QAA

Logan Lewis

9/8/2021

PART 1 - Read Quality Score Distributions

The data I analyzed for this report were located in /projects/bgmp/shared/2017_sequencing/demultiplexed/34_4H_both_S2
/projects/bgmp/shared/2017_sequencing/demultiplexed/34_4H_both_S24_L008_R2_001.fastq.gz,
/projects/bgmp/shared/2017_sequencing/demultiplexed/6_2D_mbnl_S5_L008_R1_001.fastq.gz, and
/projects/bgmp/shared/2017_sequencing/demultiplexed/6_2D_mbnl_S5_L008_R2_001.fastq.gz.
I'll be referring to these data sets as simply 34 and 6 respectivly throughout this report.

FastQC

An example FastQC batch script can be found in FastQC.sh

The graphs (Figures 1-8) show the FastQC per base quality scores and N content for all reads. We can see the the quality scores are slightly lower in the first few indexes of the reads. This is also reflected in the N content, with a sligtly higher n-content being present in the first few bases.

Python

My python script for graphing the mean base pair quality score can be found in DemuxPt1.py

The plots (Figures 9-12) given by this script look very similar to the ones given by FastQC. The run time for FastQC was a little more than 2 minutes, and my own code took about 3.5 minutes per graph. Besides the fact that FastQC is most likely very optimized, my code is in python while FastQC is written in Java. Java overall runs much faster than Python. FastQC can also run multiple files at once since it can handle multi threading meaning it can run multiple files at once, while mine cannot.

Overall the data seems to be of good quality, with good sequencing scores throughout, very low N-content, and relatively low adapter content. Some warnings flagged by FastQC such as repeated kmers and high duplication levels are to be expected during RNAseq experiments.

Part 2 - Adapter Trimming Comparison

Installed both cutadapt and trimommatic into my conda envirnment as follows:

```
conda install cutadapt
conda install trimmomatic
```

