PS1_Rmarkdown_Output

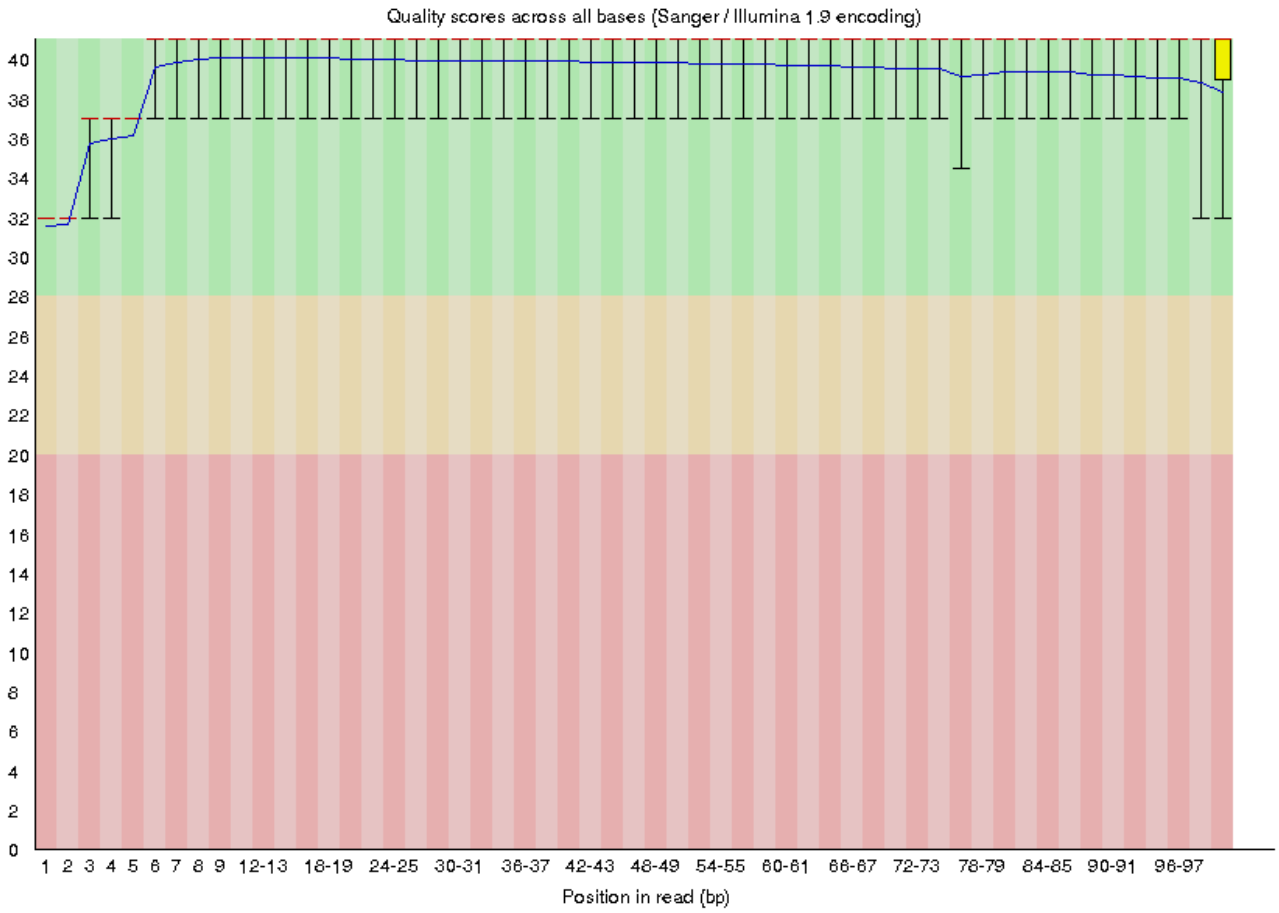
Dane Dewees

Wed Oct 4 12:43:58 2017

Part 1 – SF-Seq read quality score distributions

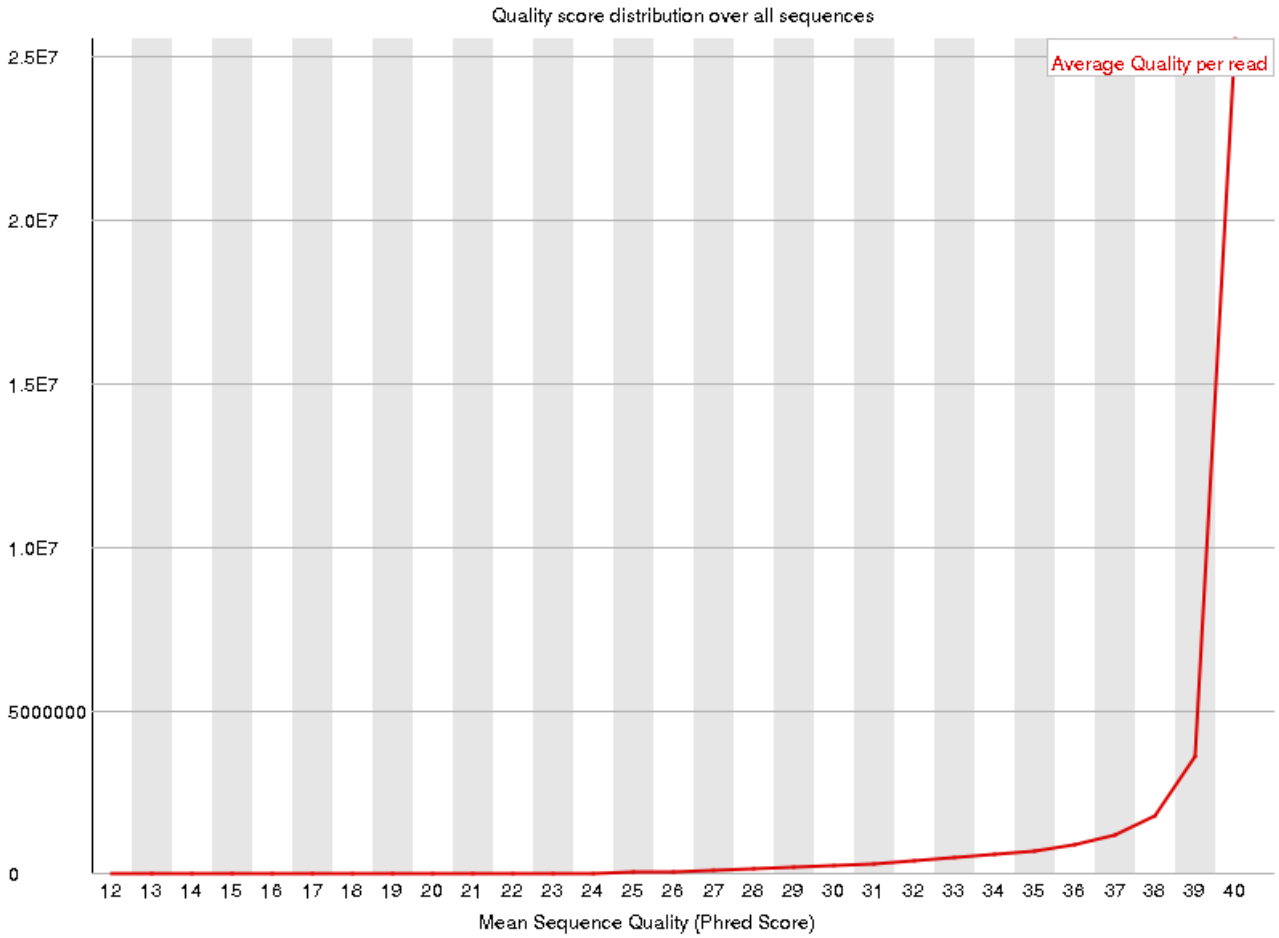
Using FastQC on Talapas, produce plots of quality score distributions for forward and reverse reads. Also, produce plots of the per-base N content, and comment on whether or not they are consistent with the quality score plots.

