QAA_report

Anna Grace Welch

2023-09-14

Contents

Part 1 **1**
22_3H_both_S16_L008 Quality Plots . . . 2
23_4A_control_S17_L008 Quality Plots . . . 4
Discussion . . . 6

Part 2 **7**
Cutadapt Results . . . 7
Trimmed Read Length Distribution after Trimmomatic . . . 7
Discussion . . . 8

Part 3 **8**
Mapped and Unmapped Reads for Libraries . . . 8
Mapped and Unmapped Reads for htseq-count Files . . . 9
Discussion . . . 9

Part 1

The goal of this project was to utilize tools for quality assessment and adaptor trimming and compare certain outputs to my own quality assessment from scripts I wrote.

The libraries examined in this project were 22_3H_both_S16_L008 and 23_4A_control_S17_L008.

The first part of this project focused on running FastQC on the read files. The per base quality score distributions, per base N content distributions, and per tile sequence quality plots were generated from this and evaluated.

The entirety of the FASTQC results for both libraries is summarized below.

FASTQC Summary Report:
A. Basic Statistics.
B. Per Base Sequence Quality.
C. Per Tile Sequence Quality.
D. Per Sequence Quality Scores.
E. Per Base Sequence Content.
F. Per Sequence GC Content.
G. Per Base N Content.
H. Sequence Length Distribution.
I. Sequence Duplication Levels.
J. Overrepresented Sequences.
K. Adapter Content.
L. Kmer Content.

Sample	A	B	C	D	E	F	G	H	I	J	K	L
22_3H_R1	PASS	PASS	FAIL	PASS	WARN	PASS	PASS	PASS	PASS	PASS	PASS	FAIL
22_3H_R2	PASS	PASS	WARN	PASS	WARN	PASS	PASS	PASS	PASS	PASS	PASS	FAIL
23_4A_R1	PASS	PASS	FAIL	PASS	FAIL	PASS	PASS	PASS	FAIL	WARN	PASS	FAIL
23_4A_R2	PASS	PASS	WARN	PASS	WARN	WARN	PASS	PASS	FAIL	WARN	PASS	FAIL

Table 1. FastQC Summarized Results. The table above shows whether each test of FastQC passed, failed, or showed a warning for each libraries' R1 and R2 reads.

22_3H_both_S16_L008 Quality Plots

FastQC Results

The per base quality score distributions and per base N content distributions were generated and looked at further.

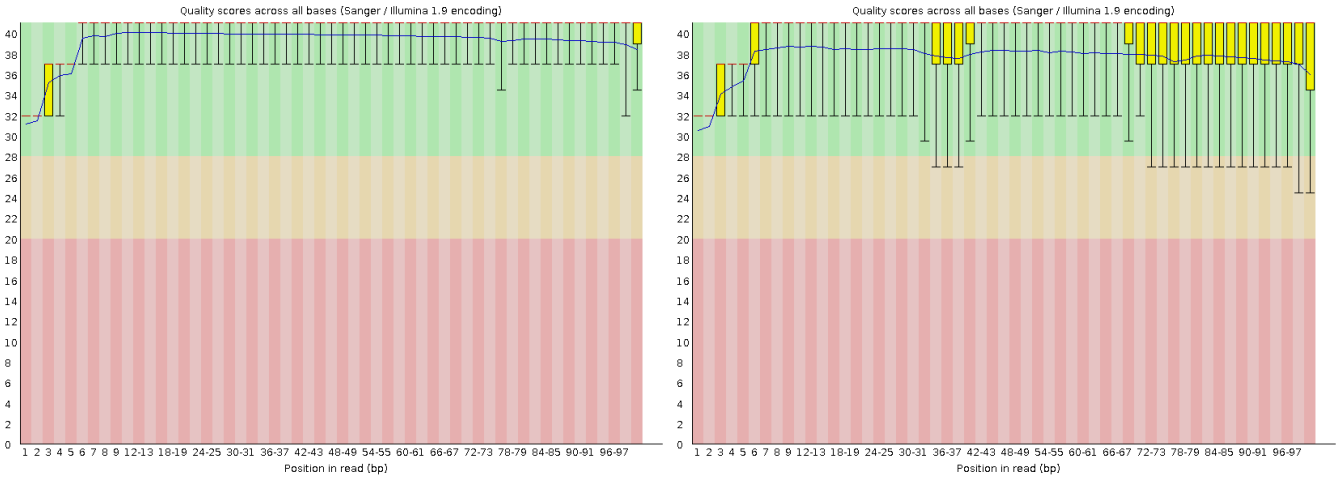

Figure 1: 22-3H-both-S16-L008 Per Base Quality Score Distribution: Read 1 (left) and Read 2 (right)