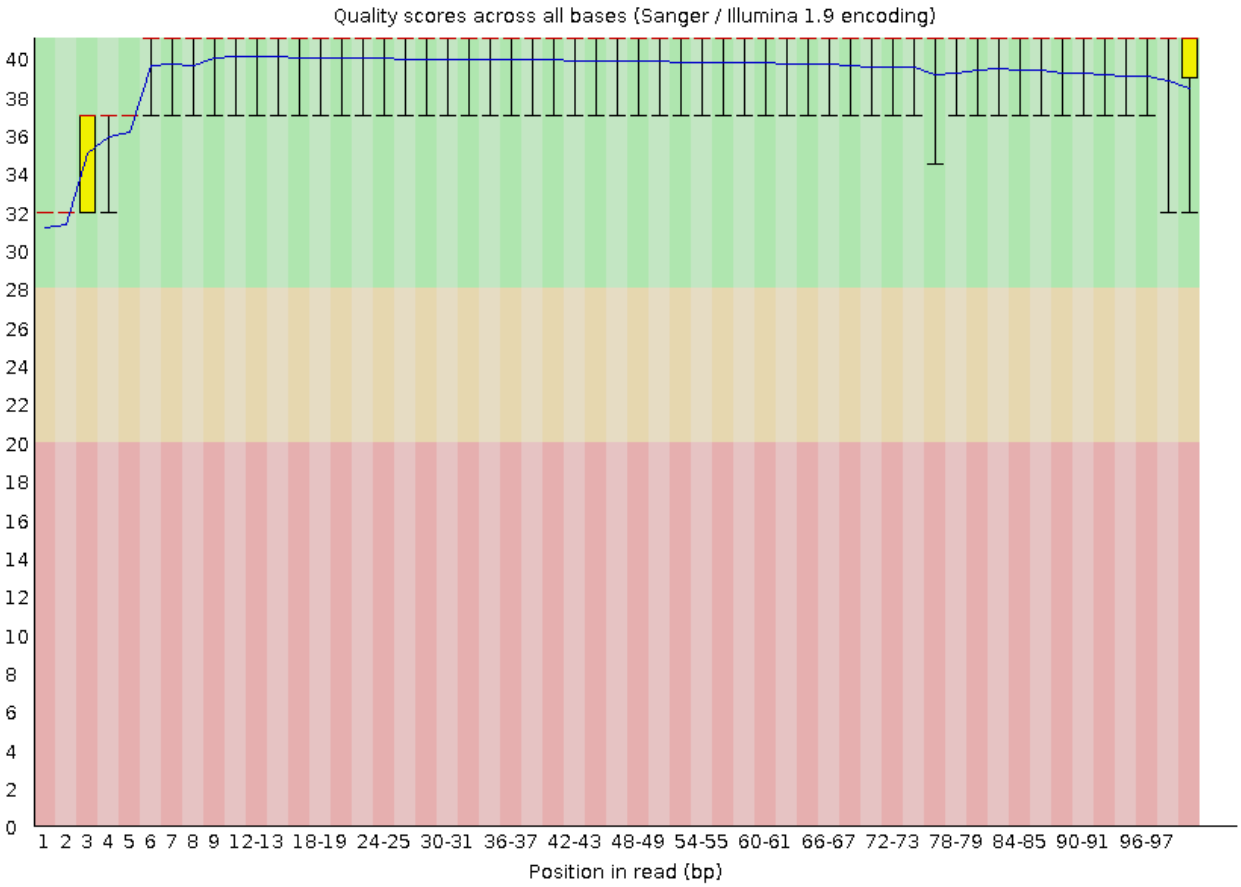

Figure 1: 34 Read 1 Per Base Quality Scores

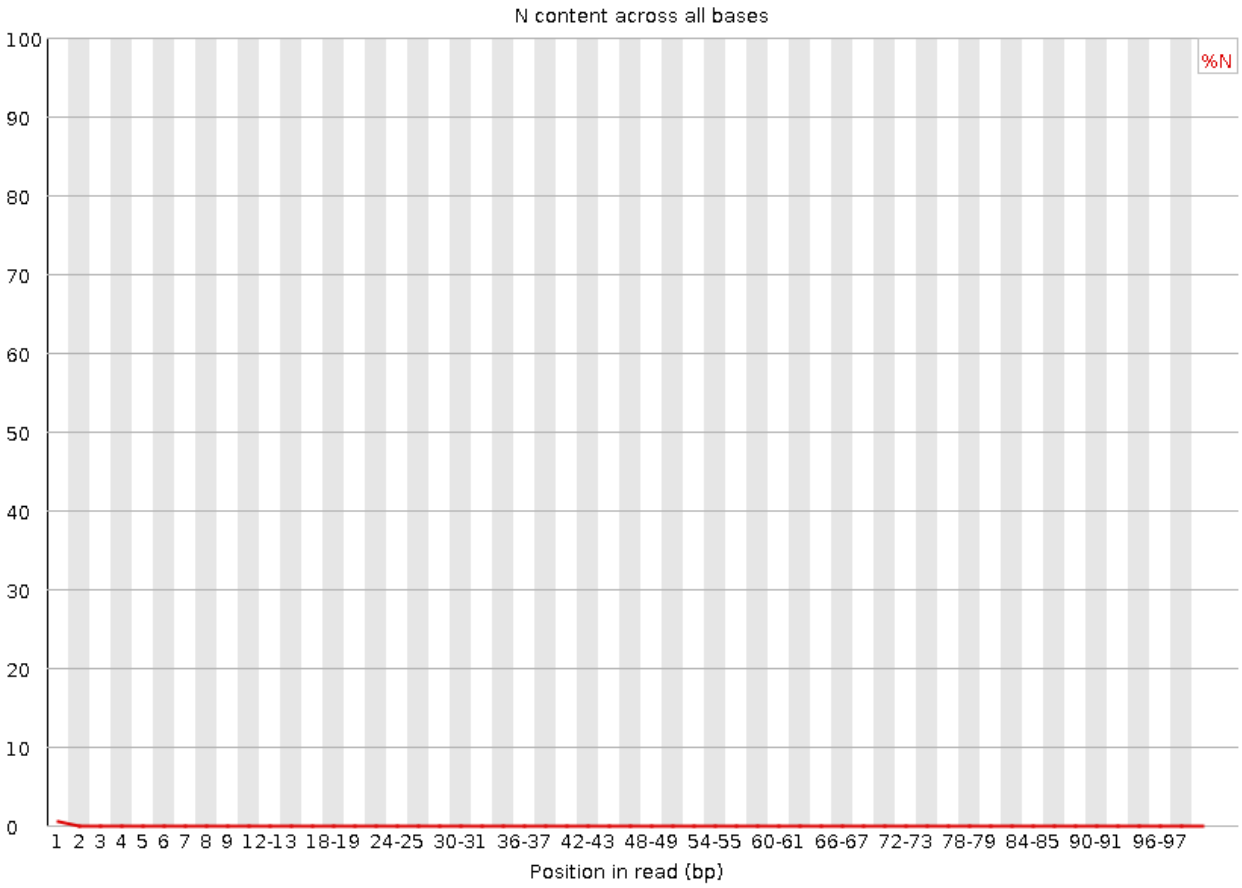

Figure 2: 34 Read 1 Per Base N Content

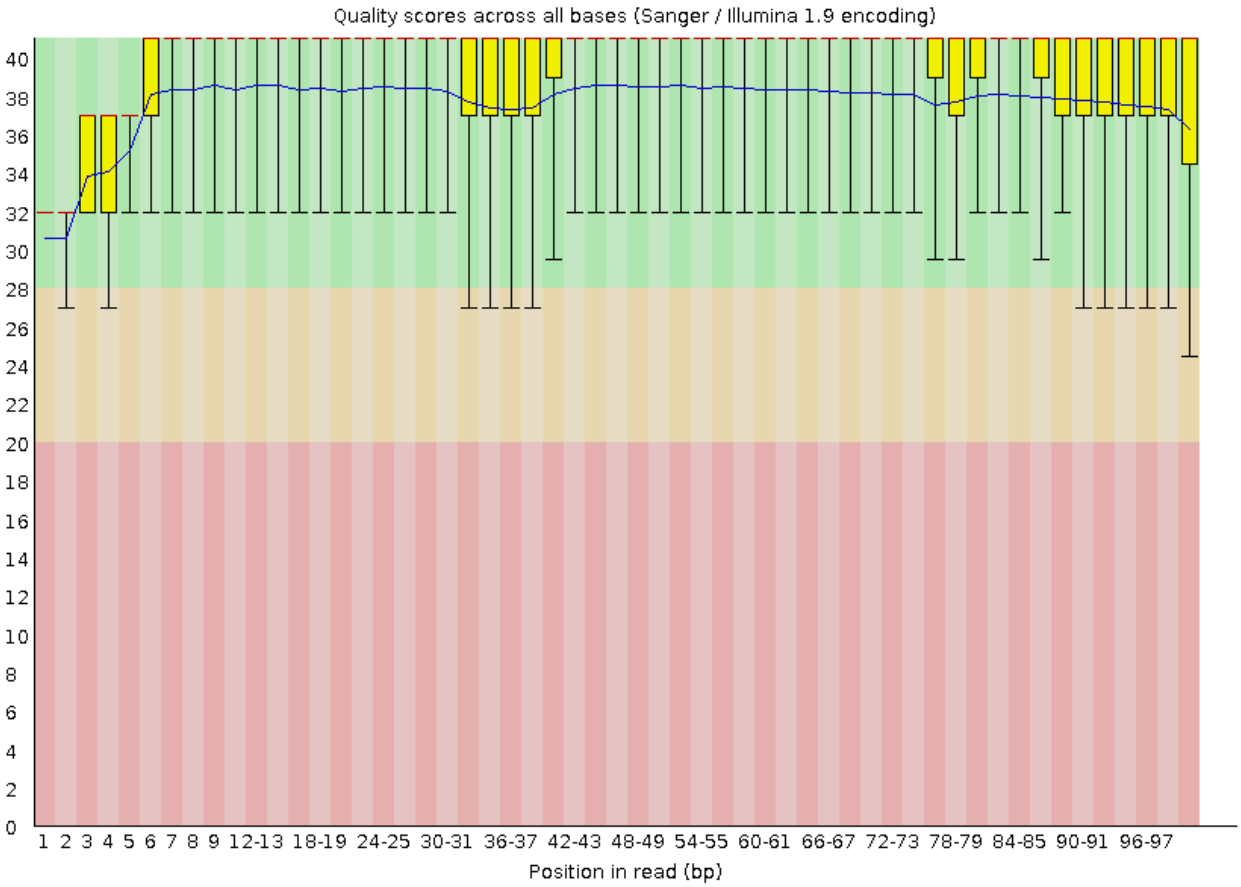

Figure 3: 34 Read 2 Per Base Quality Scores

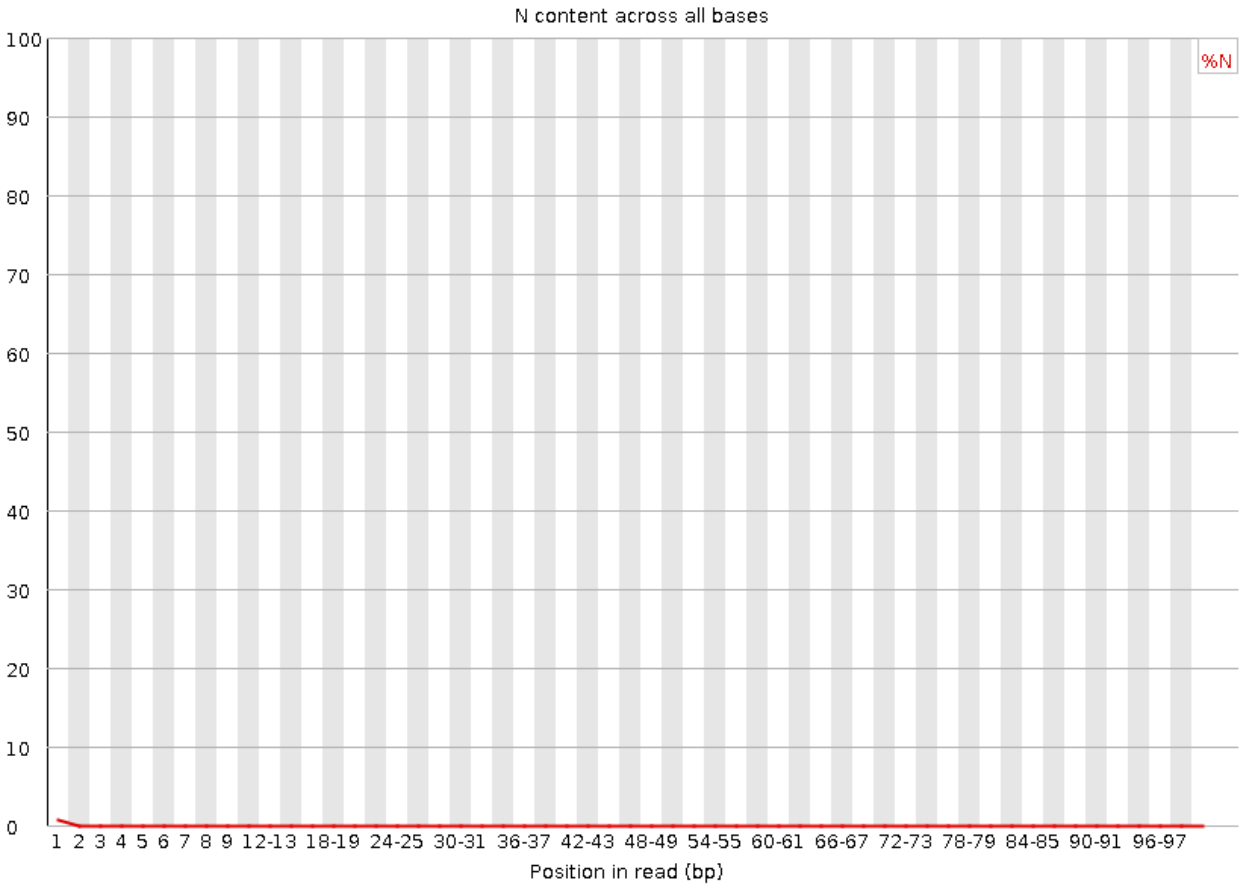

Figure 4: 34 Read 2 Per Base N Content

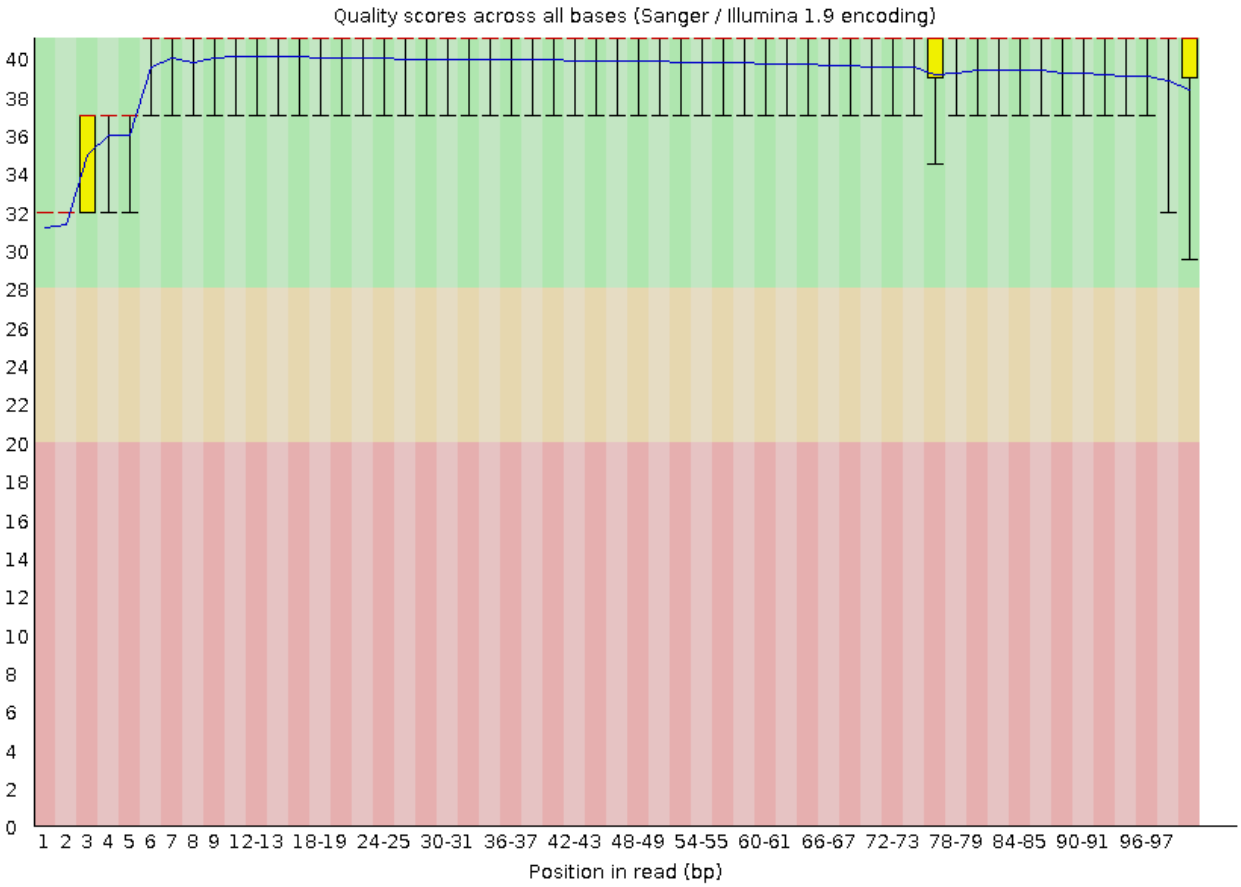

Figure 5: 6 Read 1 Per Base Quality Scores

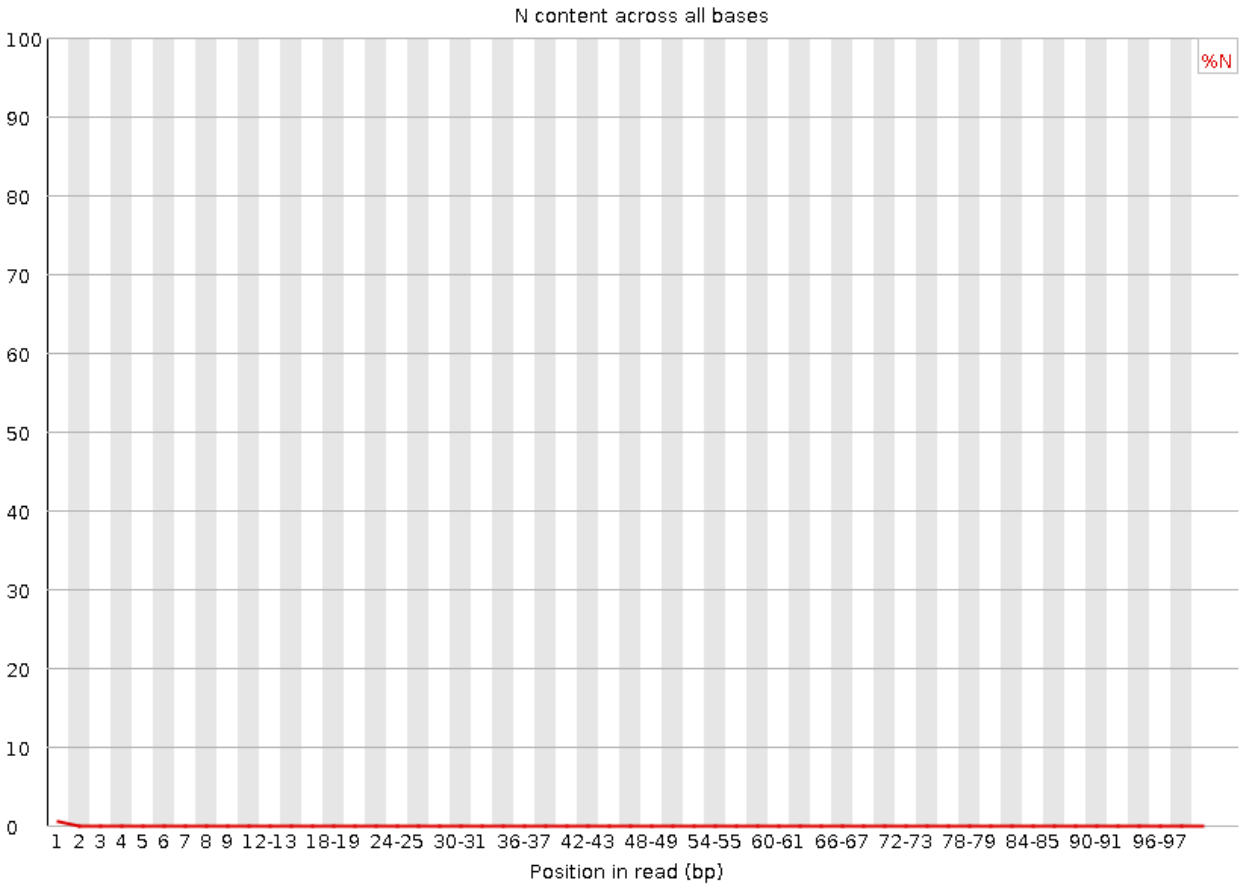

Figure 6: 6 Read 1 Per Base N Content

