QAA_report

2025-09-10

My SRR assignments are `SRR25630409` and `SRR25630385`.

Part 1

Per base quality scores and N content

Plots of the per-base quality score distributions and N content for R1 and R2 reads. We can see high quality scores down the board, along with very low N content. There is some slight, expected dip in quality scores later on in the sequences, but nothing that would correspond with higher N content.

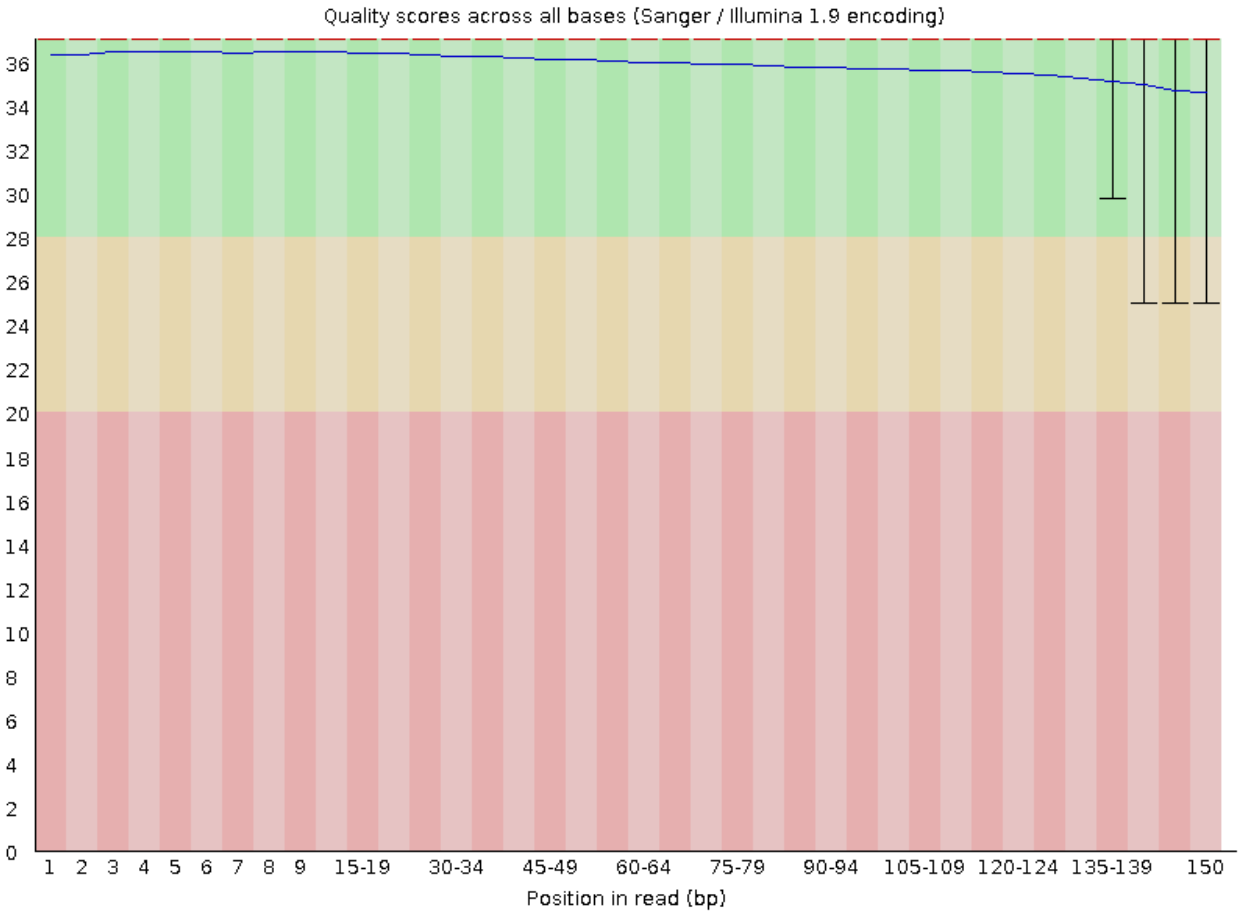