Figure 1. 22_3H_both_S16_L008 Fastqc Per Base Quality Distribution Plot. R1 (left) and R2 (right). This figure displays the distribution of quality scores across all base positions for the library 22_3H_both_S16_L008. The plot for R1 reads is shown on the left, while the plot for R2 reads is on the right. The y axis displays quality scores, and the x axis shows base pair position. blue line across the graph represents the mean quality score for each position, while the central red line is the median. The top and bottom whiskers represent the 10% and 90% points, and the yellow box represents inter-quartile range.

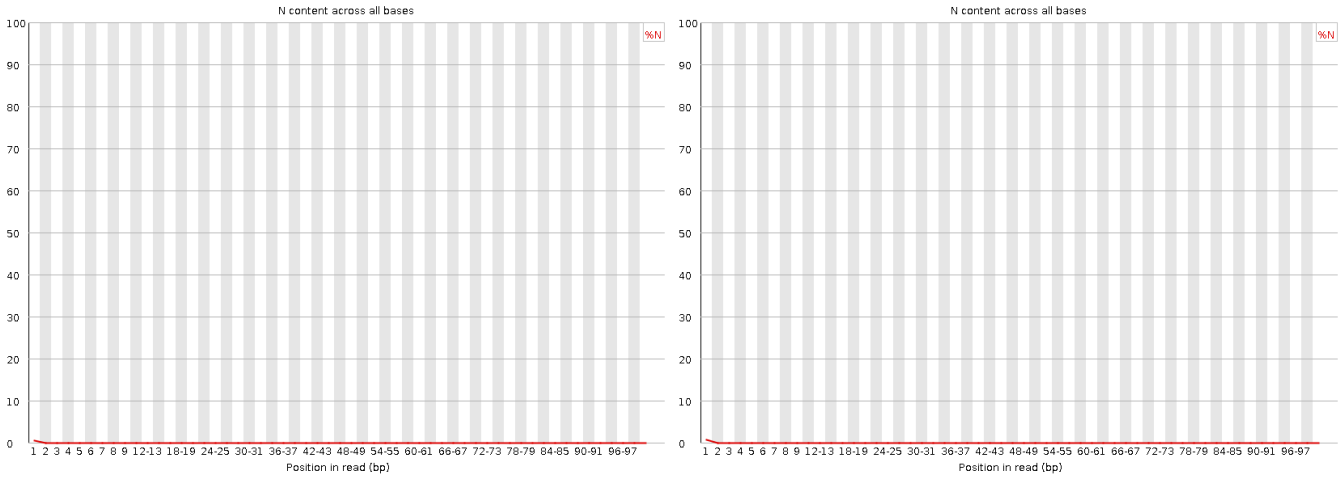

Figure 2: 22-3H-both-S16-L008 Per Base N Content: Read 1 (left) and Read 2 (right)

Figure 2. 22_3H_both_S16_L008 Per Base N Content. R1 (left) and R2 (right). This plot shows the per base N content across all base positions for R1 and R2 reads for the library 22_3H_both_S16_L008. The red line represents the N content for each base.

Python Script Quality Distribution.

The read files were also run through my Python script part1.py from the Bi622 Demultiplexing assignment. This was done in order to compare output from this script to distributions generated from fastqc.

