

Project:

(From Raw Reads to Gene Insights)

RNA-Seq Analysis of Human LCLs
(Lymphoblastoid cell lines)

Subtitle :

An exploration of gene expression in Yoruba individuals using public RNA-Seq data

Course :

Bioinformatics with Advance AI (Pulse, AIIMS Delhi)

Date :

15 July 2025

Presented by :

Muskan Kashyap (muskan0072003@gmail.com)

Project Overview :

Organism: *Homo sapiens* (Human)

Sample: YRI Lymphoblastoid Cell Line

SRA Project ID: PRJNA207555

SRA Run ID: SRR19762473



Why Chose This Dataset:

✓ **Population-Specific Insight**

The dataset represents Yoruba individuals in Ibadan, Nigeria (YRI), which is one of the most studied African populations in genomics — valuable for population genetics and expression variability.

✓ **Well-Characterized Samples**

Lymphoblastoid cell lines (LCLs) are widely used in functional genomics because they're standardized, renewable, and consistent across studies.

✓ **Part of the 1000 Genomes Project**

This makes the data reliable and extensively annotated, allowing for comparison with genomic variation data.

✓ **Publicly Available & High Quality**

The data is open-access, peer-reviewed, and associated with detailed metadata, which supports reproducibility.

✓ **Suitable for RNA-Seq Analysis**

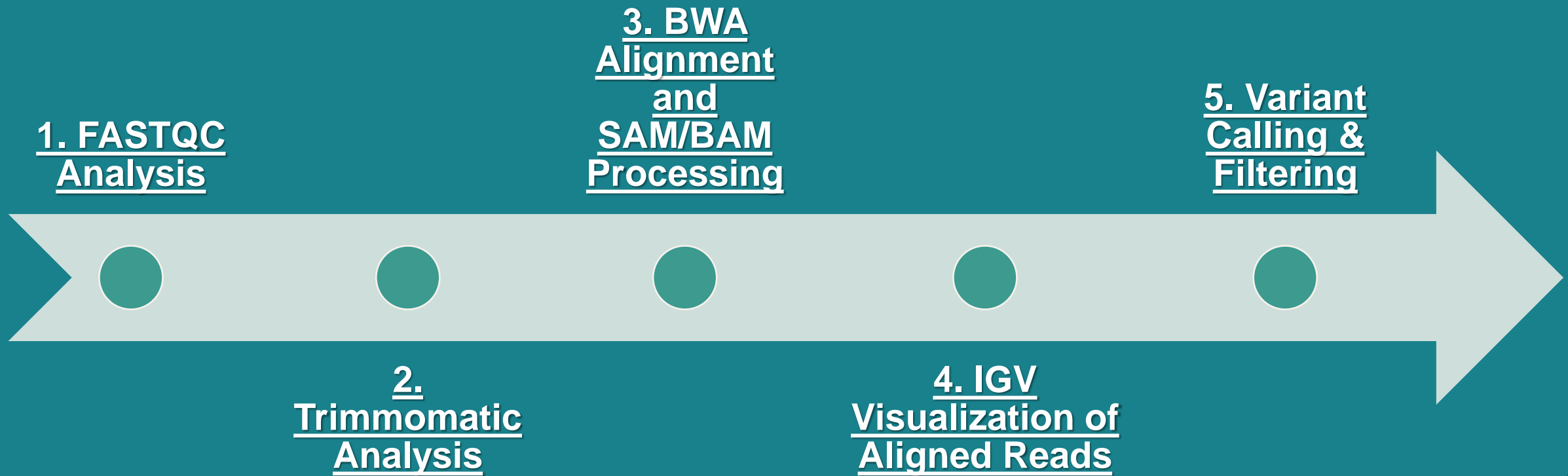
The data includes paired-end RNA-seq reads — ideal for studying gene expression, alternative splicing, and transcript assembly.

Workflow Steps :

1. **Download SRR files** from NCBI SRA
2. **Convert** .sra → .fastq using fasterq-dump
3. **Quality check** with **FASTQC** (before trimming)
4. **Trim** low-quality reads and adapters with **Trimmomatic**
5. **Re-check quality** with FASTQC (after trimming)
6. **Align reads** to human genome (**hg38**) using **BWA**
7. **Convert & sort** alignment files using **SAMtools**
8. **Visualize** alignments in **IGV**
9. Perform **variant calling**
10. **Reflect** and report biological insights



Index





FASTQC Analysis

Quality Assessment of Raw Sequencing Reads

Step 1: Download the data using prefetch

➤ `prefetch -O ./test_fqc SRR19762473`

- **Tool:** *prefetch* (from NCBI SRA Toolkit)
- **Purpose:** Downloads SRA data from NCBI to your local computer.
- `-O ./test_fqc`: Specifies the output directory (*test_fqc*) where the data should be saved.
- *SRR19762473*: The accession number of the sample to be downloaded.

