QAA Report

Evan Tizzard

2024-09-10

Part 0 - Source Data

Library A:

28_4D_mbnl_S20_L008_R1_001
28_4D_mbnl_S20_L008_R2_001

Library B:

2_2B_control_S2_L008_R1_001
2_2B_control_S2_L008_R2_001

Part 1: Read Quality Score Distributions

Part 1-2: FastQC Quality Score Distributions (1 of 4)

Library A Per-Base Quality Score Distribution

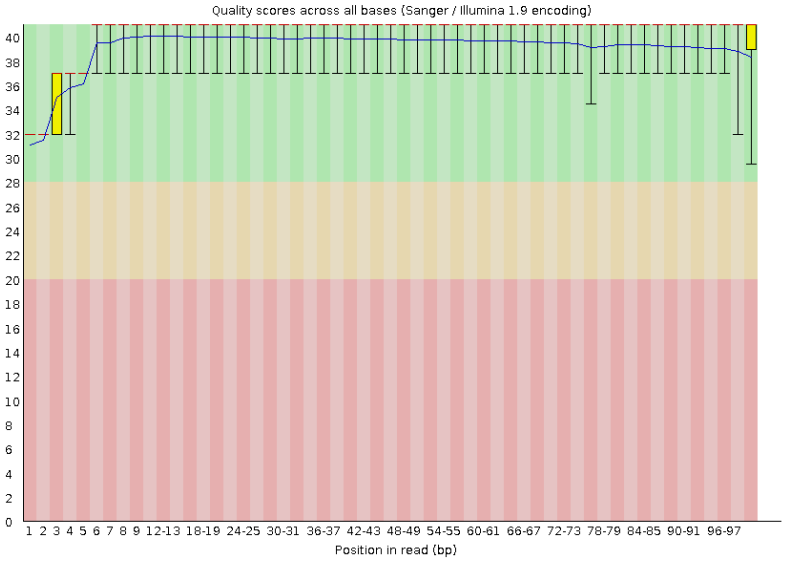

Figure 1: Library A: R1 FastQC-Generated Per-Base Quality Score Distribution Plot

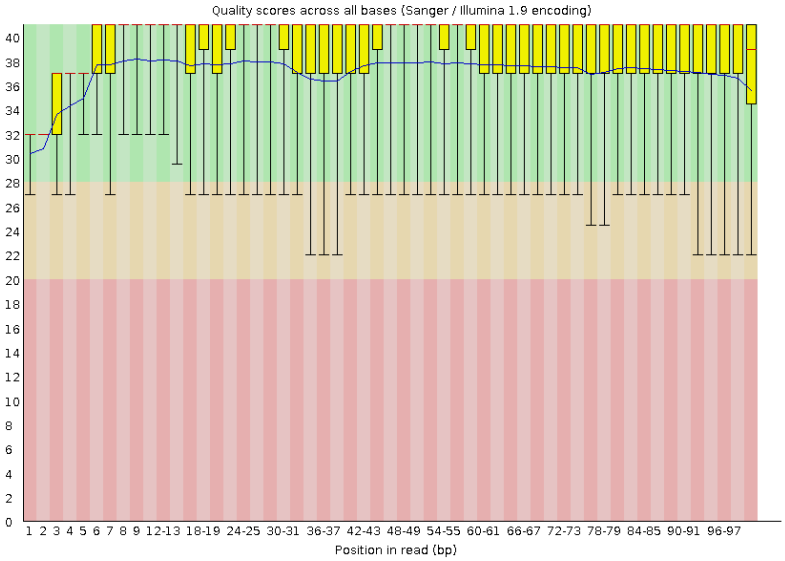

Figure 2: Library A: R2 FastQC-Generated Per-Base Quality Score Distribution Plot

Comments: R1 clearly has a much tighter spread of quality scores and has generally higher quality scores on average when compared to R2. This is normal, however, since R2 is the fourth and final read to be sequenced, so it is more prone to sequencing errors than the other three reads are.

