QAA Report

Tayana Roychowdhury

2024-08-28

Part 1: Read Quality Score Distributions

FastQC plots

First library (17_3E_fox_S13_L008)
In both of the per base quality score graphs for R1 and R2, the worst average quality scores occur within the first 5 nucleotide positions, and the first position has the lowest score. In both of the N percentage graphs, the highest percentage of N content is at the first base, which is consistent with the low average quality score at the first base in both reads. The only point where there may be a discrepancy between the per base quality score and N content graphs is towards the end of R2. The average quality score in the last few positions of R2 gradually decrease, and there is a wide spread of quality scores at the last base, with the 10th percentile score dipping into the "yellow" zone of the graph. Despite the decrease in quality scores, the N content graph for R2 does not increase at all during the last few positions.

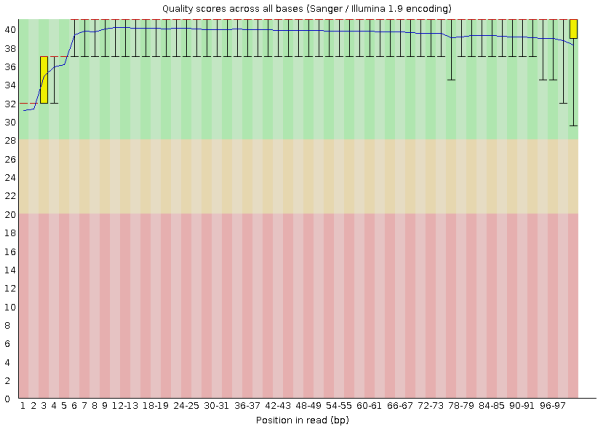

Figure 1: Average Per Base Quality Score of Reads in 17-3E-fox-S13-L008 R1. Blue line is mean quality, red line is median value, yellow box is IQR, and whiskers bound middle 80% of values.

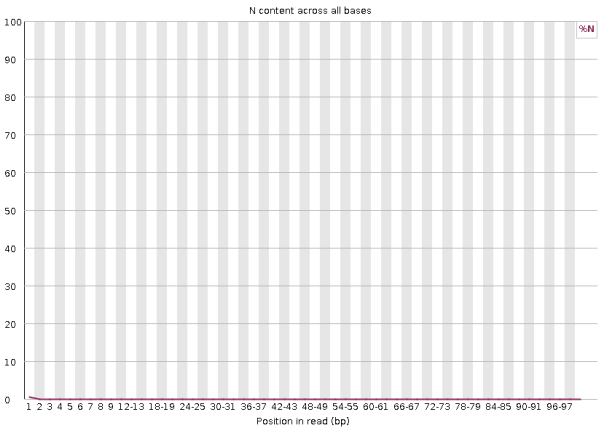

Figure 2: Percentage of N Content Per Base in 17-3E-fox-S13-L008 R1

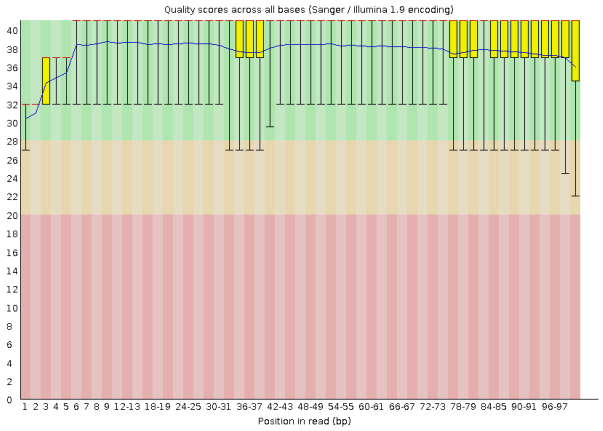

Figure 3: Average Per Base Quality Score of Reads in 17-3E-fox-S13-L008 R2. Blue line is mean quality, red line is median value, yellow box is IQR, and whiskers bound middle 80% of values.