8_2F_fox_R1: Per Base Sequence Quality output - FastQC



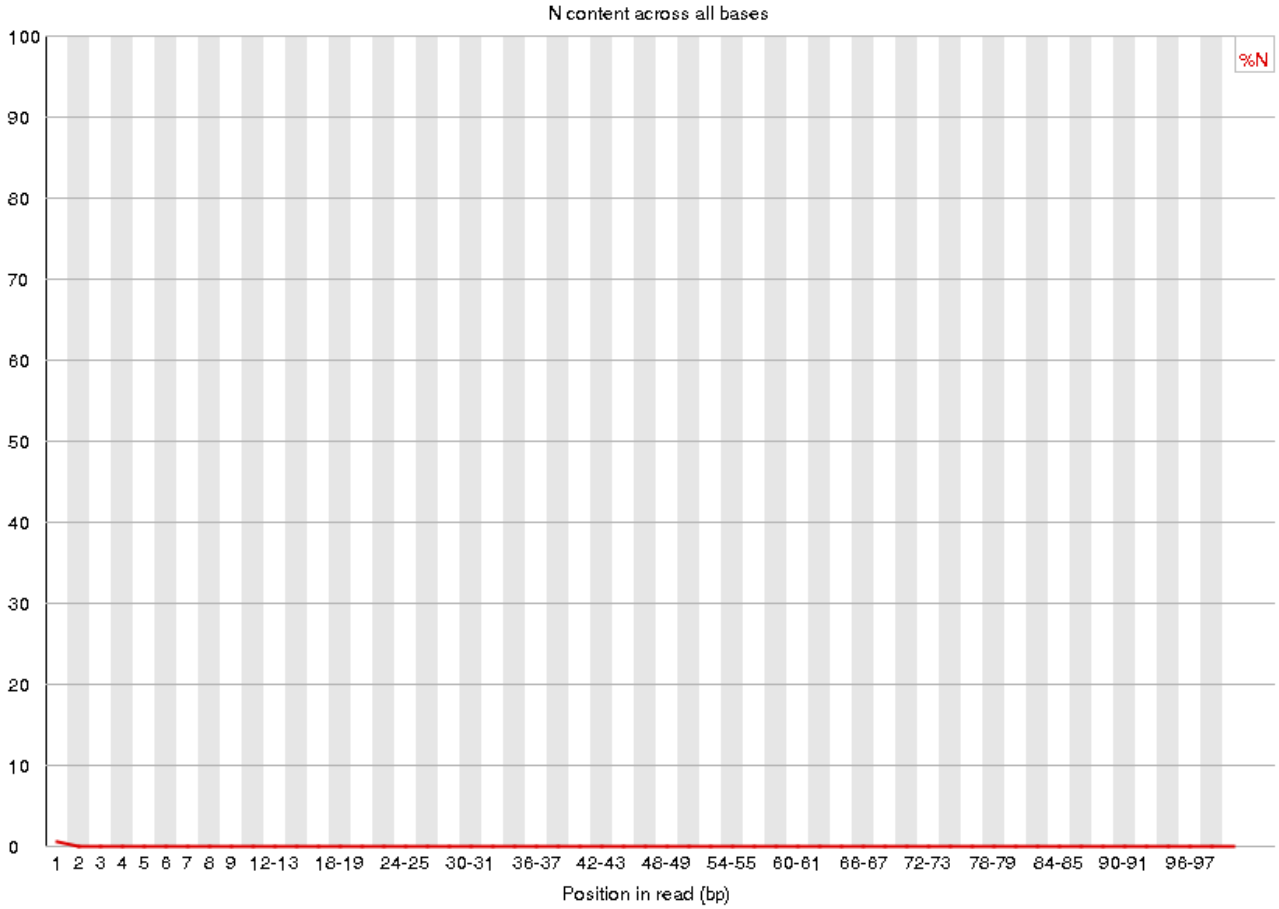
{width=550px height 550px}

8_2F_fox_R1: Per Sequence Quality output - FastQC



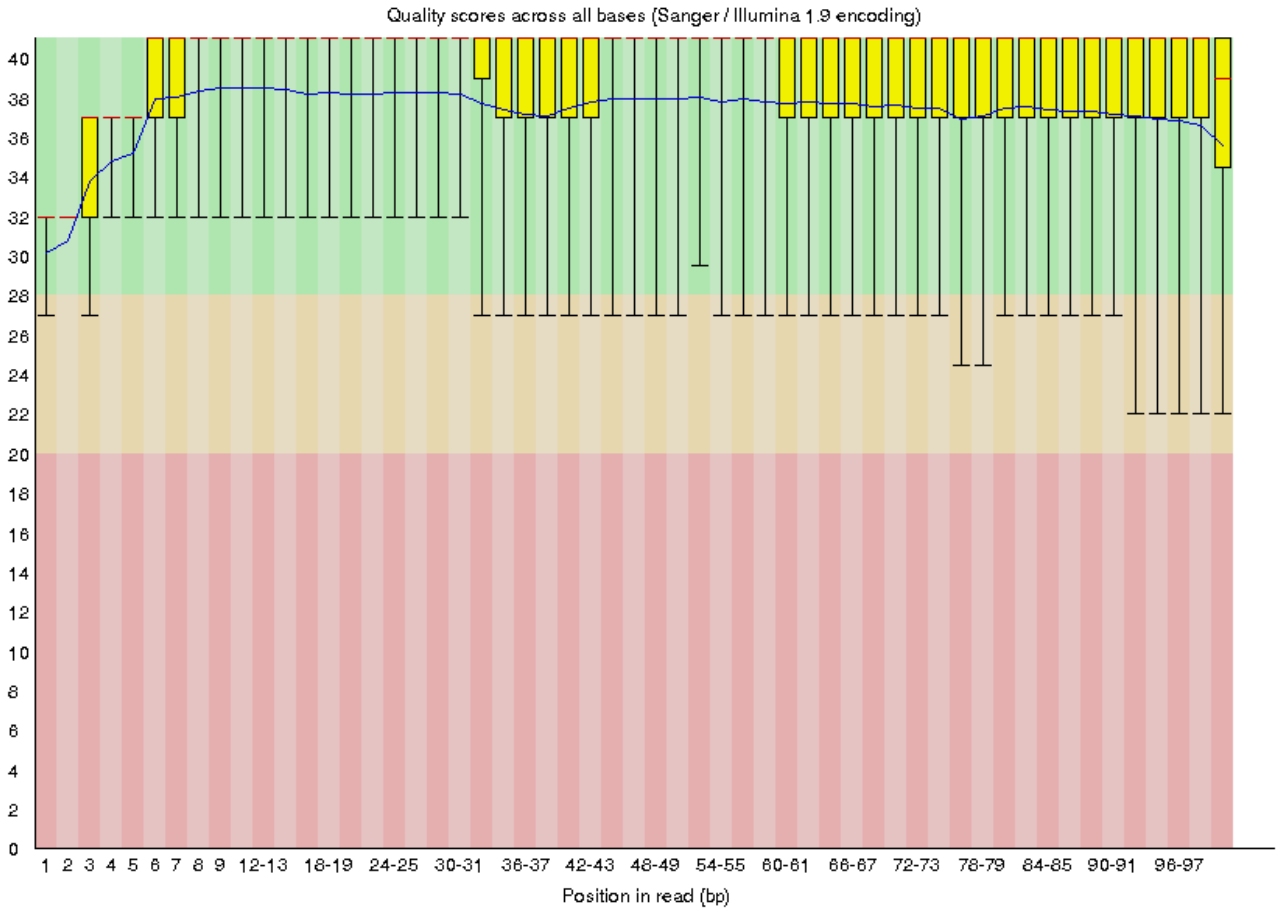
{width=550px height 550px}

8_2F_fox_R1: Per Base N Content output - FastQC



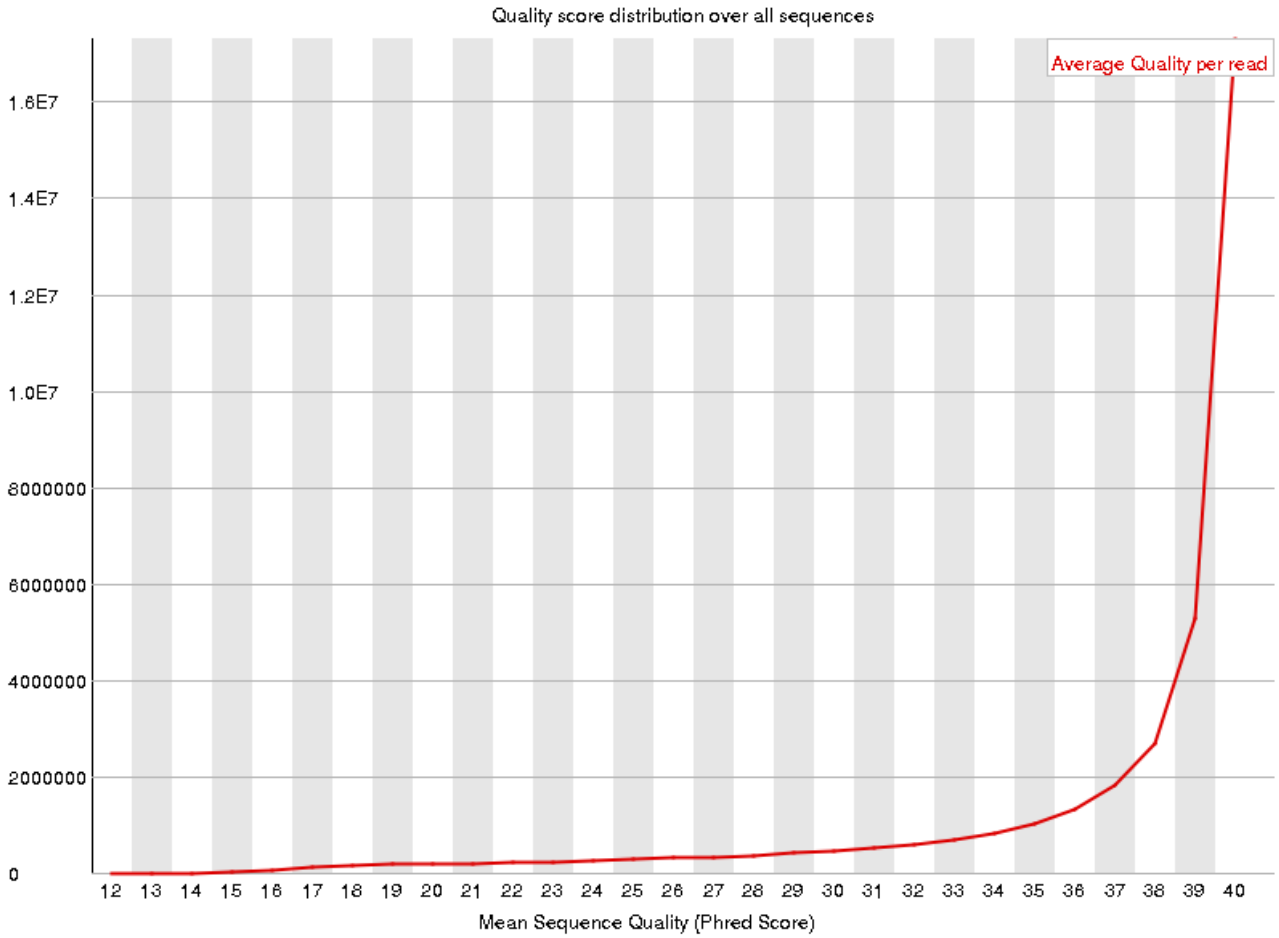
{width=550px height 550px}

8_2F_fox_R1: Per Base Sequence Quality output - FastQC



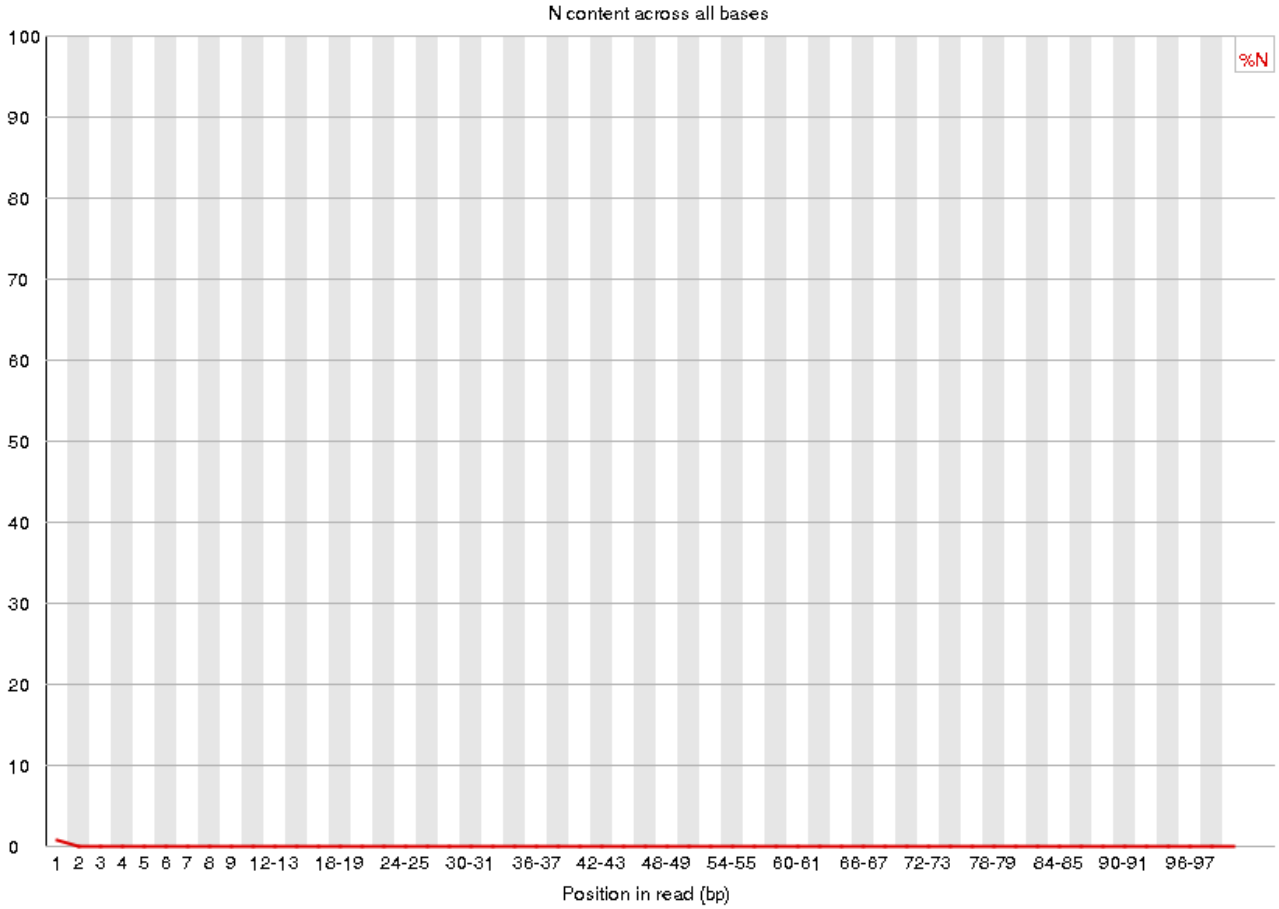
{width=550px height 550px}

8_2F_fox_R2: Per Sequence Quality output - FastQC