Part 1-2: FastQC Quality Score Distributions (2 of 4)

Library A Per-Base N Content

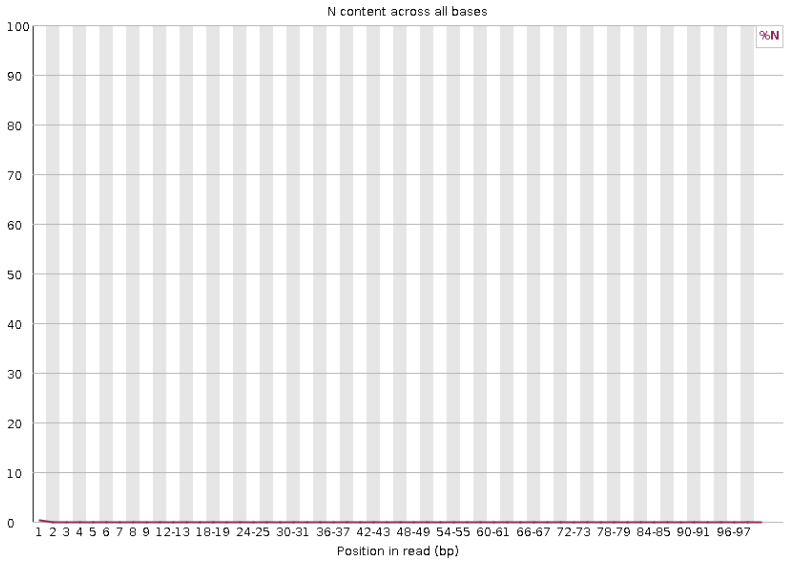

Figure 3: Library A: R1 FastQC-Generated Per-Base N Content Plot

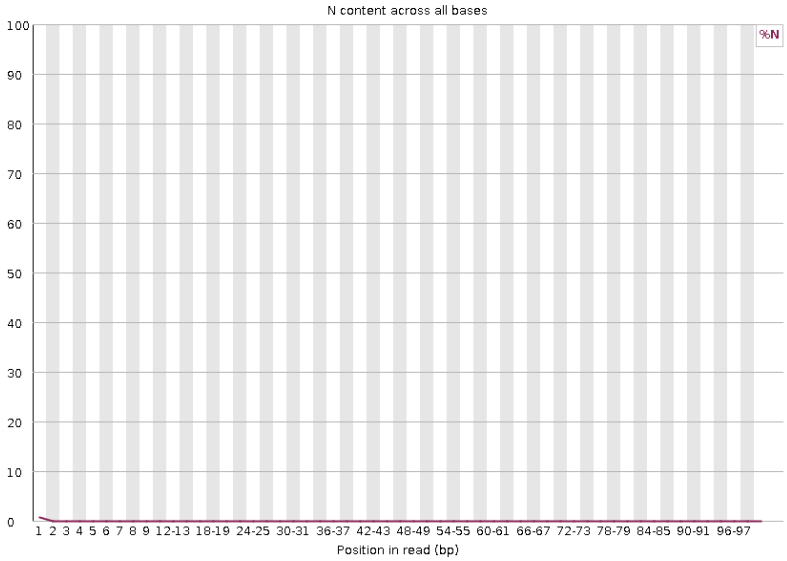

Figure 4: Library A: R2 FastQC-Generated Per-Base N Content Plot

Comments: Although R1 and R2 have quite visually distinct quality score distributions, their N content plots are nearly identical. This still makes logical sense, though, because R2 still had high quality scores generally, so Ns were not sequenced at a significantly increased rate in R2 when compared to R1.

Part 1-2: FastQC Quality Score Distributions (3 of 4)

Library B Per-Base Quality Score Distribution

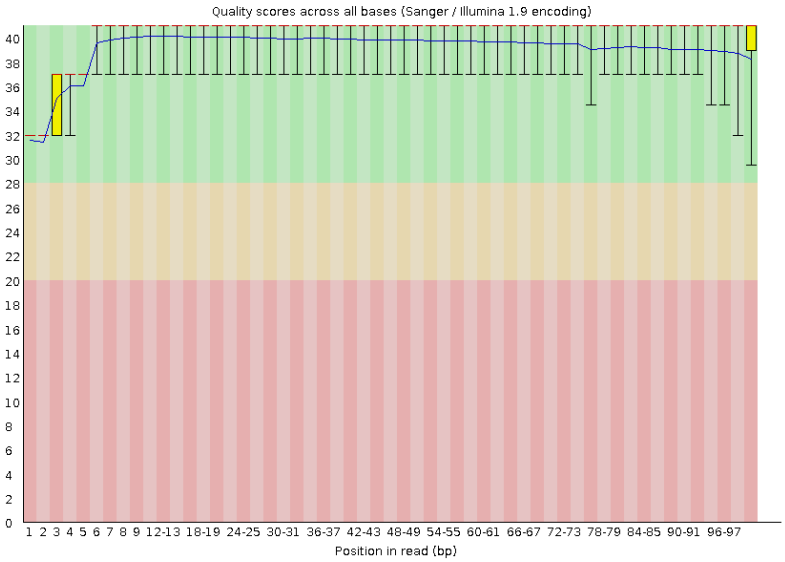

Figure 5: Library B: R1 FastQC-Generated Per-Base Quality Score Distribution Plot