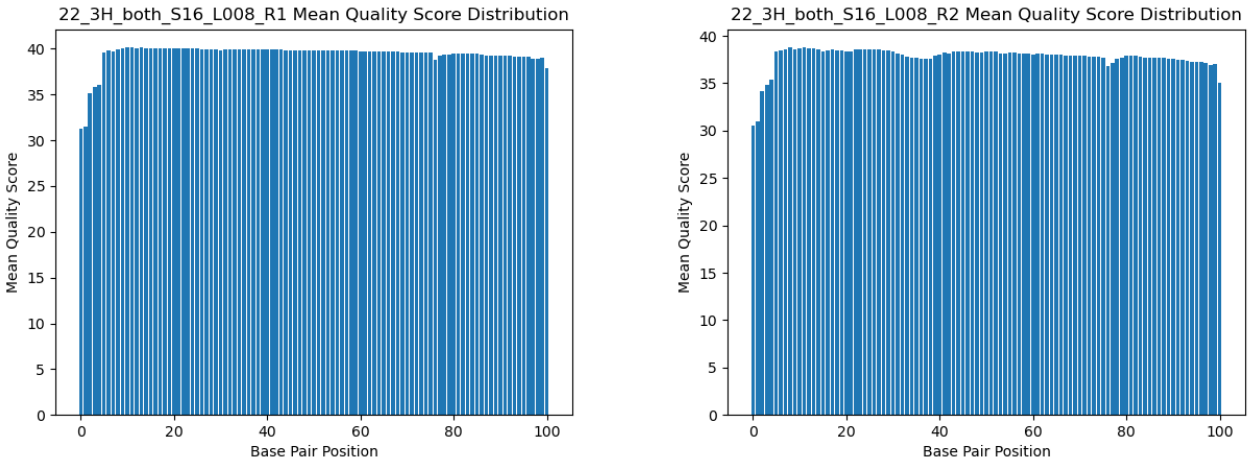

Figure 3: 22-3H-both-S16-L008 Python Quality Score Distribution: Read 1 (left) and Read 2 (right)

Figure 3. 22_3H_both_S16_L008 Python Script Per Base Quality Score Distribution. R1 (left) and R2 (right). This plot shows the quality score distribution across all base positions for R1 and R2 reads for the library 22_3H_both_S16_L008, as generated by my Python script part1.py found in my Demultiplex GitHub repository. The x-axis shows the base pair position while the y-axis shows the mean quality score for each position.

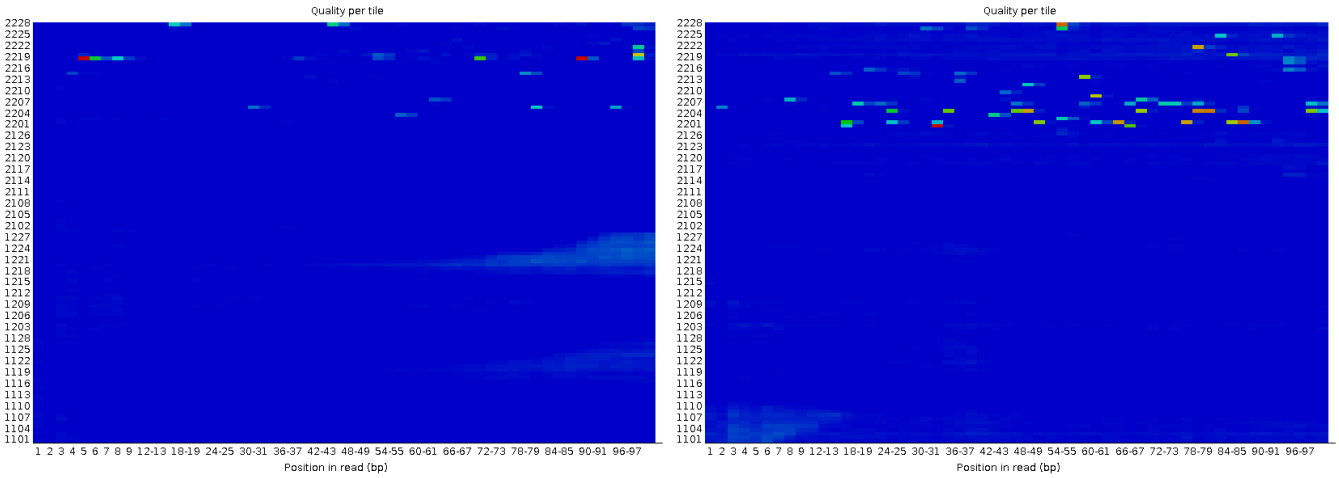

Figure 4: 22-3H-both-S16-L008 Per Tile Sequence Quality: R1 (left) and R2 (right)

Figure 4. 22_3H_both_S16_L008 Per Tile Sequence Quality: R1 (left) and R2 (right). This figure shows the quality per sequence tile from the Illumina flowcell for library 22_3H_both_S16_L008 R1 and R2 reads. It shows the deviation from average quality for each tile on a cold-to-hot scale. Cold colors such as blue or green represent positions at or above average quality for that base while hot colors such as red represent worse qualities than other tiles for that particular base.

