

QAA

Peter Pham

2022-09-07

Part 1

This is all the graphs that are output from FASTQC and Python scripts side by side.

Table 1: both_S9 FASTQC v.s Python Script

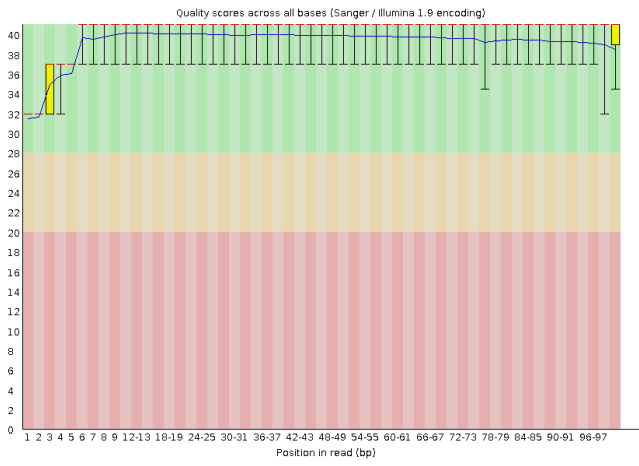
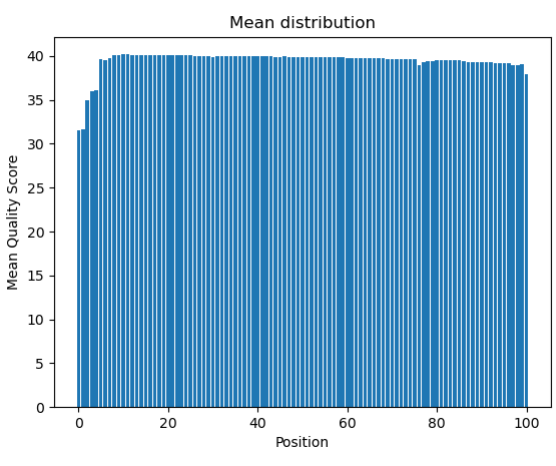
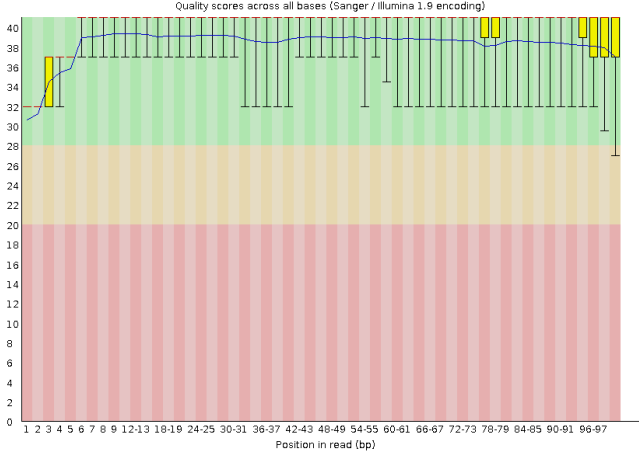
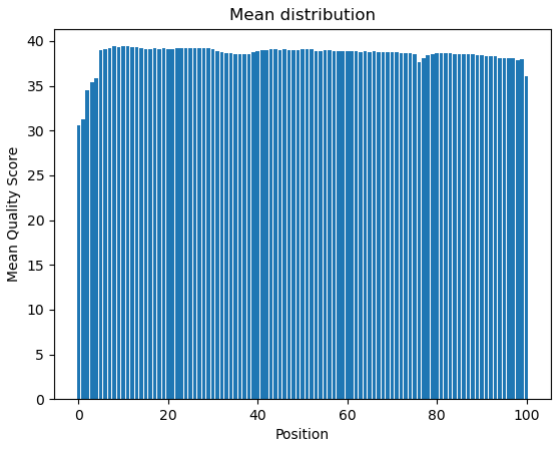
FASTQC both_S9 (R1,R2)	Python both_S9 (R1,R2)
	
	