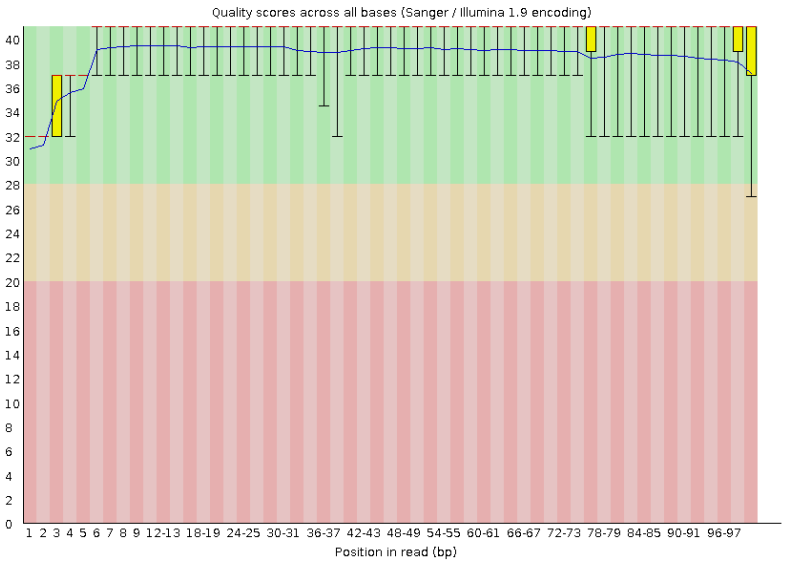

Figure 6: Library B: R2 FastQC-Generated Per-Base Quality Score Distribution Plot

Comments: Library B seems to have less variation in quality score per base position than Library A does. Simply put, these plots indicate high quality reads.

Part 1-2: FastQC Quality Score Distributions (4 of 4)

