define the experimental design and create the DGEList object
group <- factor(c(rep("normal", 5), rep("covid", 8)))
dge <- DGEList(counts = as.matrix(gene_expression_numeric), group = group)

# Identify which columns correspond to normal samples and which ones correspond to COVID samples
normal_cols <- c("NB5", "NB4", "NB1", "NB2", "NB3")
covid_cols <- c("CB4", "CB5", "CB6", "CB7", "CB2", "CB3", "CB1", "CB8")

# Filter the genes that have low expression and/or low variability
keep <- rowSums(cpm(dge[, normal_cols]) >= 1) >= 3
dge <- dge[keep, , keep.lib.sizes = FALSE]
dge <- calcNormFactors(dge)
design <- model.matrix(~group)
dge <- estimateDisp(dge, design)
fit <- glmQLFit(dge, design)
p <- glmQLFTest(fit, coef = 2)
keep <- p$table$PValue < 0.05
dge <- dge[keep, ]

# Get the number of differentially expressed genes
n_genes <- sum(p$table$PValue < 0.05 & rowMeans(cpm(dge[, normal_cols])) > 1)
```

```
## Warning in p$table$PValue < 0.05 & rowMeans(cpm(dge[, normal_cols])) > 1:
## longer object length is not a multiple of shorter object length
```

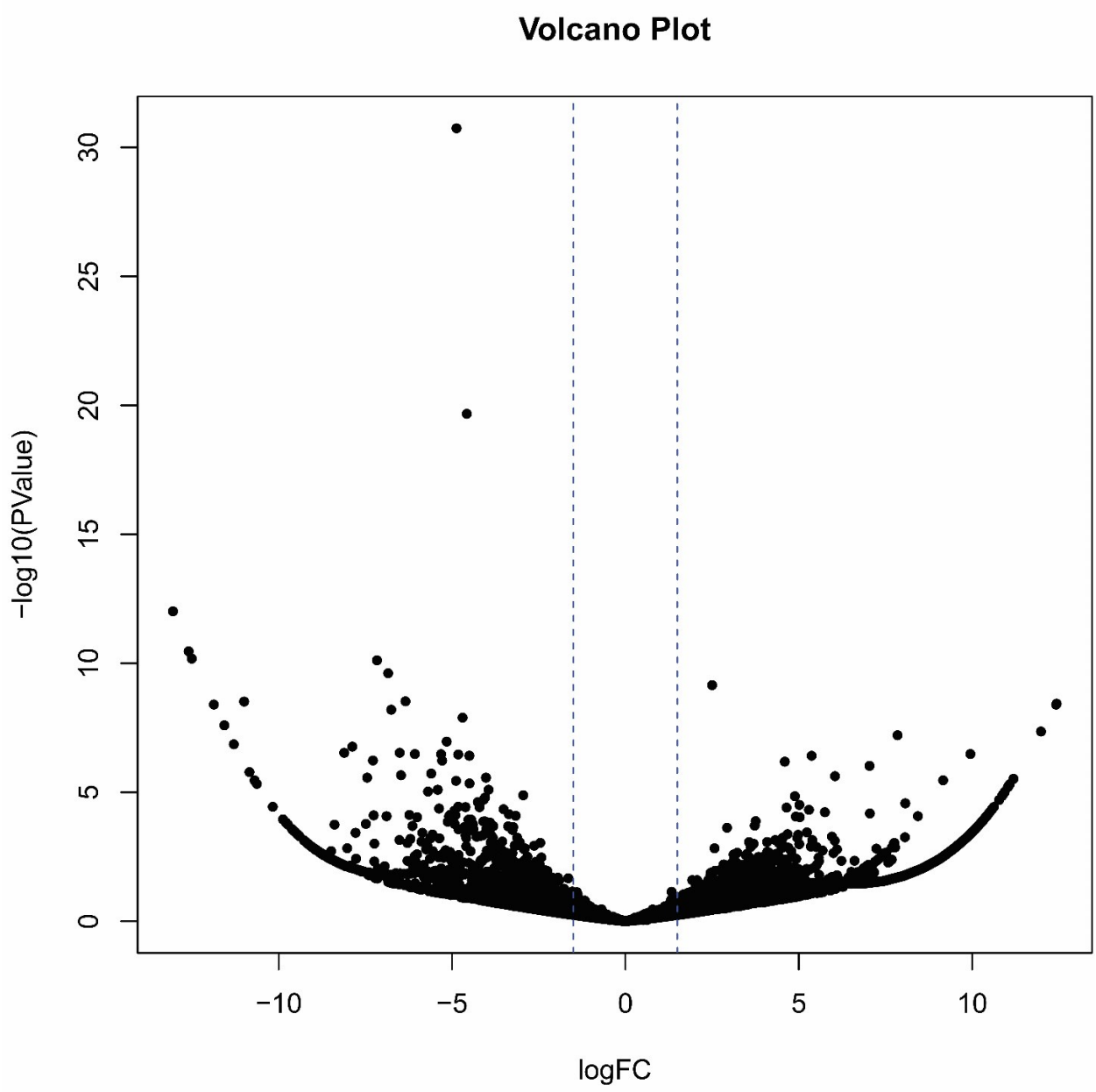
```r
cat("Number of differentially expressed genes:", n_genes, "\n")
```

```
## Number of differentially expressed genes: 1531
```

```r
# Get the top differentially expressed genes
#tags <- topTags(fit, n = n_genes)
#de_genes <- rownames(tags$table)

# Print the top differentially expressed genes with their statistics
#cat("Top differentially expressed genes:\n")
#print(tags$table)
```

To draw volcano plot of the differentially expressed genes:

```
# Create a volcano plot of the differentially expressed genes
logCPM <- cpm(dge, log = TRUE)
logCPM_normal <- rowMeans(logCPM[, normal_cols])
logCPM_covid <- rowMeans(logCPM[, covid_cols])
logFC <- logCPM_covid - logCPM_normal
pval <- p$table$PValue
plot(logFC, -log10(pval), pch = 20, cex = 0.5, col = ifelse(pval < 0.05 & abs(logFC) > 1, "red", "black"),
    xlab = "log2 Fold Change", ylab = "-log10 P-value", main = "Volcano plot",
    xlim = c(-6, 6), ylim = c(0, 20))
abline(v = c(-1, 1), lty = 2, col = "gray") # Add vertical lines at logFC = -1 and logFC = 1
abline(h = -log10(0.05), col = "gray", lty = 2) # Add a horizontal line at -log10(0.05)
```



The majority of the data in the volcano plot are clustered around log2 fold change values of 2 to 5 and -2 to -5, and -log10(p-value) values of 2 to 6, this suggests that there are a relatively large number of genes that are significantly differentially expressed between the COVID and normal samples. Genes with log2 fold changes greater than 1 or less than -1 are typically considered to be differentially expressed, and genes with p-values less than 0.05 are considered to be statistically significant. Therefore, genes that fall in the upper right and upper left quadrants of the volcano plot (i.e., high log2 fold change and low p-value) are likely to be the most differentially expressed and statistically significant genes.

How do you define statistical significance in this context?

In the context of this code, statistical significance is defined based on the p-value from a statistical test of differential expression. The p-value represents the probability of observing a difference in gene expression as large or larger than the observed difference, given that there is no true difference between the two groups or conditions being compared. A p-value less than 0.05 is commonly used as a threshold for statistical significance, indicating that there is less than a 5% chance of observing the difference in gene expression by chance alone. In the code, the line keep <- p$table$PValue < 0.05 filters the genes that have a p-value less than 0.05, indicating that they are statistically significantly differentially expressed between the COVID and normal samples.

2. Determine the percentage of differentially expressed genes with |log2FoldChange| > 1.5.