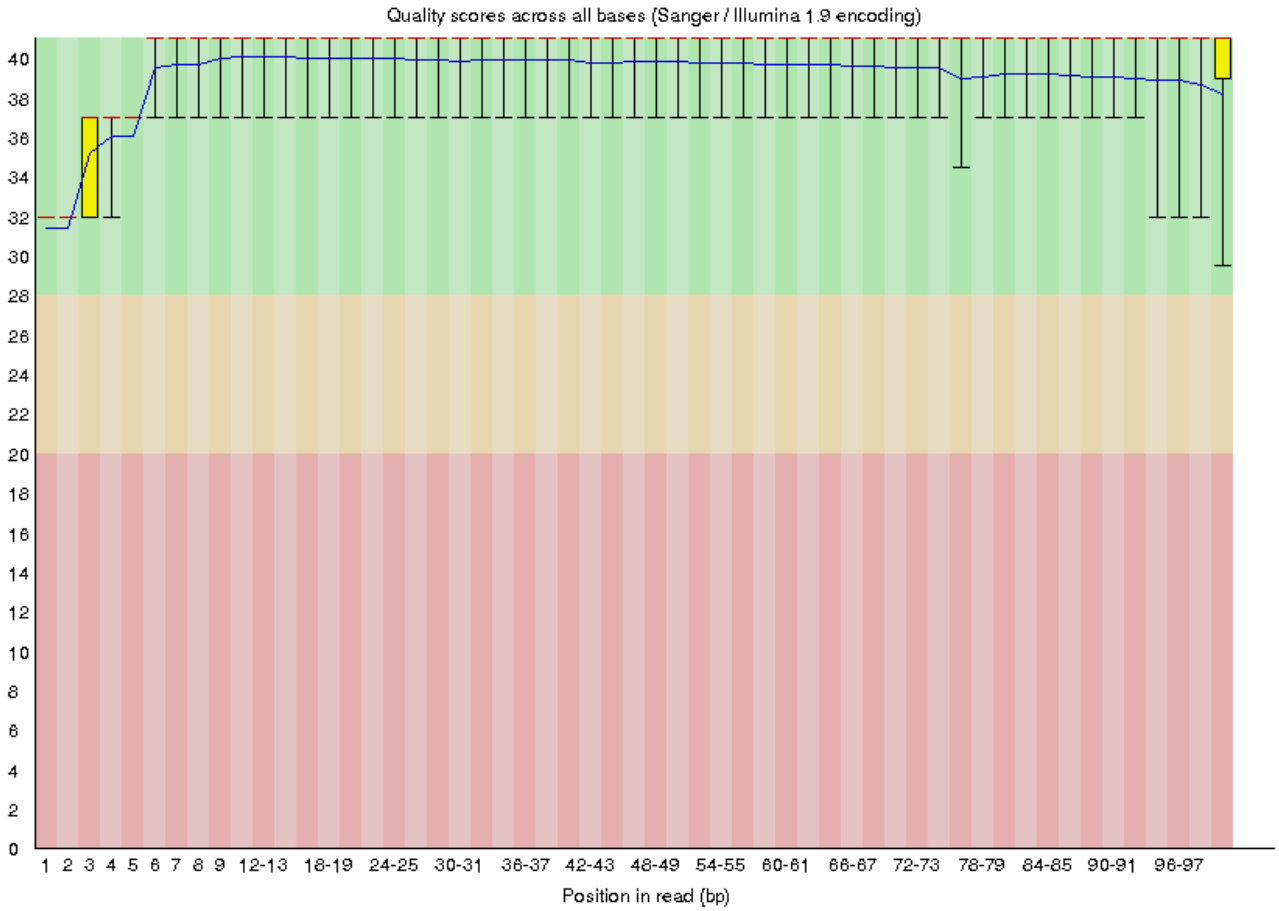
{width=550px height 550px}

8_2F_fox_R2: Per Base N Content output - FastQC



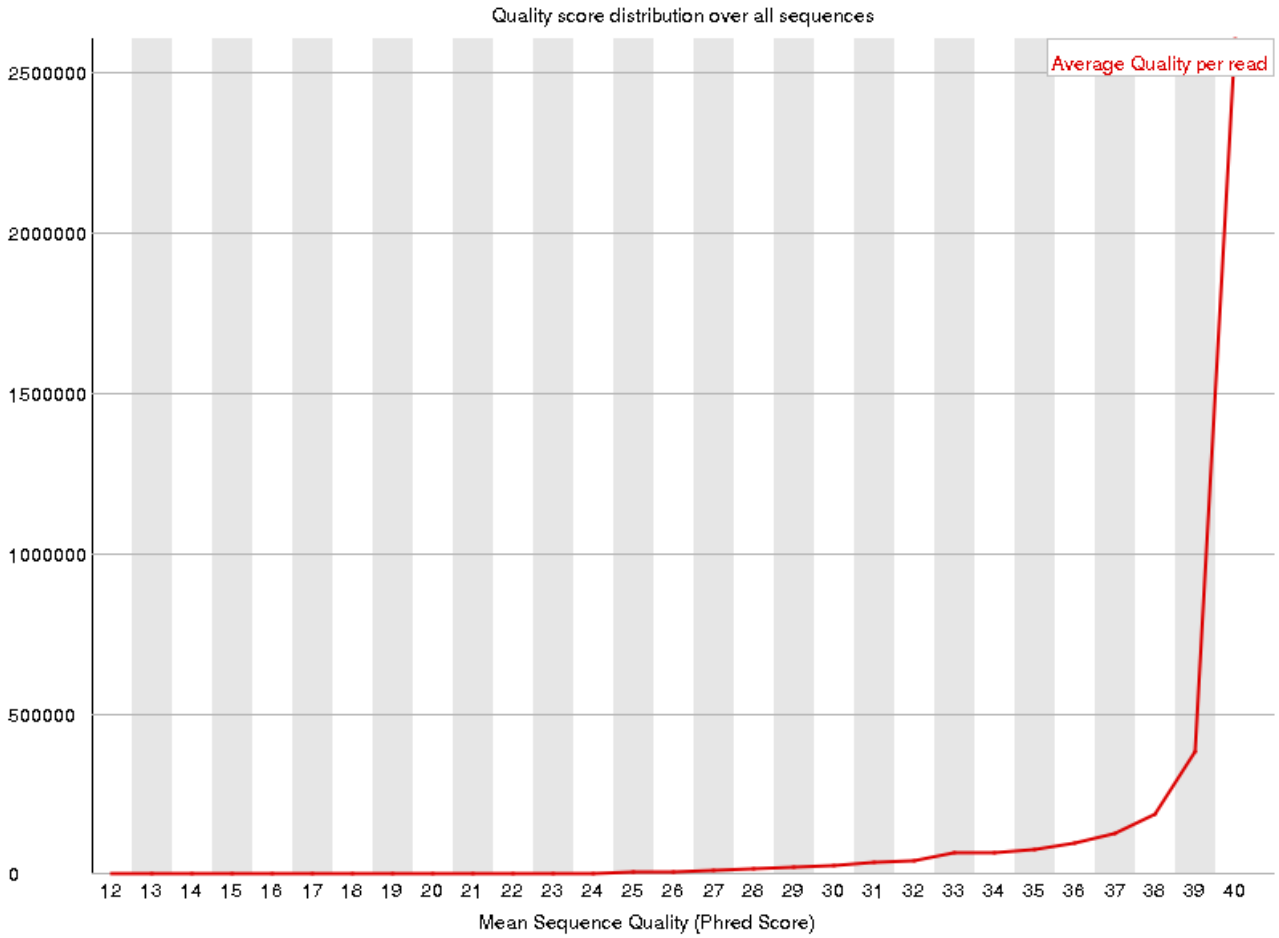
{width=550px height 550px}

31_4F_fox_R1: Per Base Sequence Quality output - FastQC



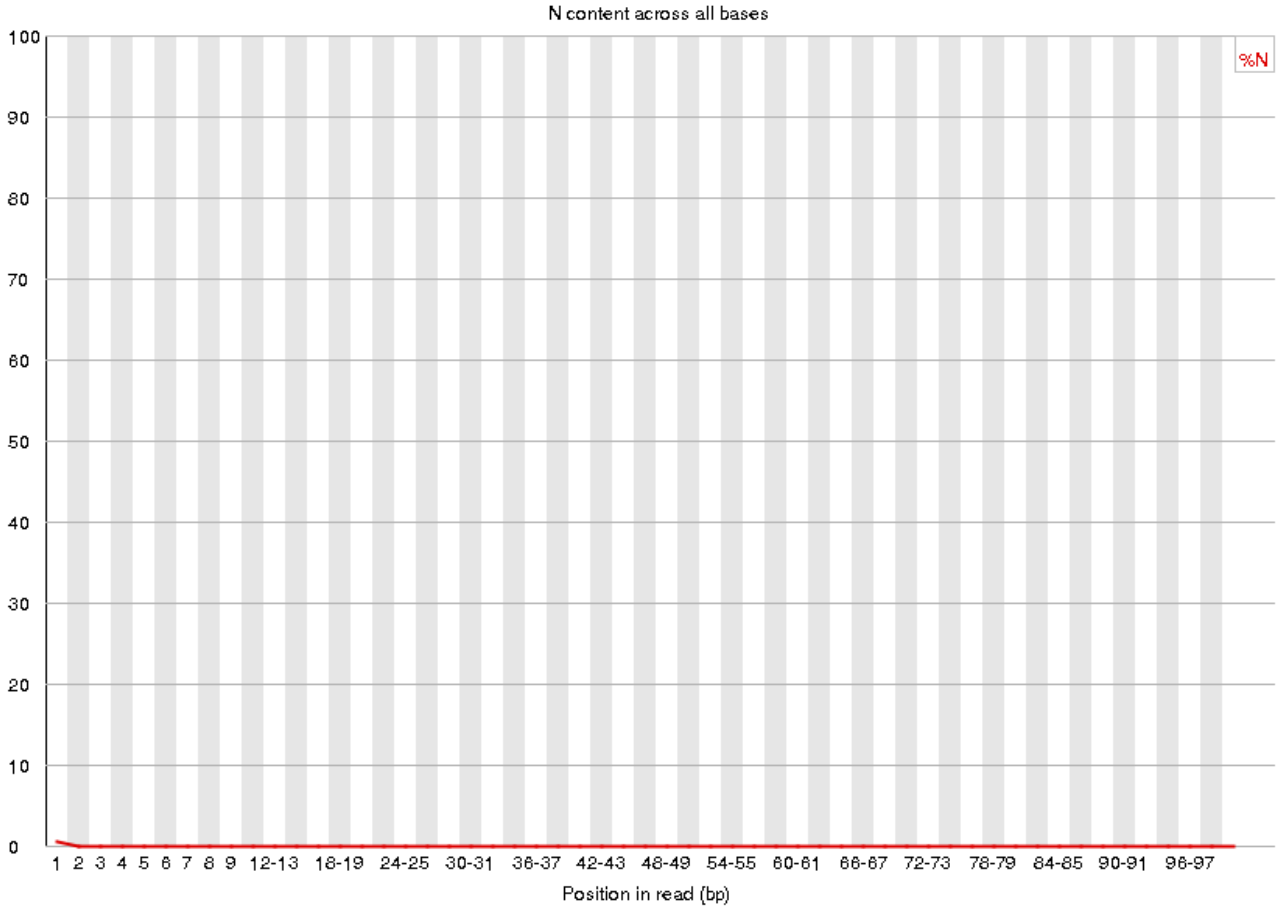
{width=550px height 550px}

31_4F_fox_R1: Per Sequence Quality output - FastQC



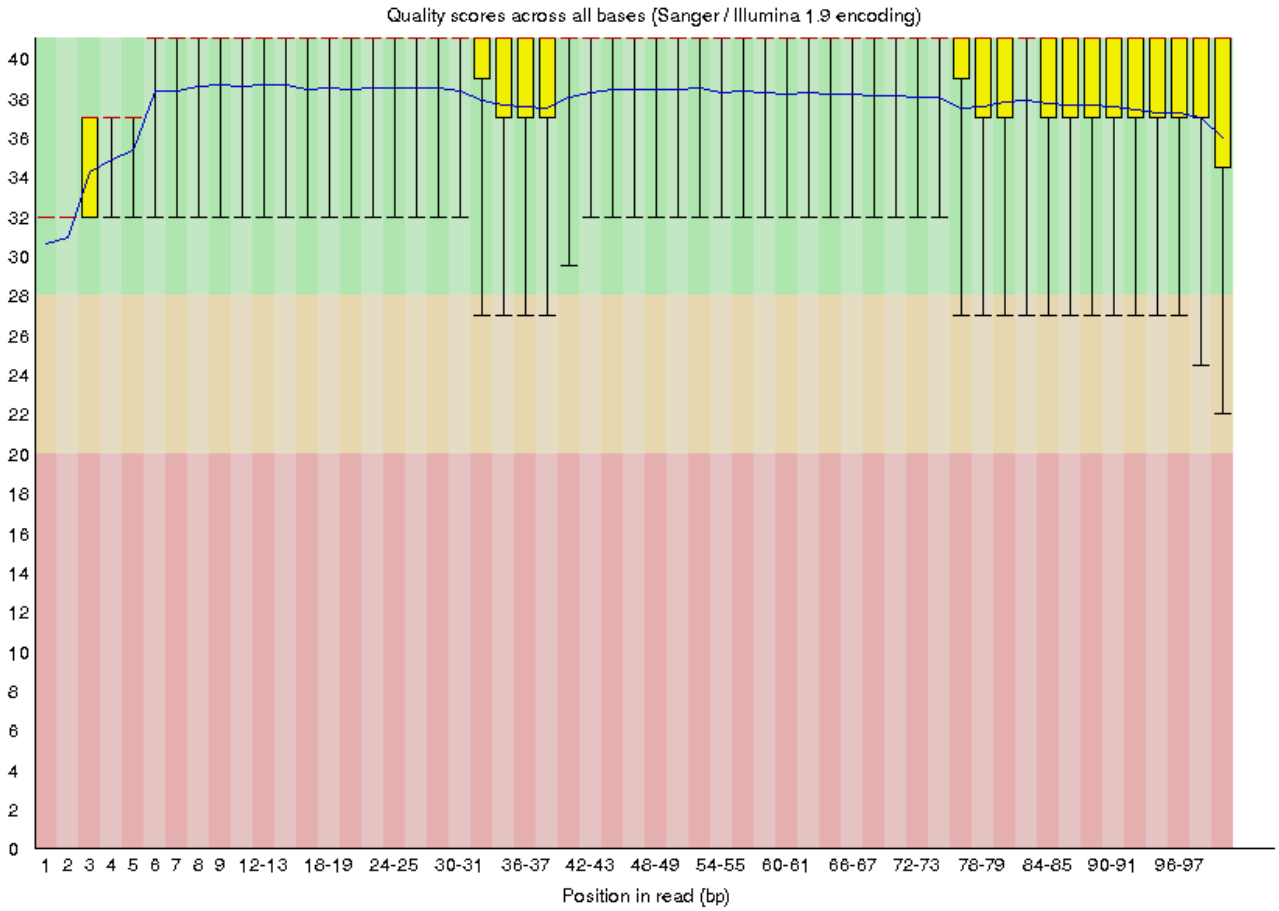
{width=550px height 550px}

31_4F_fox_R1: Per Base N Content output - FastQC



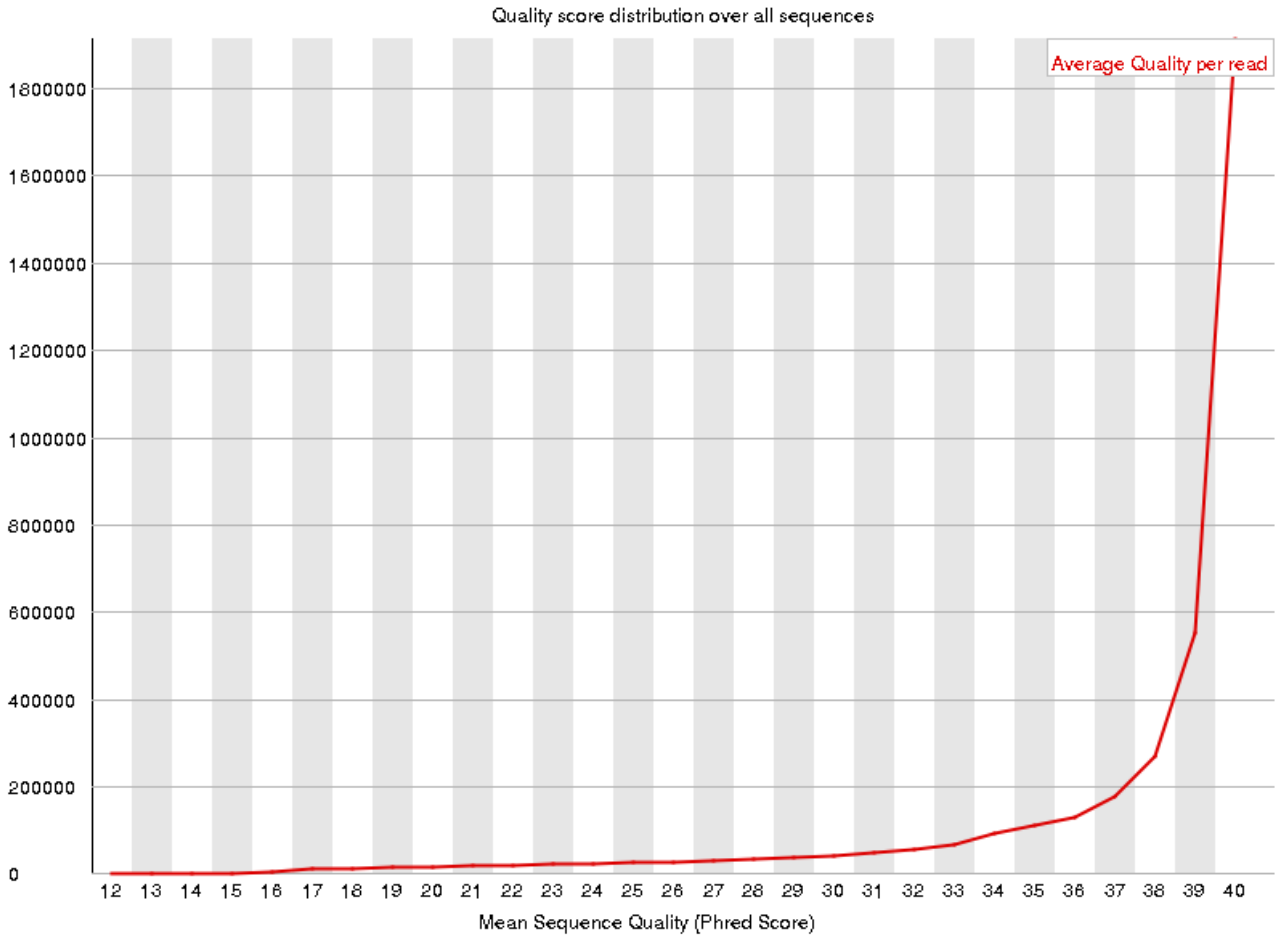