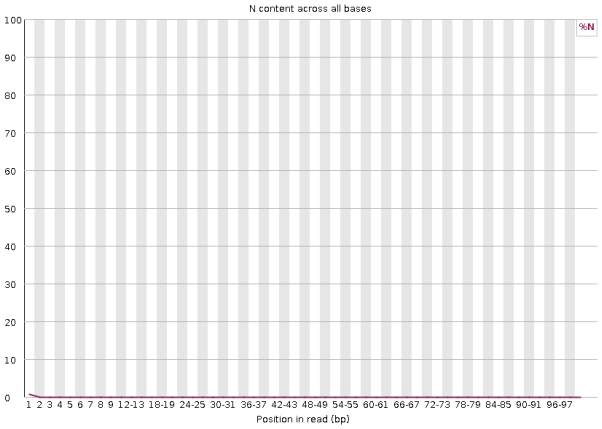

Figure 4: Percentage of N Content Per Base in 17-3E-fox-S13-L008 R2

Second library (27_4C_mbnl_S19_L008)

The per-base quality score graphs (figures 5, 7) and N content graphs (figures 6, 8) are consistent with each other for both reads. Most of the bases in the reads have a high average quality score, which is compatible with the near-zero N content across those bases. In both reads, there is a slightly higher N content within the first few bases, which agrees with the fact that both reads have their worst average quality scores at the beginning of the sequences.

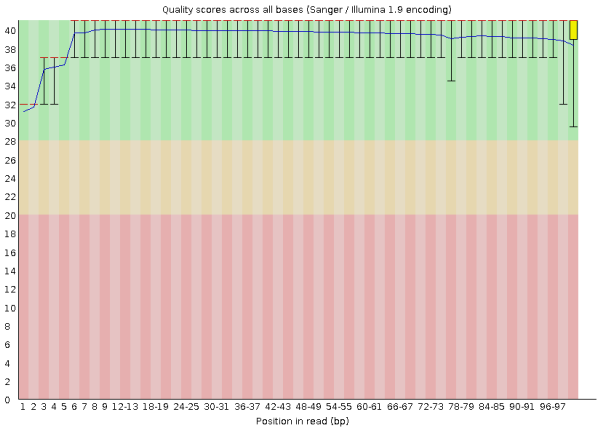

Figure 5: Average Per Base Quality Score of Reads in 27-4C-mbnl-S19-L008 R1. Blue line is mean quality, red line is median value, yellow box is IQR, and whiskers bound middle 80% of values.