Library B Per-Base N Content

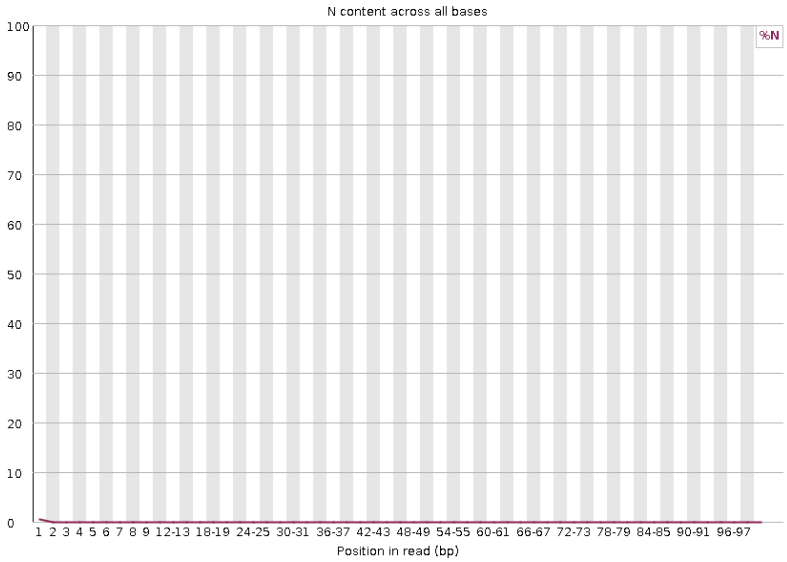

Figure 7: Library B: R1 FastQC-Generated Per-Base N Content Plot

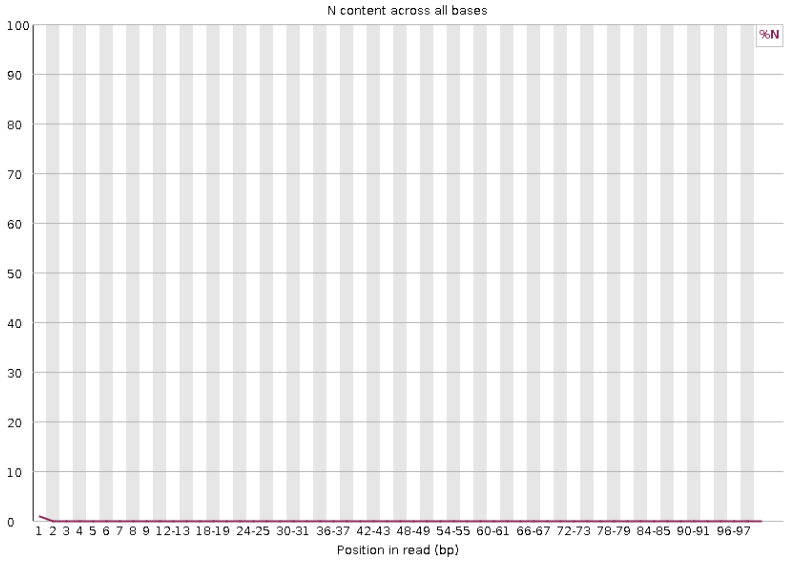

Figure 8: Library B: R2 FastQC-Generated Per-Base N Content Plot

Comments: Both the quality score distribution and N content plots for Library B indicate quite high quality reads.

Part 1-3: Self-Generated Quality Score Distributions (1 of 5)

***Library A: R1* Per-Base Quality Score Distribution**

Figure 9: Library A: R1 Self-Generated Per-Base Quality Score Distribution Plot

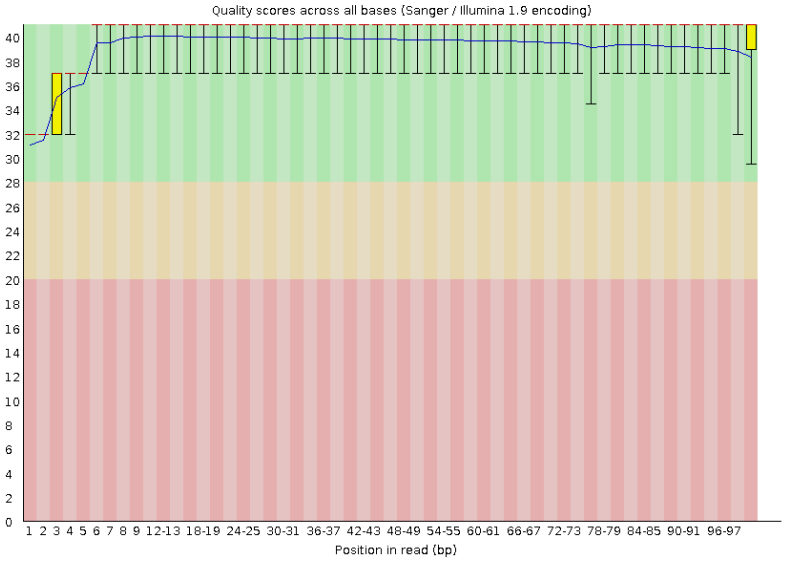

Figure 10: Library A: R1 FastQC-Generated Per-Base Quality Score Distribution Plot (presented again for comparison)

Comments: Although it may be a bit difficult to discern due to the different scaling of plots, my self-generated Library A: R1 plot is depicting the exact same data trend as the FastQC plot. My plot is the average quality score at each base position, and it thus mirrors the blue line representing the mean in the FastQC plot, albeit with more detail due to scale.

Part 1-3: Self-Generated Quality Score Distributions (2 of 5)