Table 2: fox_S21 FASTQC v.s Python Script

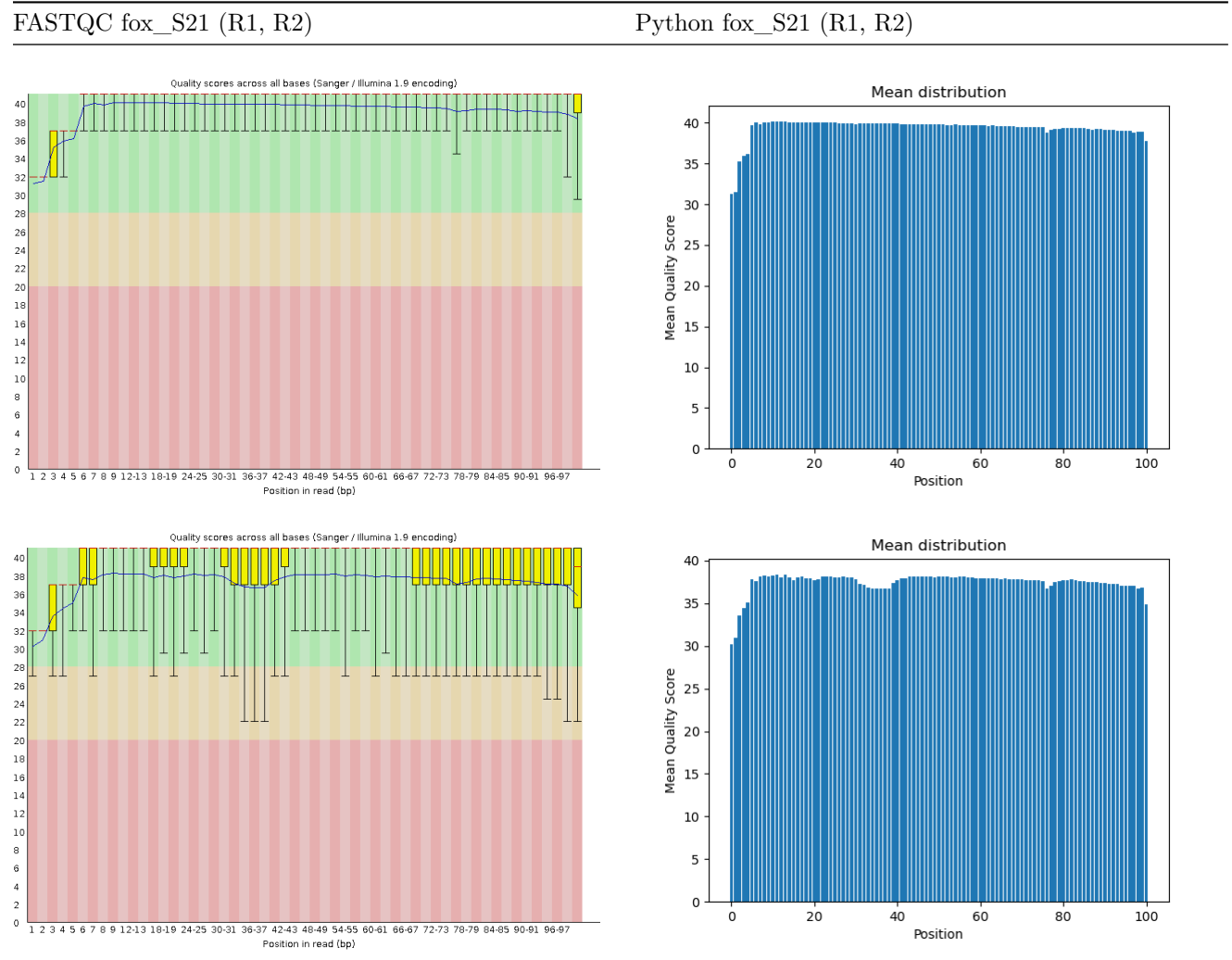