Figure 1: Per base sequence quality for Read 1 of SRR25630385

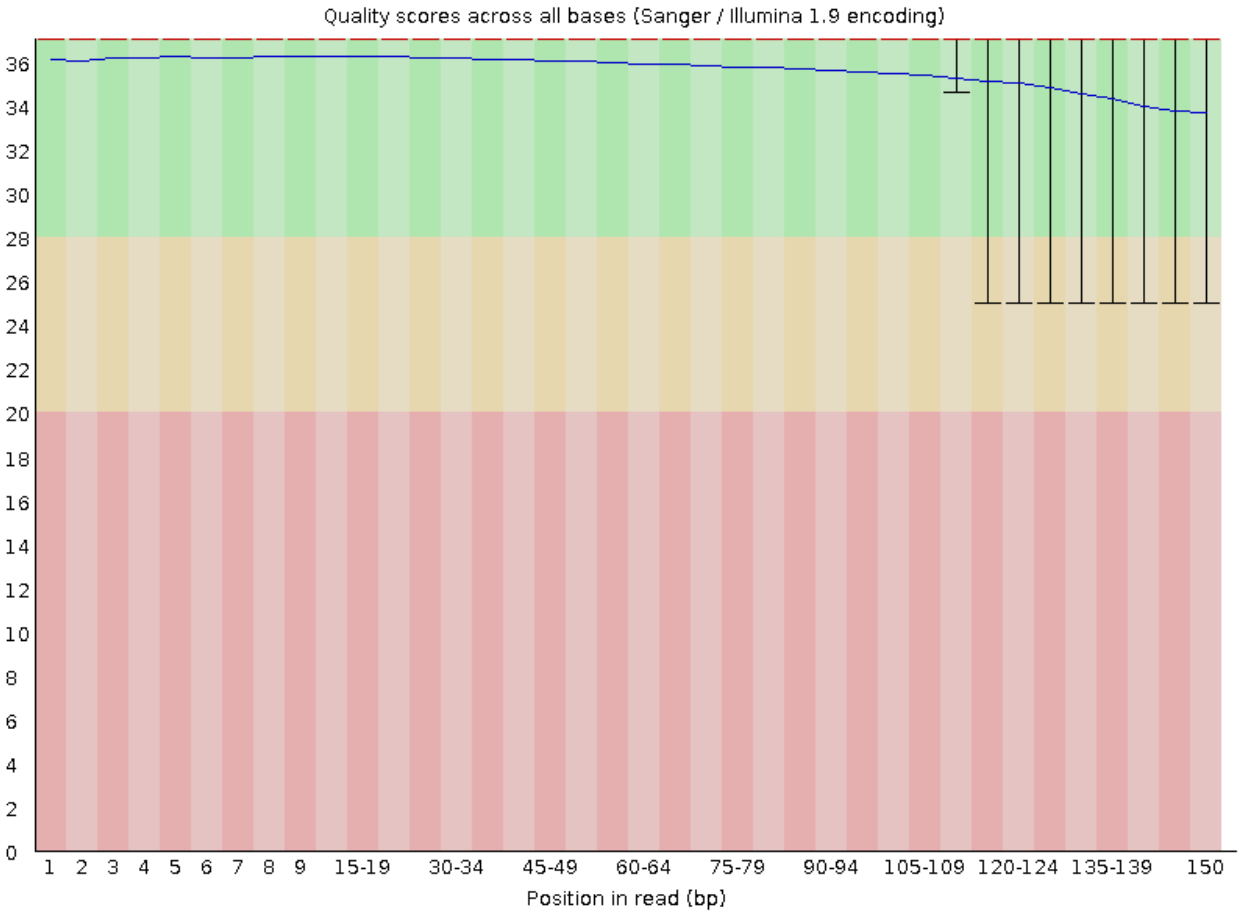

Figure 2: Per base sequence quality for Read 2 of SRR25630385

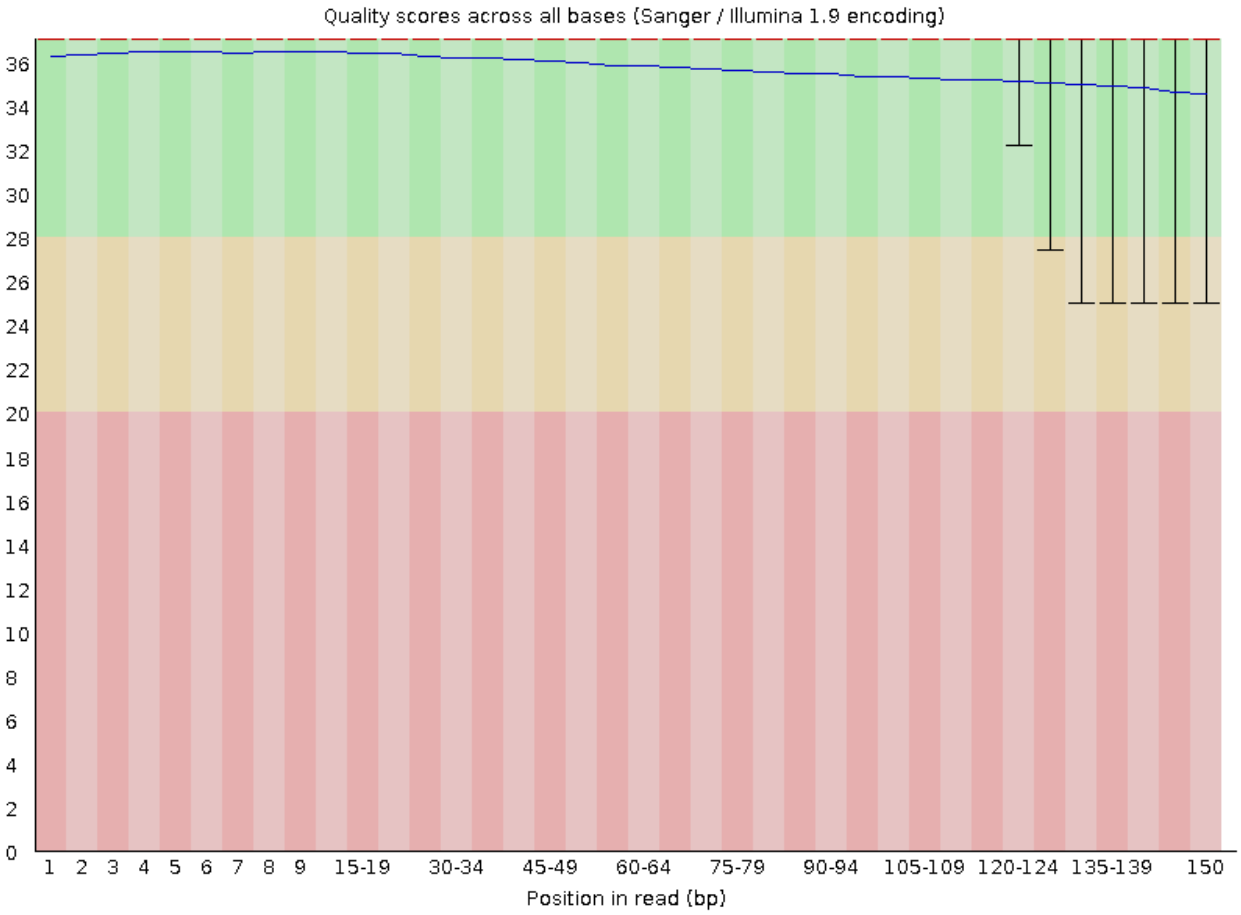

Figure 3: Per base sequence quality for Read 1 of SRR25630409

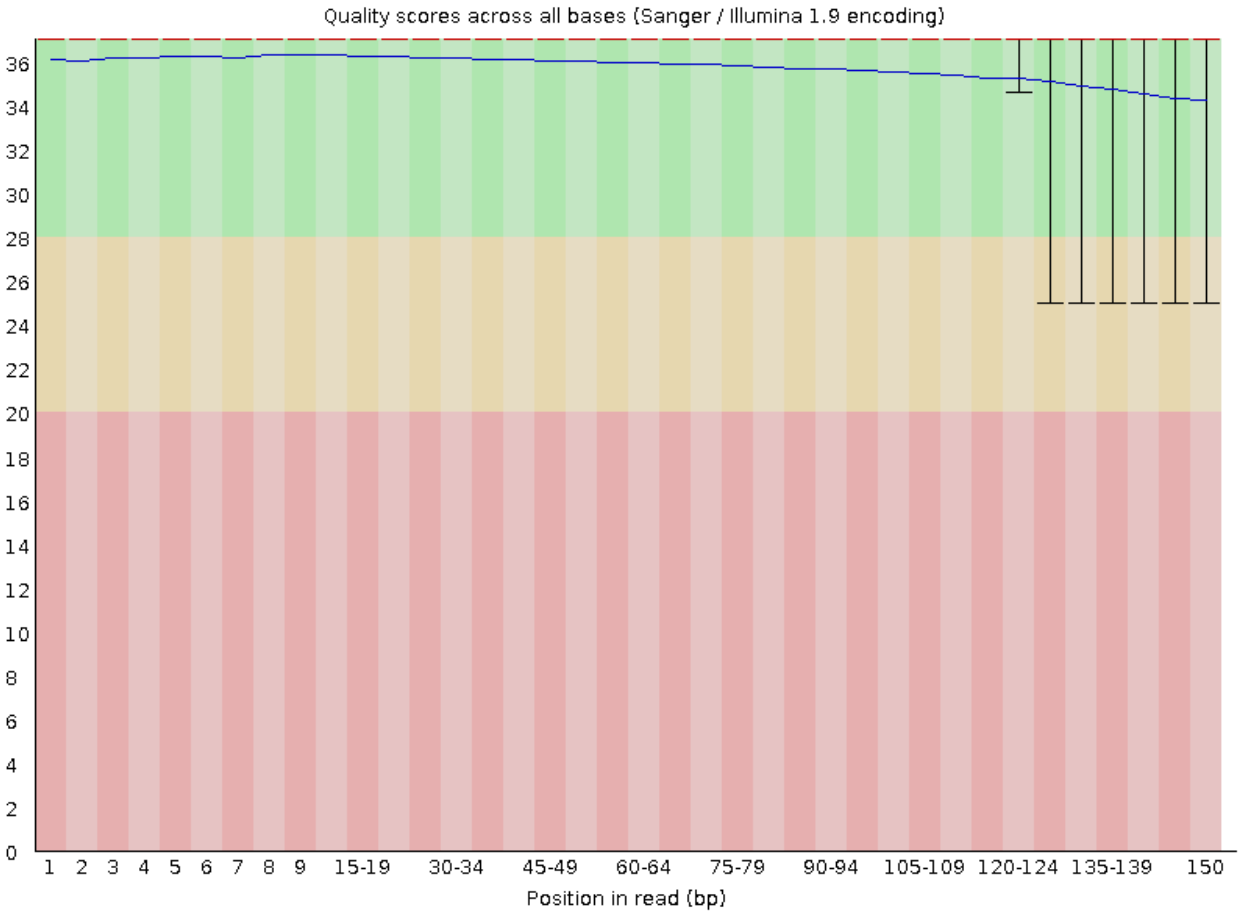

Figure 4: Per base sequence quality for Read 2 of SRR25630409

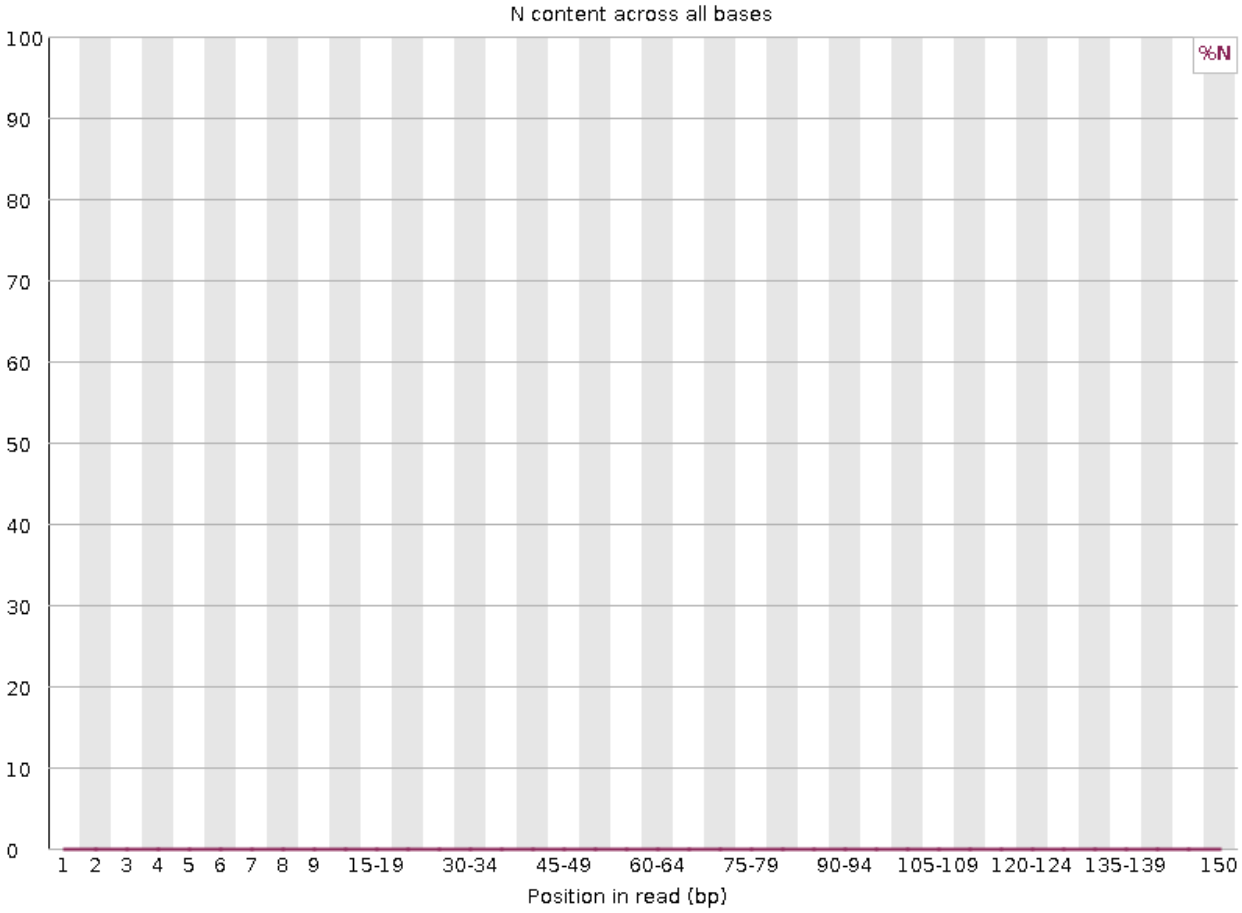