Library A: R2 Per-Base Quality Score Distribution

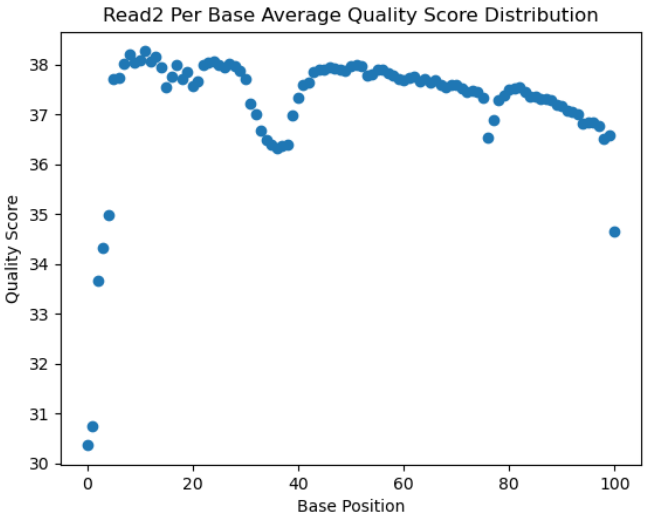

Figure 11: Library A: R2 Self-Generated Per-Base Quality Score Distribution Plot

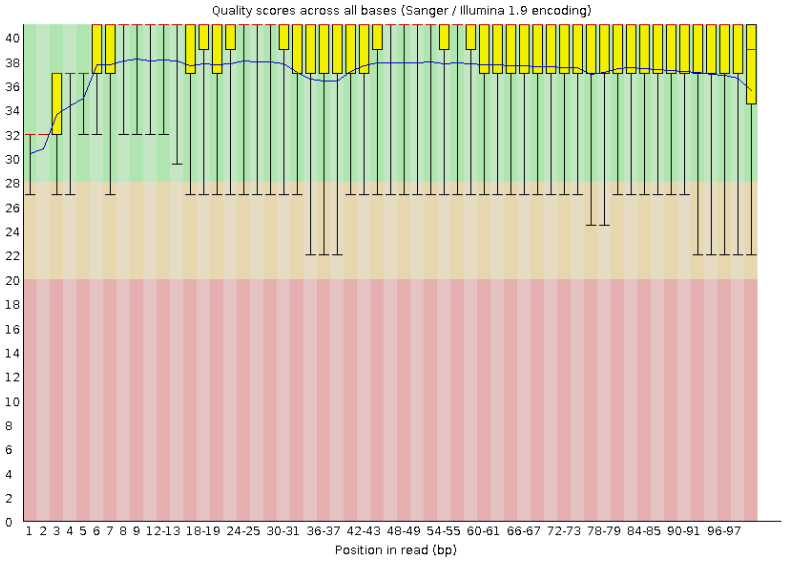

Figure 12: Library A: R2 FastQC-Generated Per-Base Quality Score Distribution Plot (presented again for comparison)

Comments: Again, my personal plot here mirrors the blue mean line of the FastQC plot. Notably, the early base positions are markedly lower quality on average, and there is a distinct dip in quality around the 35th to 40th base positions as well. This middle-position dip in quality is odd, and it is made even more obvious by my personal plot.

Part 1-3: Self-Generated Quality Score Distributions (3 of 5)

***Library B: R1* Per-Base Quality Score Distribution**

Figure 13: Library B: R1 Self-Generated Per-Base Quality Score Distribution Plot

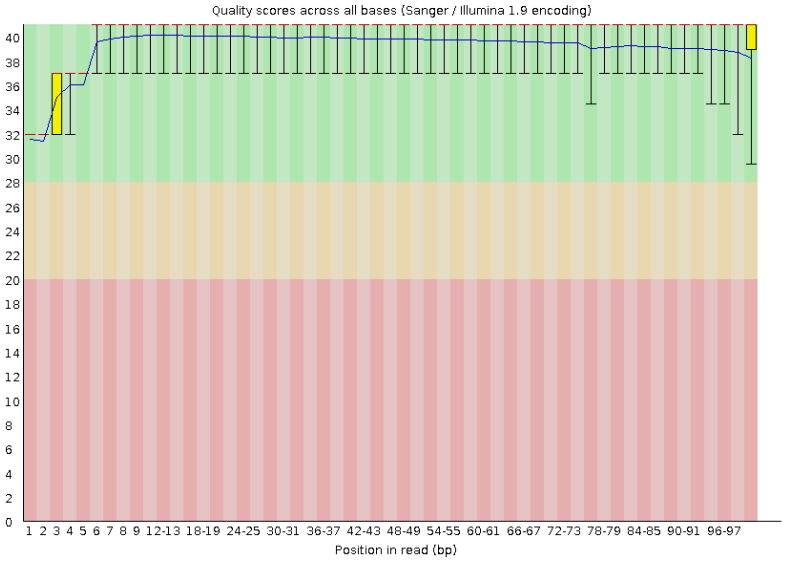

Figure 14: Library B: R1 FastQC-Generated Per-Base Quality Score Distribution Plot (presented again for comparison)

Comments: Library B reads are generally of quite high quality, and my plot matches the conclusions drawn from the FastQC plot here. This is a standard high-quality read qscore distribution - the very first base positions have lower scores on average due to calibration, then quality skyrockets and gradually decreases toward the end of the read.

Part 1-3: Self-Generated Quality Score Distributions (4 of 5)

