QAA_report

Ben Carr

2024-09-10

PART 1: Read Score Quality Distributions

To perform an initial assessment of our RNA-seq data quality, I produced plots using FastQC displaying the per-base N distribution across our four reads: "7_2E_fox_S6_L008_R1", "7_2E_fox_S6_L008_R2", "Undetermined_S0_L008_R1", and "Undetermined_S0_L008_R2". As shown in Figure. 1, N content is very low across each read of each sample. There is a small bump at base position 1, but this is to be expected and not cause for concern.

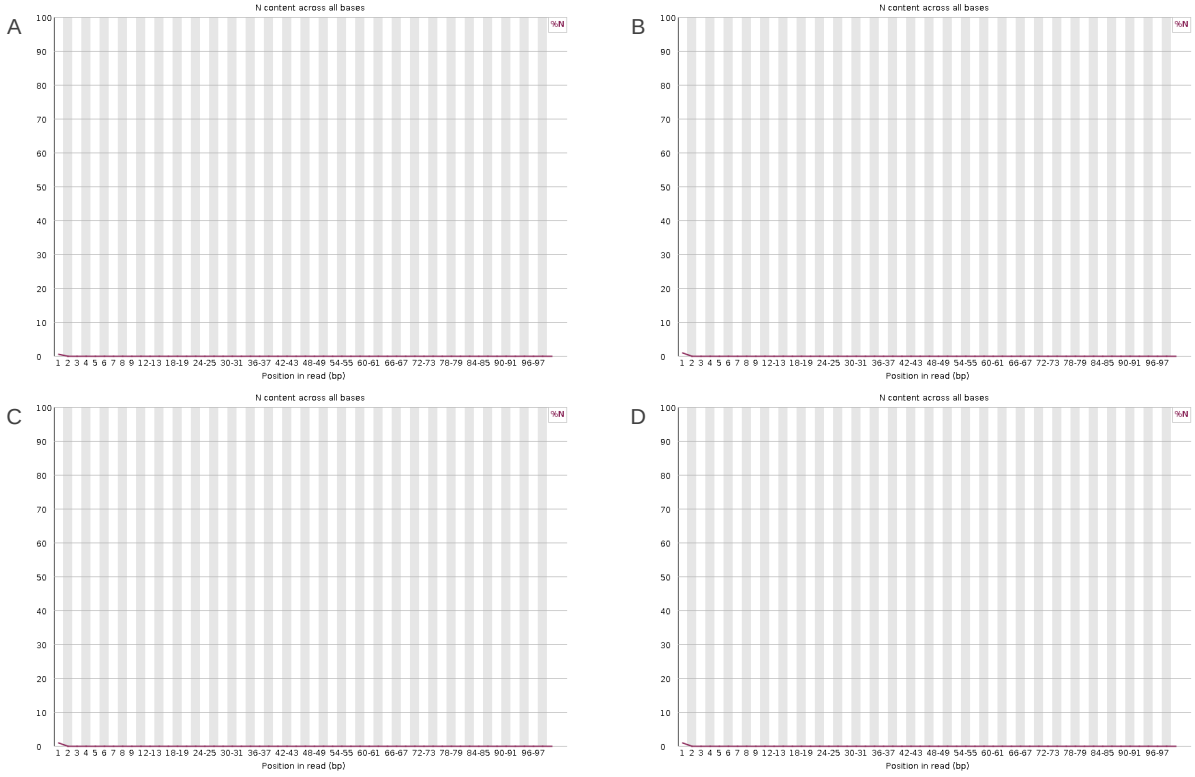

Figure 1: FastQC generated per-base N content distributions for (A) 7_2E_fox_S6_L008_R1, (B) 7_2E_fox_S6_L008_R2, (C) Undetermined_S0_L008_R1, (D) Undetermined_S0_L008_R2

Given these reassuring results, I decided to move to an assessment of per-base Phred Quality Score. To assess this metric, I used both FastQC and a custom python script. The results of these analyses are displayed side by side in figures X-Y. These quality score distributions are likely reassuring; there are no base positions with egregiously low quality scores, and the relatively low quality at the beginning of each read is an expected

result that will be removed by quality trimming. Given the high quality of sequencing data, I would feel comfortable proceeding with downstream analysis.