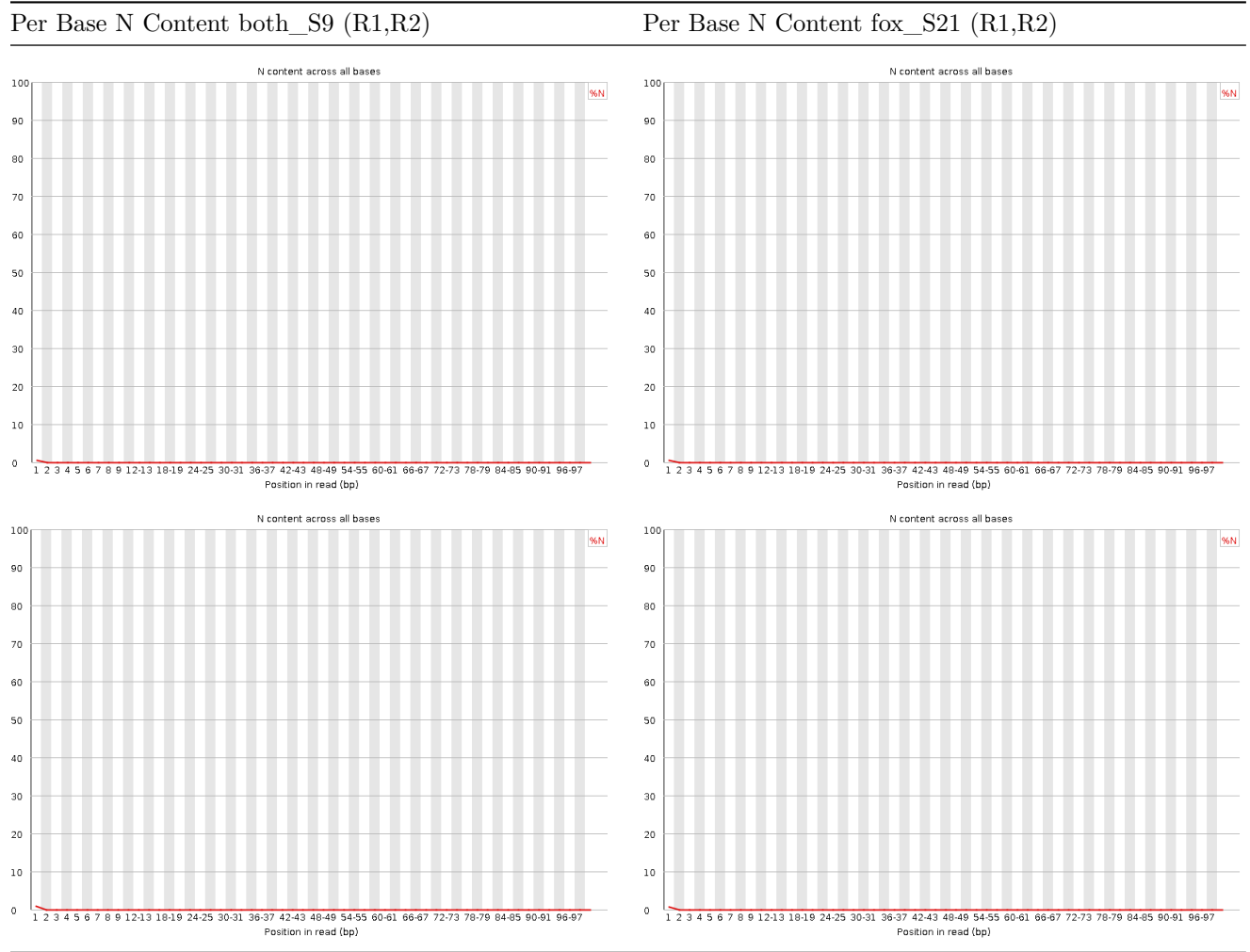