Output:

- It confirmed a successful download using HTTPS.
- The data is saved in `./test_fqc/SRR19762473` in SRA's proprietary format (*.sra*)

Step 2: Convert SRA to FASTQ using fasterq-dump

➤ `fasterq-dump ./test_fqc/SRR19762473`

- **Tool:** *fasterq-dump* (also from SRA Toolkit)
- **Purpose:** Converts the *.sra* file to standard **FASTQ** format used in downstream tools.
- Since this is **paired-end data**, it generated two files:
 - SRR19762473_1.fastq* (forward reads)
 - SRR19762473_2.fastq* (reverse reads)

Output:

spots read : 23,278,323

reads read : 46,556,646

- **"Spots"**: Each spot usually represents one fragment (i.e. a DNA insert).
- Since this is paired-end, each spot has 2 reads, hence 46 million reads from 23 million spots.

Step 3: Moving the FASTQ files into the correct folder

➤ `cd ..`

```
mv SRR19762473_*.fastq ~/test_fqc/
```

- Moved the two FASTQ files into *test_fqc* folder.
- Now *test_fqc* folder has:
 - i. *SRR19762473* (original downloaded .sra)
 - ii. *SRR19762473_1.fastq*
 - iii. *SRR19762473_2.fastq*

Step 4: Quality Check using FastQC

➤ `fastqc SRR19762473_1.fastq SRR19762473_2.fastq`

- **Tool:** FastQC
- **Purpose:** Performs a **quality check** on the raw FASTQ files. This includes –
 - i. Per base sequence quality
 - ii. GC content
 - iii. Adapter content
 - iv. Overrepresented sequences
 - v. Sequence duplication levels

Output:

- The progress lines like:

Approx 25% complete for SRR19762473_1.fastq

indicate the analysis is running.

- *null* lines at the top can be ignored — possibly just standard error output.

After analysis, **FastQC** generates:

➤ An *.html* report for each input file (*SRR19762473_1_fastqc.html*)

i. <https://muskan-fastqc-report-1.netlify.app/>

ii. <https://muskan-fastqc-report-2.netlify.app/>

➤ A *.zip* file with raw data.

Summary of Tools Used:

Step	Tool	Purpose
1	<i>prefetch</i>	Download <i>.sra</i> file from SRA database
2	<i>fasterq-dump</i>	Convert <i>.sra</i> to <i>.fastq</i>
3	<i>mv</i>	Move files to organize them
4	<i>FastQC</i>	Assess quality of raw reads



Trimmomatic Analysis

Adapter Removal and Quality Trimming of Paired-End Reads

Step 1: Update & Install Trimmomatic

- `sudo apt update`
- `sudo apt install trimmomatic`
 - `sudo apt update`: Updates the list of available packages.
 - `sudo apt install trimmomatic`: Installs Trimmomatic, a tool for trimming sequencing reads.
 - By default, Trimmomatic was installed in a `.jar` file: `/usr/share/java/trimmomatic.jar`.
- `java -jar /usr/share/java/trimmomatic.jar -version`
 - Use `java -jar` to run the `.jar` file because `trimmomatic` isn't a shell command by itself.
 - Output `0.39` confirms it's installed correctly.

Step 2: Downloaded the Adapter File

- `wget https://raw.githubusercontent.com/timflutre/trimmomatic/master/adapters/TruSeq3-PE.fa -O TruSeq3-PE.fa`
 - Trimmomatic needs the adapter file *TruSeq3-PE.fa*
 - This downloaded the adapter file to home directory.
- `realpath ~/TruSeq3-PE.fa`
 - This returned */home/piggyubuntu/TruSeq3-PE.fa*

Step 3: Run Trimmomatic

➤ `java -jar /usr/share/java/trimmomatic.jar PE -threads 4 \
SRR19762473_1.fastq SRR19762473_2.fastq \
SRR19762473_1_paired.fastq SRR19762473_1_unpaired.fastq \
SRR19762473_2_paired.fastq SRR19762473_2_unpaired.fastq \
ILLUMINACLIP:/home/piggyubuntu/TruSeq3-PE.fa:2:30:10 \
LEADING:3 TRAILING:3 SLIDINGWINDOW:4:20 MINLEN:36`

What Did This Trimmomatic Command Do?