```r
# Load the required library
library(edgeR)
```

```
## Loading required package: limma
```

```r
# Read in the count matrix from the CSV file
counts <- read.csv("merged_all_main_matrix__NB_CB.csv", header = TRUE, row.names = 1)

# Replace any NA values with 0
counts[is.na(counts)] <- 0

# Write the modified count matrix to a new CSV file
write.csv(counts, file = "modify-merge-main-matrix.csv", row.names = TRUE)

# Print a message indicating that the file has been saved
cat("Modified count matrix saved to modify-merge-main-matrix.csv\n")
```

```
## Modified count matrix saved to modify-merge-main-matrix.csv
```

```r
# Identify any non-numeric values in each column of the data frame
non_numeric_cols <- sapply(counts, function(x) any(grepl("[^0-9.]", x)))

# Exclude the non-numeric columns from the data frame
gene_expression_numeric <- counts[, !non_numeric_cols]

# Define the experimental design and create the DGEList object
group <- factor(c(rep("normal", 5), rep("covid", 8)))
dge <- DGEList(counts = as.matrix(gene_expression_numeric), group = group)

# Identify which columns correspond to normal samples and which ones correspond to COVID samples
normal_cols <- c("NB5", "NB4", "NB1", "NB2", "NB3")
covid_cols <- c("CB4", "CB5", "CB6", "CB7", "CB2", "CB3", "CB1", "CB8")

# Filter the genes that have low expression and/or low variability
keep <- rowSums(cpm(dge[, normal_cols]) >= 1) >= 3
dge <- dge[keep, , keep.lib.sizes = FALSE]
dge <- calcNormFactors(dge)
design <- model.matrix(~group)
dge <- estimateDisp(dge, design)
fit <- glmQLFit(dge, design)
lrt <- glmQLFTest(fit, coef = 2)
p <- p.adjust(lrt$table$PValue, method = "BH")
keep <- p < 0.05
dge <- dge[keep, ]

# Calculate the log2-fold change and filter the genes based on a fold change threshold of 1.5
logFC <- as.numeric(coef(lrt)[,2])
de_genes <- abs(logFC) > 1.5

# Calculate the percentage of differentially expressed genes
de_percentage <- mean(de_genes & keep) * 100

# Print the percentage of differentially expressed genes to the console
cat("Percentage of differentially expressed genes with |log2FoldChange| > 1.5 and FDR < 0.05:", de_percentage, "%\n")
```

```
## Percentage of differentially expressed genes with |log2FoldChange| > 1.5 and FDR < 0.05: 7.329635 %
```

In this case, the code reports that 7.32% of the genes passed the significance threshold and were considered differentially expressed. This means that out of all the genes tested, 7.32% showed a significant difference in expression between the normal and COVID samples based on the criteria used for significance. Suggests that there may not be widespread changes in gene expression between normal and COVID samples, or that the differences in expression levels are relatively small. It is possible that some differentially expressed genes may have been missed, and a larger sample size may be needed to detect more significant changes in expression. Identifying the biological causation of these differentially expressed genes may involve further analysis, such as functional enrichment analysis or pathway analysis, to determine the biological processes and pathways that are affected by these genes. This information can provide insight into the molecular mechanisms underlying the host response to COVID-19 and may inform the development of new treatments or interventions.

3. Explain the difference between P-value and FDR?

Answer:

Both p-value and FDR are statistical measures used to evaluate the significance of a hypothesis test. However, they differ in their interpretation and the way they control for multiple testing. A p-value is the probability of obtaining a test statistic as extreme as or more extreme than the observed value, assuming the null hypothesis is true. In other words, it is the probability of observing the data, or more extreme data, given that the null hypothesis is true. A lower p-value indicates stronger evidence against the null hypothesis, and a p-value less than the significance level (usually set at 0.05) is generally considered statistically significant. However, a significant p-value does not necessarily imply a clinically or scientifically significant result. On the other hand, FDR (false discovery rate) is a statistical method that controls for the proportion of false positives among the results that are declared significant. It is the expected proportion of false positives among all the features declared significant. FDR is a more conservative approach to multiple testing correction than the traditional Bonferroni correction because it allows for more statistically significant results to be detected while still controlling the false discovery rate.

Part e- Gene Ontology enrichment analysis

In the final step, to obtain the distinct biological functions present in cancer samples, use the GOseq package in R to perform Gene Ontology (GO) enrichment analysis. The GO enrichment analysis statistically assesses the overrepresentation of differentially expressed genes in common GO functional branches. To accomplish this step, select the genes with FDR < 0.1 and an absolute value of Log2FoldChange > 1.5. Please answer the following question after completing this step.

Answer:

To accomplish this step, select the genes with FDR < 0.1 and an absolute value of Log2FoldChange > 1.5:

***Note: My main matrix faced with lack of gene length. So at first I tried bellow code:**

"# Load the required library

```
# Load the required library
library(biomaRt)

# Define a helper function to retrieve gene length from Ensembl
get_gene_length <- function(ensembl, ensembl_ids) {
  gene_info <- NULL
  for (i in seq_along(ensembl_ids)) {
    ensembl_id <- ensembl_ids[i]
    gene_length <- try(getBM(attributes = "transcript_length",
                             filters = "ensembl_gene_id",
                             values = ensembl_id,
                             mart = ensembl),
                       silent = TRUE)
    if (!inherits(gene_length, "try-error")) {
      gene_info <- rbind(gene_info, data.frame(ENSEMBL_GENE_ID = ensembl_id,
                                               Gene_Length = gene_length))
    }
  }
  return(gene_info)
}

