RNA-seq Quality Assessment Assignment

Claire Wells

2024-09-08

RNA-seq Quality Assessment Assignment

The following report encapsulates a comprehensive pipeline of RNA-seq Quality Assessment through the use of various tools such as FastQC, Trimmomatic and others. In addition to utilizing well known tools available for use, we were able to further authenticate FastQC for preliminary quality data assessment through processing quality score data through our own developed Python scripts. For this assignment, we were each given two samples with two corresponding files for Read 1 and Read 2 respectively. For the sake of simplicity, I will be referencing the file names by a designated reference name as shown in Table 1.

File Name	Reference Name	Num. Records	File Size	Read Length
11_2H_both_S9_L008_R1_001.fastq.gz	Sample 11_R1	17919193	917M	101
11_2H_both_S9_L008_R2_001.fastq.gz	Sample 11_R2	17919193	987M	101
14_3B_control_S10_L008_R1_001.fastq.gz	Control_R1	4440378	231M	101
14_3B_control_S10_L008_R2_001.fastq.gz	Control_R2	4440378	260M	101

Table 1: Summary of file names and initial data exploration

Part 1: Read quality score distributions

Sample 11

The initial step to quality assessment is to run these files through FastQC in order to get an overall idea for read quality score distributions as well as other preliminary data for the overall quality of these files. In addition to running these files through FastQC, these files were also run through a Python script that similarly outputs an average quality score plot. These plots are shown in Figures 1 through 4 below. Another FastQC output included in this report are the per base N content for each of the files shown in Figures 5 and 6.

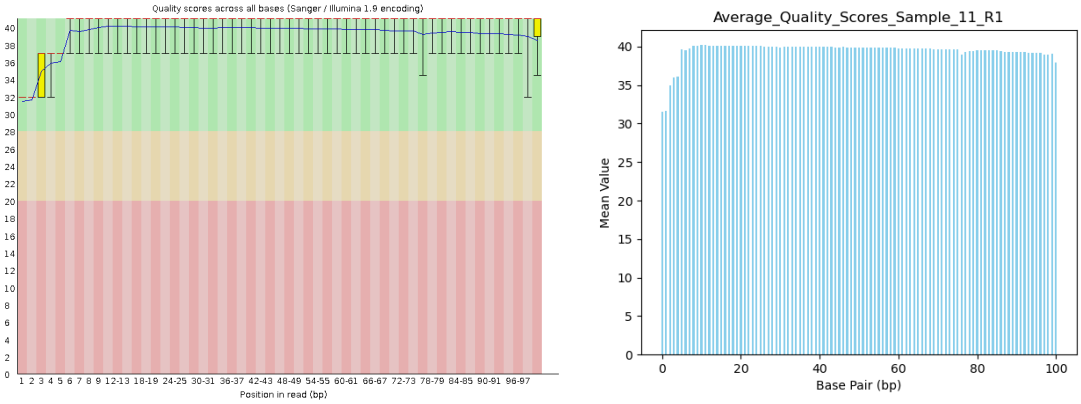

Figure 1: Read 1, Sample 11: Per Base Quality Score Graphs from FastQC (left) compared to manual Python qscore script

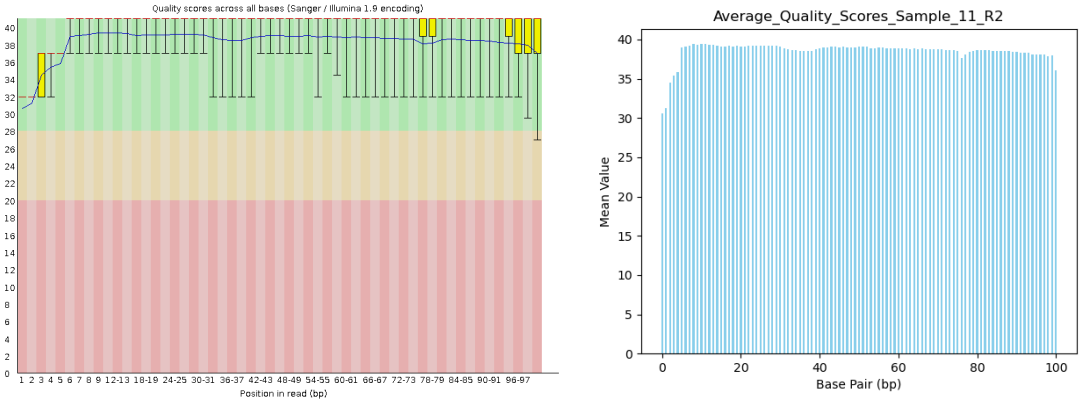

Figure 2: Read 2, Sample 11: Per Base Quality Score Graphs from FastQC (left) compared to manual Python qscore script