23_4A_control_S17_L008 Quality Plots

FASTQC Results

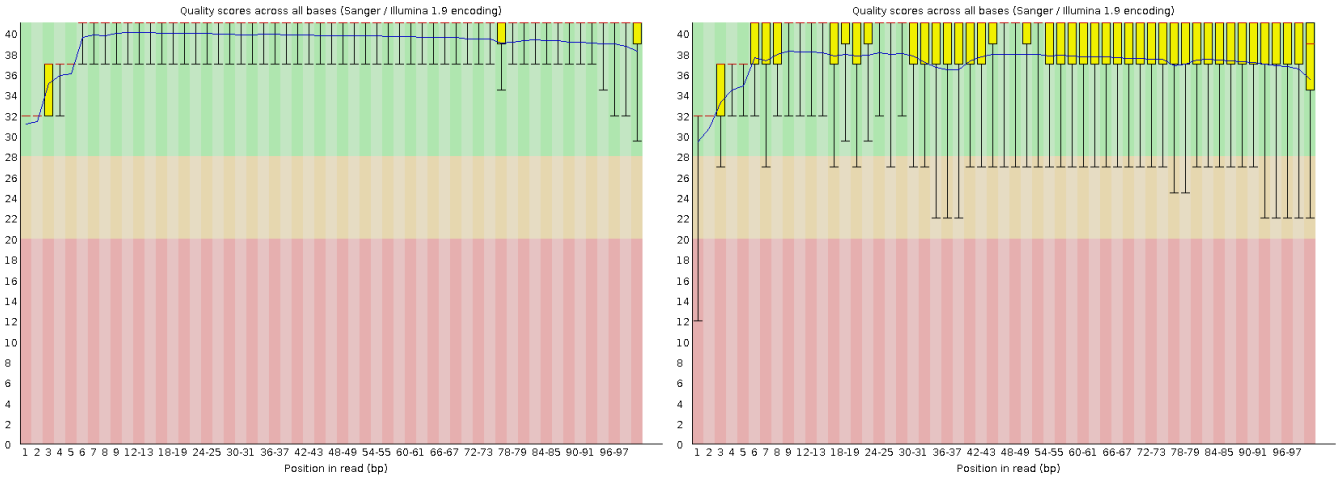

Figure 5: 23-4A-control-S17-L008 Per Base Quality Distribution: Read 1 (left) and Read 2 (right)

Figure 5. 23_4A_control_S17_L008 Fastqc Per Base Quality Distribution Plot. R1 (left) and R2 (right). This figure displays the distribution of quality scores across all base positions for the library 23_4A_control_S17_L008. The plot for R1 reads is shown on the left, while the plot for R2 reads is on the right. The y axis displays quality scores, and the x axis shows base pair position. The blue line across the graph represents the mean quality score for each position, while the central red line is the median. The top and bottom whiskers represent the 10% and 90% points, and the yellow box represents inter-quartile range.