***Library B: R2* Per-Base Quality Score Distribution**

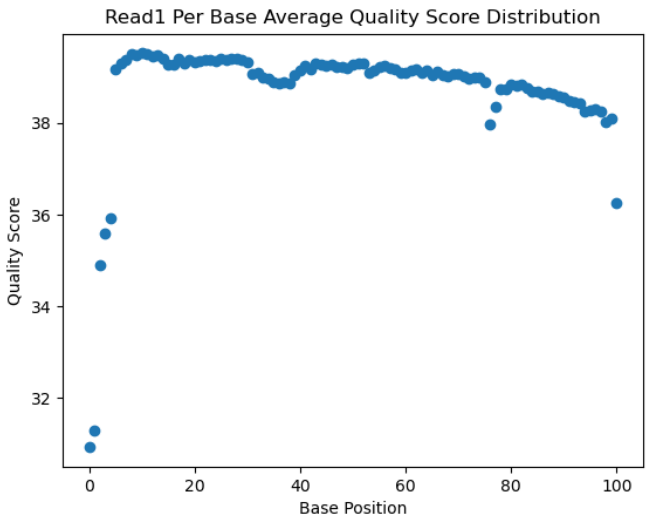

Figure 15: Library B: R2 Self-Generated Per-Base Quality Score Distribution Plot

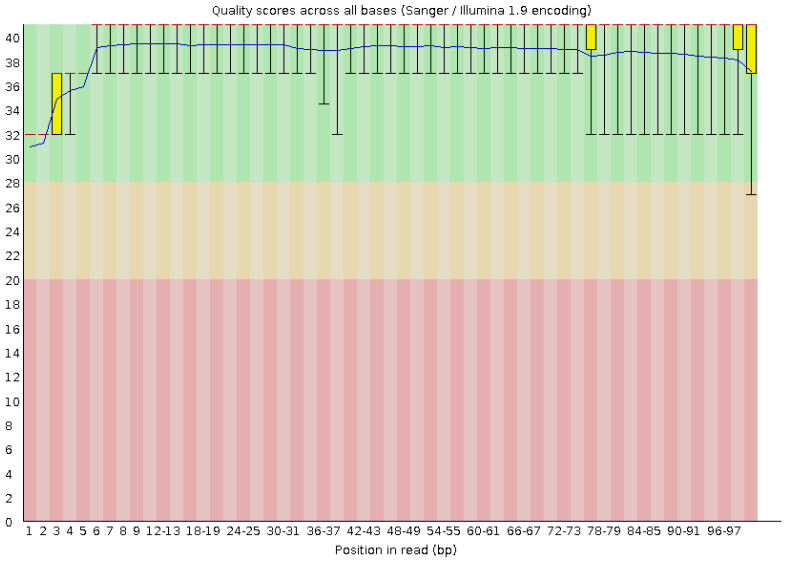

Figure 16: Library B: R2 FastQC-Generated Per-Base Quality Score Distribution Plot (presented again for comparison)

Comments: R2 of Library B is also of quite high quality, but it is naturally of lower quality than R1. My plot and the FastQC plot are in congruence here as well, with average qscore being lower at the head of the read and falling lower at the tail here than R1 did. This is expected, as this R2 was really the fourth read of the sequencing process, and the later reads are typically of lower quality than early reads.

Part 1-3: Plot Generation Comparison (5 of 5)

A. Runtime

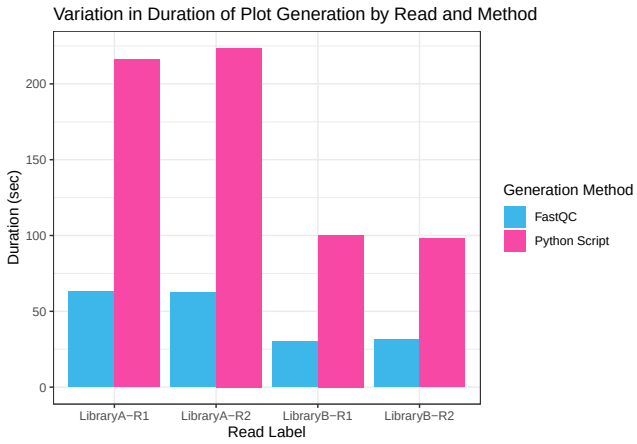

Figure 17: Runtime Comparison Plot

Comments: Note that the y-axis here is duration, so a lower value means faster performance. That said, FastQC is much faster than my Python script - around 3-4x faster on average. Not only that, but FastQC is generating many more plots and analyzing a lot more data than my script is. I would imagine that the main reason for this crazy speed of FastQC is that it runs on Java. Java is a compiled language, so it runs much faster on average than Python does. Additionally, FastQC is simply more robust and better written (and thus more efficient) than my code.