Control

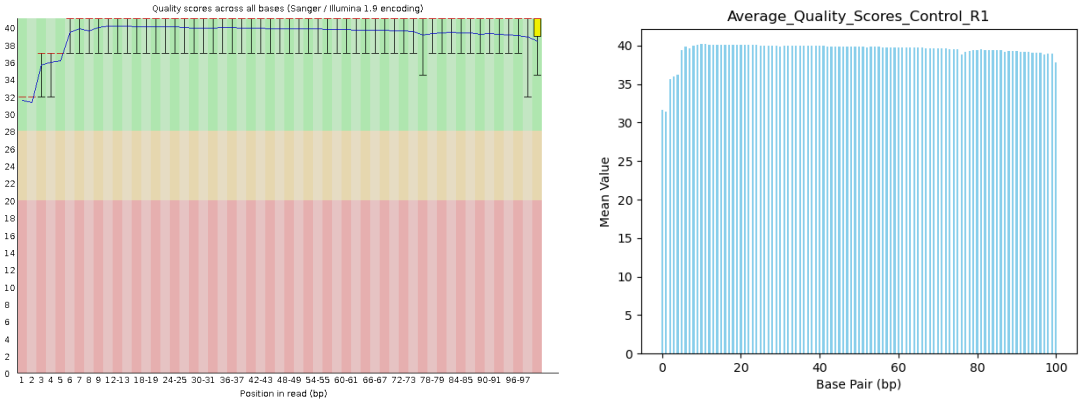

Figure 3: Read 1, Control: Per Base Quality Score Graphs from FastQC (left) compared to manual Python qscore script

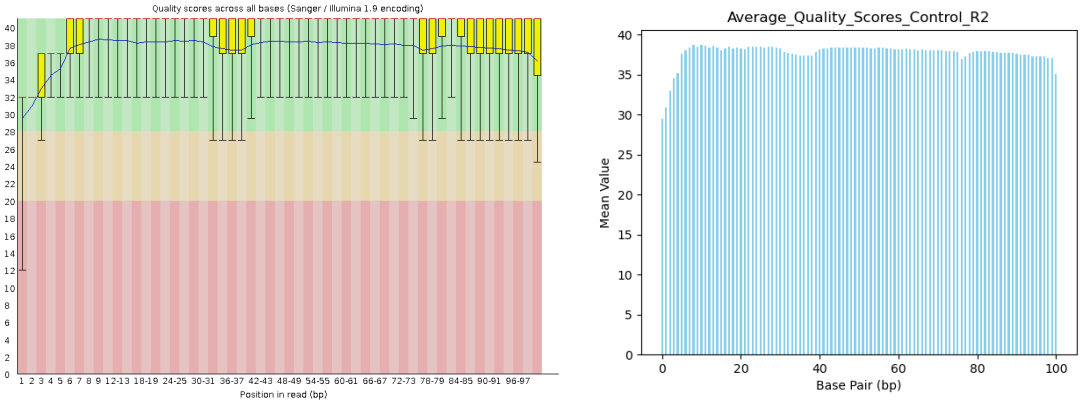

Figure 4: Read 2, Control: Per Base Quality Score Graphs from FastQC (left) compared to manual Python qscore script

