QAA

Lisa Adriani

2022-09-01

Part 1

1.1

16.3D.mbnl.S12.L008

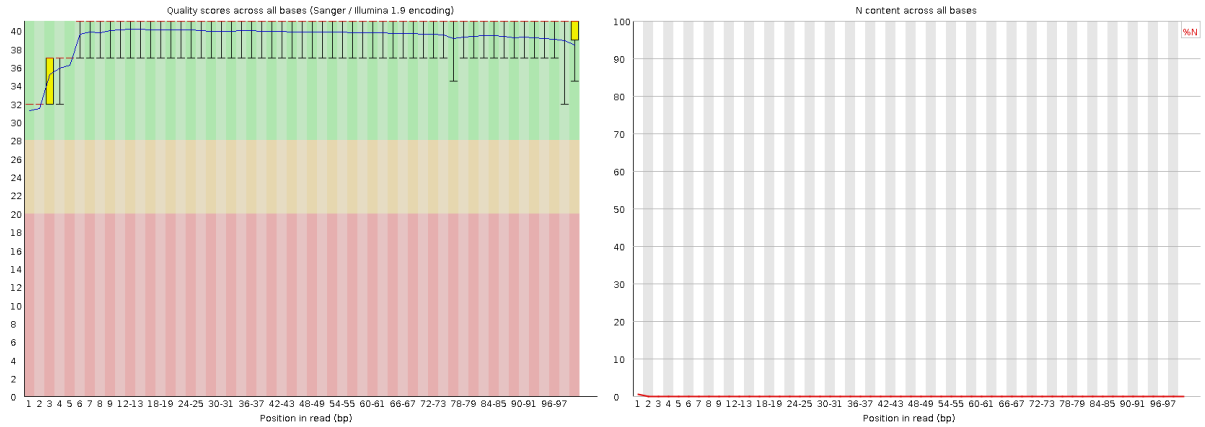

Figure 1: 16.3D.mbnl.S12.L008 – Read 1 Quality Score and N Content: provided from FastQC data output

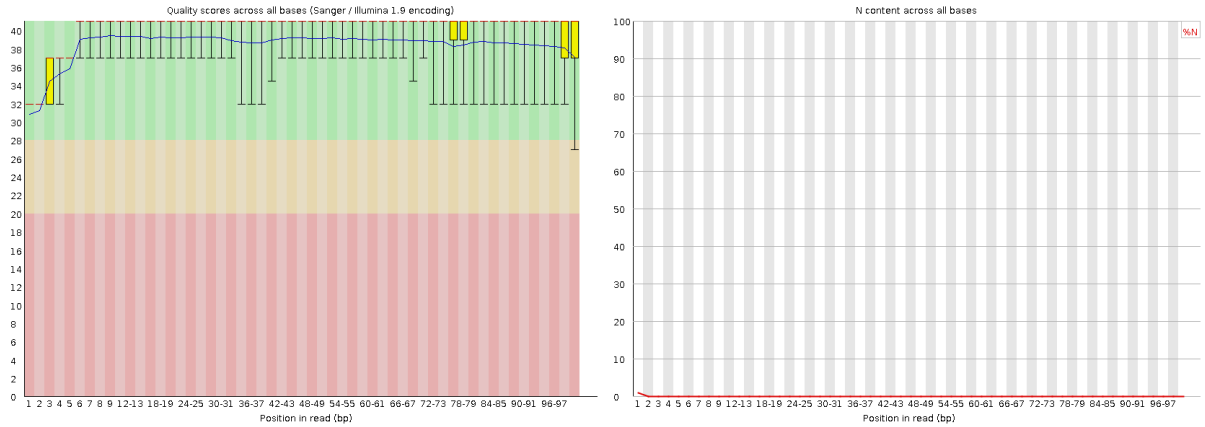

Figure 2: 16.3D.mbnl.S12.L008 – Read 2 Quality Score and N Content: provided from FastQX data output