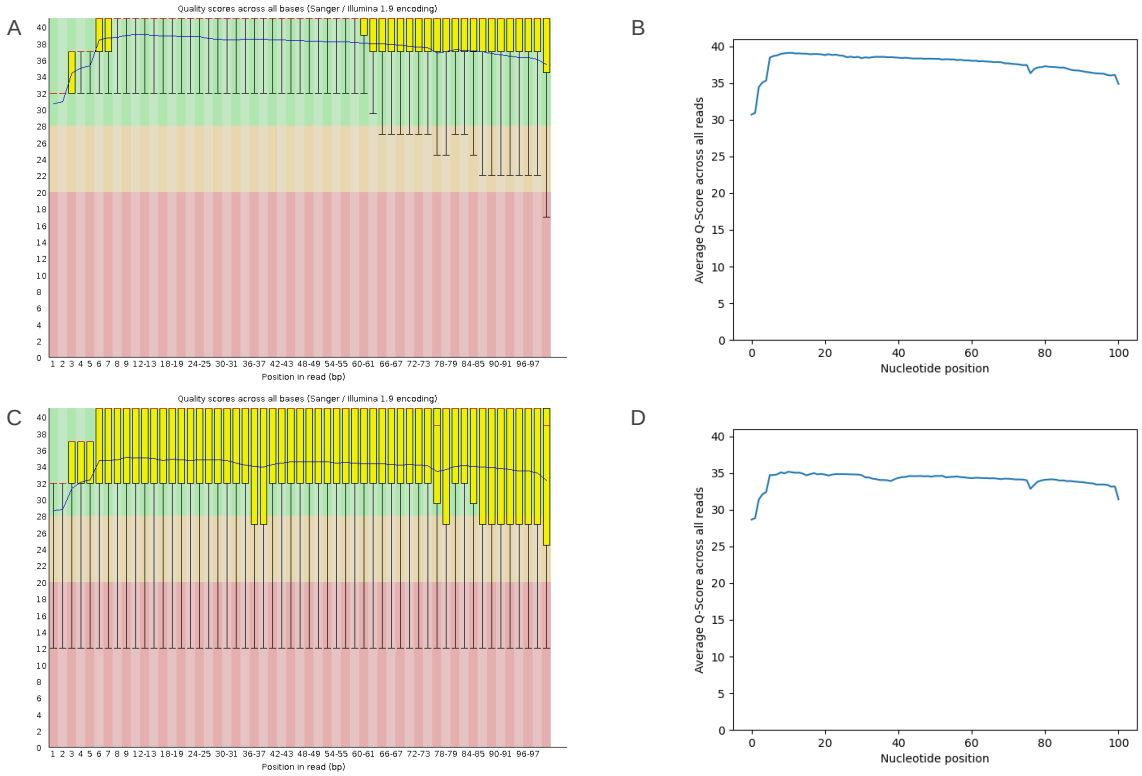

Figure 2: Per-base Phred Quality Score distributions for Undetermined_S0_L008 from (A) Read 1, FastQC, (B) Read 1, a custom python script, (C) Read 2, FastQC, and (D) Read 2, custom python script