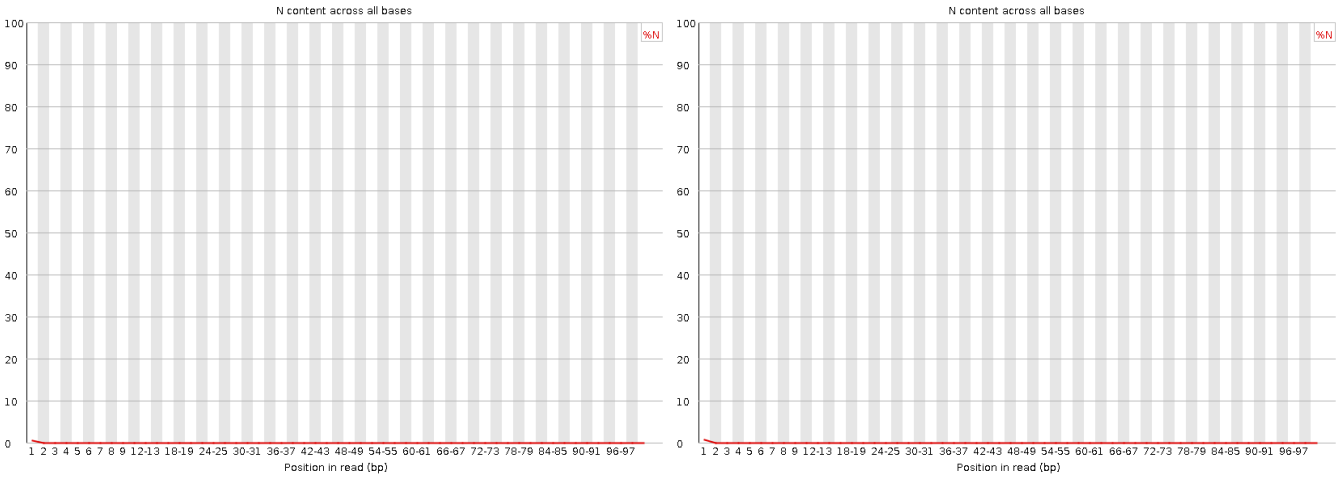

Figure 6: 23-4A-control-S17-L008 Per Base N Content: Read 1 (left) and Read 2 (right)

Figure 6. 23_4A_control_S17_L008 Per Base N Content. R1 (left) and R2 (right.) This figure shows the per base N content for R1 and R2 reads for the library 23_4A_control_S17_L008. The x-axis shows position in read. The red line across the graph represents the N content for each base position.

Python Quality Score Distribution

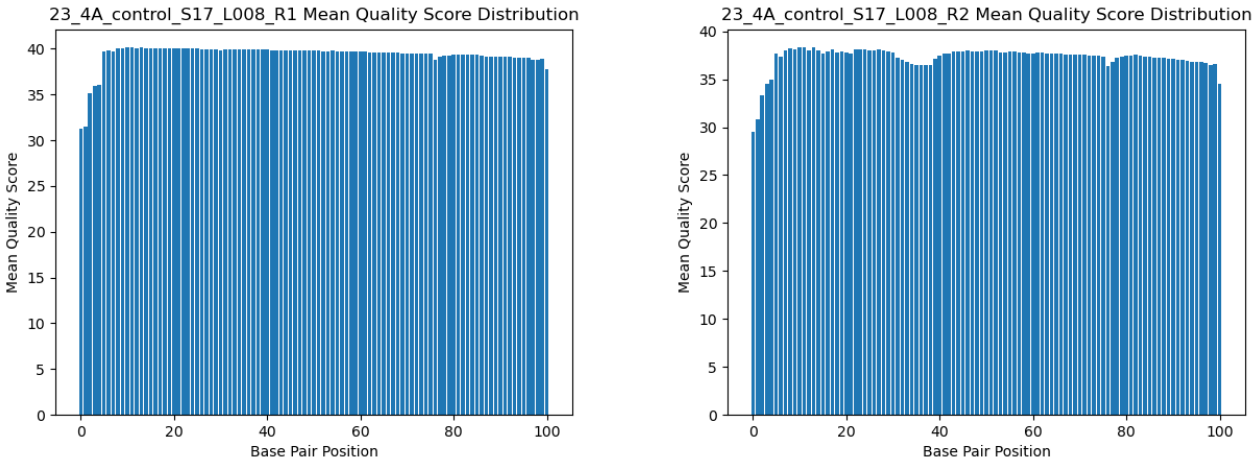

Figure 7: 23-4A-control-S17-L008 Python Quality Score Distribution: Read 1 (left) and Read 2 (right)

Figure 7. 23_4A_control_S17_L008 Python Script Per Base Quality Score Distribution. R1 (left) and R2 (right). This plot shows the quality score distribution across all base positions for R1 and R2 reads for the library 23_4A_control_S17_L008, as generated by my Python script part1.py found in my Demultiplex GitHub repository. The x-axis shows the base pair position while the y-axis shows the mean quality score for each position.