Figure 5: Per base N content for Read 1 of SRR25630385

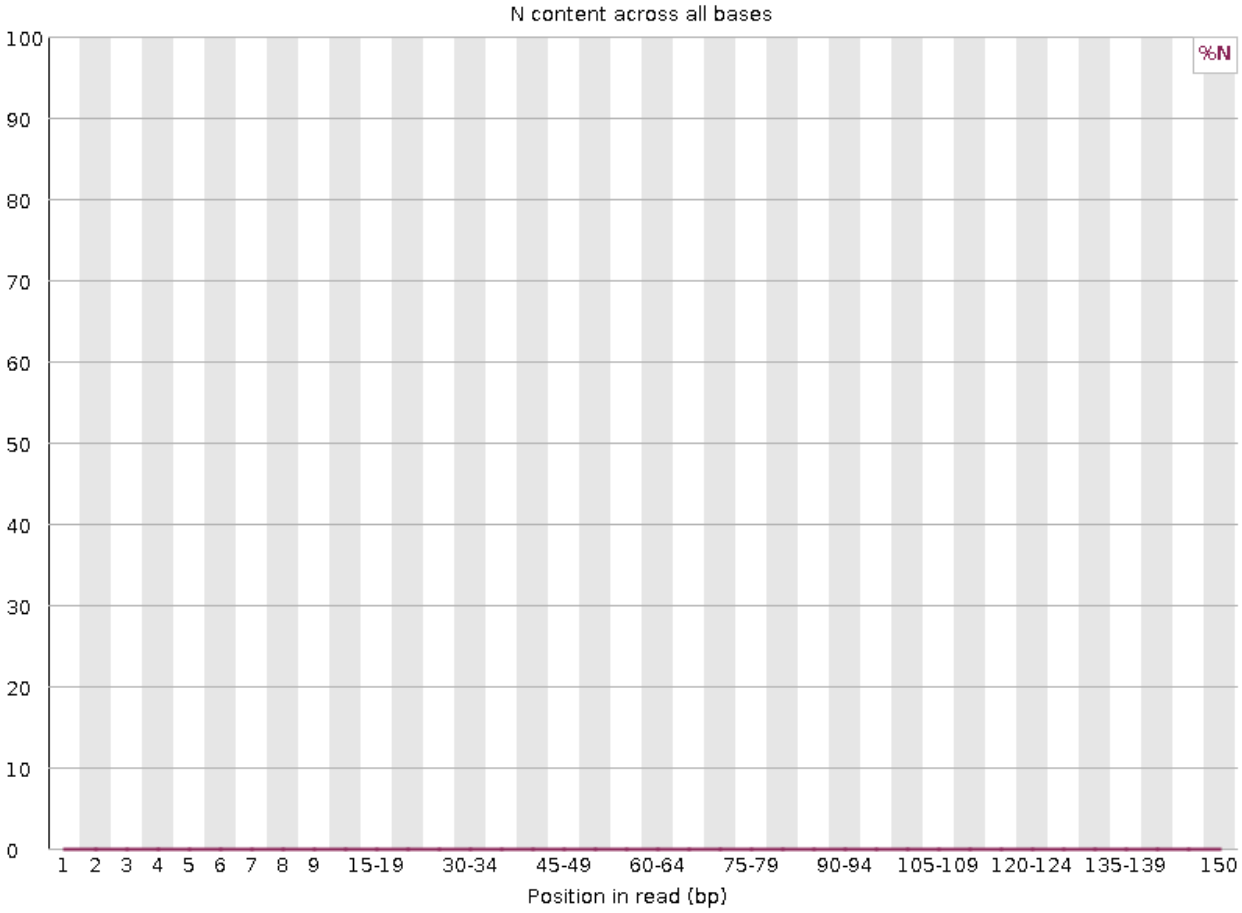

Figure 6: Per base N content for Read 2 of SRR25630385

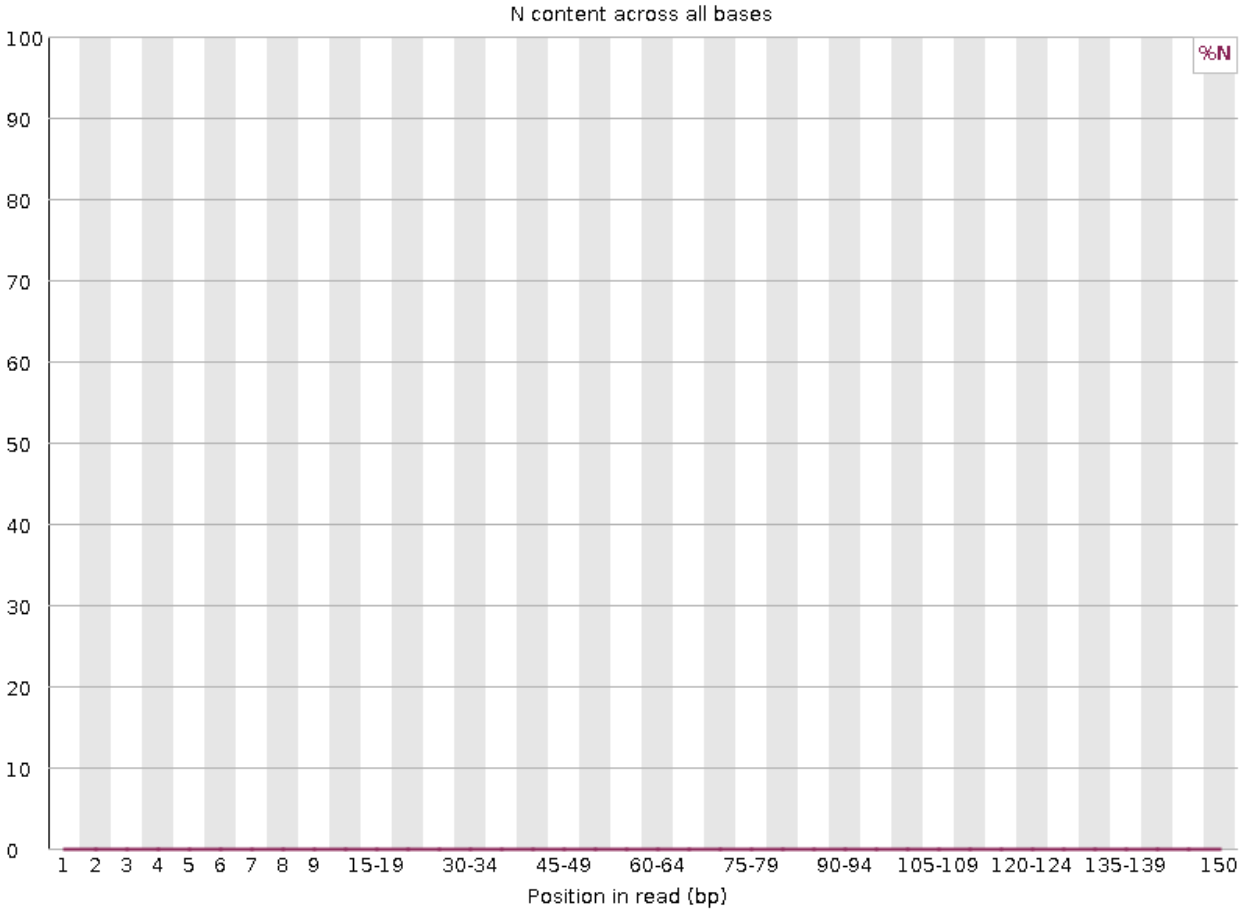

Figure 7: Per base N content for Read 1 of SRR25630409

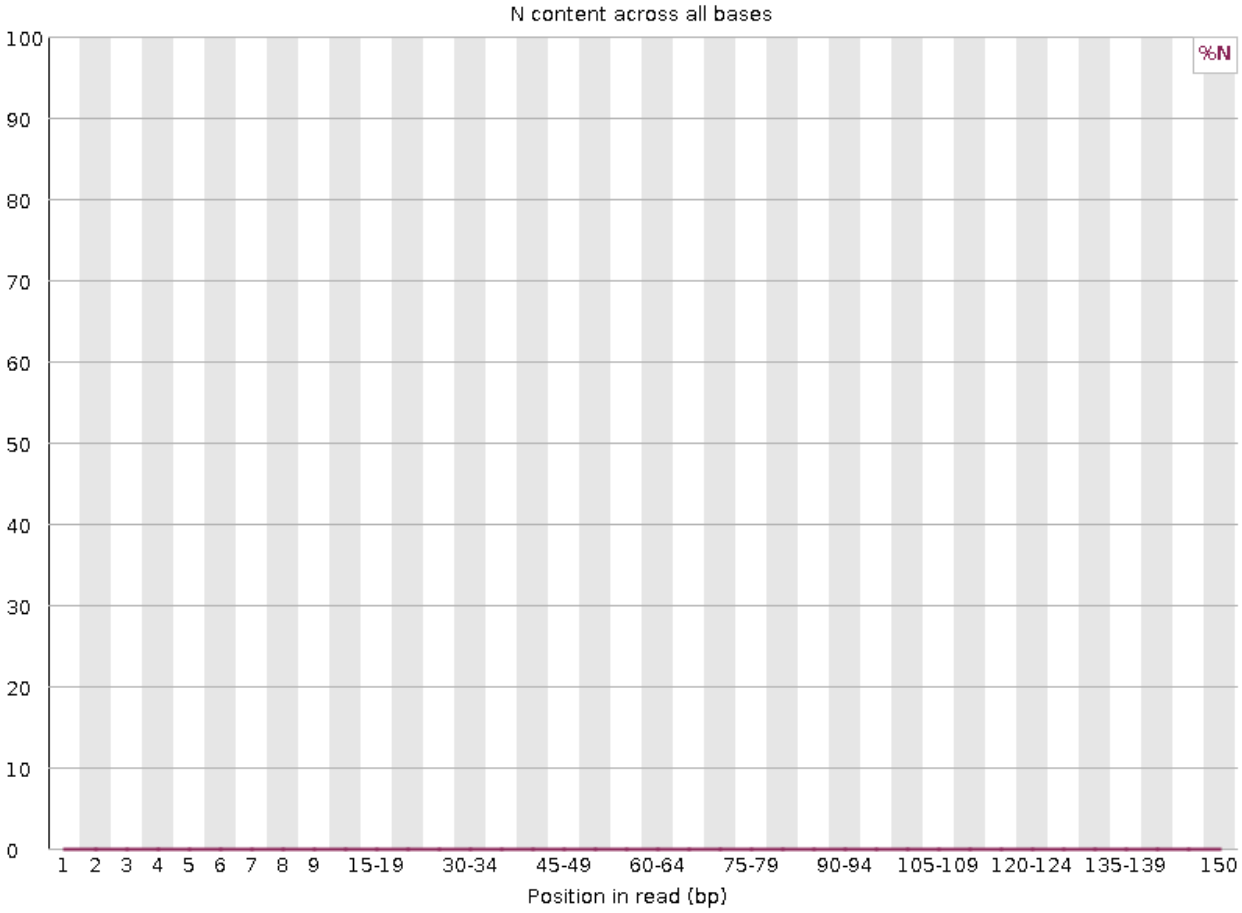

Figure 8: Per base N content for Read 2 of SRR25630409

Plotting in Python:

Per base sequence quality was plotted in python as well. Results are, as expected, similar.