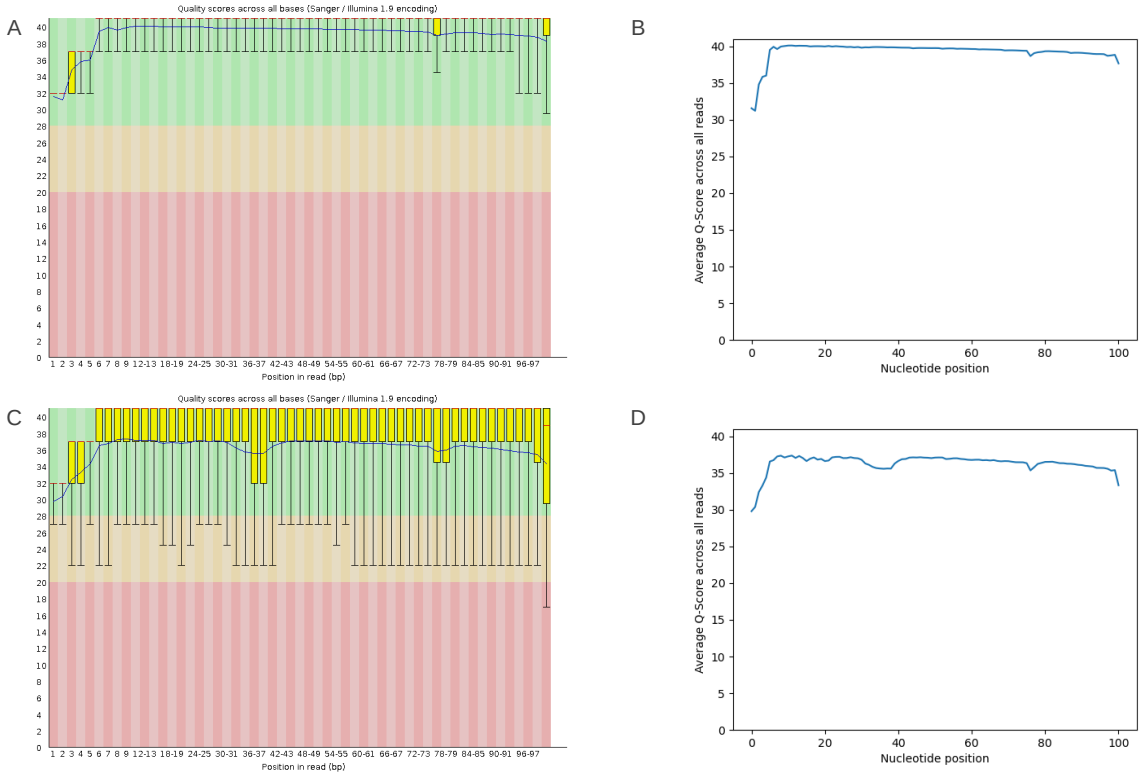

Figure 3: Per-base Phred Quality Score distributions for 7_2E_fox_S6_L008 from (A) Read 1, FastQC, (B) Read 1, a custom python script, (C) Read 2, FastQC, and (D) Read 2, custom python script

The outputs of the two programs – my python script and FastQC – have several differences between them, but still display the same general trends. FastQC contains information about the range of score distributions, rather than simply the mean, as my graph shows. These box-and-whisker plots contain valuable information, such as the very low bottom quartile in the last few base positions of 7_2E_fox_S6_L008_R2. Additionally, their tool performed much faster than mine, as summarized in the following tables:

Undetermined_R1

Metric	My Script	FastQC
Runtime	9:08.76	1:03.09
CPU	98%	97%

Undetermined_R2

Metric	My Script	FastQC
Runtime	9:03.85	1:02.22
CPU	99%	101%

Fox_R1

Metric	My Script	FastQC
Runtime	3:19.70	0:22.21
CPU	97%	101%

Fox_R2

Metric	My Script	FastQC
Runtime	3:18.34	0:23.21
CPU	99%	106%

Clearly, FastQC runs significantly faster than my python script. In terms of computational efficiency, it appears that they used similar (or if anything, slightly more) CPU capacity. It is hard to know why this is without knowing the details of their application, but it may have been written with a more efficient algorithm or in a faster programming language.

PART 2: Adaptor Trimming Distributions

Although the data are of high enough quality proceed with analysis, they still must be processed in order to trim low quality reads and accidentally sequenced adapters.To do so, the data were processed first with cutadapt to remove Adapter sequences, then Trimmomatic to remove low quality base calls. After doing this, I produced plots of sequence length distribution once adapter sequences and low quality base calls have been eliminated (Figure 4.) It is known that read 2 is generally lower quality – this is primarily due to cluster size decreasing during the second bridge amplification stage, before the reverse read is

sequenced (https://www.ecseq.com/support/ngs/why-has-reverse-read-a-worse-quality-than-forward-read-in-Illumina-sequencing). This can be seen in the below plots – R2 contains noticably more reads of shorter length compared to read 1.

Due to this same quality issue, we would expect that read 1 would be adapter trimmed more than read 2, because there may be more errors when sequencing the adapters of read 2. However, this does not imply that read 1 should have a shorter distribution of lengths, because it will not be quality trimmed as extensively.