# Connect to the Ensembl database using the biomaRt package
```

ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")

 # Read in the CSV file

counts <- read.csv("my_counts.csv", header = TRUE, row.names = "COUNTS")

 # Replace any NA values with 0

counts[is.na(counts)] <- 0

 # Retrieve gene length information from Ensembl using biomaRt

ensembl_ids <- rownames(counts)

gene_info <- get_gene_length(ensembl, ensembl_ids)

 # Merge gene length information with the counts data frame using gene IDs

counts_with_length <- merge(counts, gene_info, by.x = "COUNTS", by.y = "ENSEMBL_GENE_ID")

 # Write the updated counts data frame to the same CSV file

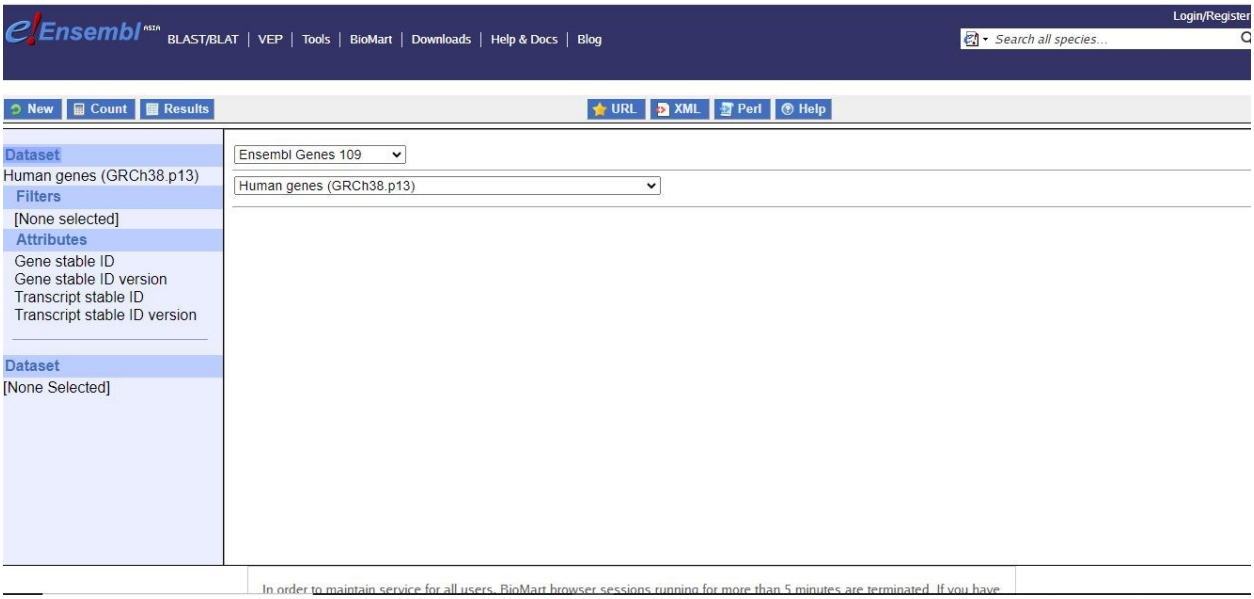
write.csv(counts_with_length, file = "my_counts.csv")

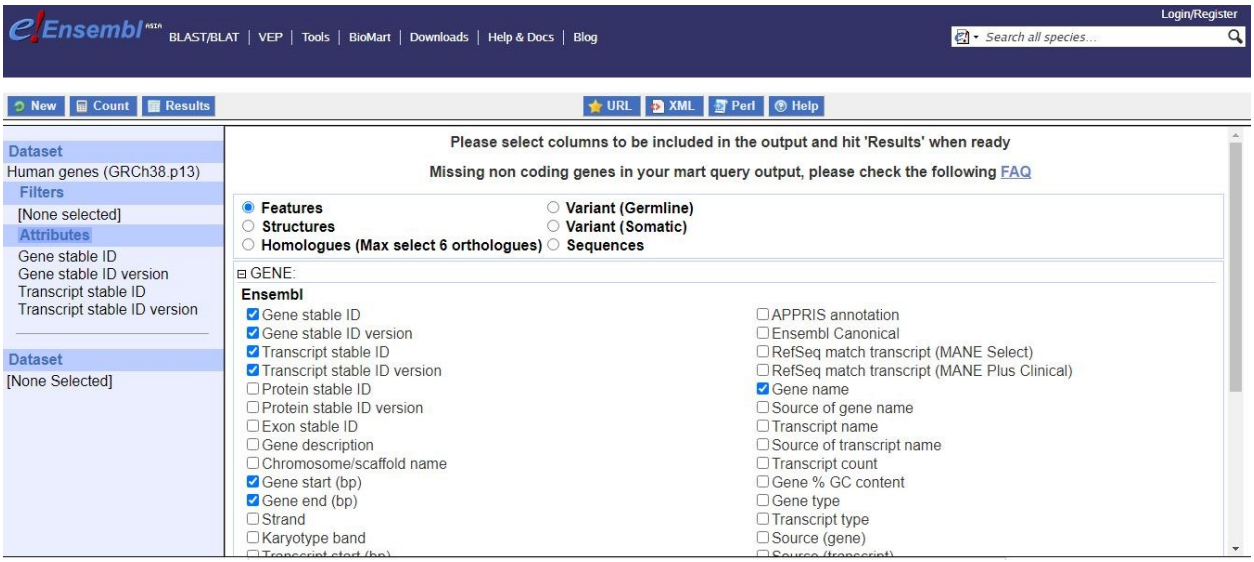
Due to internet problem above code I got error:

```
> ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")
Error in curl::curl_fetch_memory(url, handle = handle) :
  Timeout was reached: [www.ensembl.org:443] Operation timed out after 10006 milliseconds with 0 bytes received
```

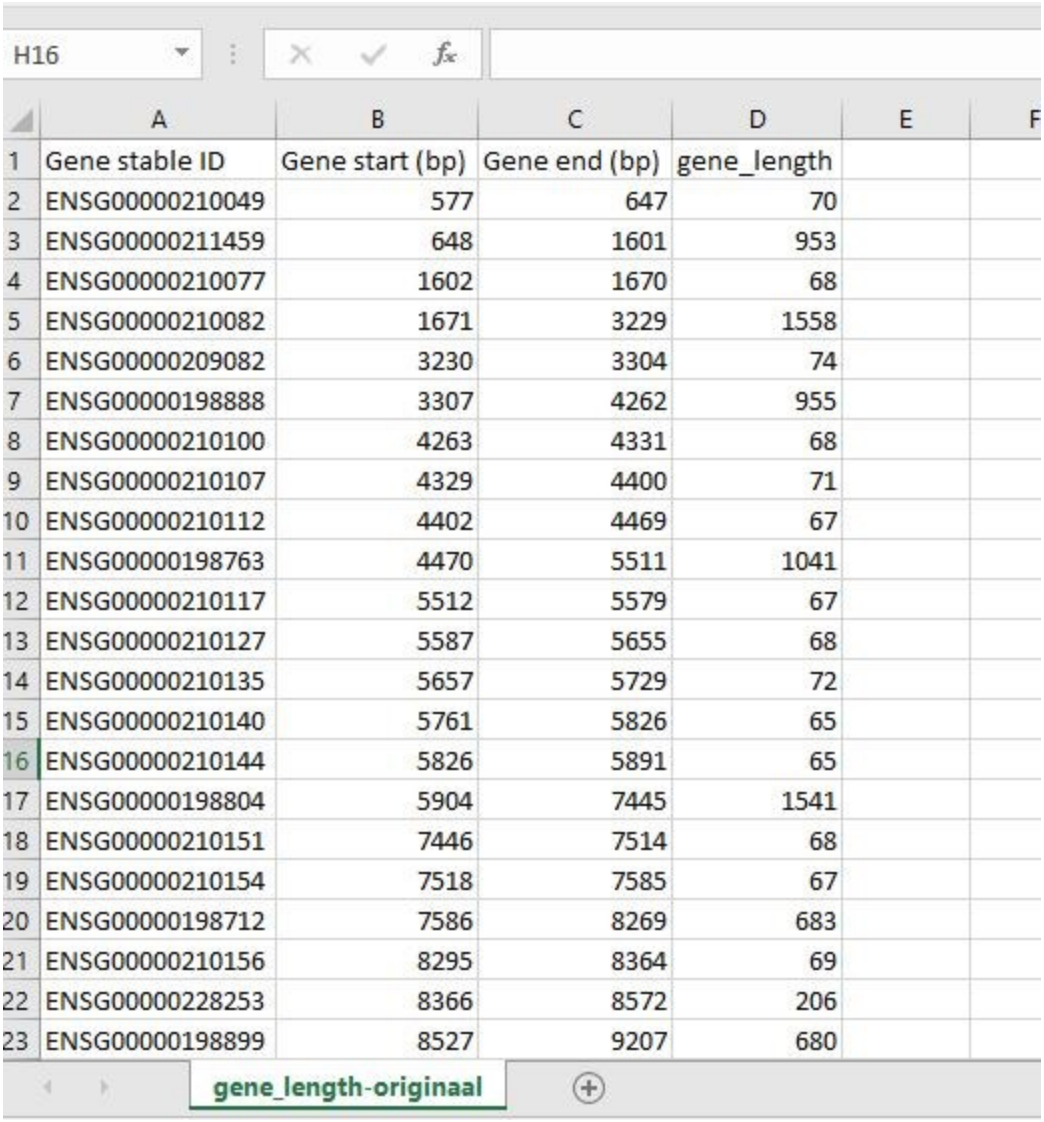
So, I decided to calculate gene length from the link:

http://asia.ensembl.org/biomart/martview/2b31973d09d9d645870a7cfde4c03598





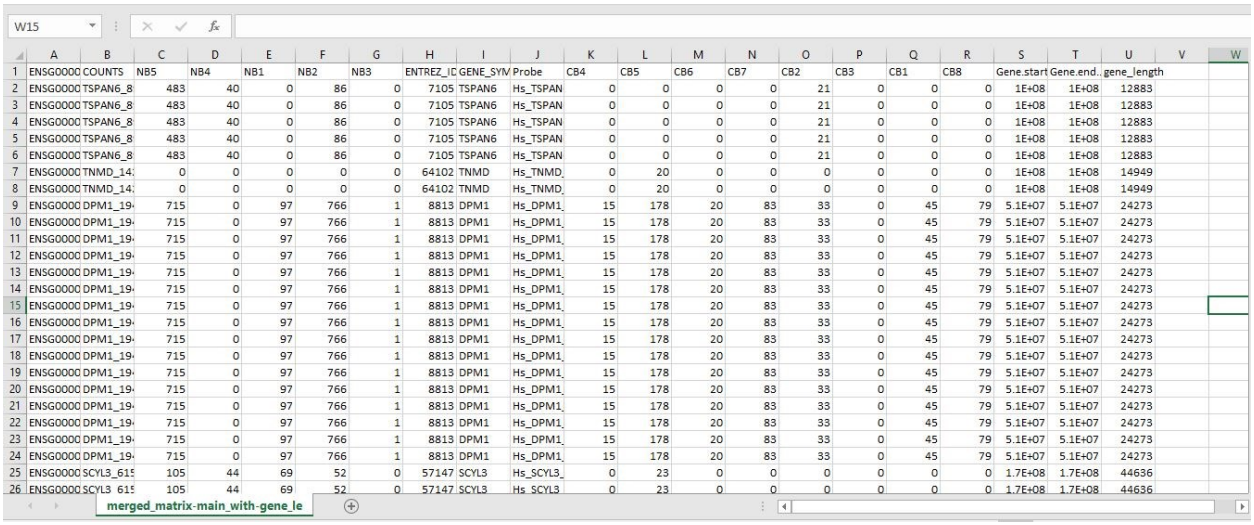
Then, from results part I save the output and in excel, by subtraction Gene end (bp) from Gene start (bp), gene lengths were calculated.



	A	B	C	D	E	F
1	Gene stable ID	Gene start (bp)	Gene end (bp)	gene_length		
2	ENSG00000210049	577	647	70		
3	ENSG00000211459	648	1601	953		
4	ENSG00000210077	1602	1670	68		
5	ENSG00000210082	1671	3229	1558		
6	ENSG00000209082	3230	3304	74		
7	ENSG00000198888	3307	4262	955		
8	ENSG00000210100	4263	4331	68		
9	ENSG00000210107	4329	4400	71		
10	ENSG00000210112	4402	4469	67		
11	ENSG00000198763	4470	5511	1041		
12	ENSG00000210117	5512	5579	67		
13	ENSG00000210127	5587	5655	68		
14	ENSG00000210135	5657	5729	72		
15	ENSG00000210140	5761	5826	65		
16	ENSG00000210144	5826	5891	65		
17	ENSG00000198804	5904	7445	1541		
18	ENSG00000210151	7446	7514	68		
19	ENSG00000210154	7518	7585	67		
20	ENSG00000198712	7586	8269	683		
21	ENSG00000210156	8295	8364	69		
22	ENSG00000228253	8366	8572	206		
23	ENSG00000198899	8527	9207	680		

In the following, the problem was the new .csv file order was not the same with my main matrix, so I merged them based on ENSEMBL_GENE_ID.