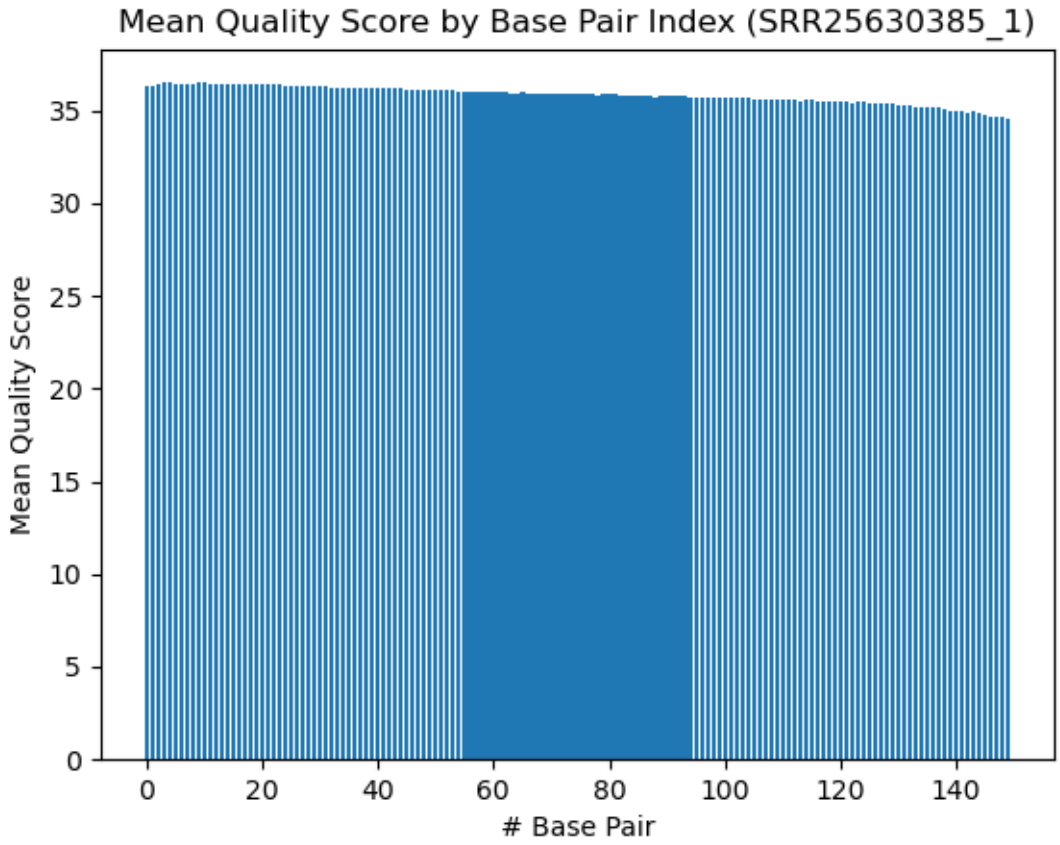

Figure 9: Per base sequence quality for Read 1 of SRR25630385 plotted in python

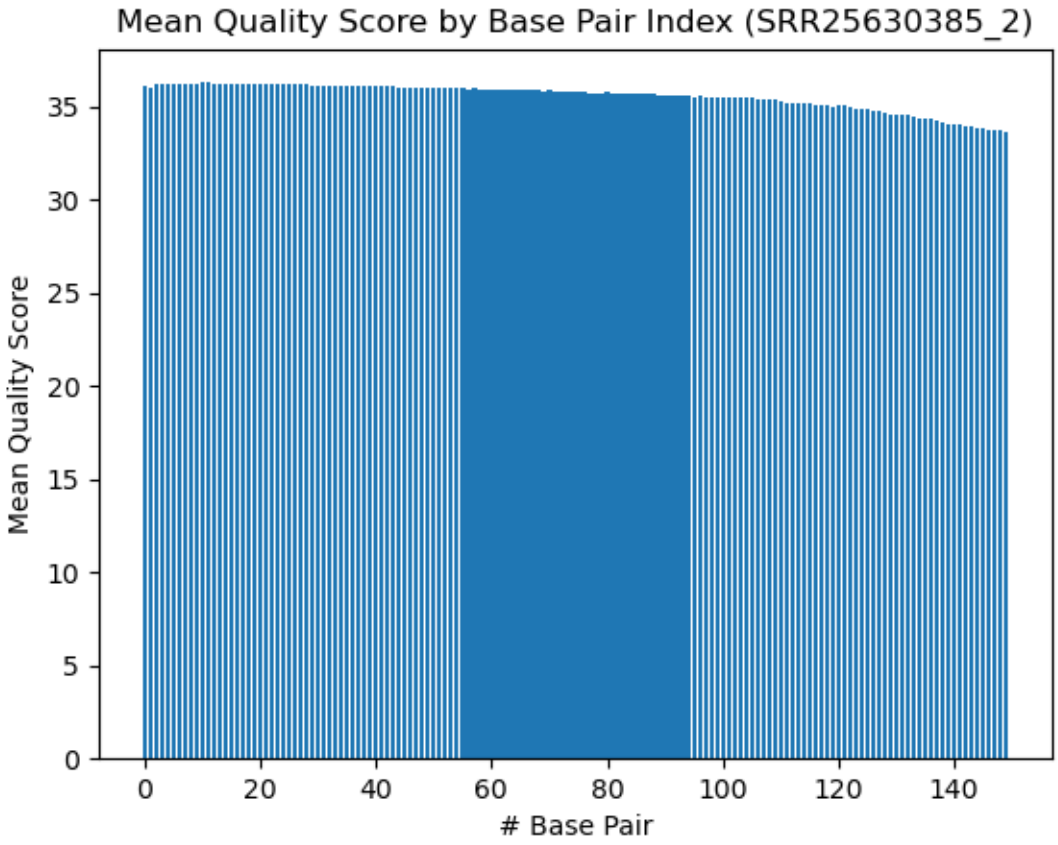

Figure 10: Per base sequence quality for Read 2 of SRR25630385 plotted in python

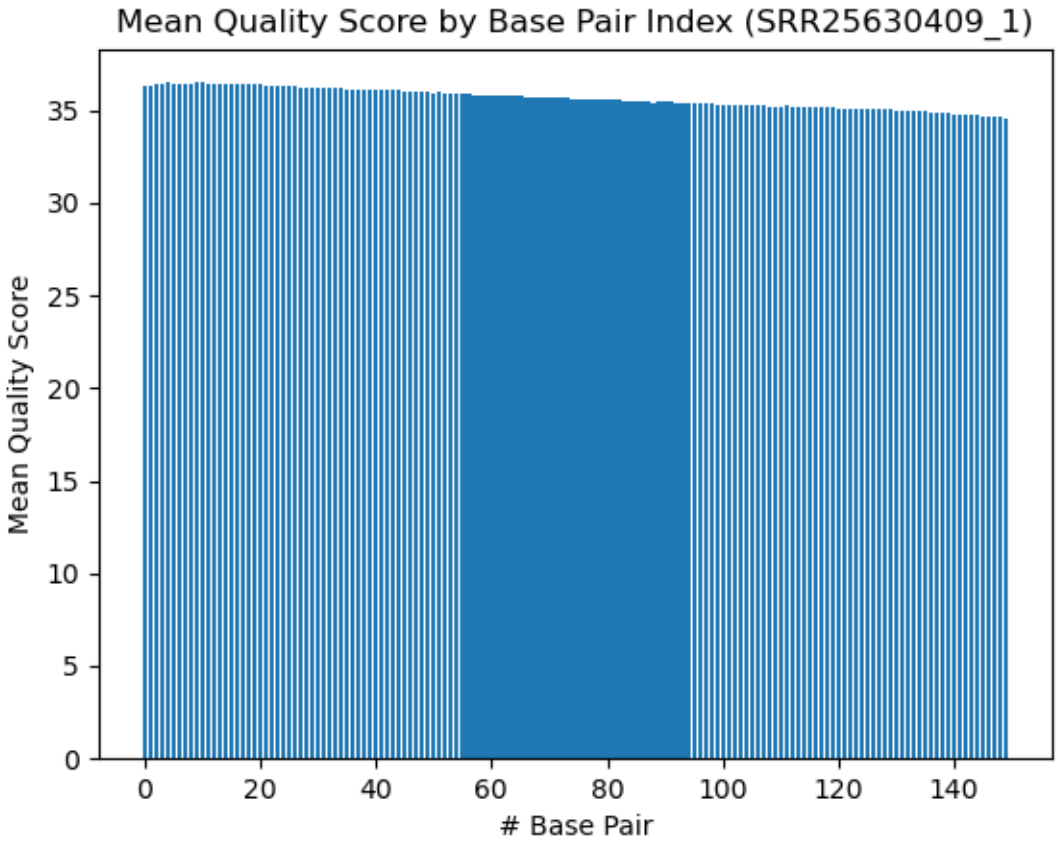

Figure 11: Per base sequence quality for Read 1 of SRR25630409 plotted in python

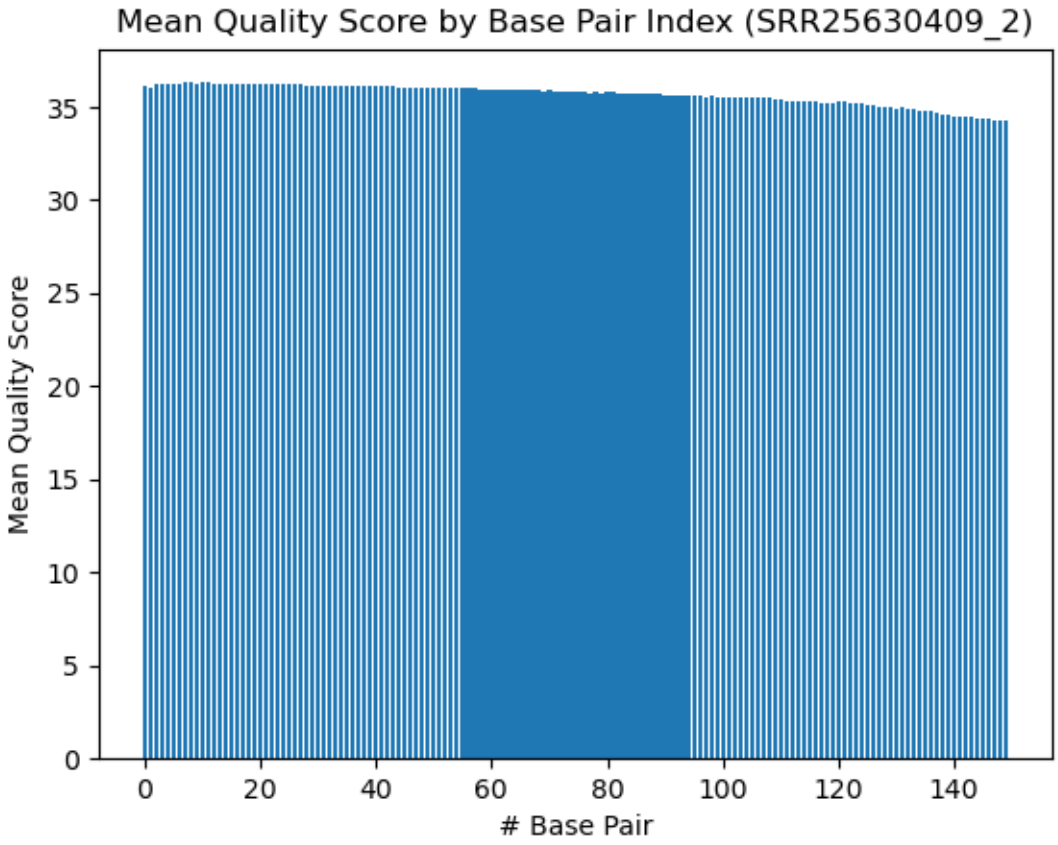

Figure 12: Per base sequence quality for Read 2 of SRR25630409 plotted in python

Resource usage comparison:

For the same two fastq files (from `SRR25630385`, no longer run in parallel) with `plot_qscores.py` sbatched with `plot_qscores.sh`:

Fast QC resource usage: For `SRR25630385` – comparable size to `SRR25630409`:

```
Percent of CPU this job got: 196%
Elapsed (wall clock) time (h:mm:ss or m:ss): 3:03.77
Maximum resident set size (kbytes): 5784548
```

Python plotting resource usage: Also for `SRR25630385`.

```
Percent of CPU this job got: 99%
Elapsed (wall clock) time (h:mm:ss or m:ss): 19:29.89
Maximum resident set size (kbytes): 71692
```

Unsurprisingly, this took a lot less memory and a lot more time than the fastqc analysis. It is notably not parallelized, and is being run on the compressed data while fastqc was run on uncompressed fastq files. Python is slower at reading files, especially so when reading compressed files, so this is expected. I assume fastqc might be uncompressing the file and loading it to memory, and then accessing it more quickly through that methodology, accounting for the increased memory usage. It may also be that there is just a lot more analysis that it is doing, which would also take more memory.