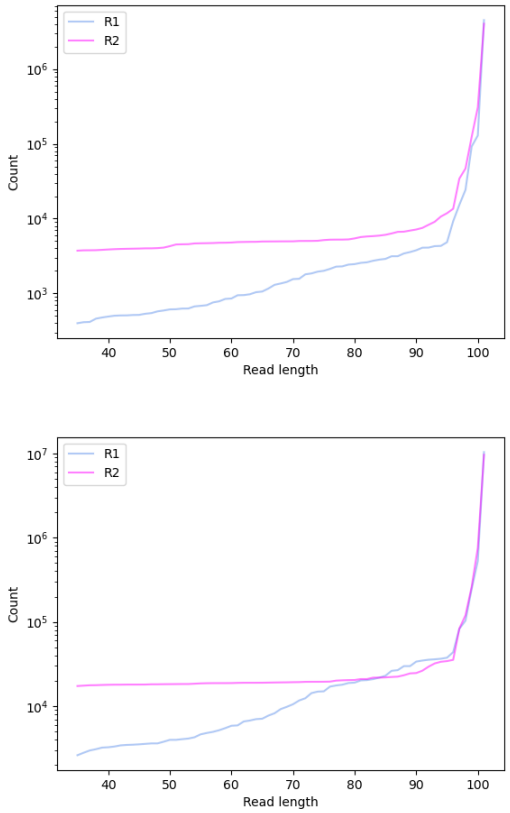

Figure 4: Read length distributions after adapter cutting and quality trimming, for 7_2E_fox_S6_L008 (top) and Undetermined_S0_L008 (bottom), log scale

PART 3: Alignment and Strand Specificity

Finally, our data were aligned to a genome and analyzed for strand-specificity. I downloaded a *Mus musculus* genome from Ensembl release 112, and used STAR to generate a reference database. The use of this particular aligner was necessary because it is splice aware, meaning that it is capable of aligning RNA reads that stretch across different exons in the reference genome. GTF files are required to take advantage of this functionality, so these were also downloaded from Ensembl.

After these files were downloaded, I created a STAR database then aligned the outputs of Trimmomatic to the reference genome. This produced one SAM file for each of our two biological samples.

After creating SAM files, the typical question is how many FASTQ reads aligned to the reference genome. To assess this, I again used two methods – a custom python script, and a publicly available tool called Htseq-count. To determine if these reads were strand-specific, Htseq was run twice with two different

parameters: –stranded=yes and –stranded=reverse. These parameters tell Htseq to treat the data as either forward stranded or reverse stranded, creating a separate output for forward and reverse reads. According to information found online (https://www.biostars.org/p/205987/), the output with the higher proportion of mapped reads is the correct setting. In our case, this is likely reverse: as seen in the tables below, in the "Fox" file, 82.46% of reads mapped under the reverse setting, whereas only 3.51% mapped in the forward. For "Undetermined", this was a little more unclear – only .68% mapped in reverse, and .054% forward. Although this is technically a large change proportionally, I don't think this should be interpreted as a significant difference. Instead, I believe this is again indicative of the fact that these data come from many different sources, and can't be reliably mapped to a genome. The output from Htseq is summarized in the table below:

Htseq Metric	Fox Reverse	Fox Forward	Undetermined Reverse	Undetermined Forward
Mapped reads	4,026,702	171,207	17,097	1,350
Total reads	4,882,703	4,882,703	2,511,252	2,511,252
% mapped	82.46%	3.51%	0.68%	0.054%

I also created a similar table for my python script data:

Script Metric	Fox	Undetermined
Mapped reads	9,424,733	163,734
Total reads	9,765,406	5,022,504
% mapped	96.51%	3.26%

There are a number of interesting things to note in these tables. First, I can confirm that the sum of the total reads for each forward and reverse file in Htseq matches the total reads of each file as analyzed by my script, a reassuring finding. Second, we can note the clear discrepancy in matched reads between the "Fox" file and the "Undetermined" file, the reason for which should be obvious from the name of the latter. The "Undetermined" data are a collection of from reads whose biological source that could not be identified, so it makes perfect sense that they do not map well to a single genome. Finally, there is a clear difference in percent of mapped reads between the Htseq output and my python script's output. This is likey due to the fact that the Htseq algorithm also takes in a GTF file as input, whereas mine only uses the information in the SAM file.