```
# Read in the two CSV files
counts <- read.csv("merged_all_main_matrix__NB_CB.csv", header=TRUE)
mart_export <- read.csv("gene_length-originaal.csv", header=TRUE)

# Merge the two dataframes based on the common column
merged_df <- merge(counts, mart_export, by.x="ENSEMBL_GENE_ID", by.y="Gene.stable.ID")

# Save the merged dataframe as a new CSV file
write.csv(merged_df, file="merged_matrix-main_with-gene_length.csv", row.names=FALSE)
```



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ENSG0000	COUNTS	NB5	NB4	NB1	NB2	NB3	ENTREZ_ID	GENE_SYM	Probe	CB4	CB5	CB6	CB7	CB2	CB3	CB1	CB8	Gene.start	Gene.end.	gene_length		
2	ENSG0000	TSPAN6_8	483	40	0	86	0	7105	TSPAN6	Hs_TSPAN	0	0	0	0	21	0	0	0	1E+08	1E+08	12883		
3	ENSG0000	TSPAN6_8	483	40	0	86	0	7105	TSPAN6	Hs_TSPAN	0	0	0	0	21	0	0	0	1E+08	1E+08	12883		
4	ENSG0000	TSPAN6_8	483	40	0	86	0	7105	TSPAN6	Hs_TSPAN	0	0	0	0	21	0	0	0	1E+08	1E+08	12883		
5	ENSG0000	TSPAN6_8	483	40	0	86	0	7105	TSPAN6	Hs_TSPAN	0	0	0	0	21	0	0	0	1E+08	1E+08	12883		
6	ENSG0000	TSPAN6_8	483	40	0	86	0	7105	TSPAN6	Hs_TSPAN	0	0	0	0	21	0	0	0	1E+08	1E+08	12883		
7	ENSG0000	TNMD_14	0	0	0	0	0	64102	TNMD	Hs_TNMD_	0	20	0	0	0	0	0	0	1E+08	1E+08	14949		
8	ENSG0000	TNMD_14	0	0	0	0	0	64102	TNMD	Hs_TNMD_	0	20	0	0	0	0	0	0	1E+08	1E+08	14949		
9	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
10	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
11	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
12	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
13	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
14	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
15	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
16	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
17	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
18	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
19	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
20	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
21	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
22	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
23	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
24	ENSG0000	DPM1_19	715	0	97	766	1	8813	DPM1	Hs_DPM1_	15	178	20	83	33	0	45	79	5.1E+07	5.1E+07	24273		
25	ENSG0000	SCYL3_615	105	44	69	52	0	57147	SCYL3	Hs_SCYL3_	0	23	0	0	0	0	0	0	1.7E+08	1.7E+08	44636		
26	ENSG0000	SCYL3_615	105	44	69	52	0	57147	SCYL3	Hs_SCYL3_	0	23	0	0	0	0	0	0	1.7E+08	1.7E+08	44636		

merged_matrix-main_with-gene_le

Then, to select genes with FDR < 0.1 and an absolute value of Log2FoldChange > 1.5:

```r
# Load the required library
library(edgeR)
```

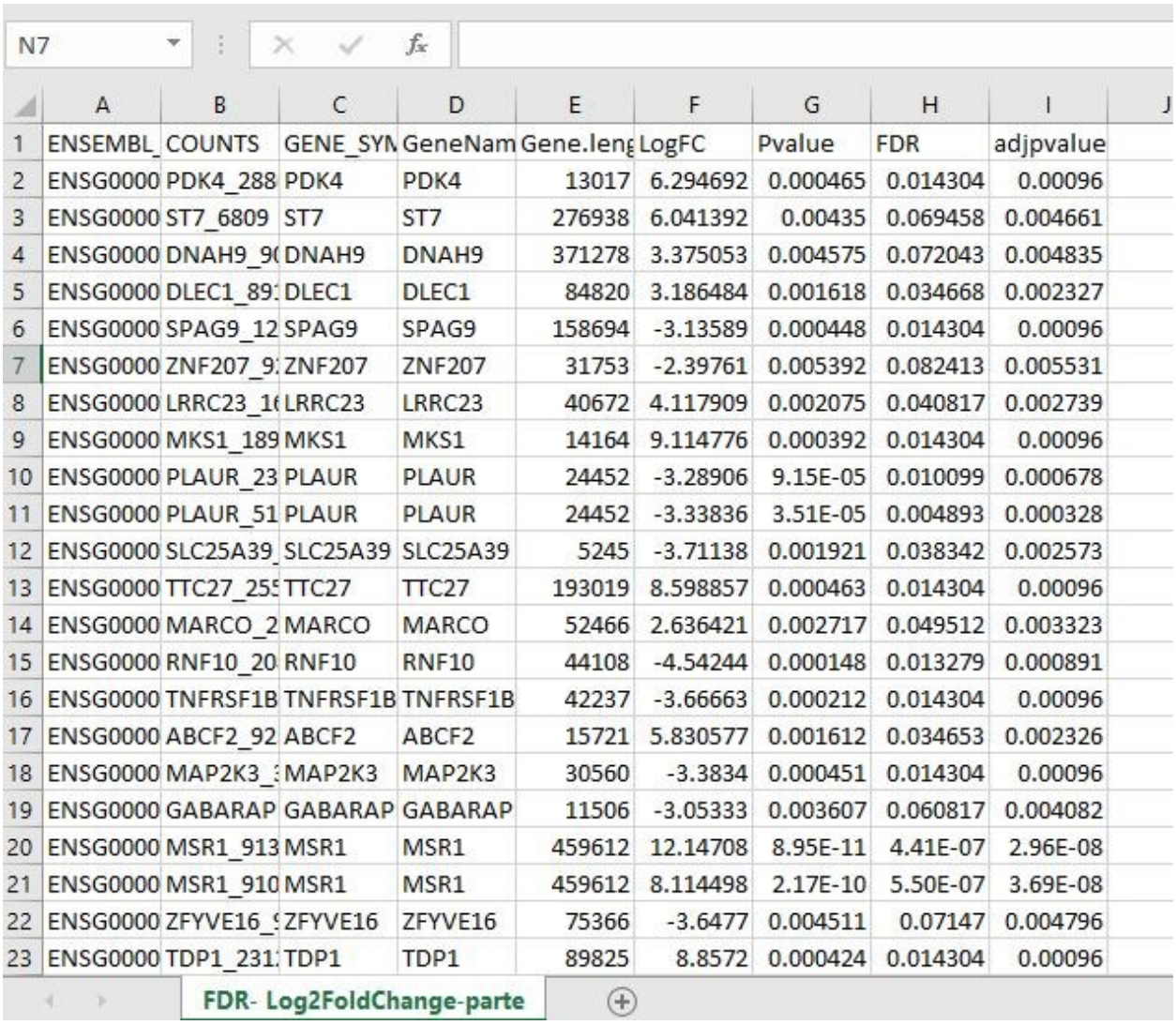
```
## Loading required package: limma
```

```r
# Read in the count matrix from the CSV file
counts <- read.csv("merged_matrix-main_with-gene_length.csv", header=TRUE, row.names=1)

# Replace any NA values with 0
counts[is.na(counts)] <- 0

# Write the modified count matrix to a new CSV file
write.csv(counts, file="modify-merge-main-matrix.csv", row.names=TRUE)

# Print a message indicating that the file has been saved
cat("Modified count matrix saved to modify-merge-main-matrix.csv\n")
```

```
## Modified count matrix saved to modify-merge-main-matrix.csv
```

```r
# Identify any non-numeric values in each column of the data frame
non_numeric_cols <- sapply(counts, function(x) any(grepl("[^0-9.]", x)))