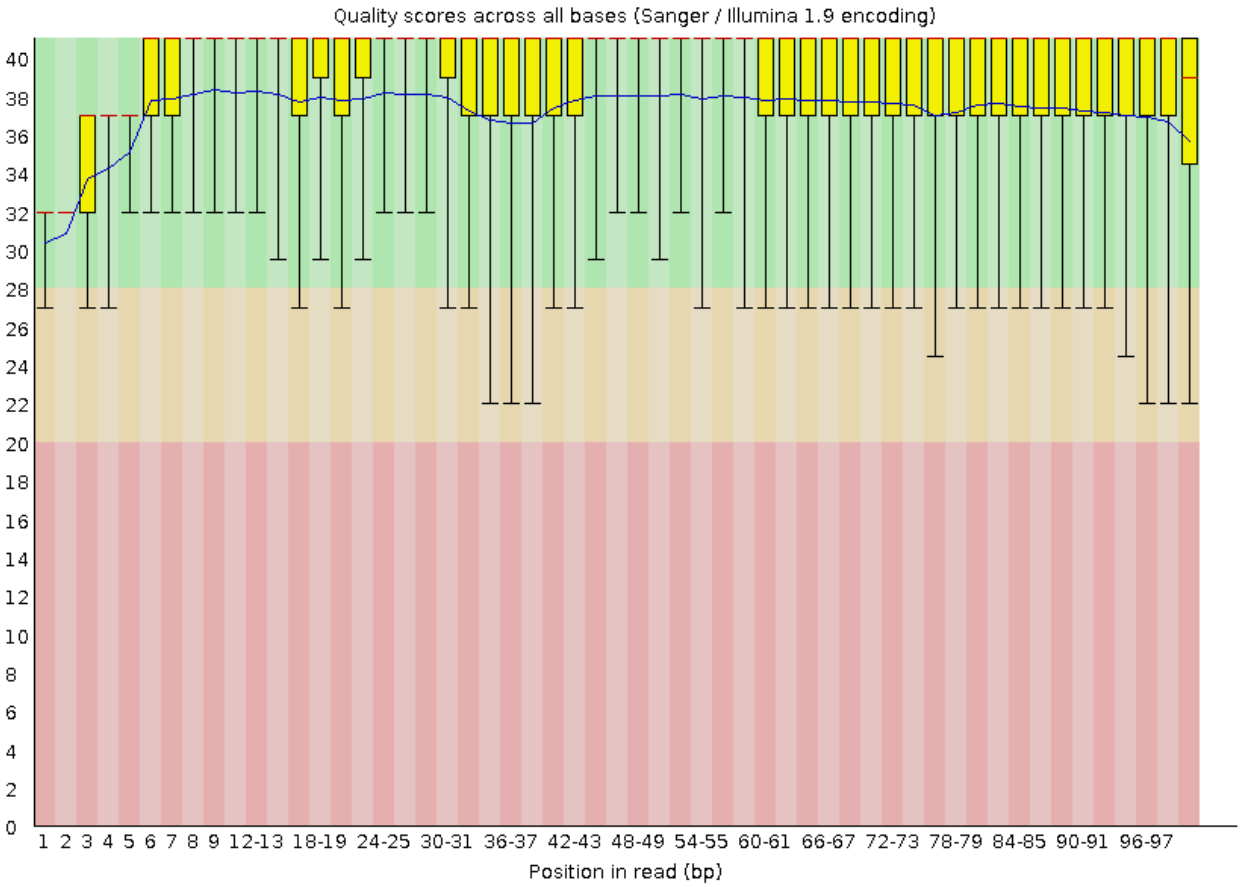

Figure 7: 6 Read 2 Per Base Quality Scores

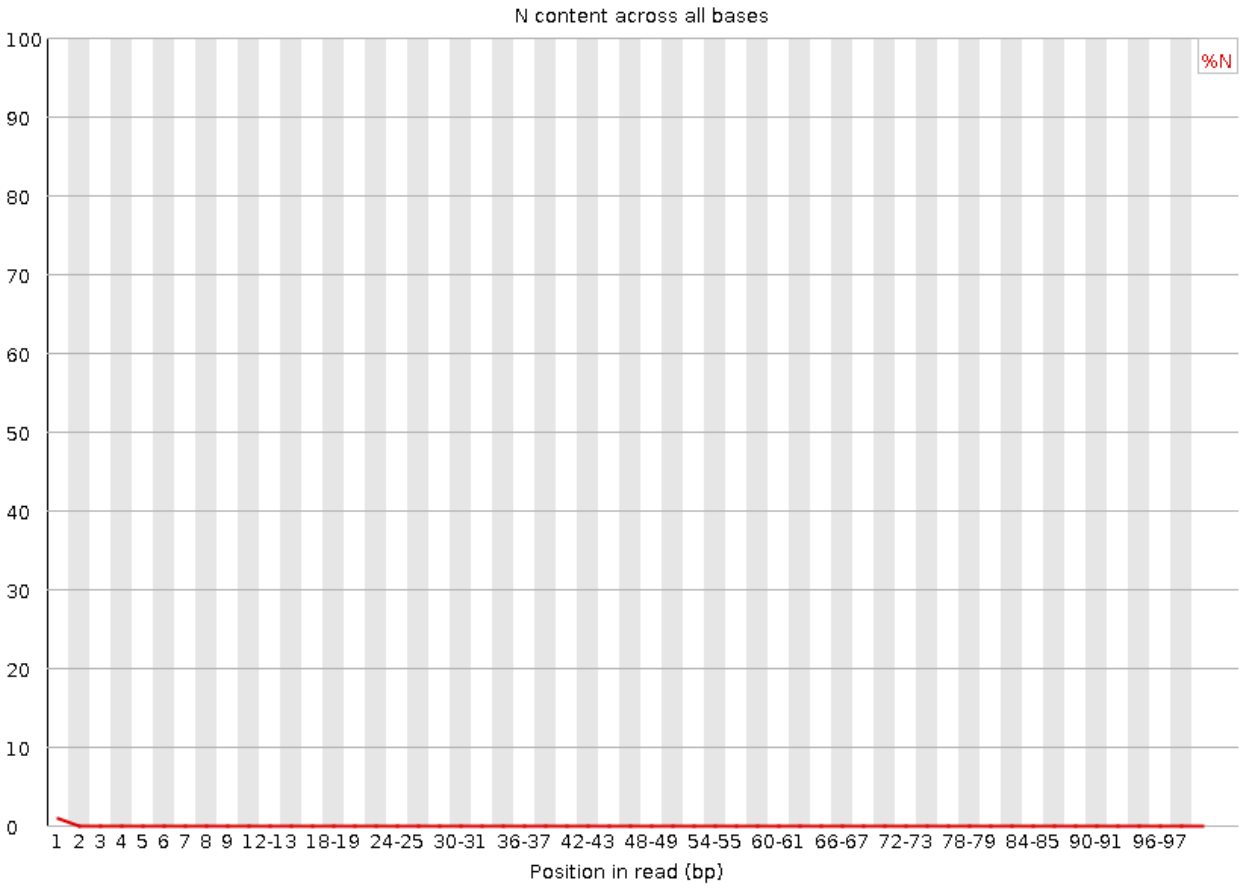

Figure 8: 6 Read 2 Per Base N Content

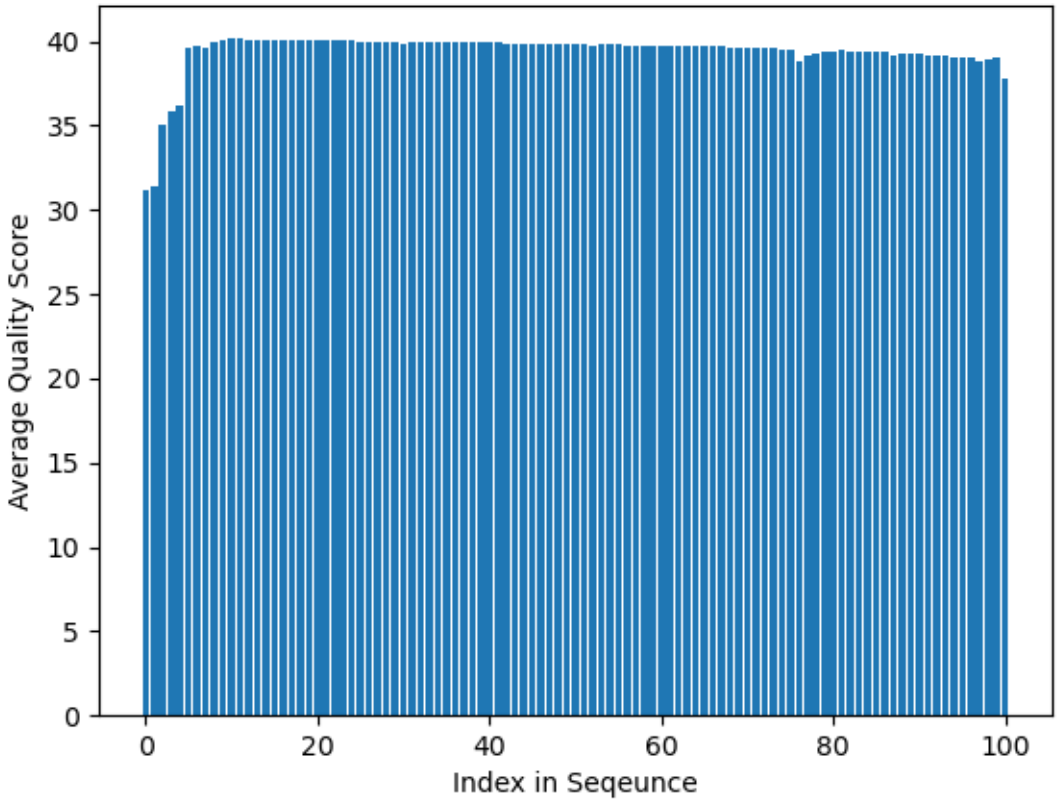

Figure 9: 34 Read 1 Average Per Base Quality Scores From Python Script

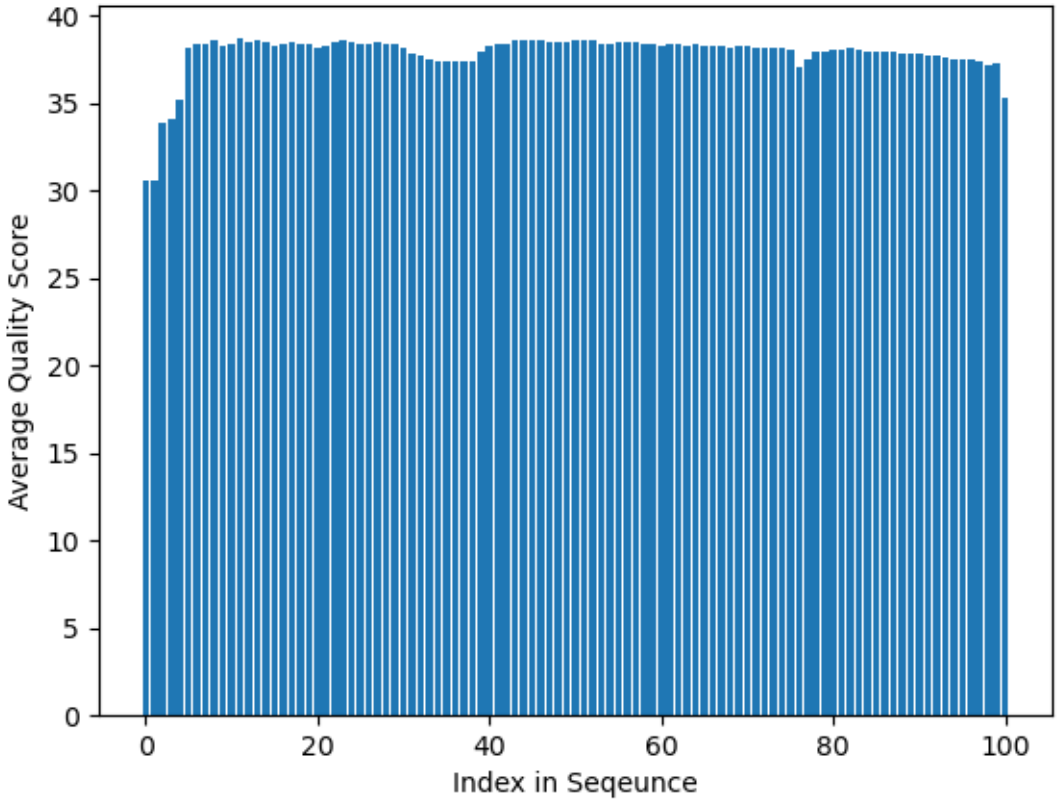

Figure 10: 34 Read 2 Average Per Base Quality Scores From Python Script

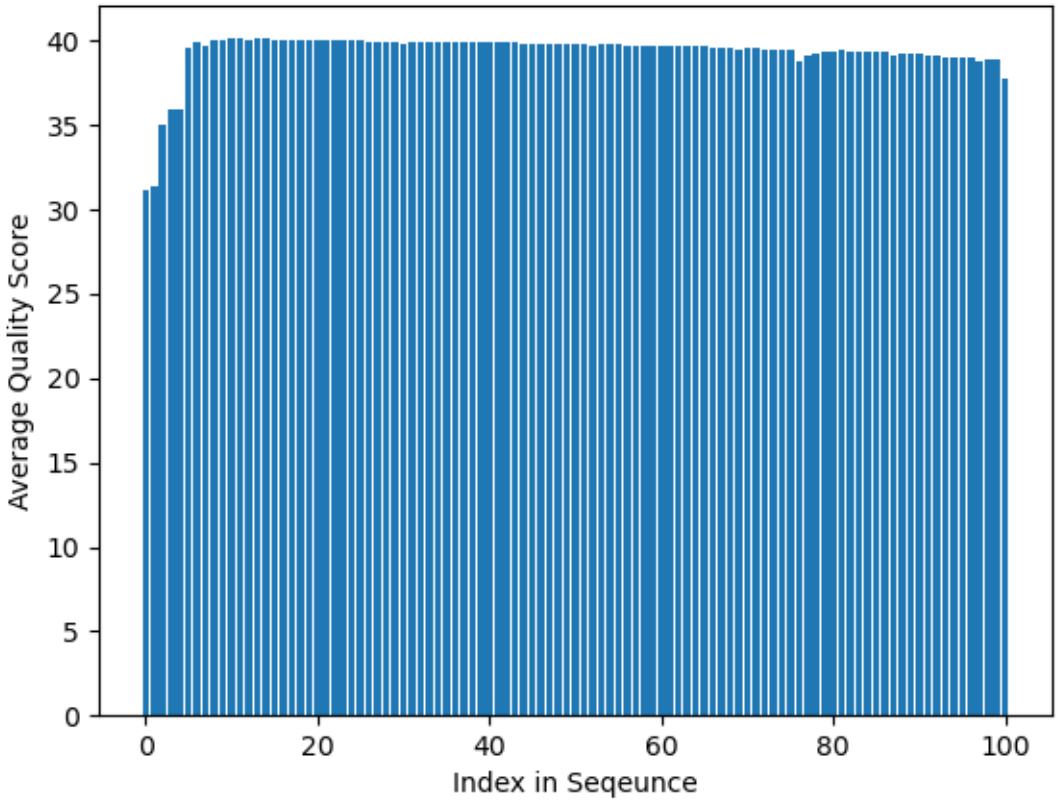

Figure 11: 6 Read 1 Average Per Base Quality Scores From Python Script