Parameter	Description
<i>PE</i>	Paired-end mode
<i>-threads 4</i>	Use 4 CPU threads
Input files	<i>SRR19762473_1.fastq</i> and <i>SRR19762473_2.fastq</i>
Output (paired reads)	<i>*_1_paired.fastq</i> , <i>*_2_paired.fastq</i>
Output (unpaired reads)	<i>*_1_unpaired.fastq</i> , <i>*_2_unpaired.fastq</i>
<i>ILLUMINACLIP:</i>	Remove adapter contamination using adapter file
<i>LEADING:3</i>	Trim low-quality bases (below 3) from the start
<i>TRAILING:3</i>	Trim low-quality bases (below 3) from the end
<i>SLIDINGWINDOW:4:20</i>	Trim if average quality over 4 bases drops below 20
<i>MINLEN:36</i>	Drop reads shorter than 36 bases



Results of Trimmomatic

From the output:




Input Read Pairs: 23278323

Both Surviving: 19338632 (83.08%)

Forward Only Surviving: 3514321 (15.10%)

Reverse Only Surviving: 237256 (1.02%)

Dropped: 188114 (0.81%)

-  **Paired** reads retained: ~83%
-  **Unpaired** reads (only one mate survived): ~16%
-  **Dropped** reads: less than 1%

Step 4: Organized Output

Moved the trimmed output files to a new folder:

- `mkdir -p ~/test_trimmomatic`
- `mv ~/test_fqc/SRR19762473_*_paired.fastq ~/test_fqc/SRR19762473_*_unpaired.fastq
~/test_trimmomatic/`

Now `~/test_trimmomatic` contains:

- i. `*_1_paired.fastq`
- ii. `*_2_paired.fastq`
- iii. `*_1_unpaired.fastq`
- iv. `*_2_unpaired.fastq`

Step 5: Re-ran FastQC on Trimmed Data

➤ `fastqc SRR19762473_*_paired.fastq SRR19762473_*_unpaired.fastq`

This checks whether:

- ✓ Adapter contamination is gone
- ✓ Quality scores improved
- ✓ Low-quality reads were effectively trimmed

FastQC produces:

- .html reports for each file
- You can open them in a browser to visually inspect quality

FastQC produces:

1. Paired reads:

- SRR19762473_1_paired.fastq

<https://muskan-srr19762473-1-paired-fastqc.netlify>

- SRR19762473_2_paired.fastq

<https://muskan-srr19762473-1-unpaired-fastqc.netlify.app>

2. Unpaired reads:

- SRR19762473_1_unpaired..fastq

<https://muskan-srr19762473-2-paired-fastqc.netlify.app>

- SRR19762473_2_unpaired.fastq

<https://muskan-srr19762473-2-unpaired-fastqc.netlify.app>

Summary

Step	Description
Installed Trimmomatic	For quality trimming and adapter removal
Downloaded Adapter File	Used <i>wget</i> to get <i>TruSeq3-PE.fa</i>
Trimmed Reads	Cleaned reads based on adapter and quality thresholds
Verified Output	Checked trimming results and moved files
Re-evaluated Quality	Ran FastQC again to confirm trimming improved quality



BWA Alignment and SAM/BAM Processing

**Mapping Trimmed Reads to the Human Reference Genome
(hg38)**

Step 1: Downloading the Reference Genome

- `wget https://hgdownload.soe.ucsc.edu/goldenPath/hg38/bigZips/hg38.fa.gz`
 - Downloads the **human reference genome (hg38)** in compressed FASTA format from UCSC.
 - *hg38.fa.gz* is a gzipped file containing all human chromosomes.
- `gunzip hg38.fa.gz`
 - Uncompresses the *hg38.fa.gz* file to *hg38.fa* , which is readable by BWA.

Step 2: Indexing the Reference Genome (Required by BWA)

➤ `bwa index hg38.fa`

- BWA builds an index from the reference genome, allowing it to **rapidly align short reads**.
- This creates several index files (*.amb*, *.ann*, *.bwt*, *.pac*, *.sa*) necessary for alignment.

Step 3: Aligning Paired-End Reads to the Reference Genome

➤ `bwa mem hg38.fa ~/test_trimmomatic/SRR19762473_1_paired.fastq
~/test_trimmomatic/SRR19762473_2_paired.fastq > SRR19762473_aligned.sam`

- Aligns **quality-trimmed paired-end reads** to the reference genome.
- Output is in **SAM format** (Sequence Alignment/Map), a human-readable alignment file.
- *bwa mem* is preferred for reads ≥ 70 bp, and outputs a SAM file showing where each read maps in the genome.