# Exclude the non-numeric columns from the data frame
gene_expression_numeric <- counts[, !non_numeric_cols]

# Define the experimental design and create the DGEList object
group <- factor(c(rep("normal", 5), rep("covid", 8)))
dge <- DGEList(counts=as.matrix(gene_expression_numeric), group=group)

# Identify which columns correspond to normal samples and which ones correspond to COVID samples
normal_cols <- c("NB5", "NB4", "NB1", "NB2", "NB3")
covid_cols <- c("CB4", "CB5", "CB6", "CB7", "CB2", "CB3", "CB1", "CB8")

# Filter the genes that have low expression and/or low variability
keep <- rowSums(cpm(dge[, normal_cols]) >= 1) >= 3
dge <- dge[keep, , keep.lib.sizes=FALSE]
dge <- calcNormFactors(dge)
design <- model.matrix(~group)
dge <- estimateDisp(dge, design)
fit <- glmQLFit(dge, design)
lrt <- glmQLFTest(fit, coef=2)
p <- p.adjust(lrt$table$PValue, method="BH")
keep <- p < 0.1 & abs(lrt$table$logFC) > 1.5
dge_selected <- dge[keep, ]
de_genes_matrix <- cbind(COUNTS = rowSums(dge_selected$counts),
                         LogFC = lrt$table$logFC[keep],
                         LogCPM = log2(cpm(dge_selected)+0.5),
                         LR = lrt$table$logLik[keep],
                         Pvalue = lrt$table$PValue[keep],
                         FDR = p[keep],
                         adjpvalue = p.adjust(lrt$table$PValue[keep], method = "BH"),
                         GENE_SYMBOL = rownames(dge_selected),
                         Gene_length = rownames(dge_selected))
# Save the selected genes to a new CSV file
write.csv(de_genes_matrix, file="FDR-Log2FoldChange-parte.csv", row.names=FALSE)

# Print a message indicating that the file has been saved
cat("Selected genes matrix saved to selected_genes_matrix.csv\n")
```

```
## Selected genes matrix saved to selected_genes_matrix.csv
```



	A	B	C	D	E	F	G	H	I
1	ENSEMBL_	COUNTS	GENE_SYN	GeneNam	Gene.leng	LogFC	Pvalue	FDR	adjpvalue
2	ENSG0000	PDK4_288	PDK4	PDK4	13017	6.294692	0.000465	0.014304	0.00096
3	ENSG0000	ST7_6809	ST7	ST7	276938	6.041392	0.00435	0.069458	0.004661
4	ENSG0000	DNAH9_9(	DNAH9	DNAH9	371278	3.375053	0.004575	0.072043	0.004835
5	ENSG0000	DLEC1_89:	DLEC1	DLEC1	84820	3.186484	0.001618	0.034668	0.002327
6	ENSG0000	SPAG9_12	SPAG9	SPAG9	158694	-3.13589	0.000448	0.014304	0.00096
7	ENSG0000	ZNF207_9:	ZNF207	ZNF207	31753	-2.39761	0.005392	0.082413	0.005531
8	ENSG0000	LRRC23_1(	LRRC23	LRRC23	40672	4.117909	0.002075	0.040817	0.002739
9	ENSG0000	MKS1_189	MKS1	MKS1	14164	9.114776	0.000392	0.014304	0.00096
10	ENSG0000	PLAUR_23	PLAUR	PLAUR	24452	-3.28906	9.15E-05	0.010099	0.000678
11	ENSG0000	PLAUR_51	PLAUR	PLAUR	24452	-3.33836	3.51E-05	0.004893	0.000328
12	ENSG0000	SLC25A39_	SLC25A39	SLC25A39	5245	-3.71138	0.001921	0.038342	0.002573
13	ENSG0000	TTC27_255	TTC27	TTC27	193019	8.598857	0.000463	0.014304	0.00096
14	ENSG0000	MARCO_2	MARCO	MARCO	52466	2.636421	0.002717	0.049512	0.003323
15	ENSG0000	RNF10_20	RNF10	RNF10	44108	-4.54244	0.000148	0.013279	0.000891
16	ENSG0000	TNFRSF1B	TNFRSF1B	TNFRSF1B	42237	-3.66663	0.000212	0.014304	0.00096
17	ENSG0000	ABCF2_92	ABCF2	ABCF2	15721	5.830577	0.001612	0.034653	0.002326
18	ENSG0000	MAP2K3_3	MAP2K3	MAP2K3	30560	-3.3834	0.000451	0.014304	0.00096
19	ENSG0000	GABARAP	GABARAP	GABARAP	11506	-3.05333	0.003607	0.060817	0.004082
20	ENSG0000	MSR1_913	MSR1	MSR1	459612	12.14708	8.95E-11	4.41E-07	2.96E-08
21	ENSG0000	MSR1_910	MSR1	MSR1	459612	8.114498	2.17E-10	5.50E-07	3.69E-08
22	ENSG0000	ZFYVE16_9	ZFYVE16	ZFYVE16	75366	-3.6477	0.004511	0.07147	0.004796
23	ENSG0000	TDP1_231	TDP1	TDP1	89825	8.8572	0.000424	0.014304	0.00096

FDR- Log2FoldChange-parte

Now want to find which of these genes are meaningful and acceptable (DEG genes).

So, with considering FDR < 0.05 and LogFC > 1.5, I collected up regulated genes and FDR < 0.05 and LogFC> -1.5, I collected down regulated genes.

```
count_data <- read.csv("FDR- Log2FoldChange-parte.csv", header = TRUE)
# Filter the data based on significance thresholds and retrieve FDR and gene length
up_regulated_genes <- count_data[which(count_data$FDR < 0.05 & count_data$LogFC > 1.5), c("ENSEMBL_GENE_ID", "FDR", "Gene.length")]
down_regulated_genes <- count_data[which(count_data$FDR < 0.05 & count_data$LogFC < -1.5), c("ENSEMBL_GENE_ID", "FDR", "Gene.length")]

# Save the gene lists to CSV files
write.csv(up_regulated_genes, file = "up_regulated_genes.csv", row.names = FALSE)
write.csv(down_regulated_genes, file = "down_regulated_genes.csv", row.names = FALSE)

# Print a message indicating that the files have been saved
cat("Up-regulated genes saved to up_regulated_genes.csv\n")
```