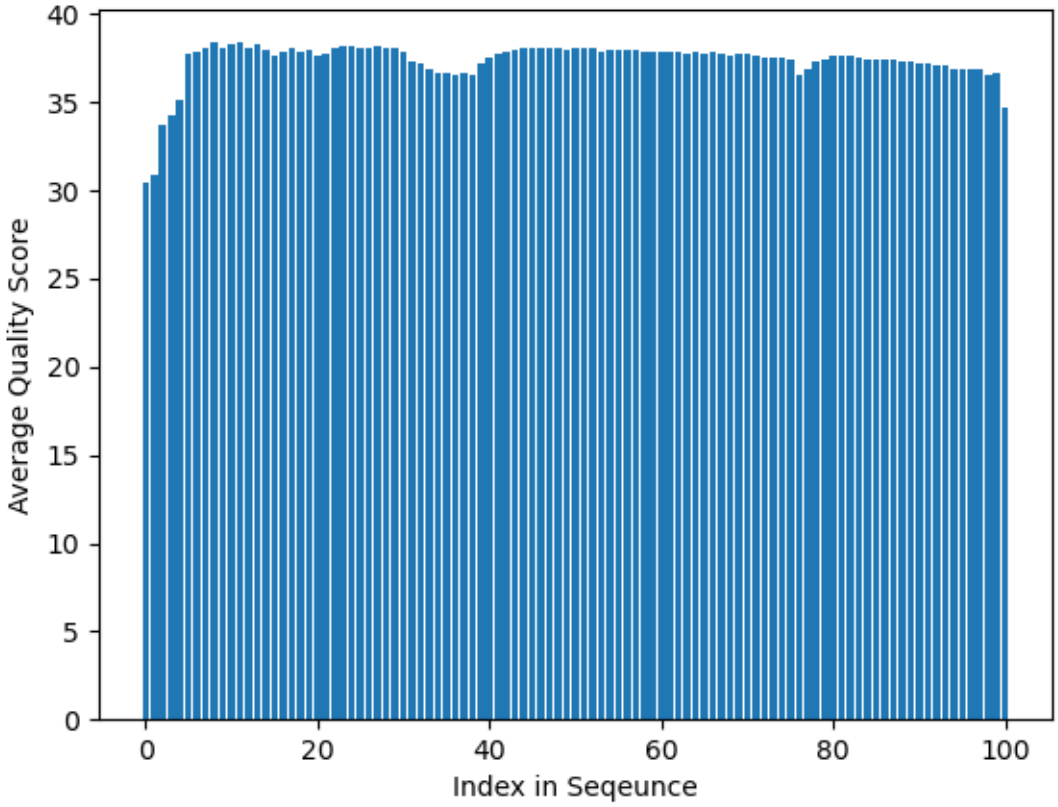

Figure 12: 6 Read 2 Average Per Base Quality Scores From Python Script

Cutadapt

I found the adapter sequnces at https://support-docs.illumina.com/SHARE/AdapterSeq/Content/SHARE/AdapterSeq/TruSeq/UDIndexes.htm

Confirming adapters:

```
gunzip -c /projects/bgmp/shared/2017_sequencing/demultiplexed/34_4H_both_S24_L008_R1_001.fastq.gz |
grep -c "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
138880