{width=550px height 550px}

31_4F_fox_R2: Per Base Sequence Quality output - FastQC



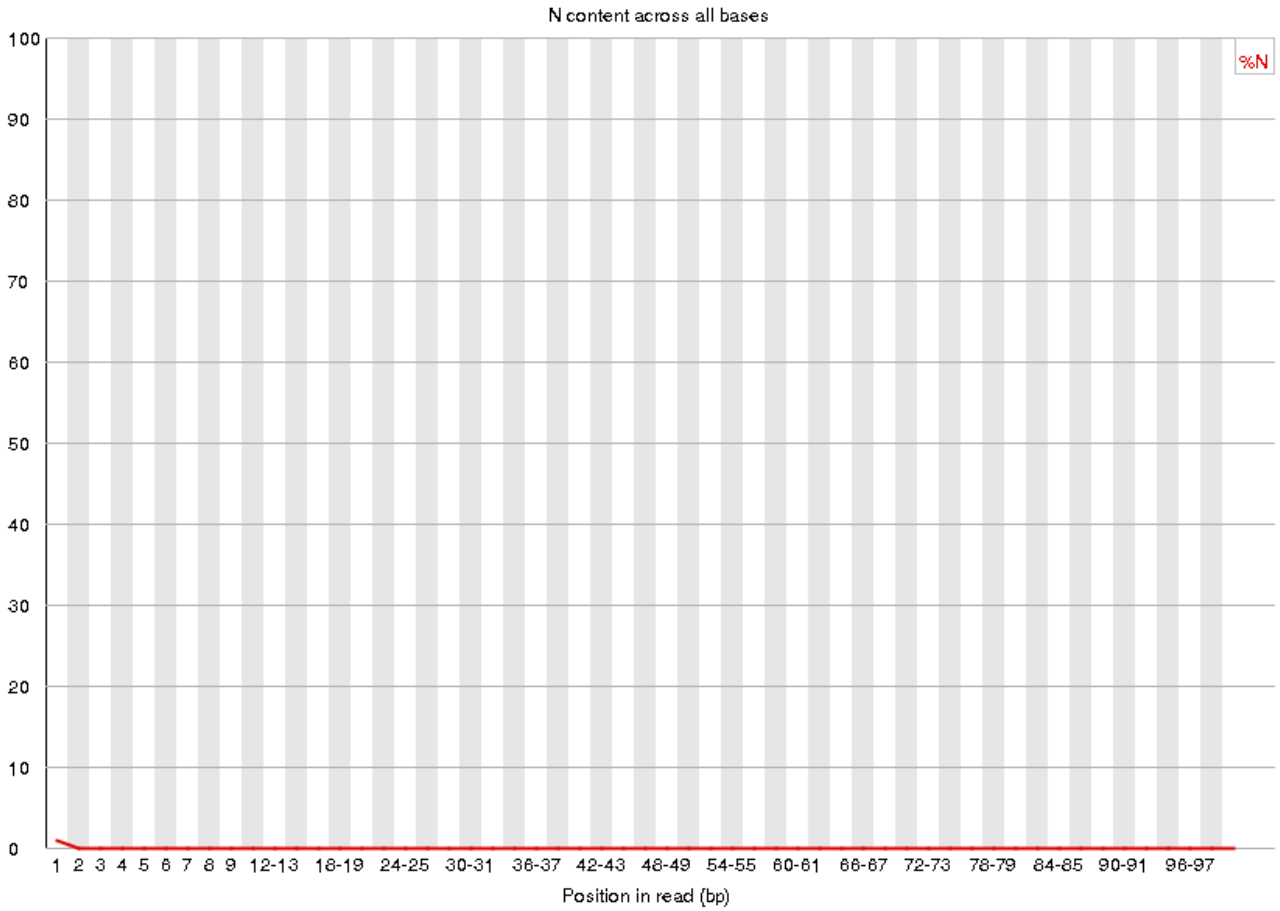
{width=550px height 550px}

31_4F_fox_R2: Per Sequence Quality output - FastQC



{width=550px height 550px}

31_4F_fox_R2: Per Base N Content output - FastQC

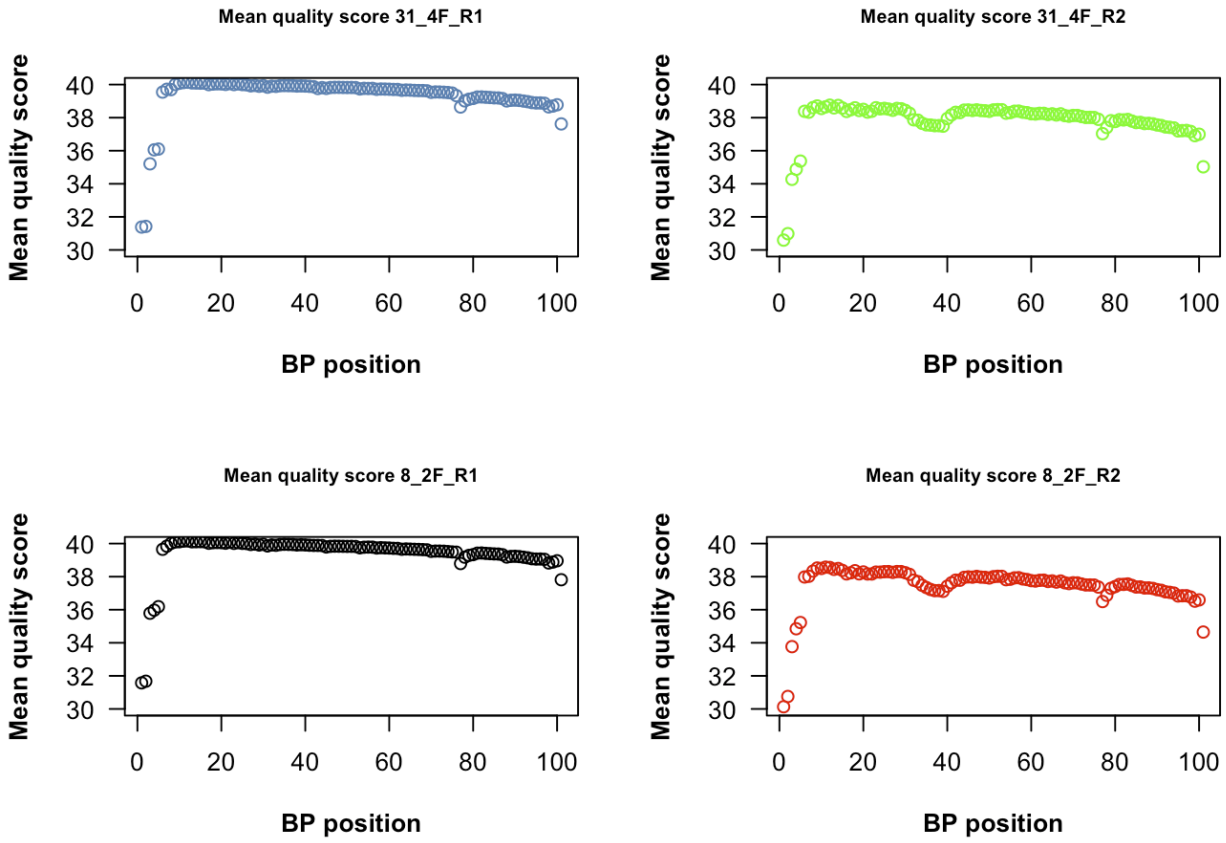


{width=550px height 550px}

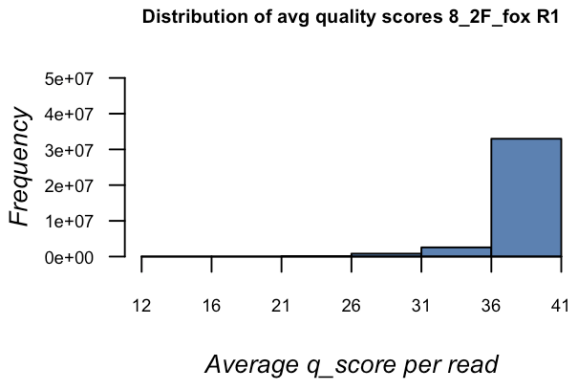
comment on whether or not they are consistent with the quality score plots.

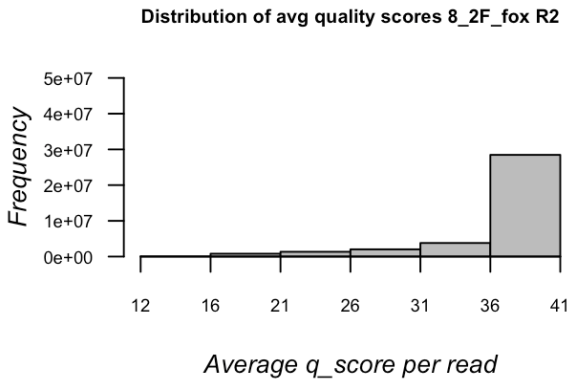