Further data quality analysis

Here is the summary table output from fastQC:

For `SRR25630385_1.fastq`:

```
PASS    Basic Statistics    SRR25630385_1.fastq
PASS    Per base sequence quality   SRR25630385_1.fastq
WARN    Per tile sequence quality   SRR25630385_1.fastq
PASS    Per sequence quality scores SRR25630385_1.fastq
FAIL    Per base sequence content   SRR25630385_1.fastq
PASS    Per sequence GC content SRR25630385_1.fastq
PASS    Per base N content  SRR25630385_1.fastq
PASS    Sequence Length Distribution    SRR25630385_1.fastq
FAIL    Sequence Duplication Levels SRR25630385_1.fastq
WARN    Overrepresented sequences   SRR25630385_1.fastq
FAIL    Adapter Content SRR25630385_1.fastq
```

For `SRR25630385_2.fastq`:

```
PASS    Basic Statistics    SRR25630385_2.fastq
PASS    Per base sequence quality   SRR25630385_2.fastq
PASS    Per tile sequence quality   SRR25630385_2.fastq
PASS    Per sequence quality scores SRR25630385_2.fastq
FAIL    Per base sequence content   SRR25630385_2.fastq
PASS    Per sequence GC content SRR25630385_2.fastq
PASS    Per base N content  SRR25630385_2.fastq
PASS    Sequence Length Distribution    SRR25630385_2.fastq
FAIL    Sequence Duplication Levels SRR25630385_2.fastq
WARN    Overrepresented sequences   SRR25630385_2.fastq
FAIL    Adapter Content SRR25630385_2.fastq
```

For `SRR25630409_1.fastq.gz`

```
PASS    Basic Statistics    SRR25630409_1.fastq.gz
PASS    Per base sequence quality   SRR25630409_1.fastq.gz
WARN    Per tile sequence quality   SRR25630409_1.fastq.gz
PASS    Per sequence quality scores SRR25630409_1.fastq.gz
FAIL    Per base sequence content   SRR25630409_1.fastq.gz
WARN    Per sequence GC content SRR25630409_1.fastq.gz
PASS    Per base N content  SRR25630409_1.fastq.gz
PASS    Sequence Length Distribution    SRR25630409_1.fastq.gz
FAIL    Sequence Duplication Levels SRR25630409_1.fastq.gz
FAIL    Overrepresented sequences   SRR25630409_1.fastq.gz
WARN    Adapter Content SRR25630409_1.fastq.gz
```

For `SRR25630385_2.fastq`

```
PASS    Basic Statistics    SRR25630385_2.fastq
PASS    Per base sequence quality   SRR25630385_2.fastq
PASS    Per tile sequence quality   SRR25630385_2.fastq
PASS    Per sequence quality scores SRR25630385_2.fastq
FAIL    Per base sequence content   SRR25630385_2.fastq
```

```
PASS    Per sequence GC content SRR25630385_2.fastq
PASS    Per base N content  SRR25630385_2.fastq
PASS    Sequence Length Distribution    SRR25630385_2.fastq
FAIL    Sequence Duplication Levels SRR25630385_2.fastq
WARN    Overrepresented sequences   SRR25630385_2.fastq
FAIL    Adapter Content SRR25630385_2.fastq
```

Let's look at some revelevant plots:

Per base sequence content

These should be flat and at the same level in random data, and they are not, notably so at the start of the reads. This is due to biases in sequencing, and will be improved by trimming.