gunzip -c /projects/bgmp/shared/2017_sequencing/demultiplexed/34_4H_both_S24_L008_R2_001.fastq.gz |
grep -c "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
0

gunzip -c /projects/bgmp/shared/2017_sequencing/demultiplexed/34_4H_both_S24_L008_R1_001.fastq.gz |
grep -c "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
0
gunzip -c /projects/bgmp/shared/2017_sequencing/demultiplexed/34_4H_both_S24_L008_R2_001.fastq.gz |
grep -c "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
137879
```

By greping for the adapters, we can see that the R1 apadter was found in the R1 file but not the R2 file and vice versa.

Cutadapt batch scripts can be found in cutadapt_34.sh and cutadapt_6.sh

Proportion of reads that were trimmed:

For 34:
Read 1 with adapter: 819,166 (9.1%)
Read 2 with adapter: 886,595 (9.8%)

For 6:
Read 1 with adapter: 416,045 (3.8%)
Read 2 with adapter: 502,045 (4.6%)

Trimmommatic

Batch script for trimommatic are trim.sh and trim2.sh

Python script used to generate read length distributions can be found in Readlen.py

We can see that the Read 2's for both the graphs (Figures 13-14) have a higher frequency of lower read lengths, meaning that they contained more adapter sequences. This is what we would expect since the Read 2's are taken after the sample has been on the flowcell for longer, which leads to it being degraded.

Part 3 - Alignment and Strand Specificity

Downloaded the mouse DNA fasta and GTF from ensembl release 104
Files:
Mus_musculus.GRCm39.104.gtf.gz
Mus_musculus.GRCm39.dna.primary_assembly.fa.gz