```
## Up-regulated genes saved to up_regulated_genes.csv
```

```
cat("Down-regulated genes saved to down_regulated_genes.csv\n")
```

```
## Down-regulated genes saved to down_regulated_genes.csv
```

To do further preparation to start Gene Ontology (GO) enrichment analysis I removed duplicated genes:

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

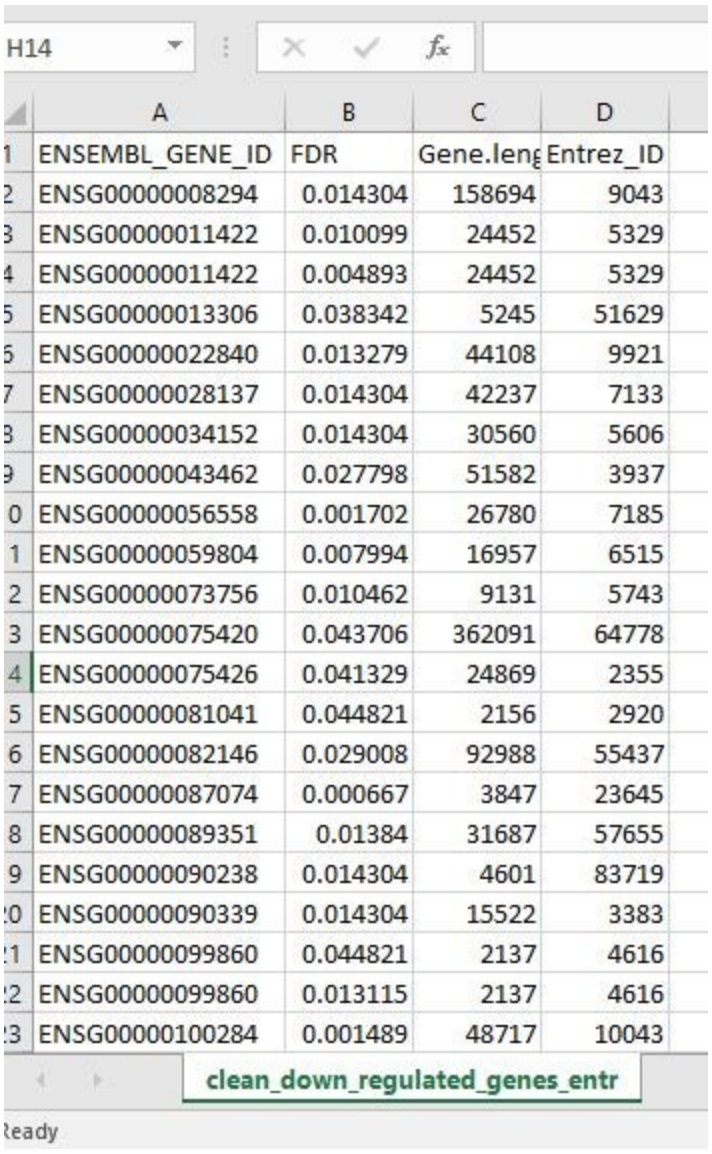
```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# Load data
df1 <- read.csv("up_regulated_genes.csv")
df2 <- read.csv("down_regulated_genes.csv")

# Check for duplicates in df1
df1_dup <- df1[duplicated(df1$Entrez_ID),]
if (nrow(df1_dup) > 0) {
  # Remove duplicates from df1
  df1 <- distinct(df1, Entrez_ID, .keep_all=TRUE)
  message(paste0("Removed ", nrow(df1_dup), " duplicate(s) from df1"))
}

# Check for duplicates in df2
df2_dup <- df2[duplicated(df2$Entrez_ID),]
if (nrow(df2_dup) > 0) {
  # Remove duplicates from df2
  df2 <- distinct(df2, Entrez_ID, .keep_all=TRUE)
  message(paste0("Removed ", nrow(df2_dup), " duplicate(s) from df2"))
}
# Check for duplicates in df1
if (any(duplicated(df1$Entrez_ID))) {
  message("df1 contains duplicates")
}

# Check for duplicates in df2
if (any(duplicated(df2$Entrez_ID))) {
  message("df2 contains duplicates")
}
# Save cleaned data frames to CSV files
write.csv(df1, "clean_up_regulated_genes.csv", row.names=FALSE)
write.csv(df2, "clean_down_regulated_genes.csv", row.names=FALSE)
```

Then, due to goseq package need Entrez gene IDs I add this parameter to my input files:

```r
library(biomaRt)

# Load data frames from CSV files
df1 <- read.csv("up_regulated_genes.csv")
df2 <- read.csv("down_regulated_genes.csv")

# Connect to BioMart database
ensembl <- useMart("ensembl", dataset = "hsapiens_gene_ensembl")

# Get mapping between ENSEMBL_GENE_ID and Entrez_ID
mapping <- getBM(attributes=c("ensembl_gene_id", "entrezgene_id"),
                 filters="ensembl_gene_id",
                 values=unique(c(df1$ENSEMBL_GENE_ID, df2$ENSEMBL_GENE_ID)),
                 mart=ensembl)

# Merge mapping with cleaned data frames
df1 <- merge(df1, mapping, by.x="ENSEMBL_GENE_ID", by.y="ensembl_gene_id", all.x=TRUE)
df2 <- merge(df2, mapping, by.x="ENSEMBL_GENE_ID", by.y="ensembl_gene_id", all.x=TRUE)