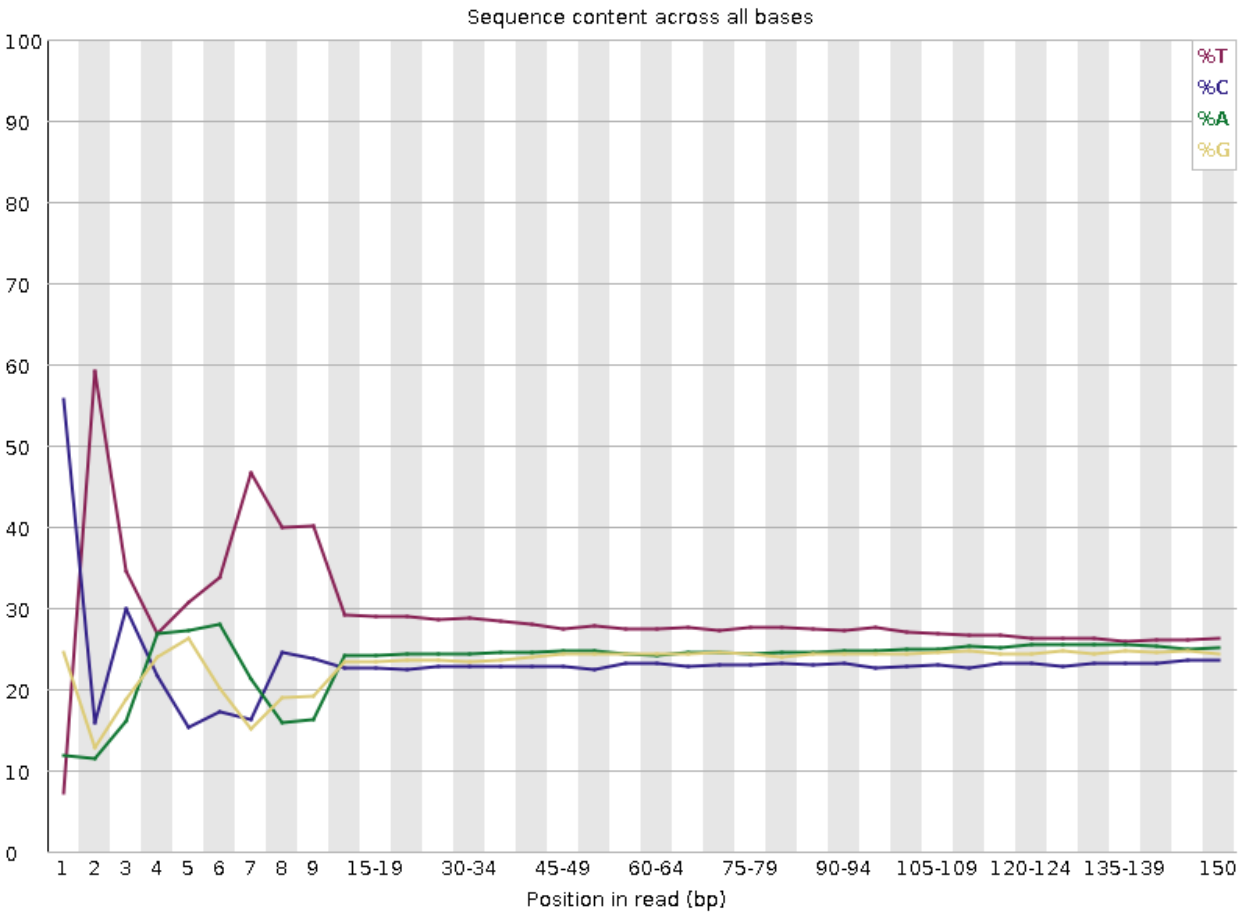

Figure 13: Per base sequence content for Read 1 of SRR25630385

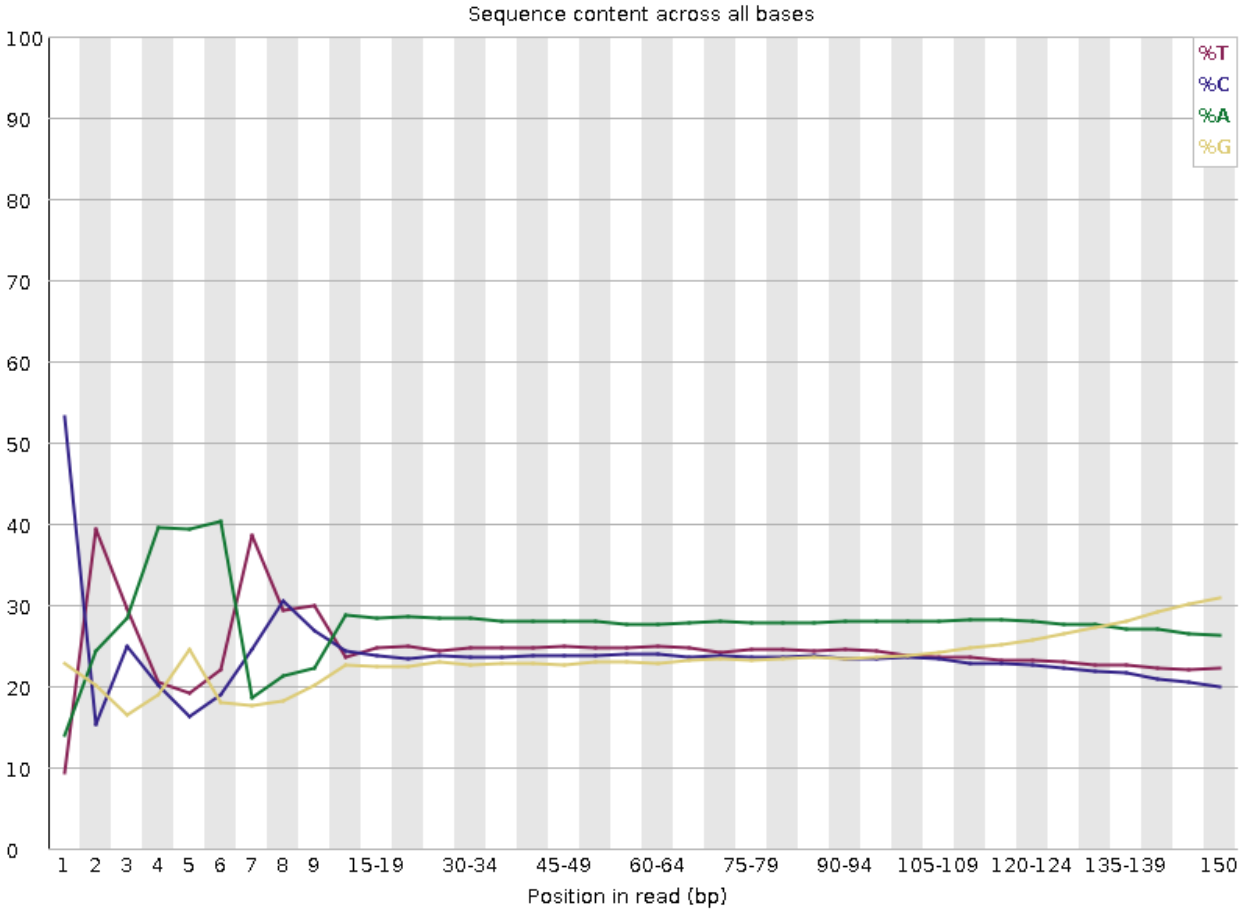

Figure 14: Per base sequence content for Read 2 of SRR25630385

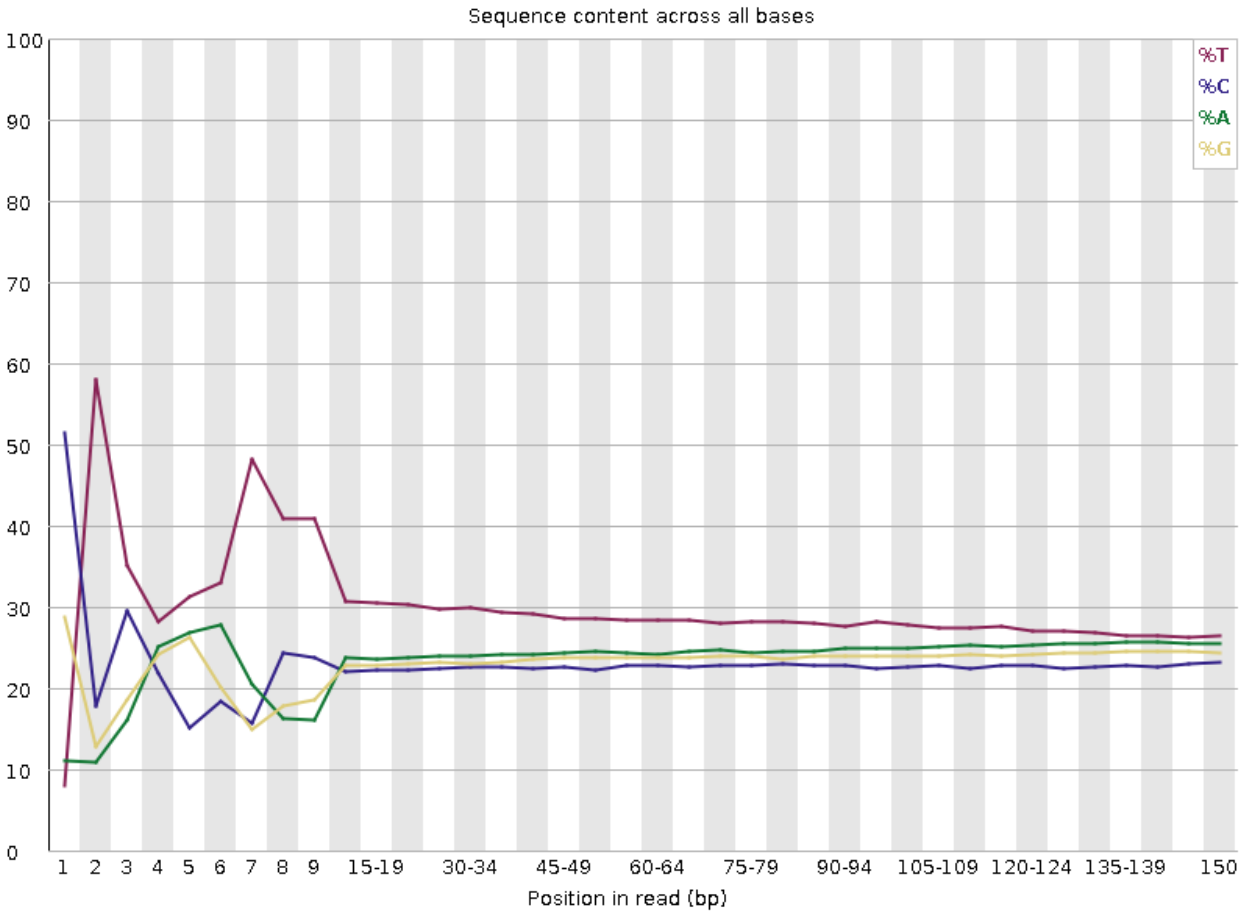

Figure 15: Per base sequence content for Read 1 of SRR25630409

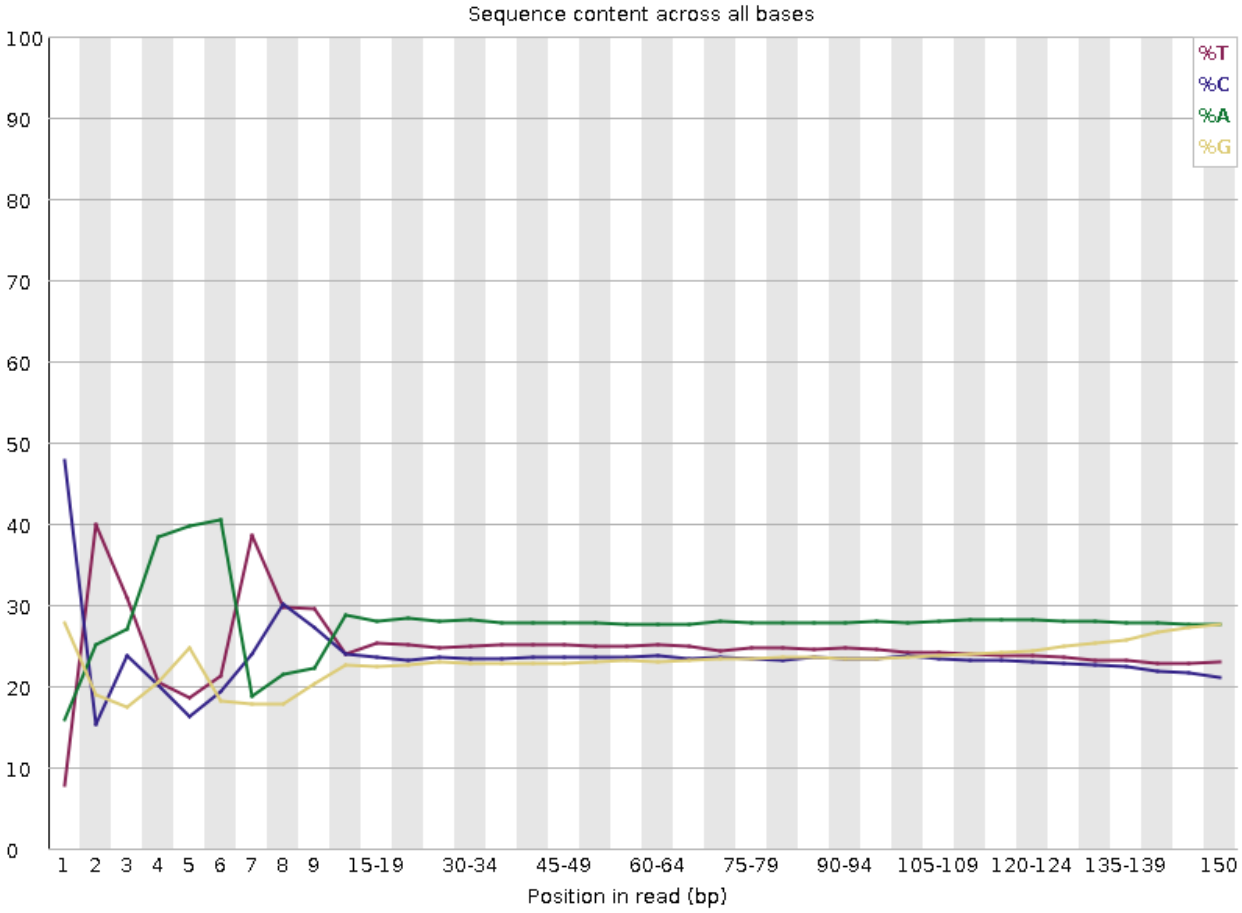

Figure 16: Per base sequence content for Read 2 of SRR25630409

Duplication levels

By looking at duplication levels of our reads, we can see if there are potential issues with PCR over-amplification. It looks like there is some of that going on, with a decent number of sequences being duplicated over 10,000 times (in the first 100,000 reads, given that is all this plots). Hopefully accounting for this later with our picard duplication is effective.