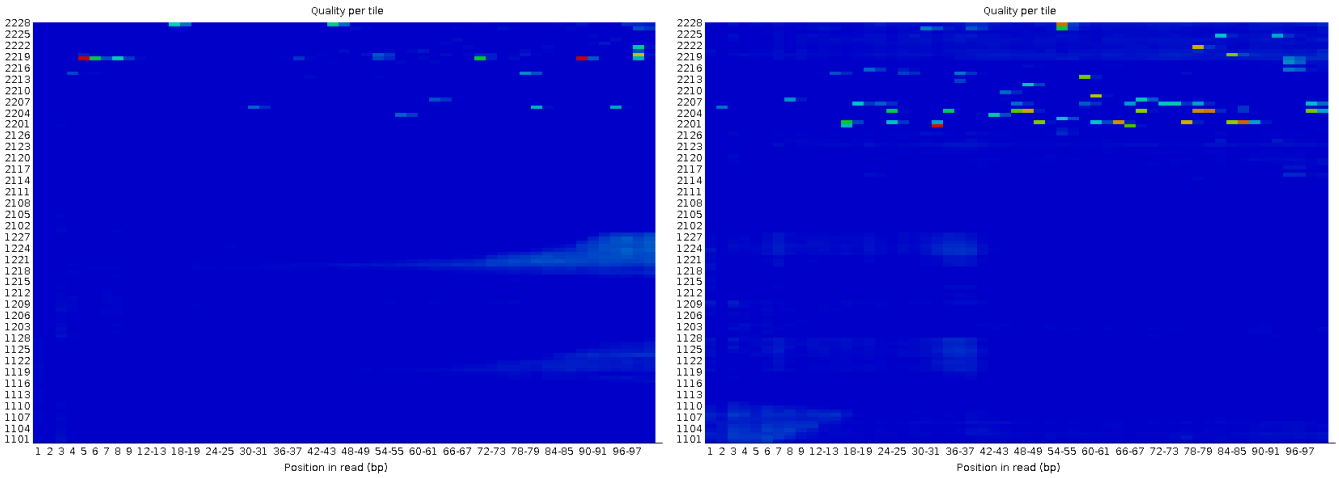

Figure 8: 23-4A-control-S17-L008 Per Tile Sequence Quality: R1 (left) and R2 (right)

Figure 8. Per Tile Sequence Quality: R1 (left) and R2 (right). This figure shows the quality per sequence tile from the flowcell for library 23_4A_control_S17_L008 R1 and R2 reads. It shows the deviation from average quality for each tile on a cold-to-hot scale. Cold colors such as blue or green represent positions at or above average quality for that base while hot colors such as red represent worse qualities than other tiles for that particular base.

Discussion

Figure 1 shows relatively high quality scores across all base pair positions in R1 and R2 reads for library 22_3H_both_S16_L008. However, R2 has more variability in quality, with lower quality scores being observed. This is to be expected as R2 is on the sequencer much longer than R1. The beginning of both reads has the lowest scores which is also to be expected as more sequencing error is often seen at the start. Similarly, Figure 4 shows decently high quality scores across all positions for R1 reads for library 23_4A_control_S17_L008, and R2 has much more variability with the whiskers dipping farther up and down from the mean line.

Figures 2 and 6 show per base N content that is consistent with observed quality distribution. There is generally low N content across all base positions for both reads in both libraries, with a slight uptick at the beginning of the sequence.

Compared to the fastqc-generated per base quality score distributions displayed in Figures 1 and 5, Figures 3 and 7 show a similar distribution of scores. Quality scores remain high across all positions, with the lowest scores being observed at the beginning of each sequence. There are slightly lower scores across the distribution for R2 reads.

FastQC was run on both reads of a library simultaneously. For library 22_3H_both_S16_L008, the run time was 42.64 seconds to create distribution plots for both reads. In contrast, my Python script took 58.74 seconds for R1 and 59.19 for R2 reads. Similarly, for library 23_4A_control_S17_L008, the FastQC run time was 6 minutes and 24.30 seconds to create distribution plots for R1 and R2 reads. My Python script took 10 minutes and 37.72 seconds and 10 minutes and 34.38 seconds for R1 and reverse reads, respectively. This difference in run time is likely due to the fact that FastQC has been optimized for speed and efficiency while my Python script was simply written for correctness of output.