32.4G.both.S23_L008

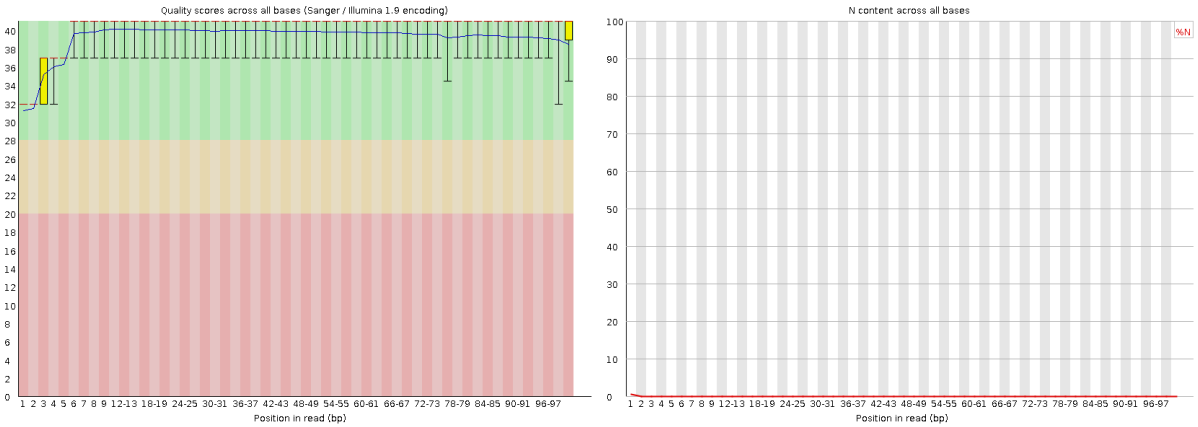

Figure 3: 32.4G.both.S23.L008 – Read 1 Quality Score and N Content: provided from FastQC data output

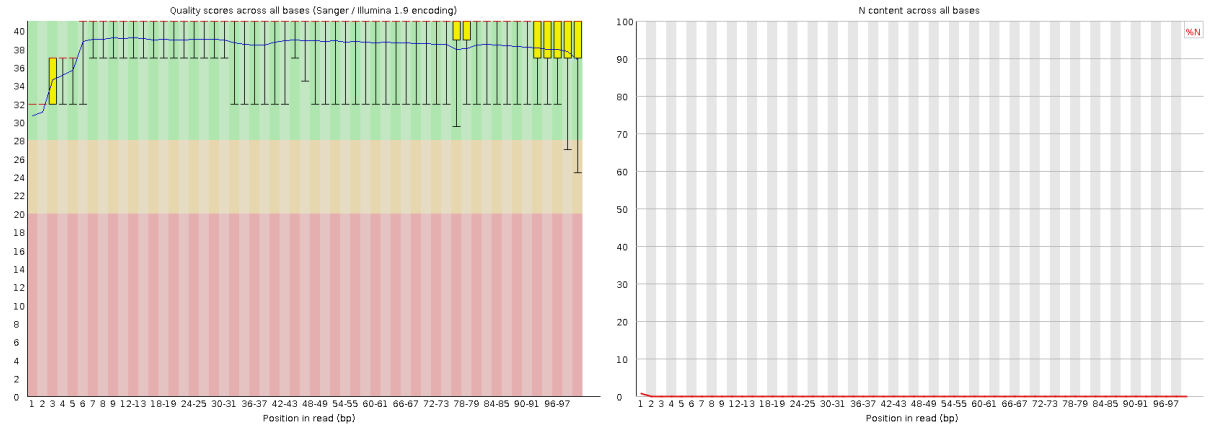

Figure 4: 32.4G.both.S23.L008 – Read 2 Quality Score and N Content: provided from FastQC data output

For each of these reads, we can see the greatest drop of Quality Score at the beginning of the sequence, in the first five bases or so. When compared to the N content, while there are a minimal amount of N's, we can see the greatest number of N's show up in the first position of the reads. Overall, the quality score is consistent with the N content of each of my reads.

1.2

The differences between the quality per position graphs of FastQC and the Demultiplex Assignment at first glance extremely different because the FastQC data has error bars, which the Demultiplex graphs do not. Overall, the graphs show similar trends within samples, but the variance information leads the FastQC graphs to be much more useful. FastQC was also able to run QC analyses and return graphed data for all four zipped files in just under 4 minutes, while my Demultiplex histogram code needed 3.5-5 minutes *per* file, just in order to calculate and graph the average Quality Score. First off, FastQC was not written by a new bioinformatics student, and I'm sure I do not have the most elegant code. To add to that, FastQC is written in java, which is a compiled program language, which typically runs in a more efficient manner than Python.