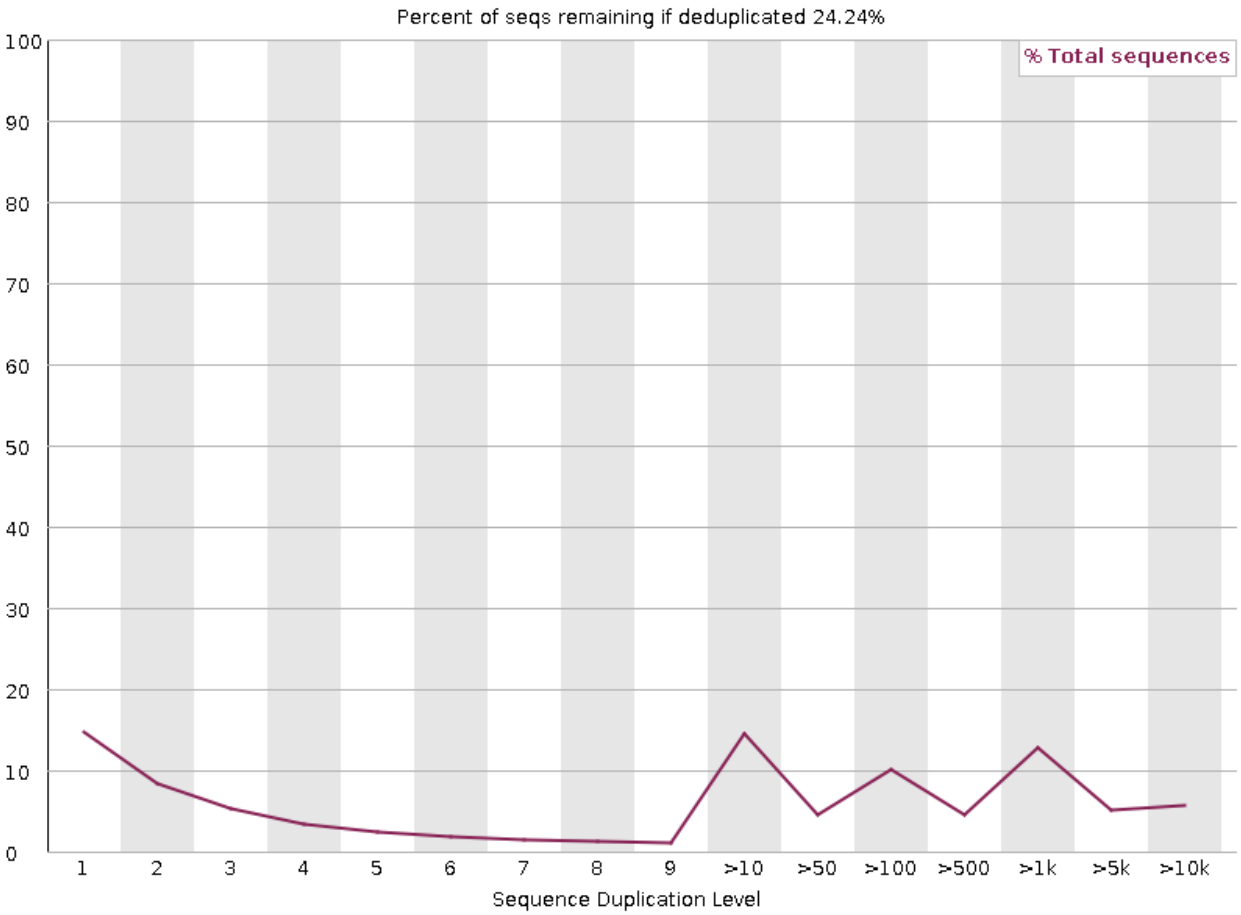

Figure 17: Duplication levels for Read 1 of SRR25630385

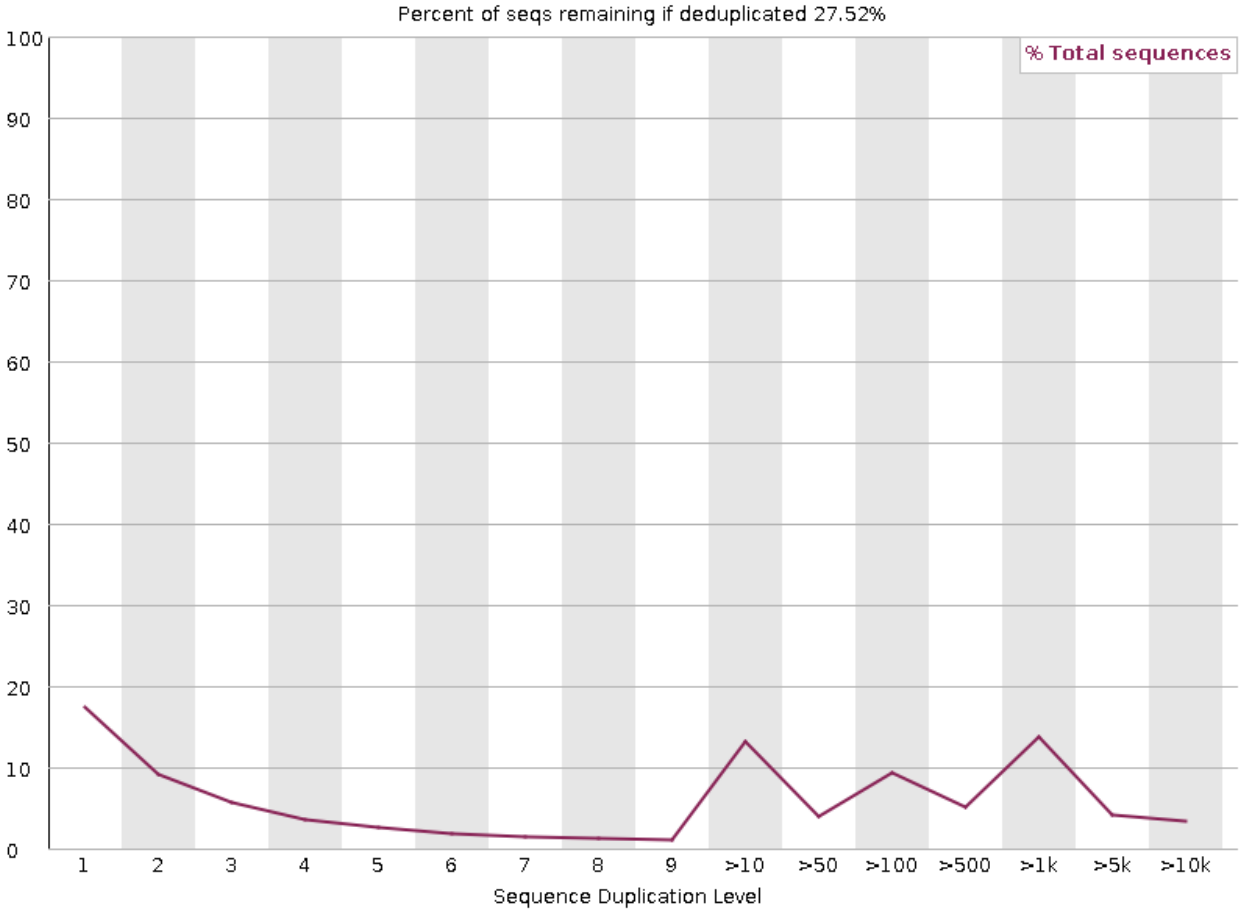

Figure 18: Duplication levels for Read 2 of SRR25630385

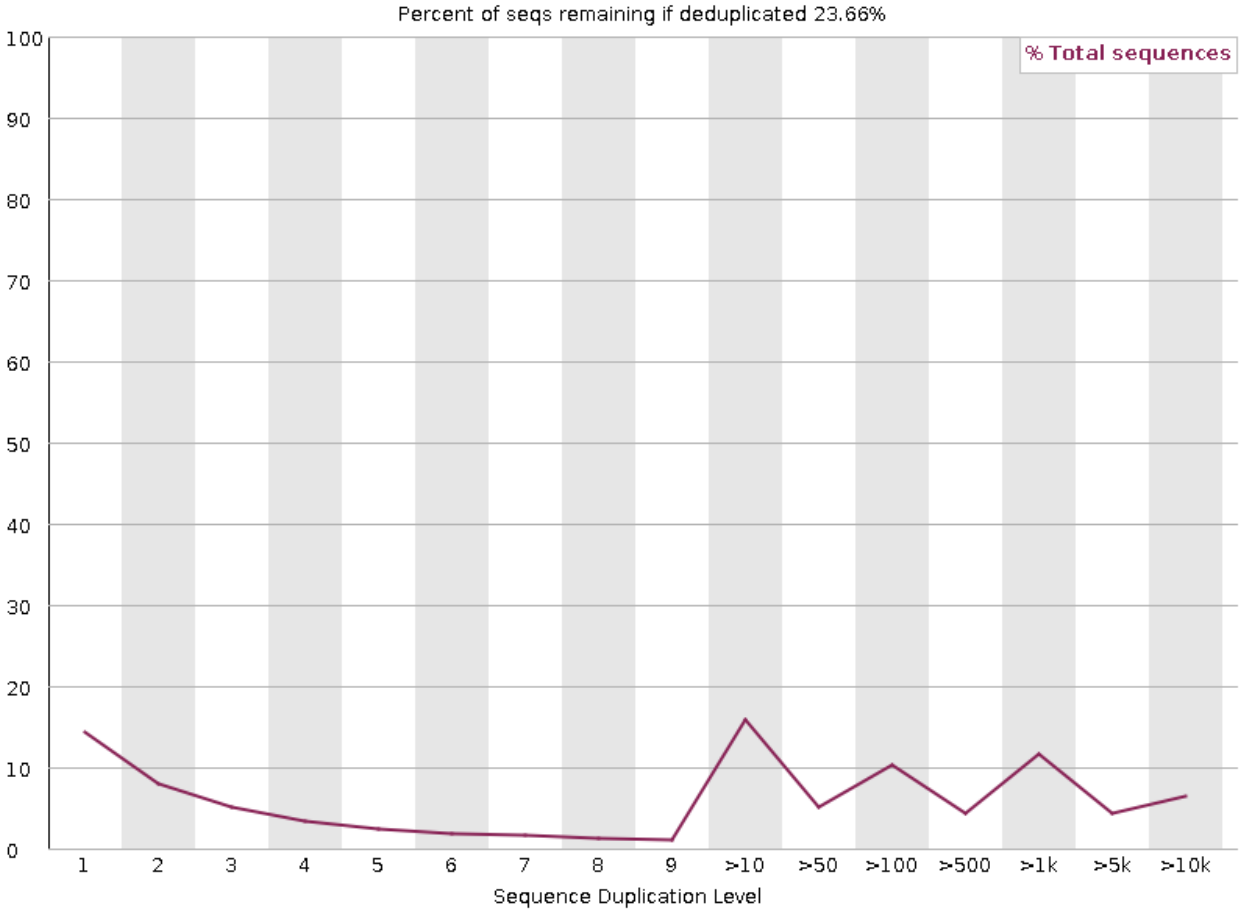

Figure 19: Duplication levels for Read 1 of SRR25630409

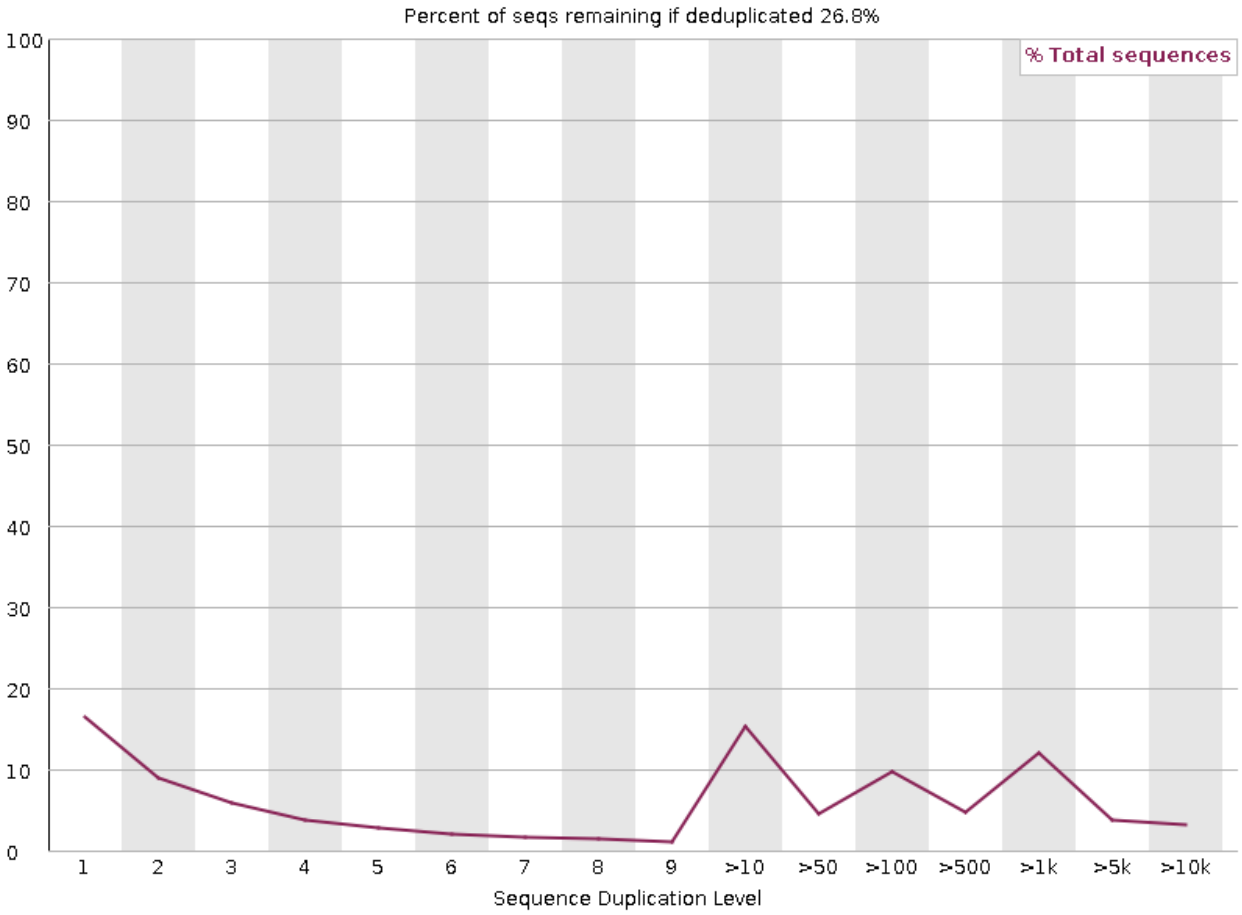

Figure 20: Duplication levels for Read 2 of SRR25630409

Part 2

cutadapt

Adapters were trimmed from ~23% of each read for `SRR25630385` and ~15% of reads from `SRR25630409`.