From the images above, you can see there were no differences in the per-base N content plots to the q_score distribution plots. All indicating that there were little to no N-content present in the data sets.

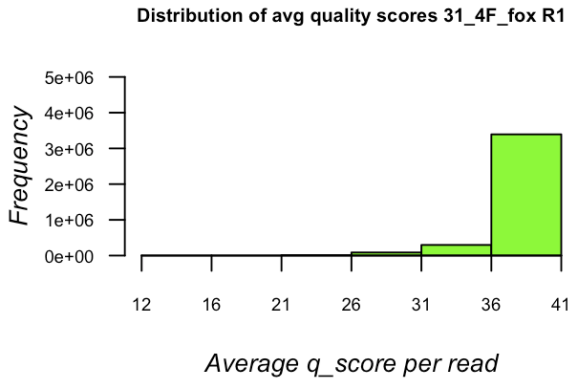
Plotting mean QS per base pair posiiton data sets from my script

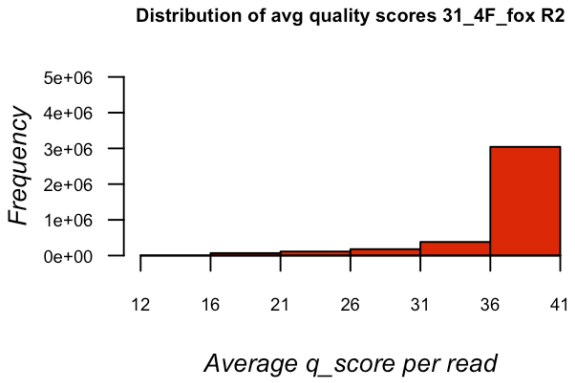


Plotting frequency distribution per QS data sets from my script









Describe how the FastQC quality score distribution plots compare to your own. If different, propose an explanation. Also, does the runtime differ? If so, why?

There seems to be little difference between my plots and the FastQC plots with regards to the mean quality score distrbution across the basepair positions. Each roughly hovered around 39-40 Qscore for the forward reads and between 37-38 for the reverse reads. This showed that the libraries generated had little to no error probability when run through Illumina sequencing (Reference HTML files uploaded to hithub to cross-reference the figures there to the R scripts above). I noticed an error bar difference among the FastQC plots.