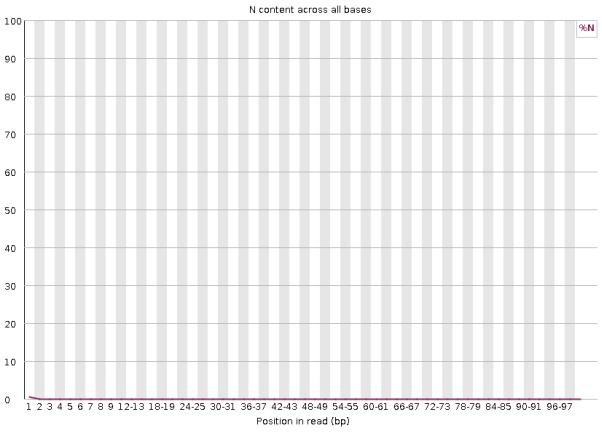

Figure 6: Percentage of N Content Per Base in 27-4C-mbnl-S19-L008 R1

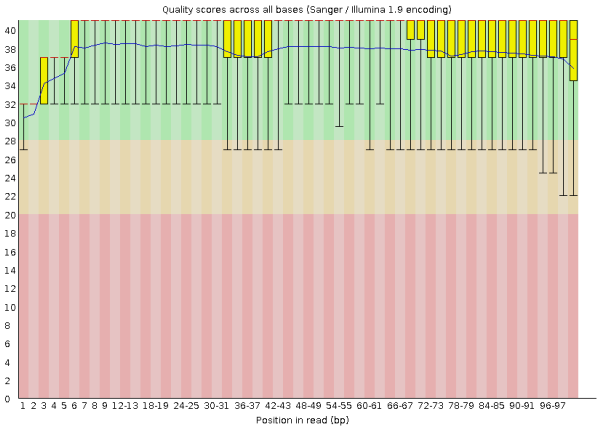

Figure 7: Average Per Base Quality Score of Reads in 27-4C-mbnl-S19-L008 R2. Blue line is mean quality, red line is median value, yellow box is IQR, and whiskers bound middle 80% of values.

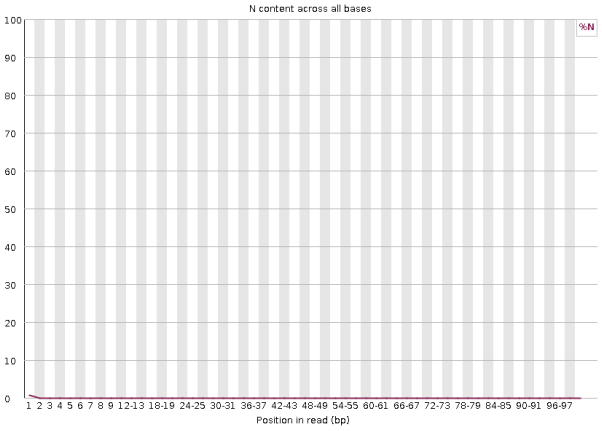

Figure 8: Percentage of N Content Per Base in 27-4C-mbnl-S19-L008 R2

Plots generated with Python script

First library (17__3E__fox__S13__L008)

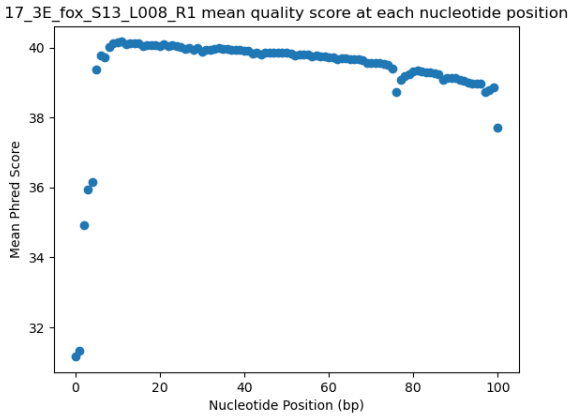

Figure 9: 17-3E-fox-S13-L008 R1 Quality Score Distribution. Blue dots represent mean quality score.

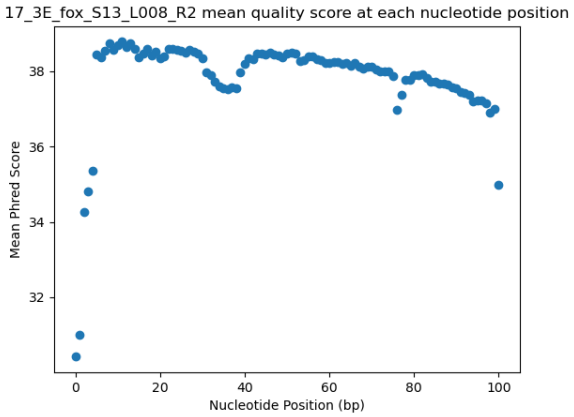

Figure 10: 17-3E-fox-S13-L008 R2 Quality Score Distribution. Blue dots represent mean quality score.