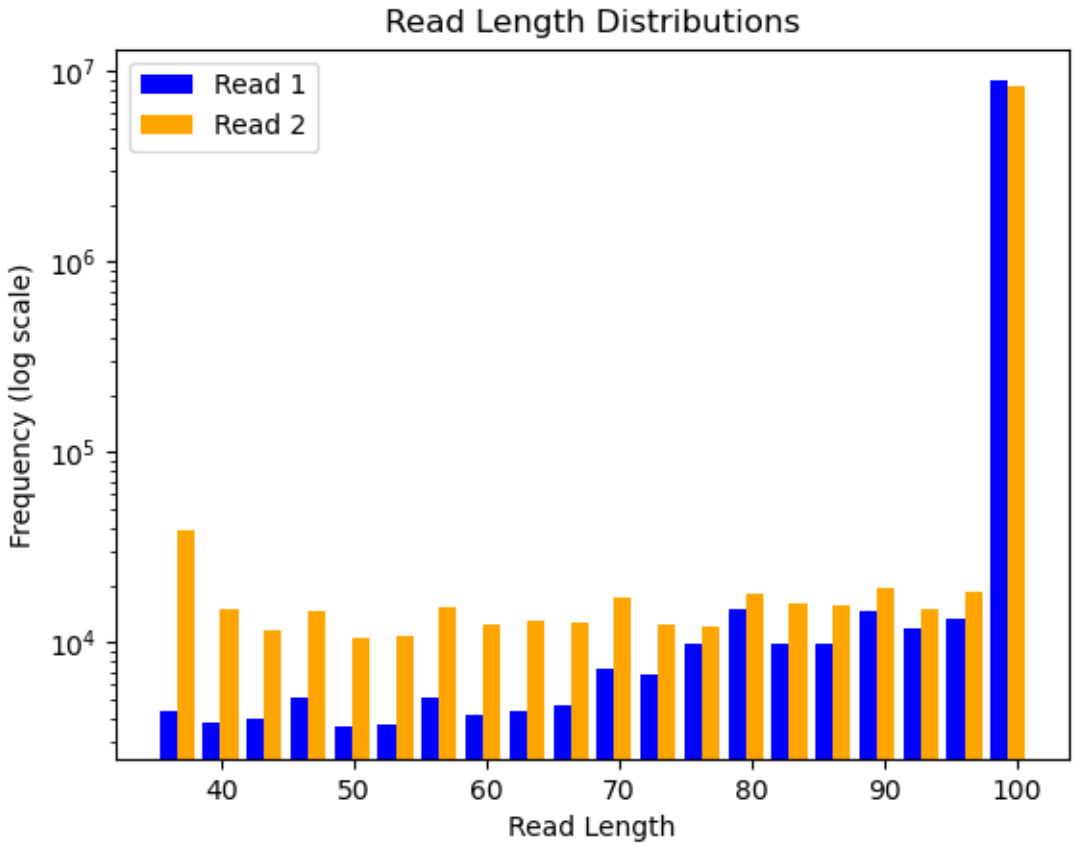

Figure 13: 34 Read Length Distributions

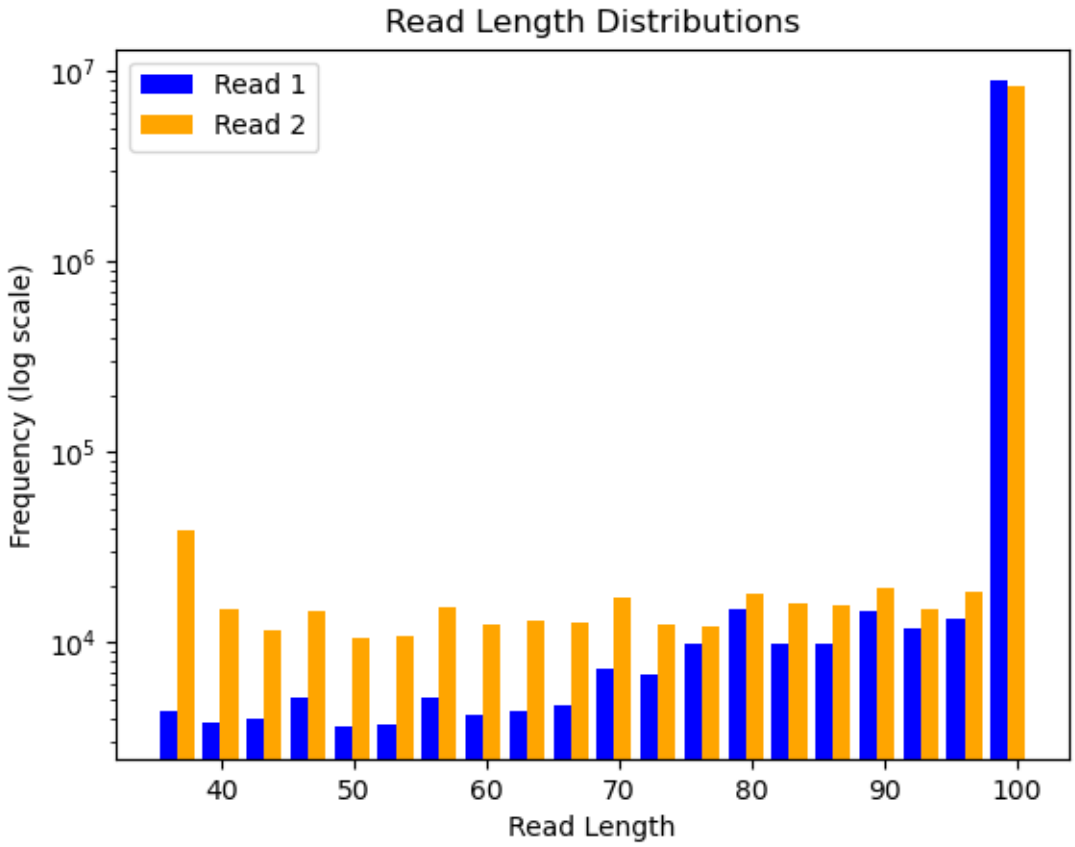

Figure 14: 34 Read Length Distributions

STAR

The batch script used to generate the STAR database can be found in STAR_setup.sh

The batch script used to align the sequences can be found in STAR_indexing.sh and STAR_indexing2.sh

Using the STAR aligner software I found the following mapped and unmapped features:
For 34:
Mapped: 16213331
Unmapped: 1130391

For 6:
Mapped: 20049211
Unmapped: 878529

The python script used to generate these counts can be found in SAM_parser.py

HTSeq-count

As per the HTSeq documentation the .sam files were first sorted by name using samtools

```
samtools sort -l 0 -o Sorted_Out34_Aligned.out.sam -n Out34_Aligned.out.sam
```

The batch scripts used to run HTSeq-count can be found in HTSeq34y.sh, HTSeq34n.sh, HTSeq6y.sh, and HTSeq6n.sh. The output of each of these have the same name with an added "Out_" prefix and replace ".sh" with ".txt"

Here we look at the output of HTSeq, with a pattern of
1. amount mapped
2. total reads
3. proportion mapped

34 Stranded: awk '{if($1~"___") {sum += $2} else {sum1 += $2}} END {print sum1; print sum1 + sum; print sum1 / (sum + sum1)}' Out_HTSeq34y.txt
425753
8671861
0.0490959

34 Unstranded: awk '{if($1~"___") {sum += $2} else {sum1 += $2}} END {print sum1; print sum1 + sum; print sum1 / (sum + sum1)}' Out_HTSeq34n.txt
6898365
8671861
0.795488

6 Stranded: awk '{if($1~"___") {sum += $2} else {sum1 += $2}} END {print sum1; print sum1 + sum; print sum1 / (sum + sum1)}' Out_HTSeq6y.txt
374287
10463870
0.0357695

6 Unstranded: awk '{if($1~"___") {sum += $2} else {sum1 += $2}} END {print sum1; print sum1 + sum; print sum1 / (sum + sum1)}' Out_HTSeq6n.txt
8281028
10463870
0.791392

The reads are most likely not strand specific as we can see a much higher percent of mapped from the unstranded files than from the stranded files.