B. CPU Usage

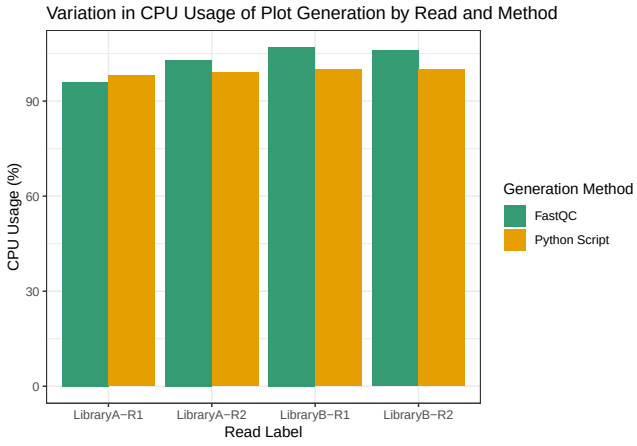

Figure 18: CPU Usage Comparison Plot

Comments: On average, FastQC utilized a higher percentage of the CPU than my Python script did, often exceeding 100%. I would imagine that this simply means FastQC runs efficiently, so the CPU can constantly carry out the computations required of FastQC without waiting for the program to provide the next input. Again, I think this is a result of FastQC being both well-written and Java-based.

Part 1-4: Overall Data Quality

I believe that both of these libraries are of high quality; however, Library B is clearly of higher quality than Library A. Both libraries have excellent mean per-base quality score distributions - Library B simply has higher quality scores on average when compared to Library A. Moreover, the spread of average quality scores is quite tight within Library B. So, not only are Library B's average quality scores at each base position high, but its tendency for a quality score at any given position to vary far from the mean is very low - its quality scores are consistently within a tight range centered on the mean. Library A, on the other hand, is not quite as neat. Although R1 of Library A is very high quality, R2 is all over the place. Despite Library A: R2's high average quality score per base position, the scores at essentially any position in this read have a much wider spread (variance) from the mean. This means that when a base is called, it is more likely to be of lower quality at any given postion than all the other library reads. Library A: R2 is by far the lowest quality. That being said, I would feel quite confident using all of this data to continue analysis. Although Library A: R2 has a fair amount of scores deviating into sub Q30 range, this is still uncommon all things considered, and the vast majority of the base calls are of perfectly good quality. If Library A: R2 had wide variance as well as an average quality score less than or near 30, then I would be much more skeptical of proceeding with downstream analysis.

Part 2: Adapter Trimming Comparison

Part 2-6: Trimmed Reads

Trimming Stats

After using cutadapt to trim adapter sequences from all four read files, I then calculated the percentage of total reads that were trimmed by comparing total line counts. According to my math:

Library A R1: 3.262% of total reads were trimmed

Library A R2: 1.184% of total reads were trimmed

Library B R1: 2.6% of total reads were trimmed

Library B R2: 0.559% of total reads were trimmed

Adapter Sequences

As for the adapter sequences themselves, I had a great deal of trouble trying to find them on my own. After letting the cat out of the bag, I took the given adapter sequences and used grep to find matches in the corresponding read files. I also tested the R1 adapter sequence on the R2 files, and vice versa, to ensure that the adapters were totally unambiguous. When searching for the incorrect adapter, grep returned nothing for all 4 read files. This made me very confident that the adapters were indeed the genuine article. Additionally, the number of matches returned by grep for each read pair were nearly identical - all of these factors stratified my confidence in these adapter sequences.

```
#Library A: R1
$ zcat /projects/bgmp/shared/2017_sequencing/\
  demultiplexed/28_4D_mbnl_S20_L008_R1_001.fastq.gz \
  | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA" | wc -l
    56758

#Library B: R1
$ zcat /projects/bgmp/shared/2017_sequencing/\
  demultiplexed/2_2B_control_S2_L008_R1_001.fastq.gz \
  | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA" | wc -l
    31917

#Library A: R2
$ zcat /projects/bgmp/shared/2017_sequencing/\
  demultiplexed/28_4D_mbnl_S20_L008_R2_001.fastq.gz \
  | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc -l
    57626

#Library B: R2
$ zcat /projects/bgmp/shared/2017_sequencing/\
  demultiplexed/2_2B_control_S2_L008_R2_001.fastq.gz \
  | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc -l
    31965
```