Second library (27__4C__mbnl__S19__L008)

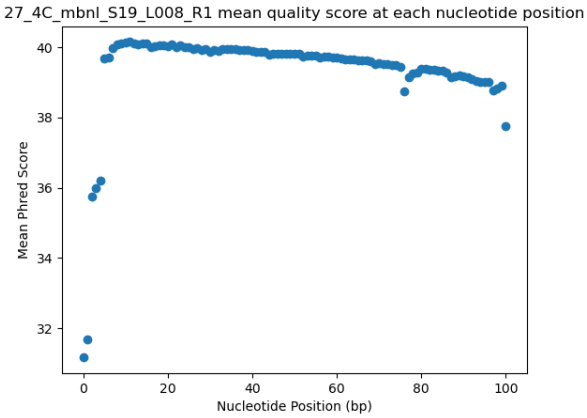

Figure 11: 27-4C-mbnl-S19-L008 R1 Quality Score Distribution. Blue dots represent mean quality score.

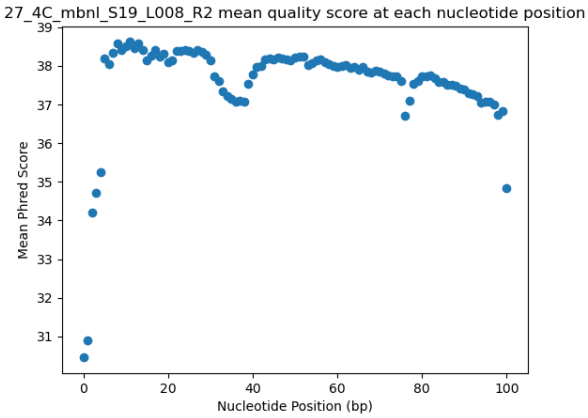

Figure 12: 27-4C-mbnl-S19-L008 R2 Quality Score Distribution. Blue dots represent mean quality score.

Compared to the FastQC output, the per-base quality score distributions I plotted for both reads in each library seem to have the same average quality score values as their respective graphs in the FastQC output.

Table 1: 17_3E_fox_S13_L008 Comparison Between FastQC and Script Metrics

	Elapsed Time (m:ss)	Memory (kbytes)	CPU usage (%)
FastQC	1:48.55	269520	97
R1 Script	6:03.45	62924	98
R2 Script	6:06.48	65024	99

Table 2: 27_4C_mbnl_S19_L008 Comparison Between FastQC and Script Metrics

	Elapsed Time (m:ss)	Memory (kbytes)	CPU usage (%)
FastQC	1:09.62	295768	97
R1 Script	3:44.57	62548	99
R2 Script	3:47.51	62456	99

From the tables above, I can see that the FastQC runs took less time, used more memory, and used slightly less CPU than my script did when generating the per base quality score distributions for each set of paired-end reads. This is likely because FastQC processed both reads at the same time so it was faster and more computationally intensive than my script.