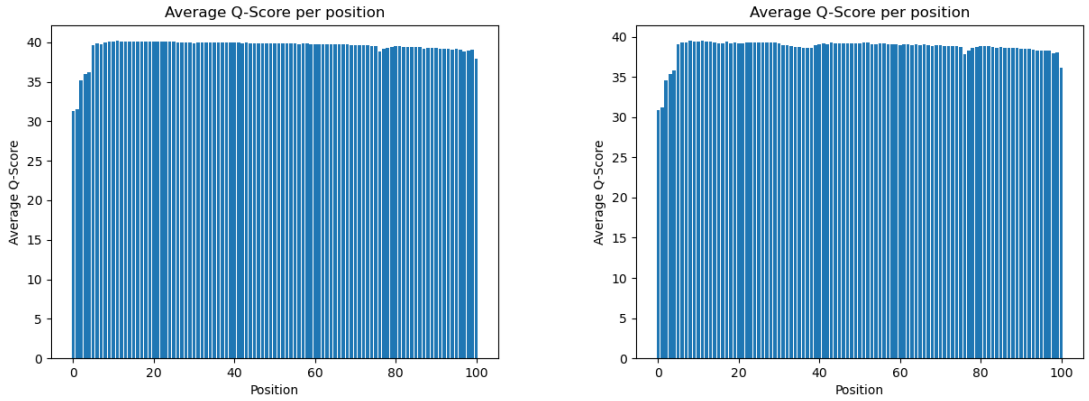

Figure 5: 16.3D.mbnl.S12.L008 – Read 1 and Read 2 Quality Score Distribution from Demultiplex Algorithim

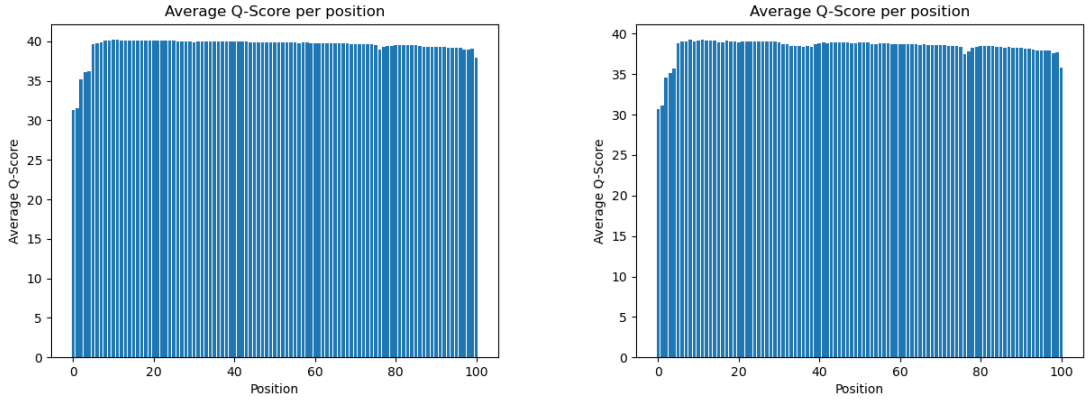

Figure 6: 32.4G.both.S23 – Read 1 and Read 2 Quality Score Distribution from Demultiplex Algorithim

1.3

Overall, the quality of my libraries is great! There is an extremely low N Percentage, the quality scores keep their average per base to be quite high as well, typically about 35 after the first five bases. The second read for both samples has a bit more variance in the quality score, especially towards the ends of the reads, but that is to be generally expected due to time on the machine and potential reagent shortage.

Part 2

2.5

From the cutadapt output for **16.3D.mbnl.S12.L008**, I can see that 12.2% of Read 1 was trimmed and 13% of read 2 was trimmed.

Read 1 with adapter: 1,002,983 (12.2%)

Read 2 with adapter: 1,069,893 (13.0%)

From the cutadapt output for **32.4G.both.S23**, I can see that 5.3% of Read 1 was trimmed and 6.1% of Read 2 was trimmed.

Read 1 with adapter: 631,720 (5.3%)

Read 2 with adapter: 725,571 (6.1%)

2.5 Sanity Check:

I checked to make sure that there were no longer adapter sequences by using a grep command:

```
cat output_16.R1.fastq | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
cat output_16.R2.fastq | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
cat output_16.R1.fastq | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
cat output_16.R2.fastq | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
cat output_32.R2.fastq | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
cat output_32.R1.fastq | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
cat output_32.R1.fastq | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
cat output_32.R2.fastq | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
```

This grep command created no output, so I know that these adaptor sequences were pulled from the data. I double checked that they were in the original sequences by running the same commands (with zcat instead) on the files given to me.

2.7

I would expect Read 2's to be trimmed at a higher rate than Read 1. We can expect the best sequencing to occur at the beginning of the reaction, because by the end of Read 2, the samples have been on the sequencer for an incredibly long time, and the reagents may start to degrade or bind to the incorrect positions on the insert. If we look at Figure 7, this looks like it is what happened, as there are slightly higher rates of trimming in both Read 2's rather than Read 1's.