The runtime was rather different when looking at FastQC compared to my python script. FastQC took roughly 00:06:00 for 3 of the 4 data sets (one took around 15 minutes), whereas when ran on my script, the three files that took 6 minutes took around 15 minutes and the larger files took around 2 hours. I think this is simply do to our python scripts not being optimized and also the FastQC being written in a lnaguage that is more ideal for these types of processings (JAVA). This, in turn, can produce faster results and generate output quicker than our raw scripts. IF we were able to optimize our scripts, I believe it would not have taken as long. Overall, there is just a functional difference when looking at JAVA to Python and each utalized for vairous things.

Part 2 - Adaptor Trimming Comparision

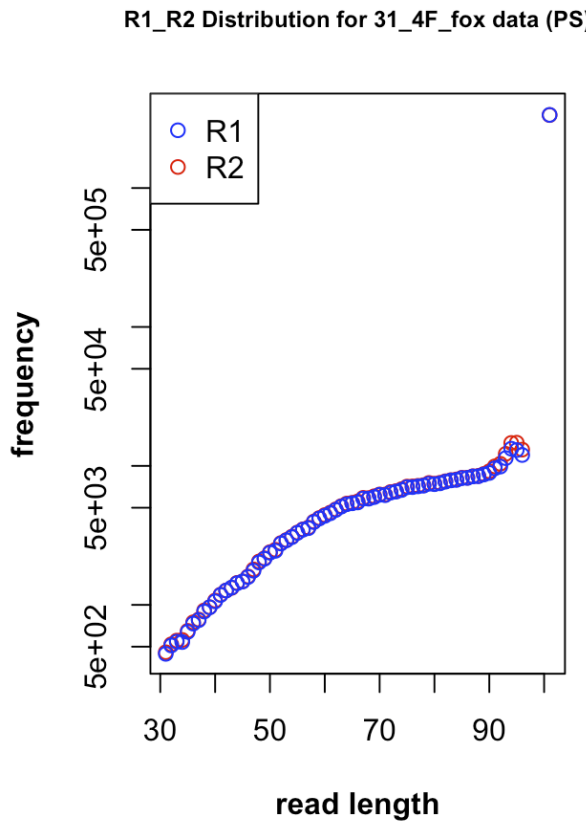
Look into the adaptor trimming options for cutadapt, process_shortreads, and Trimmomatic (all on Talapas), and briefly describe the differences.

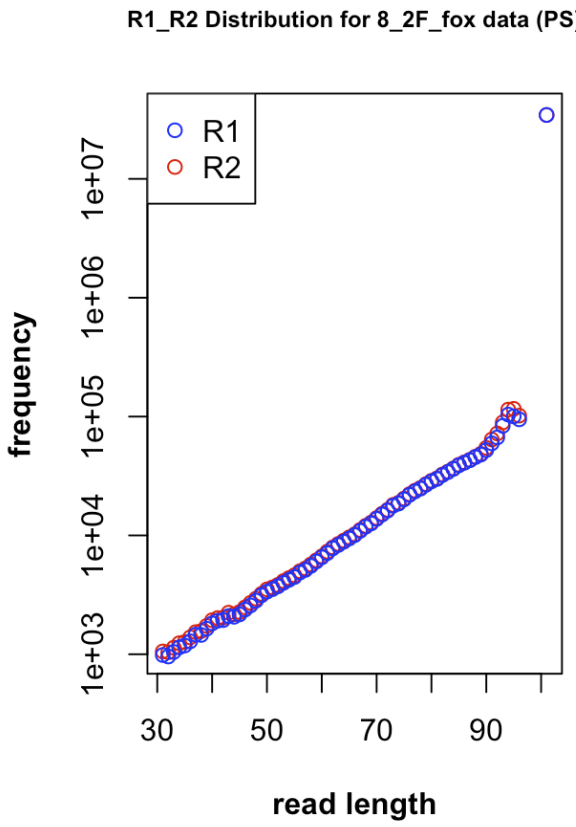
Cutadapt finds and removes adapter sequences, primers, poly-A tails and other types of unwanted sequence from your high-throughput sequencing reads. It is N-tolerant and it has a default setting that enables up to ~10% error rate

Process_shortreads will trim reads that are below quality threshold instead of discarding them, making it useful for genomic assembly or other analyses. It requires adapters(barcodes) and can take both forward/reverse reads.

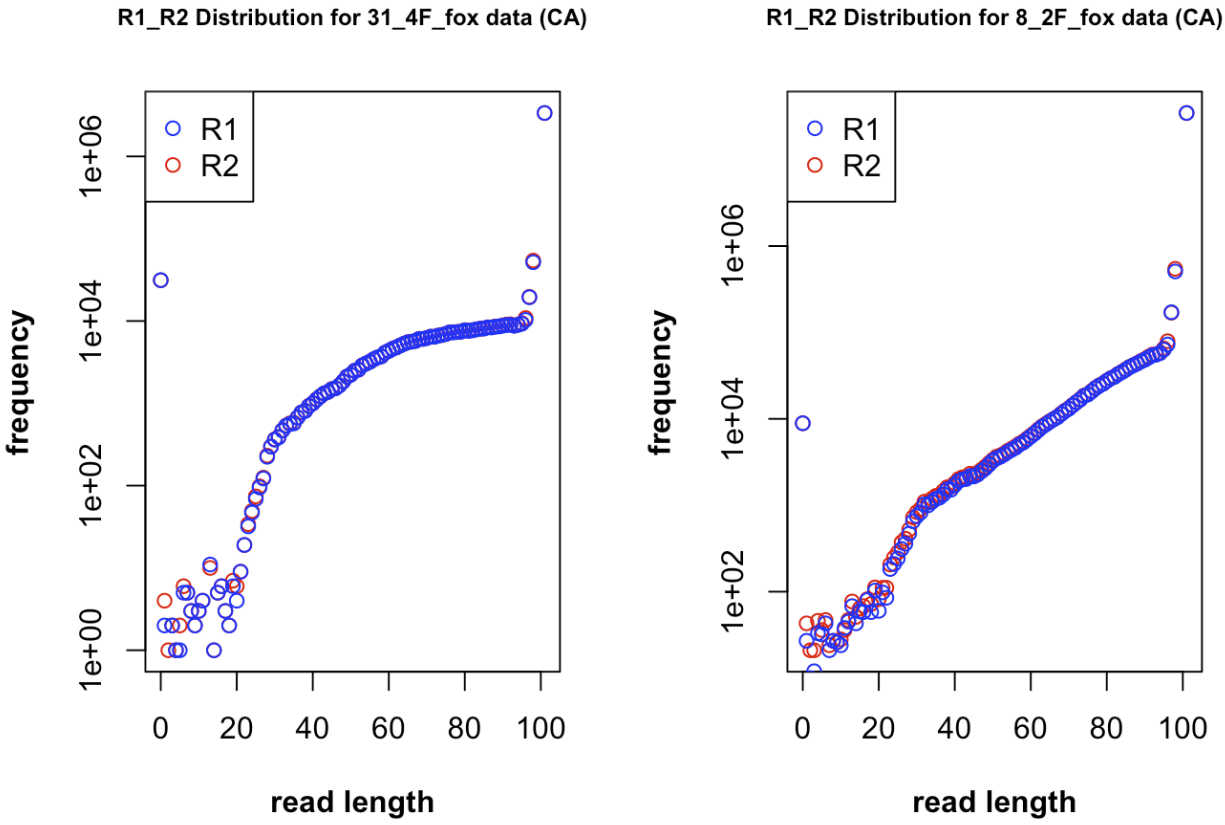
Trimmomatic performs a variety of useful trimming tasks for illumina paired-end and single ended data.It requires a FASTA file that has the associated adapters.

Plots for process_shortreads data sets





Plots for cutadapt data sets



Given the fact that the average size of BP in our fragment analyzer output file, there shouldn't be much to any adapter contamination. Given that I only found 6% and 12% in my R1/R2 reads for both 8_2F and 31_4F, this matched with the corresponding libraries shown in the fragment analyzer output dataset.

=== Summary 8_2F_fox_R1/R2 data ===

Total read pairs processed: 36,482,601 Read 1 with adapter: 2,145,571 (5.9%) Read 2 with adapter: 2,307,693 (6.3%) Pairs written (passing filters): 36,482,601 (100.0%)

=== Summary 31_4F_fox_R1/R2 data ===

Total read pairs processed: 3,788,343 Read 1 with adapter: 456,153 (12.0%) Read 2 with adapter: 407,186 (10.7%) Pairs written (passing filters): 3,788,343 (100.0%)

Part 3 - rRNA reads and strand - specificity

Align the SF-Seq reads to your mouse rRNA database and report the proportion of reads that likely came from rRNAs.