Overall Data Quality of Both Libraries

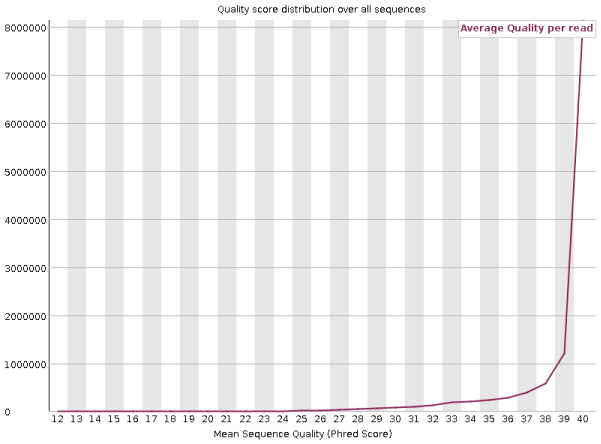

Figure 13: Average Sequence Quality Score Distribution of 17-3E-fox-S13-L008 R1

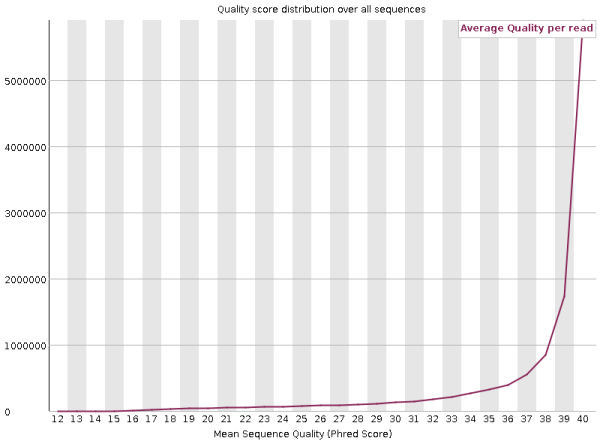

Figure 14: Average Sequence Quality Score Distribution of 17-3E-fox-S13-L008 R2

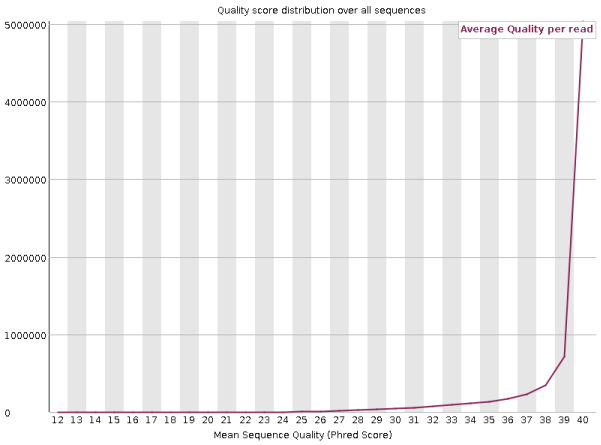

Figure 15: Average Sequence Quality Score Distribution of 27-4C-mbnl-S19-L008 R1

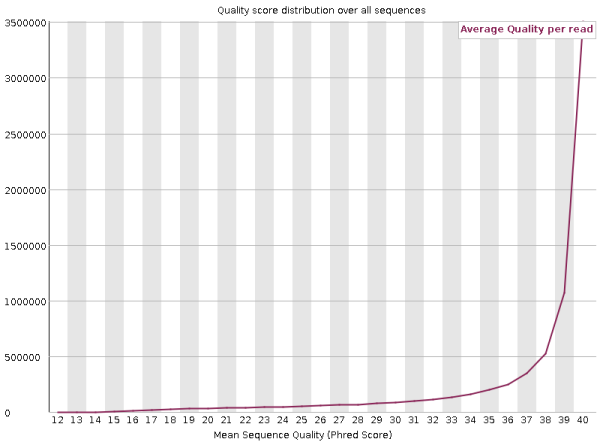

Figure 16: Average Sequence Quality Score Distribution of 27-4C-mbnl-S19-L008 R2

Based on the per-base quality score and N content graphs, as well as the average sequence quality score distributions above, both libraries have very high quality data that is appropriate for downstream analysis. In figures 13-16, the majority of sequences in all the files have average quality scores of at least 30. There are also exponentially more sequences with quality scores closer to 40 in each of the files.

Part 2: Adapter Trimming Comparison

Cutadapt

For these libraries, the R1 3' adapter was "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA" and the R2 3' adapter was "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT". Before running cutadapt, I confirmed that the appropriate adapters were towards the 3' ends in each of the untrimmed fastq files to ensure the expected sequence orientations. I did this simply by grepping for the expected adapter sequence in each of the R1 and R2 files:

```
zcat <read1.fq> | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
zcat <read2.fq> | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
```

Since most of the adapter sequences appeared towards the righthand (3') sides of the reads, they were in the proper orientation.

Table 3: Percentage of reads that were adapter-trimmed with Cutadapt in each file. Default settings for paired-end reads were used.