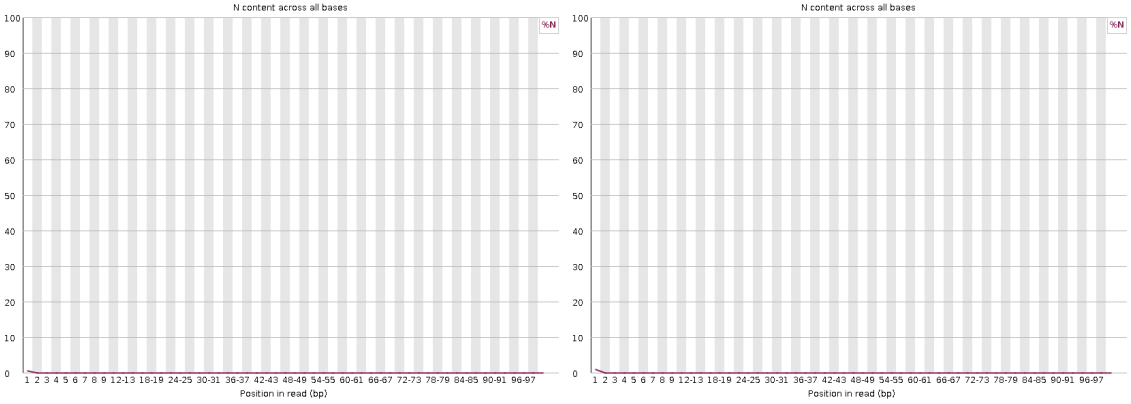

Figure 5: Per Base N Content for Sample 11 for Read 1 (left) and Read 2 (right)

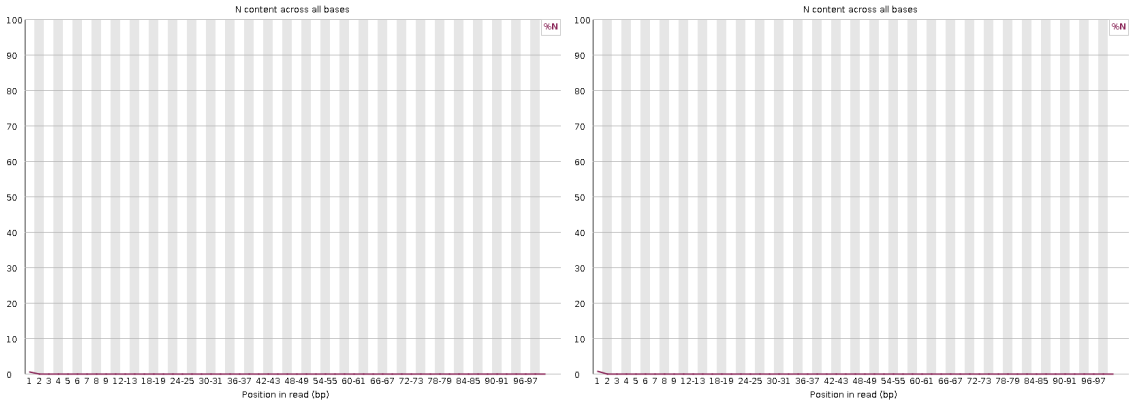

Figure 6: Per Base N Content for the Control for Read 1 (left) and Read 2 (right)

Comparing FastQC outputs to My Own Python Script

When looking at these plots, they are consistent with one another in terms of general trend. However, it is important to note that though they depict similar trends, the graphs output by FastQC provide more information such as the interquartile ranges which provide information regarding the spread of the data. In addition, FastQC also provides information about what might be considered an optimal cutoff for quality scores. With that, a downside to the FastQC Per Base Quality Score graph compared to the graph output by my own script is that FastQC's graph is binned on the x-axis which does result in a loss of some data.

Additionally, when looking at the FastQC plots of per-base N content, we see consistency with the quality score plots. In all the N-content plots, we are seeing a small spike in N at the beginning of the plot and we see a corresponding drop in quality on plots depicting quality score. In terms of the runtime between FastQC and my own script, I would say that though there was a significant difference, it was not nearly as signficant as the difference some of my peers saw due to do how I chose to run my sbatch scripts. Instead of writing one sbatch script with all four commands, I instead chose to write an sbatch script for each individual file for the sake of my own organization.

As a result, the run times for these two programs differed only slightly. For FastQC, run time was on average 67.3 seconds for Sample 11 files and an average of 19.49 seconds for Control files which were substantially smaller than Sample 11 files. In comparison, my own script took an average of 233.97 seconds for Sample 11 and 36.53 seconds for Control Samples. As for CPU usage, for both FastQC and my own script, the amount of CPU used was comparable at around 99%. It's interesting that despite FastQC having far more outputs than my own script, it is still faster. The reasoning for this is because FastQC is written in java which is computationally faster to process in comparison to my own script which is written in python. Additionally, my own script was written quickly by me in the span of a few days while FastQC has been adapted by many individuals since 2010 by many different individuals.