Step 4: Converting SAM to BAM (Binary Format) & Sorting the BAM File

➤ `samtools view -bS SRR19762473_aligned.sam > SRR19762473_aligned.bam`

- Converts the large SAM file to a BAM file (binary, compressed version of SAM).
- BAM is faster to process and compatible with most downstream tools.

➤ `samtools sort SRR19762473_aligned.bam -o SRR19762473_aligned_sorted.bam`

Sorts reads by their coordinates in the genome, which is essential for:

- Variant calling
- Visualization (e.g. in IGV)
- Duplicate marking

Step 5: Indexing the Sorted BAM File

➤ `samtools index SRR19762473_aligned_sorted.bam`

- Creates an index (.bai) that allows rapid access to any region of the BAM file.
- Required for tools like **IGV**, **bcftools**, or **GATK** to access alignments quickly.

✓ Result

- i. SRR19762473_aligned_sorted.bam → sorted alignment file
- ii. SRR19762473_aligned_sorted.bam.bai → index for that BAM

✓ Ready for **visual inspection** (e.g., IGV) or **variant calling** (e.g., with *bcftools* or GATK)



IGV Visualization of Aligned Reads

To visualize the aligned sequencing reads (*.bam*) on the human reference genome (**hg38**) using **Integrative Genomics Viewer (IGV)**

Steps Performed:

1. Launched IGV Desktop App -

Version: 2.19.5 with built-in Java

2. Loaded Reference Genome -

Selected **Human hg38** from IGV genome list

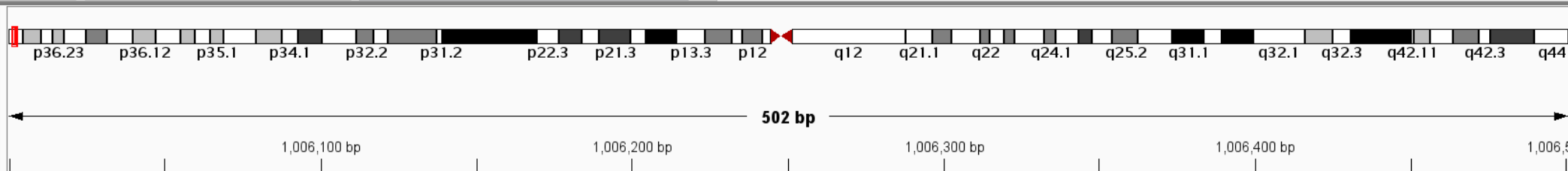
3. Loaded Aligned Data -

Loaded *SRR19762473_aligned_sorted.bam* and its index *.bai*

4. Navigation –

Jumped to region *chr1:1,006,000–1,006,500* using the search bar.

Zoomed in to observe aligned reads and mismatches.



SRR19762473_aligned_sorted.bam
overage

[0 - 10.00]

SRR19762473_aligned_sorted.bam

Refseq Select



□ IGV Visualization of Aligned Reads :

This image shows a zoomed-in view of **aligned sequencing reads** from the file *SRR19762473_aligned_sorted.bam* visualized using **IGV** over the **hg38 reference genome** at region *chr1:1,006,000–1,006,500*.

Interpretation:

- **Gray bars** represent individual reads successfully aligned to the reference genome.
- A **red bar** indicates a **mismatch** (potential SNP or sequencing error) at that base position.
- The **bottom sequence track** shows the reference genome bases.
- The "**Coverage**" track (gray histogram above the reads) is visible but shallow, indicating **low depth** (~1–10 reads) at this locus.



Variant Calling & Filtering

Tool Used: BCFtools

Step 1: Variant Calling with bcftools mpileup and call

```
bcftools mpileup -Ou -f ~/bwa_align/hg38.fa  
~/bwa_align/SRR19762473_aligned_sorted.bam | \  
bcftools call -mv -Ob -o ~/bwa_align/SRR19762473_variants.bcf
```

What's happening?

- *bcftools mpileup* examines the **aligned reads** (in your BAM file) against the **reference genome** (*hg38.fa*) and identifies **potential variant sites**.
- The *.ou* flag outputs **uncompressed BCF** data to stream directly into the next command.
- The *bcftools call* command actually **calls the variants** — identifies whether the differences are **SNPs (Single Nucleotide Polymorphisms)** or **indels (insertions/deletions)**.

Output:

- *SRR19762473_variants.bcf* : A **compressed binary VCF (BCF)** file containing **all detected variants**, regardless of quality.