Table 3: N content per base



N-Plots consistent with quality score There is no correlation between the N bases present and the quality score. If we take a look at fox_S21_R2 read and compare it with its respective n distribution, it stays low while there are bad reads within the graph.

Overall Data Quality and Comparison The fastQC score distribution is very similar to the one that I produced. The files differ in how the positions are binned. The python graphs made from matplotlib did not have binned x axis which showed a larger spread of data. The fastQC distrubtion graph did have bins which slightly adjusted the statics. The fastQC data also shows a box and whisker plot that shows us the range of each position, mean, median and quartiles. The python output file only displays the mean. Outside of that, the general pattern both the fastQC and python graphs follow is similar. FastQC is much quicker than the python script. This is probably due to the level of coding as they have optimized fastQC to work well while i have made my script after only a couple months of experience.

Looking at the data, Read 1 has a higher average position score than that of Read 2 for both paired end samples. This makes sense because the second read takes place after it had been sitting on the sequencer for a longer period compared to Read 1. So the average score dropped. The second read of fox_S21 had some really bad reads compared to fox_s21 first read.

Part 2

Table 4: Proportion of reads that were trimmed

Sample	Percent of adapters that were trimmed
both_S9_R1	4.9%
both_S9_R2	5.7%
fox_S21_L008_R1	7.5%
fox_S21_L008_R2	8.3%

Adapters R1 = AGATCGGAAGAGCACACGTCTGAACTCCAGTCA

R2 = AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT

Confirming adapters For this I used the sanity check to get the adapters. To determine that they were there i used grep.

```
zcat 11_2H_both_S9_L008_R1_001.fastq.gz | grep "^AGATCGGAAGAGCACACGTCTGAACTCCAGTCA" | wc
```

which got me some hits

```
zcat 11_2H_both_S9_L008_R1_001.fastq.gz | grep "^AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc
```

which got no hits

```
zcat 11_2H_both_S9_L008_R2_001.fastq.gz | grep "^AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc
```

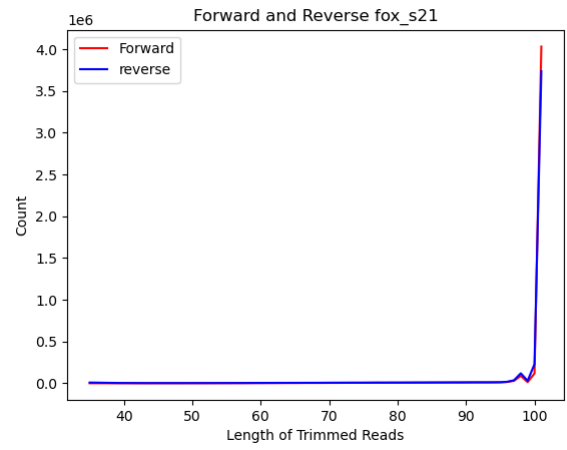
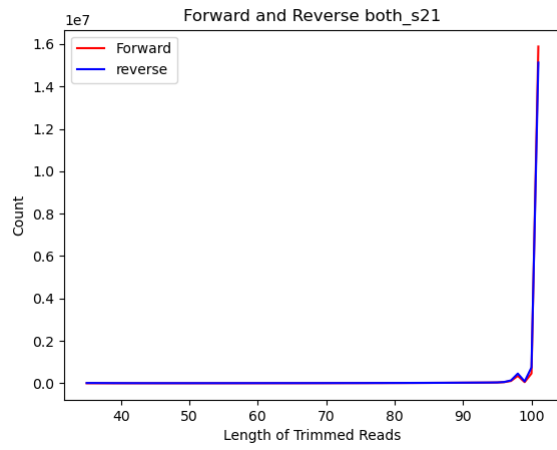
which got some hits

Because the forwards reads got hit on the adapter sequence for R1 and not for R2 we can say that it is there. When we take a look at R2 with relationship to the R2 adapters we also get hits.

Table 5: Comparing the number of trimmed reads R1 v.s R2

Trimmed reads both_S9 (R1,R2)

Trimmed reads fox_S21 (R1,R2)



Part 3

Table 6: Mapped and Unmapped reads using STAR

Sample	Mapped	Unmapped
both_S9	33,637,730	1,293,496
fox_S21	8,883,012	260,796

Table 7: HTSeq matched counts

Sample	Stranded	Reverse
both_S9	602,072	13,819,257
fox_S21	189,376	3,860,217

Strand or Not Strand Specific I propose that these files are strand specific. This is because when we are looking at the htseq counts the values are matching. When both stranded and stranded reverse were run we would get the same number. Because our values are different in matches we have a stranded library prep.