To determine the proportion of reads that likely originated from rRNA, we can count the number of reads that did NOT map to any rRNA sequences, and subtract that from the total number of reads to determine the portion of reads that did map to rRNA (See equations below)

31_4F_fox_R1 reads vs total reads in gsnap_out_31_4F_R1.nomapping

```
1 - (3720606/3788343)
```

```
## [1] 0.01788038
```

31_4F_fox_R2 reads vs total reads in gsnap_out_31_4F_R2.nomapping

```
1 - (3720901/3788343)
```

```
## [1] 0.01780251
```

8_2F_fox_R1 reads vs total reads in gsnap_out_8_2F_R1.nomapping

```
1 - (35637300/36482601)
```

```
## [1] 0.02316998
```

8_2F_fox_R2 reads vs total reads in gsnap_out_8_2F_R2.nomapping

```
1 - (35641880/36482601)
```

```
## [1] 0.02304444
```

As you can see, each dataset showed roughly 1-2% of all reads likely originated from rRNA based on the gsnap alignment. For the 31_4F_fox library, you see around 1.7% that alligned with rRNA and around 2.3% for the 8_2F_fox library datasets. Our libraries coould potentially be mostly rRNA reads if the polyA selection went poorly during preportation. A cell's rRNA, in theory, can achieve up to 95% of the total RNA in the cell, which is the exact reason why we wanted to get rid of it in the first place. Seeing that both libraries are rather low, I'd conclude that the library prep was sucessful.

How do we know our library is strand-specific?

The easiest method (which we experienced in Clay's class) was finding out what proporiton of the R1/R2 reads contains a piece of the poly-A tail (targeting homoploymer regions). Since we used a poly-A tail method to target and mark mRNA sequences with the tails, one should find a certain number of reads in either dataset, with it's corresponding set having fewer. The same should be the case for Poly-T tails as well.

Library for 31_4F_fox_trimmed files

[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_31_4F_fox_R1_001.fastq | grep "AAAAAAAAAAAAAAAAAAAAAA" | wc -l

940

[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_31_4F_fox_R2_001.fastq | grep "AAAAAAAAAAAAAAAAAAAAAA" | wc -l

2124

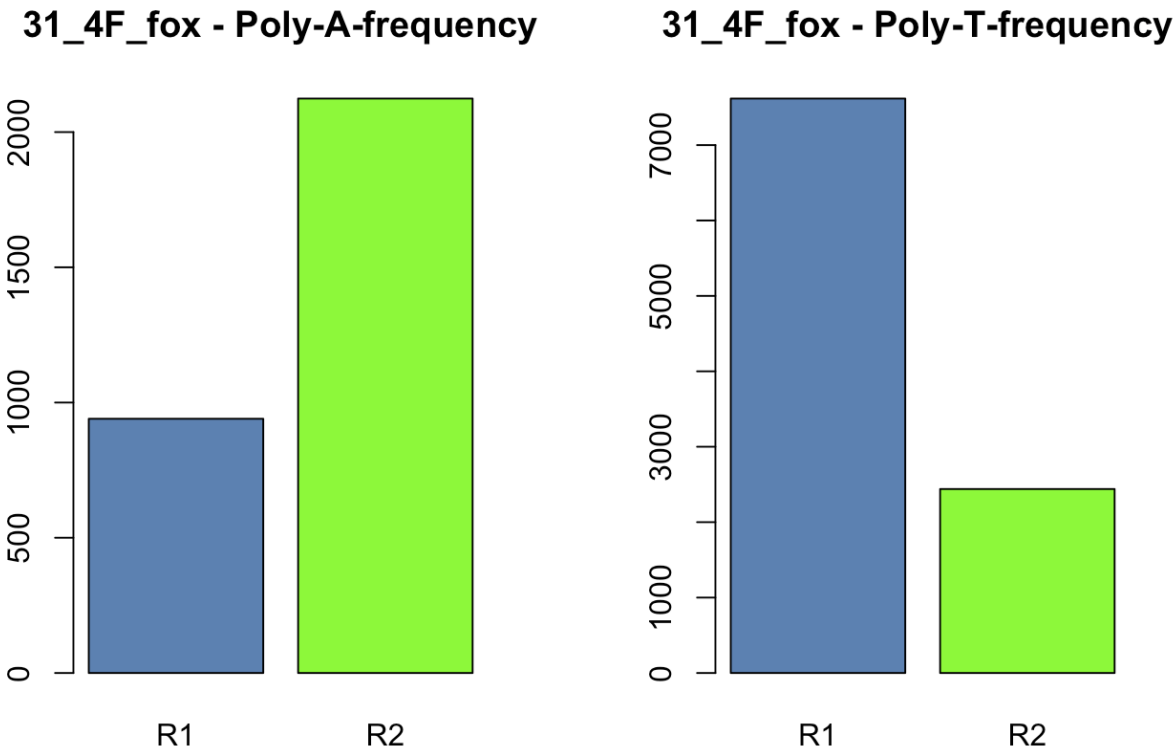
[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_31_4F_fox_R1_001.fastq | grep "TTTTTTTTTTTTTTTTTTTTTT" | wc -l

7617

[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_31_4F_fox_R2_001.fastq | grep "TTTTTTTTTTTTTTTTTTTTTT" | wc -l

2439

Plotting distribtution of rRNA in both trimmed libraries for 31_4F_fox-R1/R2 reads



Library for 8_2F_fox_trimmed files

[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_8_2F_fox_R1_001.fastq | grep "AAAAAAAAAAAAAAAAAAAAAA" | wc -l

11395

[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_8_2F_fox_R2_001.fastq | grep "AAAAAAAAAAAAAAAAAAAAAA" | wc -l

32969

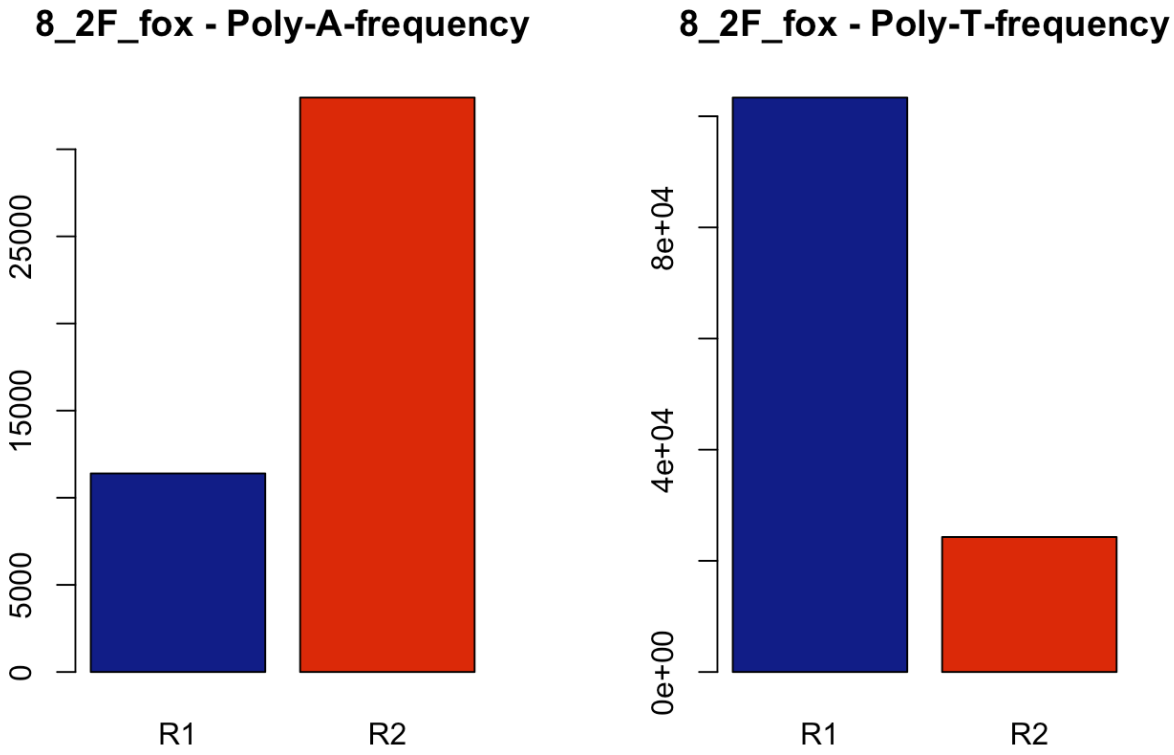
[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_8_2F_fox_R1_001.fastq | grep "TTTTTTTTTTTTTTTTTTTTTT" | wc -l

103366

[daned@ln1 (mailto:daned@ln1) ps1]$ awk 'NR % 4 == 2' trimmed_8_2F_fox_R2_001.fastq | grep "TTTTTTTTTTTTTTTTTTTTTT" | wc -l

24287

Plotting distribtution of rRNA in both trimmed libraries for 8_2F_fox-R1/R2 reads



As you see from the figures above, there is a constant trend with regards to the abundance of Poly-A tails and Poly-T tails found in both reads of the library datasets. The 8_2F_fox data set showed a far greater number of poly-A sequences in R2 than in R1, and the inverse of that for Poly-T sequences. The same is shown in the 31_4F_fox data set but with substentially less counts total compared to the other library data set.