To further assess the quality of my sequencing libraries, I examined the per tile sequence quality plots shown in Figures 4 and 8. A good tile quality plot would be all blue. However, while there is some red tiles present in the plots, it is generally only an issue for a couple cycles for the base pairs affected. It is not usually a

consistent low quality seen for any particular base position. This means the red or orange tiles are likely due to a short-term problem while sequencing such as a bubble passing through the flowcell.

The quality score distributions, per base N content plots, and per tile sequence quality maps all indicate that our libraries are likely of a high enough quality to proceed. Our mean quality scores per base do not dip below 30, meaning it is unlikely to have significant sequencing errors with an expected base call accuracy of 99.9%. Additionally, the per base N content plots had very, very few N's present in the sequences, with the ones that were present congregating at the beginning. This also suggests high quality data, along with the red tiles in the per tile plots being far apart and only persisting for one or two cycles for the affected positions. This makes it likely that our libraries have a small enough sequencing error to proceed without incident for downstream analysis and alignment.

Part 2

The goal of the second part of this project was to trim adapters from sequences and quality trim our data. This was done using cutadapt and trimmomatic.

Cutadapt Results

Below is the proportion of reads that were adapter-trimmed using cutadapt.

Sample	Proportion of Adapter-Trimmed Reads
22_3H_both_S16_L008_R1	3.8%
22_3H_both_S16_L008_R2	4.6%
23_4A_control_S17_L008_R1	3.1%
23_4A_control_S17_L008_R2	3.7%

Table 1: Proportion of reads that were adapter-trimmed. This table displays the amount of R1 and R2 reads that were trimmed by cutadapt for the libraries 22_3H_both_S16_L008 and 23_4A_control_S17_L008.

Trimmed Read Length Distribution after Trimmomatic

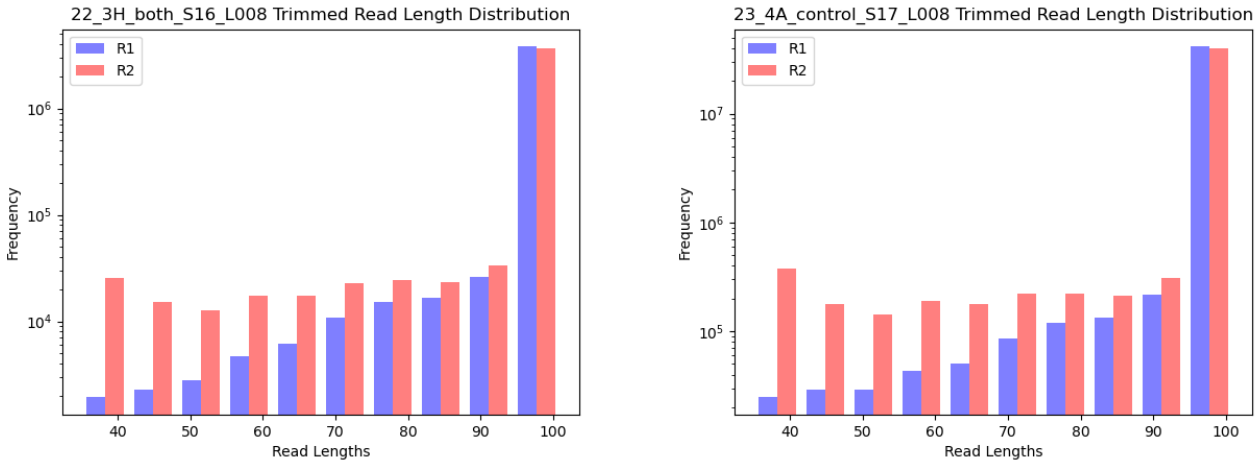

Figure 9: Trimmed Read Length Distribution: 22-3H-both-S16-L008 (left) and 23-4A-control-S17-L008 (right)

Figure 9. Trimmed Read Length Distribution. The distribution of trimmed read lengths after cutadapt and trimmomatic for library 22_3H_both_S16_L008 is shown on the left while the distribution for library 23_4A_control_S17_L008 is shown on the right. The x-axis shows read lengths binned from 40 to 100. The y-axis displays the frequency of each observed read length. Blue bars represent R1 reads, and red bars represent R2 reads.