# Rename Entrez_ID column
colnames(df1)[ncol(df1)] <- "Entrez_ID"
colnames(df2)[ncol(df2)] <- "Entrez_ID"

# Save updated data frames to CSV files
write.csv(df1, "clean_up_regulated_genes_entrez-2.csv", row.names=FALSE)
write.csv(df2, "clean_down_regulated_genes_entrez-2.csv", row.names=FALSE)
```

The output is:

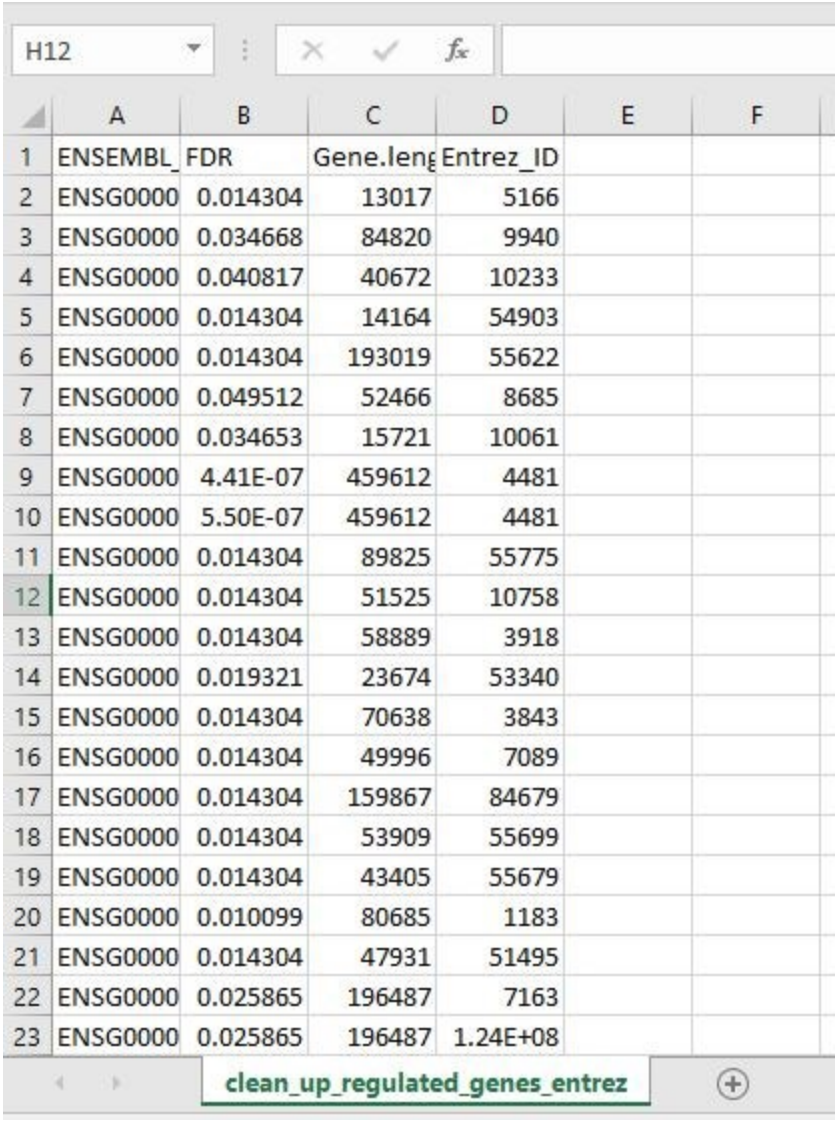


H14

	A	B	C	D
1	ENSEMBL_GENE_ID	FDR	Gene.leng	Entrez_ID
2	ENSG00000008294	0.014304	158694	9043
3	ENSG00000011422	0.010099	24452	5329
4	ENSG00000011422	0.004893	24452	5329
5	ENSG00000013306	0.038342	5245	51629
6	ENSG00000022840	0.013279	44108	9921
7	ENSG00000028137	0.014304	42237	7133
8	ENSG00000034152	0.014304	30560	5606
9	ENSG00000043462	0.027798	51582	3937
10	ENSG00000056558	0.001702	26780	7185
11	ENSG00000059804	0.007994	16957	6515
12	ENSG00000073756	0.010462	9131	5743
13	ENSG00000075420	0.043706	362091	64778
14	ENSG00000075426	0.041329	24869	2355
15	ENSG00000081041	0.044821	2156	2920
16	ENSG00000082146	0.029008	92988	55437
17	ENSG00000087074	0.000667	3847	23645
18	ENSG00000089351	0.01384	31687	57655
19	ENSG00000090238	0.014304	4601	83719
20	ENSG00000090339	0.014304	15522	3383
21	ENSG00000099860	0.044821	2137	4616
22	ENSG00000099860	0.013115	2137	4616
23	ENSG00000100284	0.001489	48717	10043

clean_down_regulated_genes_entr

Ready



H12

	A	B	C	D	E	F
1	ENSEMBL_	FDR	Gene.leng	Entrez_ID		
2	ENSG0000	0.014304	13017	5166		
3	ENSG0000	0.034668	84820	9940		
4	ENSG0000	0.040817	40672	10233		
5	ENSG0000	0.014304	14164	54903		
6	ENSG0000	0.014304	193019	55622		
7	ENSG0000	0.049512	52466	8685		
8	ENSG0000	0.034653	15721	10061		
9	ENSG0000	4.41E-07	459612	4481		
10	ENSG0000	5.50E-07	459612	4481		
11	ENSG0000	0.014304	89825	55775		
12	ENSG0000	0.014304	51525	10758		
13	ENSG0000	0.014304	58889	3918		
14	ENSG0000	0.019321	23674	53340		
15	ENSG0000	0.014304	70638	3843		
16	ENSG0000	0.014304	49996	7089		
17	ENSG0000	0.014304	159867	84679		
18	ENSG0000	0.014304	53909	55699		
19	ENSG0000	0.014304	43405	55679		
20	ENSG0000	0.010099	80685	1183		
21	ENSG0000	0.014304	47931	51495		
22	ENSG0000	0.025865	196487	7163		
23	ENSG0000	0.025865	196487	1.24E+08		

clean_up_regulated_genes_entrez

At last, I tried bellowing code but I was unable to resolve the error within the given deadline.

```
# Load the required library
library(edgeR)

# Read in the count matrix from the CSV file
counts <- read.csv("merged_all_main_matrix__NB_CB.csv", header=TRUE, row.names=1)

# Replace any NA values with 0
counts[is.na(counts)] <- 0

# Write the modified count matrix to a new CSV file
write.csv(counts, file="modify-merge-main-matrix.csv", row.names=TRUE)
```

```
# Print a message indicating that the file has been saved
cat("Modified count matrix saved to modify-merge-main-matrix.csv\n")

# Identify any non-numeric values in each column of the data frame
non_numeric_cols <- sapply(counts, function(x) any(grepl("[^0-9.]", x)))

# Exclude the non-numeric columns from the data frame
gene_expression_numeric <- counts[, !non_numeric_cols]

# Define the experimental design and create the DGEList object
group <- factor(c(rep("normal", 5), rep("covid", 8)))
dge <- DGEList(counts=as.matrix(gene_expression_numeric), group=group)

# Identify which columns correspond to normal samples and which ones correspond to COVID
samples
normal_cols <- c("NB5", "NB4", "NB1", "NB2", "NB3")
covid_cols <- c("CB4", "CB5", "CB6", "CB7", "CB2", "CB3", "CB1", "CB8")

# Filter the genes that have low expression and/or low variability
keep <- rowSums(cpm(dge[, normal_cols]) >= 1) >= 3
dge <- dge[keep, , keep.lib.sizes=FALSE]
dge <- calcNormFactors(dge)
design <- model.matrix(~group)
dge <- estimateDisp(dge, design)
fit <- glmQLFit(dge, design)
lrt <- glmQLFTest(fit, coef=2)
p <- p.adjust(lrt$table$PValue, method="BH")
keep <- p < 0.1 & abs(lrt$table$logFC) > 1.5
dge_selected <- dge[keep, ]
de_genes_matrix <- cbind(COUNTS = rowSums(dge_selected$counts),
                         LogFC = lrt$table$logFC[keep],
                         LogCPM = log2(cpm(dge_selected)+0.5),
                         LR = lrt$table$logLik[keep],
                         Pvalue = lrt$table$PValue[keep],
                         FDR = p[keep],
                         adjpvalue = p.adjust(lrt$table$PValue[keep], method = "BH"),
                         GENE_SYMBOL = rownames(dge_selected))

# Save the selected genes to a new CSV file
write.csv(de_genes_matrix, file="selected_genes_matrix.csv", row.names=FALSE)

# Print a message indicating that the file has been saved
cat("Selected genes matrix saved to selected_genes_matrix.csv\n")
```