Part 2-8: Trimmed Read Length Distributions

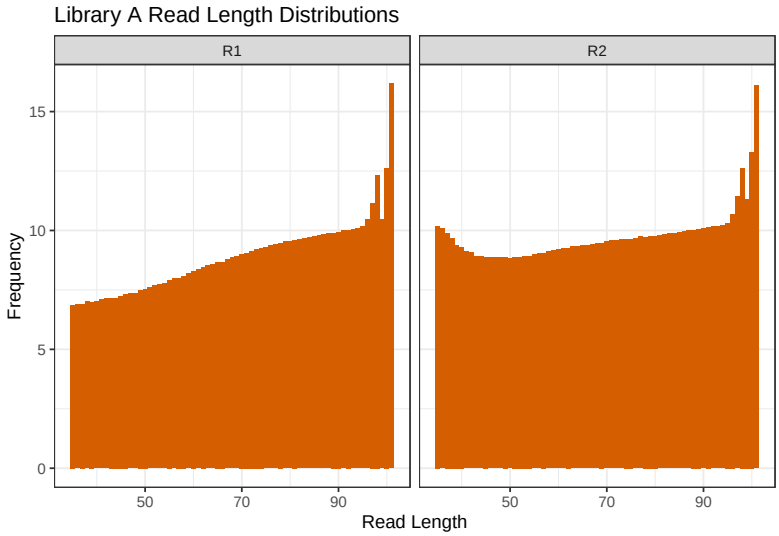

Figure 19: Library A: Read Length Distribution Plots

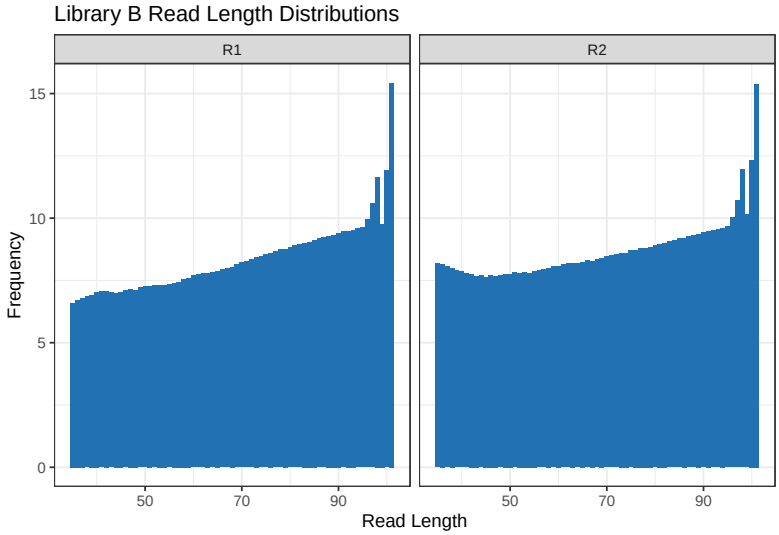

Figure 20: Library B: Read Length Distribution Plots

Comments: Looking at these plots, it is discernable that the R1 of both libraries are more heavily trimmed than R2. Although the plots between R1 and R2 look quite similar, especially in regions toward the end of the read length, the left side of the plots are noticeably distinct. Both R2 plots exhibit a "bump" at the lower read positions, and the frequencies are generally slightly higher at every position. I would think that this pattern of R1 being more adapter-trimmed than R2 is due to sequencing bias or library preparation. Since R1 is sequenced first, I would suggest that the sequencer detection algorithm is still calibrating to the adapter at the beginning of sequencing, so the early R1 reads are low quality and are thus more heavily trimmed. Conversely, the R1 adapter sequence chosen and used during library preparation could have also introduced sequencing bias to R1 reads.

Part 3: Alignment and Strand-Specificity

Part 3-12: Number of (Un)Mapped Reads

Table 1: Count of Mapped and Unmapped Reads Between Mouse Genome and RNA-Seq Libraries

Library	Mapped Reads	Unmapped Reads
A	22657642	793158
B	11078796	226280

Part 3-14: RNA-Seq Library Strand-Specificity

I would postulate that this data originated from strand-specific RNA-Seq libraries. In my simple analysis of the htseq-count data, it is clear to me that there are vastly **(nearly 23x)** more feature mappings in the reverse count than there are in the forward count. I believe that such an extreme discrepancy in feature mapping counts could not have occurred due to pure random chance, thus, it must be due to strand-specificity. I reached this conclusion after performing the following:

```
#Forward R1 Feature Map Count
$ grep -v "^_" htseq_count_fwstranded.txt | awk '{s+=$2} END {print s}'
  411314

#Forward R2 Feature Map Count
$ grep -v "^_" htseq_count_fwstranded.txt | awk '{s+=$3} END {print s}'
  219590

#All forward feature mappings: 630,904
411314 + 219590 = 630904

#Reverse R1 Feature Map Count
$ grep -v "^_" htseq_count_rvstranded.txt | awk '{s+=$2} END {print s}'
  9700357

#Reverse R2 Feature Map Count
$ grep -v "^_" htseq_count_rvstranded.txt | awk '{s+=$3} END {print s}'
  4805275

#All reverse feature mappings: 14,505,632
9700357 + 4805275 = 14505632

#Feature mapping comparison
14505632 / 630904 = 22.992

#The reverse count yielded essentially 23x more feature mappings than the
#forward count did. Interesting...
```