Discussion

I determined the adapter sequences for my reads by first looking at my FASTQC results detailing the "Adapter Content". For R1 and R2 reads for both libraries, the only adapter visible on the graph was Illumina Universal Adapter. I was able to then web search to investigate what the sequence of this adapter was and found the sequences at https://dnatech.genomecenter.ucdavis.edu/wp-content/uploads/2019/03/illumina-adapter-sequences-2019-1000000002694-10.pdf.

The adapter sequences are detailed below:

R1 3' Adapter Sequence: AGATCGGAAGAGCACACGTCTGAACTCCAGTCA

R2 3' Adapter Sequence: AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT

I used the following commands to confirm adapter sequence orientations for R1 and R2 reads, respectively:

```
$ zcat <R1_file_path> | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
$ zcat <R1_file_path> | grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"

$ zcat <R2_file_path> | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
$ zcat <R1_file_path> | grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
```

I used grep to search for the adapter sequences in both R1 and R2 reads to confirm that the sequences were present in the expected reads and not present in the other. The first adapter was found in R1 reads while the second adapter was not. Similarly, the second adapter was found in R2 reads while the first adapter was not. This confirms the expected sequence orientations.

R2 adapters were trimmed more frequently than R1 reads by cutadapt, which is different than expected. If cutadapt worked perfectly, we would expect the exact same amount of adapters to be trimmed from R1 and R2 reads. This is because if the adapter is present on the R2 read, it should also be present on the R1 read. This means errors must have occurred when cutadapt was trimming the adapters, since they were not trimmed equally.

Figure 9 displays the trimmed read length distribution for 22_3H_both_S16_L008 and 23_4A_control_S17_L008. For both libraries, R2 reads were trimmed at much higher rates than R1 reads, resulting in more reads of a shorter length. This is to be expected as R2 reads generally tend to be of lower quality as the sample has been on the sequencer for a much longer time at that point through many cycles of washes.

Part 3

Mapped and Unmapped Reads for Libraries

The third and final part of this project aimed to create a database from mouse genome fasta files obtained from Ensembl release 1.10 and align our library reads to this database. This was done using the software STAR to generate SAM files of aligned reads.

Then, using my Python script parse_sam.py located in my QAA GitHub repository, mapped and nonmapped read counts were generated for both libraries using the SAM files outputted for STAR.

Sample	Mapped Reads	Unmapped Reads
22_3H_both_S16_L008	7677914	124340
23_4A_control_S17_L008	79472970	4640182

Table 2. Mapped and Unmapped Reads from Libraries 22_3H_both_S16_L008 and 23_4A_control_S17_L008. The table above displays mapped and unmapped reads generated from my Python script for both libraries.

Mapped and Unmapped Reads for htseq-count Files

Additionally, reads mapping to features for both libraries were counted utilizing htseq-count, once with the parameter stranded==yes and once with the parameter stranded==reverse.

Sample	Mapped Reads	Unmapped Reads	Percentage of Mapped Reads
reverse_22_3H_both_S16_L008.tsv	3394113	204965	87%
reverse_23_4A_control_S17_L008.tsv	32949127	4165856	78.3%
stranded_22_3H_both_S16_L008.tsv	145550	3521004	3.7%
stranded_23_4A_control_S17_L008.tsv	1549481	36148685	3.7%

Table 3. Htseq-count Output for Libraries 22_3H_both_S16_L008 and 23_4A_control_S17_L008. This table shows the amount of mapped reads, unmapped reads, and percent mapped for htseq-count stranded==yes and stranded==reverse.

Discussion

Based on the htseq-count results, I believe that the data is from "strand-specific" RNA-seq libraries, meaning that the identity of the coding DNA strand is preserved. Table 3 shows that stranded==reverse for 22_3H_both_S16_L008 and 23_4A_control_S17_L008 libraries had 87% and 78.3% mapped reads, respectively. In contrast, stranded==yes showed 3.7% mapped reads for both libraries. If it were not strand-specific, we would expect the htseq-count results for stranded==yes and stranded==reverse to be closer to 50% mapped reads for both. Instead, since we presumably used a kit not made by Illumina, many more reads mapped to features when running stranded==reverse since the antisense strand is preferentially sequenced.