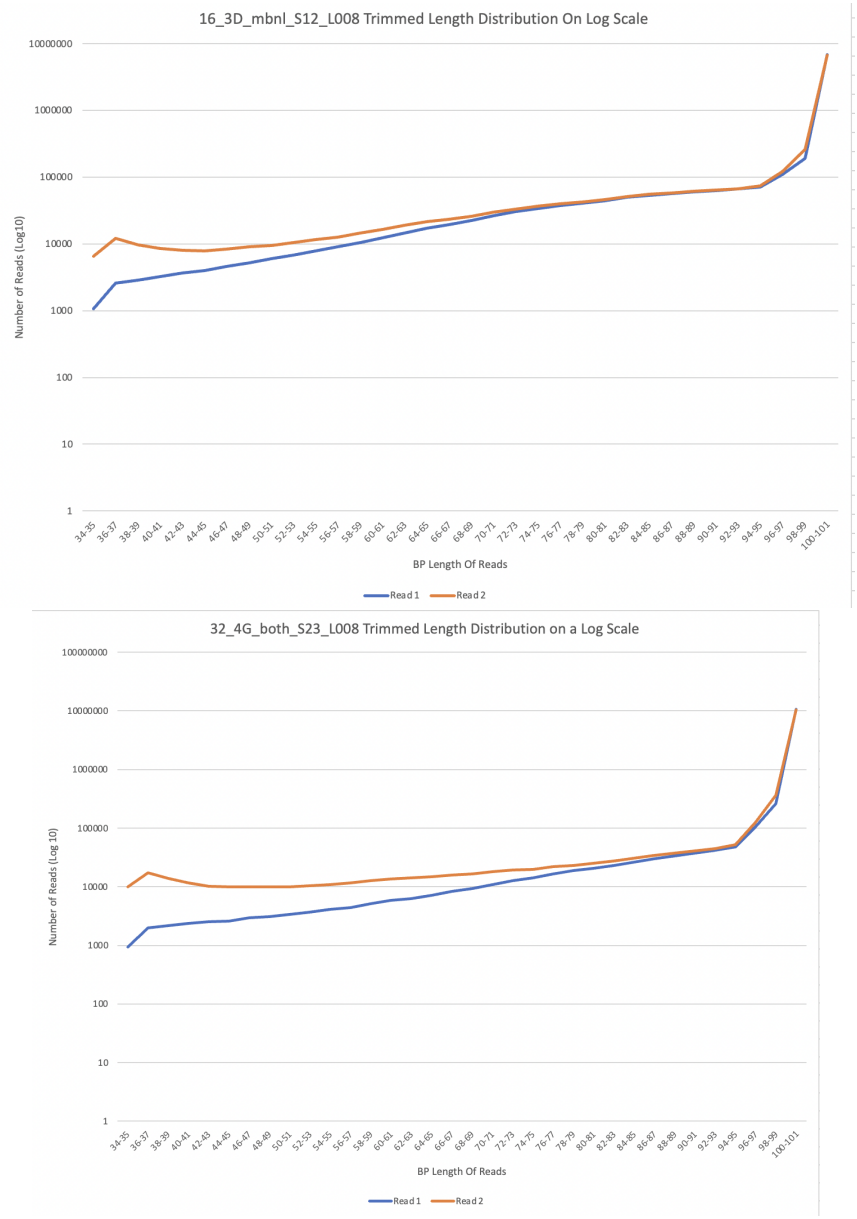

Figure 7: Trimmed Length Distribution on a Log Scale: Results from FastQC output after cutadapt and trimming with Trimmomatic

Part 3

3.12

Data	Mapped	UnMapped
16_3D_mbnl_S12_L008	15662613	365703
32_4G_both_S23_L008	22404329	533603

Figure 8: Table of output from map.py: Mapped VS Unmapped reads

```
#Used in order to calculate the mapped reads percentage the data using two different settings.
read1="htseq_output/32_htseq_yes.txt"
read2="htseq_output/32_htseq_reverse.txt"
read3="htseq_output/16_htseq_yes.txt"
read4="htseq_output/16_htseq_reverse.txt"
Map1=$(awk '$1~"ENSMU"{sum+= $2} END {print sum}' $read1)
Map2=$(awk '$1~"ENSMU"{sum+= $2} END {print sum}' $read2)
Map3=$(awk '$1~"ENSMU"{sum+= $2} END {print sum}' $read3)
Map4=$(awk '$1~"ENSMU"{sum+= $2} END {print sum}' $read4)
total1=$(awk '{sum+= $2} END {print sum}' $read1)
total2=$(awk '{sum+= $2} END {print sum}' $read2)
total3=$(awk '{sum+= $2} END {print sum}' $read3)
total4=$(awk '{sum+= $2} END {print sum}' $read4)
$Map1/$total1 = 3.96%
$Map2/$total2 = 86.4%
$Map3/$total3 = 4.16%
$Map4/$total4 = 85.8%
```

I propose that these data are both strand specific. When htseq-count was set to "reverse", over 85% of both of their reads were mapped to a feature. Setting htseq count to reverse means that the strands are considered paired end reads, where the second strand has to be the first read and the opposite strand has to be the first read. Only about 4% for both of the data were mapped to a feature when the data was treated as a single end read.