	R1	R2
17_3E_fox_S13_L008	8.7	9.4
27_4C_mbnl_S19_L008	10.4	11.1

Trimmomatic

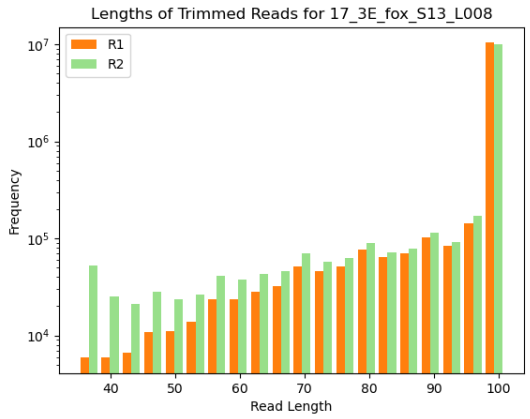

Figure 17: Length Distribution of 17-3E-fox-S13-L008 Reads. Files were adapter-trimmed with cutadapt and then quality-trimmed with Trimmomatic. Y axis is in log scale.

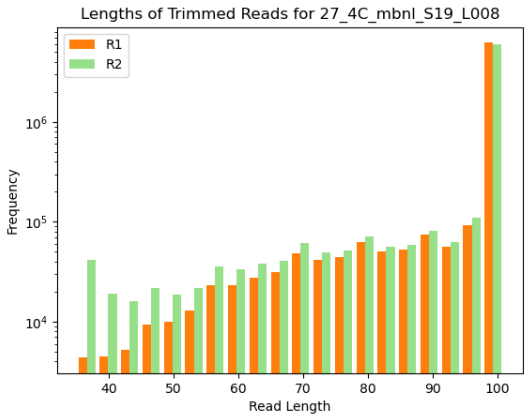

Figure 18: Length Distribution of 27-4C-mbnl-S19-L008 Reads. Files were adapter-trimmed with cutadapt and then quality-trimmed with Trimmomatic. Y axis is in log scale.

In general, I would expect R1 and R2 to be adapter-trimmed at similar rates in high quality sequencing data. Read-through adapter contamination in sequencing data usually occurs when insert sizes are too short, and since optimal insert sizes should be selected for during library prep, it is easy to control for adapter contamination from this cause.

FastQC second run on trimmed reads

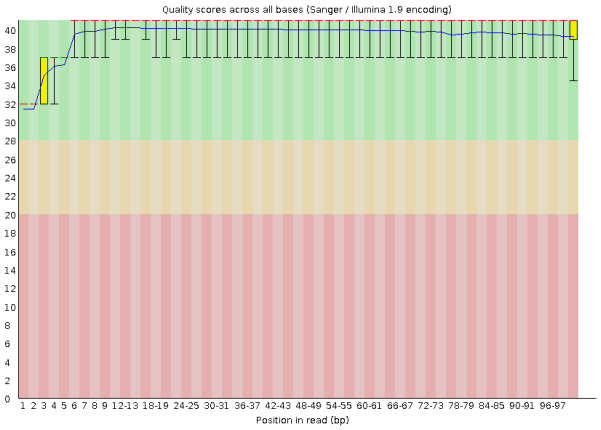

Figure 19: FastQC average per base quality for trimmed sequences in 17-3E-fox R1. Input data was adapter-trimmed with cutadapt and quality-trimmed with Trimmomatic.

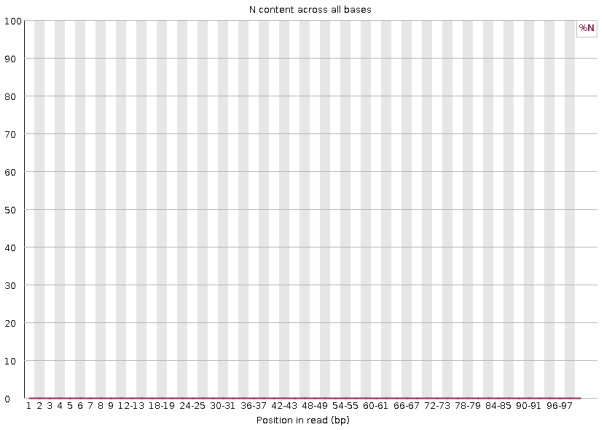

Figure 20: FastQC per base N content for trimmed sequences in 17-3E-fox R1. Input data was adapter-trimmed with cutadapt and quality-trimmed with Trimmomatic.