Overall Data Quality

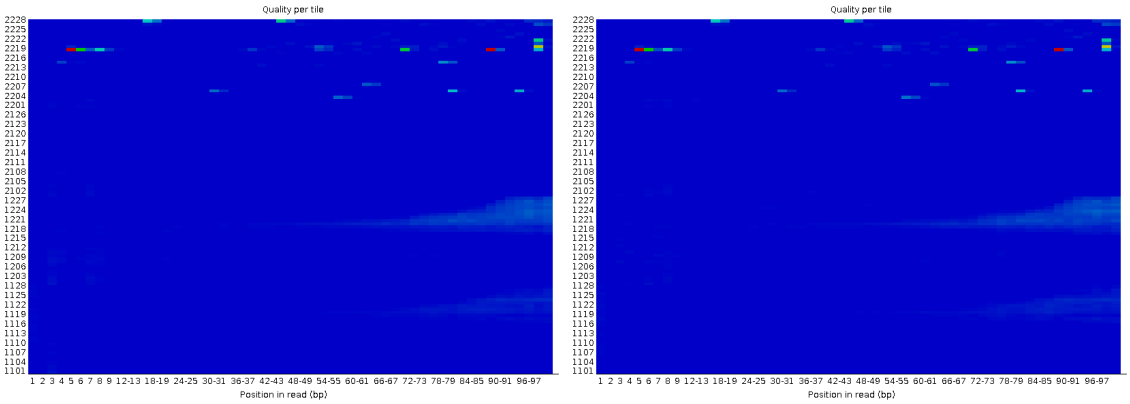

Figure 7: Per Tile Sequence Quality FastQC Output for Read 1 in Sample 11 (Left) and Control (Right)

In general, the overall data quality for both Sample 11 and the Control Sample are of high enough quality for further analysis. While quality score distributions demonstrate that these reads are of good enough quality for analysis, we can further support that our data is of good quality by looking to the Per Tile Sequence Quality FastQC output (Figure 7). In these particular figures, we are able to gain further insight into whether or not there was any deviation in quality at a given area on the flow cell. The figures depicted are for Read 1 from each of the two samples and we can clearly see that although there are some deviations in average quality indicated by the red tiles, in general, there aren't any clear patterns of decline in quality at any particular area. We can further support the overall quality of our data by looking at Per Base N-Content. As shown in Figures 5 and 6, we can see that across all samples, there are no N's with the exception of a slight uptick in N's at the very beginning of the graph. Since this uptick is so minor in addition to the fact that there are no other upticks across all the graphs, we can affirm that this data is of high enough quality to proceed.

Part 2: Adapter trimming comparison

Once we established that the data was of high enough quality we proceeded to run our data through Cutadapt and Trimmomatic. Cutadapt was used to trim adapter sequences. When analyzing Cutadapt outputs, we found that only 4.9% of Read 1 and 5.7% of Read 2 in Sample 11 contained adapters. For the Control sample, 6.0% of Read 1 and 6.7% of Read 2 had adapter sequences. Following Cutadapt, Trimmomatic was then used for quality trimming.

Read 1: AGATCGGAAGAGCACACGTCTGAACTCCAGTCA

Read 2: AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT

Figure 8: Output of bash command highlighting orientation of adapter sequences in Sample 11 Read 2

Adapter sequences were initially recognized by searching for common Illumina adapters and grep'ing for them within our file in order to narrow down what the potential adapter might be. From there, we were able to confirm the adapter sequences by looking at the hint provided in the ReadME. In order to confirm the presence of adapter sequences in the file, we ran a simple bash command that allowed us to see where the adapter sequences were oriented in the read

(Figure 9). An example output of that is demonstrated in Figure 10. As you can see in the figure, we see the bulk of the adapter sequences on the right hand side of the sequence which makes logical sense because Illumina sequences reads from the 5' to 3' end.

```
zcat /path/to/file | grep "ADAPTOR"| less -p ADAPTOR
```

Figure 9: Bash Command Used to Visualize Adapter Sequence Orientation

After performing adapter and quality trimming using Cutadapt and Trimmomatic, we then visualized trimmed read length distributions for Read 1 and Read 2 reads for each sample compiled into one plot for comparison purposes. As shown in both Figures 11 and 12, we can clearly see that R1 has fewer short reads while R2 is more heavily trimmed in comparison to R1. We might expect Read 1 and Read 2 to be adapter trimmed at different rates because Read 2 has more adapters than Read 1 as shown from the Cutadapt output. This is a result of Read 2 being the last thing to be sequenced on the flow cell resulting in degradation of DNA and therefore resulting in lower quality. For Read 2, because it is the last thing being sequenced, it is being overrepresented in comparison to Read 1 resulting in us seeing a higher proportion of adapters in R2 in comparison to R1 because of signal degradation meaning wells are calling adapters over the signal of good quality reads which is weaker in comparison.