Step 2: Converting BCF to Human-Readable VCF

```
bcftools view ~/bwa_align/SRR19762473_variants.bcf -Ov -o  
~/bwa_align/SRR19762473_variants.vcf
```

What's happening?

- Converts the binary *.bcf* file into **standard VCF format** (*.vcf*), which is plain-text and easier to read or analyze further.
- *-ov* = Output VCF format, uncompressed.

Output:

- *SRR19762473_variants.vcf* : Contains all the raw variant calls in human-readable text format.

Step 3: Filtering High-Confidence Variants

```
bcftools filter -i 'QUAL>30 && DP>10' \  
~/bwa_align/SRR19762473_variants.vcf \  
-Ov -o ~/bwa_align/SRR19762473_variants_filtered.vcf
```

What's happening?

Filters out **low-quality or low-confidence variant calls** based on:

QUAL > 30: Variant quality score must be >30 (Phred-scaled, meaning ~99.9% accuracy).

DP > 10: Read **depth** (coverage) must be greater than 10 reads at that site.

-i : include variants that meet this expression.

Output:

SRR19762473_variants_filtered.vcf: Final set of **high-confidence variants** for downstream interpretation.

Checking File Sizes

➤ `ls -lh ~/bwa_align/*variants*`

 What is shown:

- `SRR19762473_variants.bcf` → **14 MB** (binary, compressed)
- `SRR19762473_variants.vcf` → **84 MB** (uncompressed, includes all calls)
- `SRR19762473_variants_filtered.vcf` → **19 MB** (filtered, high-quality calls)

Summary

Step	Tool	Purpose
1	<code>bcftools mpileup + call</code>	Identify variants from aligned reads
2	<code>bcftools view</code>	Convert binary BCF to readable VCF
3	<code>bcftools filter</code>	Keep only high-quality variants

Visualize Variants in IGV:

1. Open IGV on your Desktop

2. Set Genome to hg38 -

Top left: Select "Genome" → "Load Genome" → "hg38"

3. Load the BAM file -

- Go to **File** → **Load from File...**
- Navigate to your *bwa_align* directory
- Select *SRR19762473_aligned_sorted.bam*

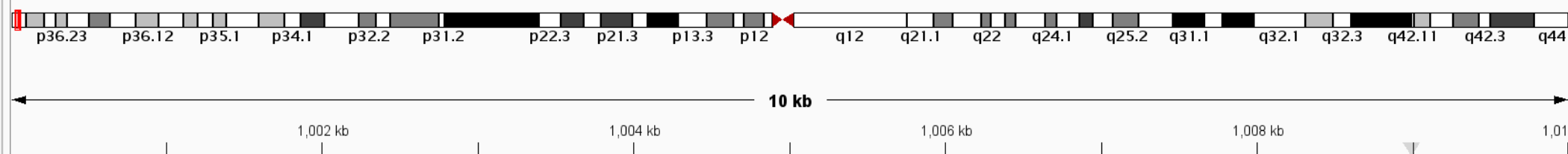
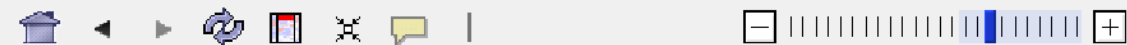
4. Load the VCF file -

Again go to **File** → **Load from File...**

Select *SRR19762473_variants_filtered.vcf*

5. Navigate to a region with variants -

- Use Ctrl+F or the **search bar** at the top (type: chr1:1000000-1010000)
- You'll see reads in the BAM track and variant markers (colored) in the VCF track



```
/home/piggyubun...gned_sorted.bar
```

[0 - 42]

SRR19762473_aligned_sorted.bam

ES4



What We See in the Visualization:

- **Coverage Track** (*SRR19762473_aligned_sorted.bam*):

Displays read depth across the selected genomic region. The height of the vertical bars indicates how many sequencing reads align at each base — giving a sense of coverage.

- **Alignments Track:**

Shows individual sequencing reads as gray rectangles.

- **Colored marks** (e.g., red, blue, purple) highlight **mismatches, insertions, or deletions** compared to the reference genome.
- This helps visually confirm the presence of real variants versus sequencing errors.

- **VCF Track** (*SRR19762473_variants_filtered.vcf*):

Displays high-confidence variants called from the data.

- Only variants passing the filter $QUAL > 30 \ \&\& \ DP > 10$ are shown.
- These represent reliable single-nucleotide variants (SNVs) or indels discovered in the sample.



Thank you

Muskan Kashyap

Connect with me on LinkedIn