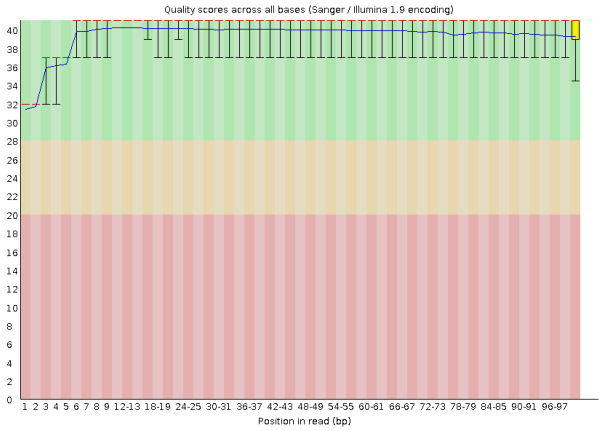

Figure 21: FastQC average per base quality for trimmed sequences in 27-4C-mbnl R1. Input data was adapter-trimmed with cutadapt and quality-trimmed with Trimmomatic.

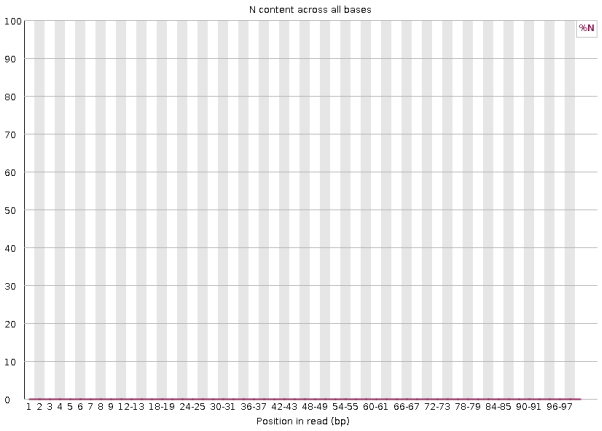

Figure 22: FastQC per base N content for trimmed sequences in 27-4C-mbnl R1. Input data was adapter-trimmed with cutadapt and quality-trimmed with Trimmomatic.

I ran FastQC for a second time on the trimmed reads in both libraries, and selected Figures 19-22 above from the output to present. I included some graphs only for R1 in both libraries for conciseness, but the R2 reads had similar results. After trimming the data, it is clear from comparing these plots to the FastQC plots in part 1 (figures 1-8) that the average per base quality for the reads have become much less variable and also increased slightly. Additionally, the trimming removed virtually all N content from the reads, whereas there were bases with slightly higher N percentages in the untrimmed data. Overall, the trimmed reads are obviously higher in quality than the untrimmed reads for both libraries.

Part 3: Alignment and strand-specificity

SAM files

Table 4: Number of mapped and unmapped reads in each SAM file from alignment to mouse reference genome. Counts printed by PS8 Python script.

	Mapped	Unmapped
17_3E_fox_S13_L008	21532811	948721
27_4C_mbnl_S19_L008	13320032	433878

htseq-count

To determine whether the data from both libraries are "strand-specific", I determined the percentages of mapped reads in the forward and reverse htseq count files. I used the following Unix commands to find the number of reads mapping to features, and total reads, in each file:

```
grep -v "^__" <file.counts> | awk '{sum+=$2} END {print sum}'
awk '{sum+=$2} END {print sum}' <file.counts>
```

I believe the data from the 17_3E_fox_S13_L008 library are strand-specific, because 79.63% of the reads map to features on the reverse DNA strand, whereas only 3.69% mapped to the foward strand. Similarly, the data from the 27_4C_mbnl_S19_L008 library are also strand-specific since 82.7% of the reads mapped to the reverse strand and only 3.91% mapped to the forward strand. Data from unstranded libraries would have closer to a 50/50 split in reads mapping to the forward and reverse strands.