SRR25630385

Read 1:

```
Total reads processed:              33,181,752
Reads with adapters:                 7,592,116 (22.9%)
Reads written (passing filters):    33,181,752 (100.0%)
```

Read 2:

```
Total reads processed:              33,181,752
Reads with adapters:                 7,416,044 (22.3%)
Reads written (passing filters):    33,181,752 (100.0%)
```

SRR25630409

Read 1:

```
Total reads processed:              43,543,075
Reads with adapters:                 6,476,589 (14.9%)
Reads written (passing filters):    43,543,075 (100.0%)
```

Read 2:

```
Total reads processed:              43,543,075
Reads with adapters:                 6,531,478 (15.0%)
Reads written (passing filters):    43,543,075 (100.0%)
```

Plotting sequence length of trimmed reads

Sequence lengths distributions were gathered with a bash script and plotted in R.

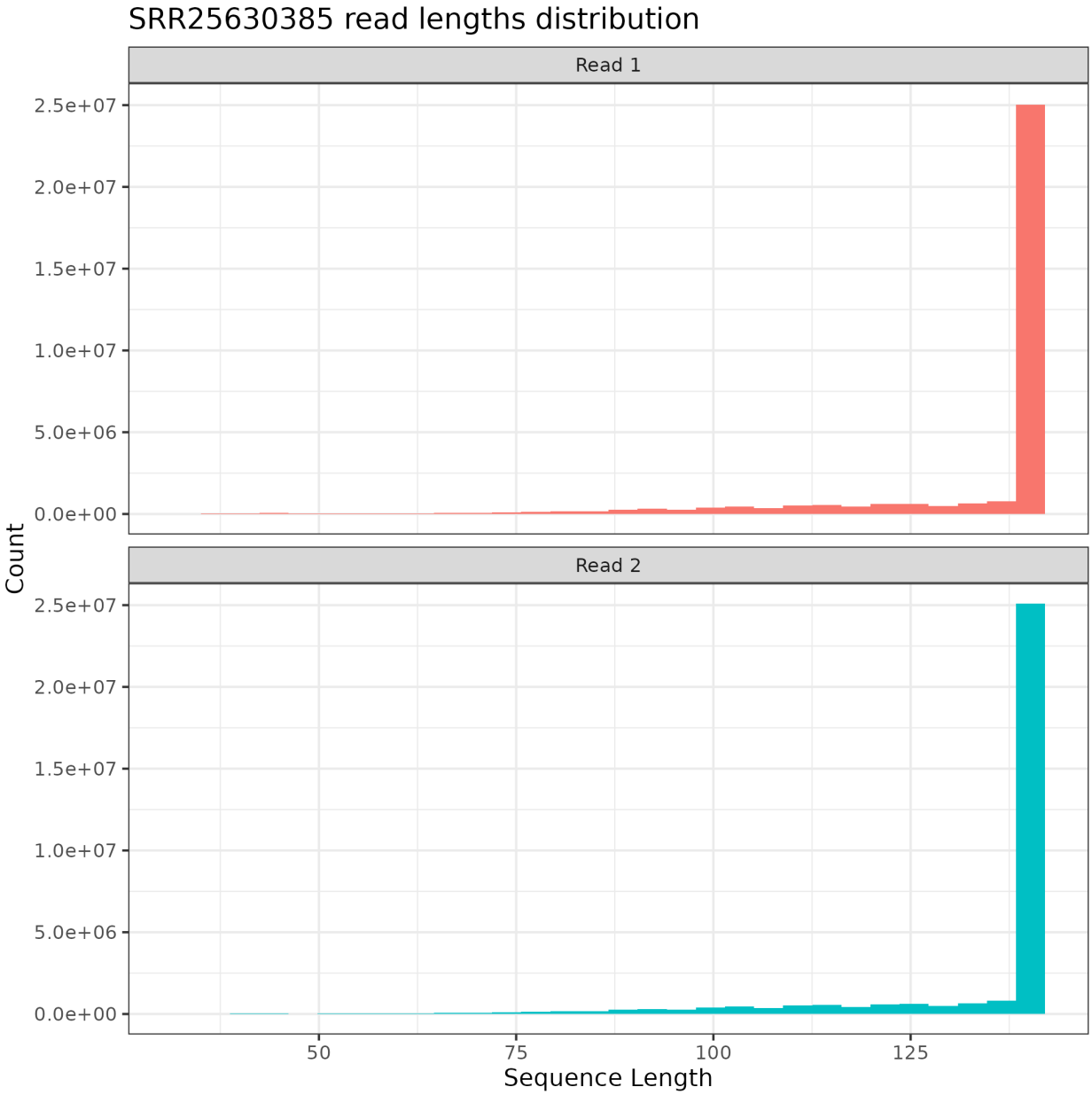

Figure 21: Sequence lengths of reads from SRR25630385 trimmed with cutadapt.

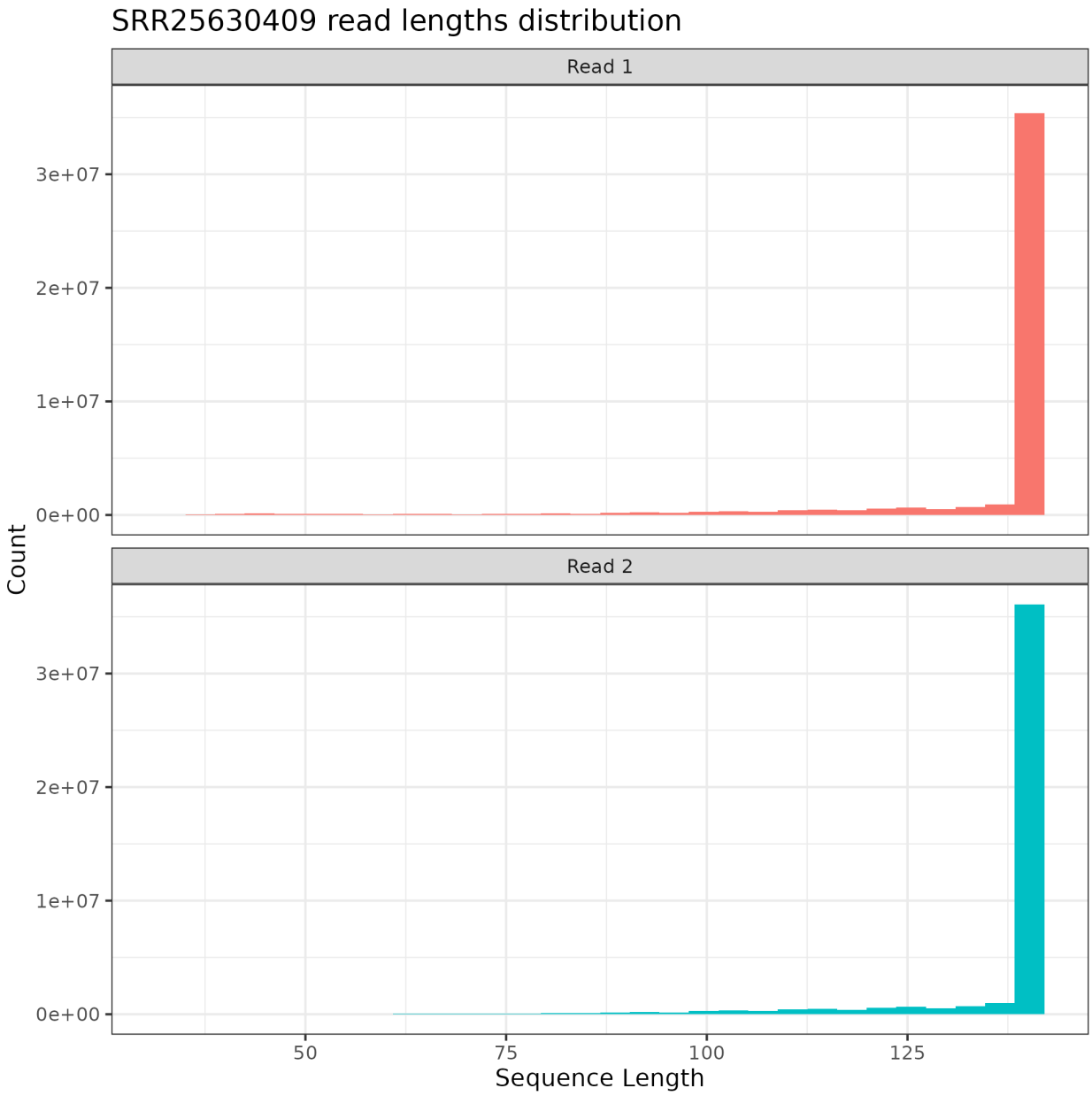

Figure 22: Sequence lengths of reads from SRR25630409 trimmed with cutadapt.

Part 3

mapped and unmapped reads

After aligning with star and deduplication with picard, here are the numbers of mapped and unmapped reads:

SRR	Mapped	Unmapped
SRR25630385	22910539	17341635
SRR25630409	33240914	17431172

Table 1: Counts of mapped and unmapped reads after alignment and deduplication for each SRR id.

htseq-count

SRR	Forward	Reverse
SRR25630385	0.06779142	0.4786234
SRR25630409	0.08263348	0.5089198

Table 2: Ratio of non-zero RNAseq counts.

Looking at table 2, we can see that there are not a lot of genes which have counts interpreting these as forward reads, with only 6.7% and 8.2% of genes having sequences. Reverse reads, on the other hand, have far more genes with non-zero counts, with 47.8% and 50.8% of genes having non-zero counts. If this library was not strand specific, we would expect these ratios to be the same within each SRR. As such, we can be confident that this should be interpreted as reverse reads.