Figure 10: Output of bash command highlighting orientation of adapter sequences in Sample 11 Read 2

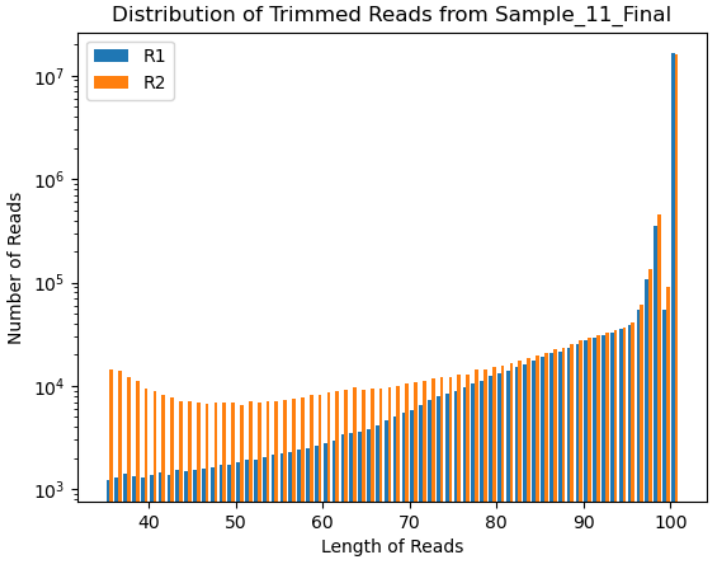

Figure 11: Trimmed Distribution from Cutadapt and Trimmomatic Outputs of Read 1 and Read 2 Reads from Sample 11

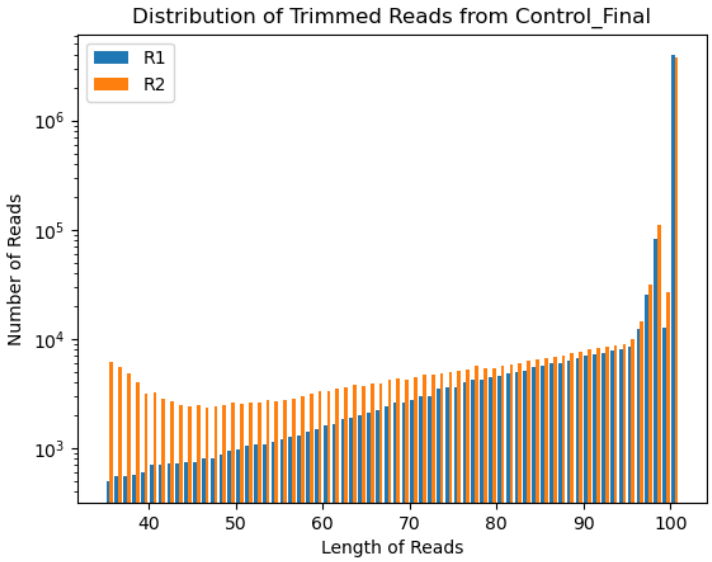

Figure 12: Trimmed Distribution from Cutadapt and Trimmomatic Outputs of Read 1 and Read 2 Reads from the Control Sample

Part 3: Alignment and strand specificity

After trimming reads for adapters and quality, we then moved on to aligning our reads using STAR and HTSeq Count. After aligning reads using STAR, we used htseq-count to count the number of reads in features from our STAR outputs. We ran htseq-count a total of four times with the stranded argument being stranded = yes and stranded = reverse for each of our respective alignments. In doing this, we found that in both the Control and Sample 11, the

percent of mapped reads was substantially higher at 86.3% (Control) and 79.1% (Sample 11) respectively when using the stranded = reverse argument (Table 2). In comparison, when using the stranded = yes argument for the Control and Sample, the percentage of mapped reads plummeted significantly to 3.8% (Control) and 3.1% (Sample 11) of reads mapped (Table 2). Given these results, we can see that there is a clear preference for mapped reads when stranded = reverse implying that this is a strand specific library.

Sample	Stranded	Percent Mapped Reads
Control	Yes	3.8%
Control	Reverse	86.3%
Sample 11	Yes	79.1%
Sample 11	Reverse	3.2%

Table 2: Percent Mapped Reads with Varying Stranded Arguments in htseq-count

Sample	Mapped Reads	Unmapped Reads
Control	8312390	180914
Sample 11	33637699	1293527

Table 3: The number of mapped and unmapped reads from PS8 script