RNA-seq Quality Assessment Assignment (QAA)

Christina Zakarian

9-4-2021

Objectives

The objectives of this assignment are to use existing tools for quality assessment and adapter trimming, compare the quality assessments to those from own software, and to summarize other important information about the RNA-Seq data set.

Input Files

The two samples I will be working with and the file locations of the paired end FASTQ files:

10_2G_both_S8

```
/projects/bgmp/shared/2017_sequencing/demultiplexed/10_2G_both_S8_L008_R1_001.fastq.gz
/projects/bgmp/shared/2017_sequencing/demultiplexed/10_2G_both_S8_L008_R2_001.fastq.gz
```

31_4F_fox_S22

```
/projects/bgmp/shared/2017_sequencing/demultiplexed/31_4F_fox_S22_L008_R1_001.fastq.gz
/projects/bgmp/shared/2017_sequencing/demultiplexed/31_4F_fox_S22_L008_R2_001.fastq.gz
```

Github repository containing files/data in this report: https://github.com/czakarian/QAA

All sbatch scripts mentioned can be found in the github repository in the sbatch_scripts folder.
All python scripts mentioned can be found in the github repository in the python_scripts folder.
Output from fastQC and htseq-count can be found in fastQC_ouput and htseq_output folders.

Part 1 – Read quality score distributions

1. FastQC generated plots of quality score distributions and N content distributions

10_2G_both_S8 FastQC Plots:

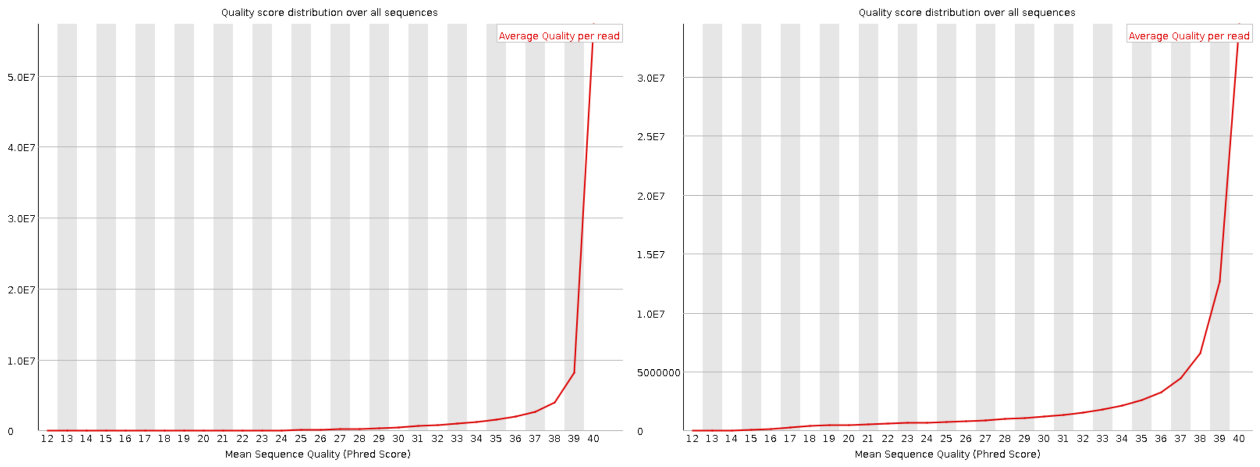


Figure 1: Per Sequence Quality Scores. Distribution of average quality scores per sequence for read 1 (left) and read 2 (right).

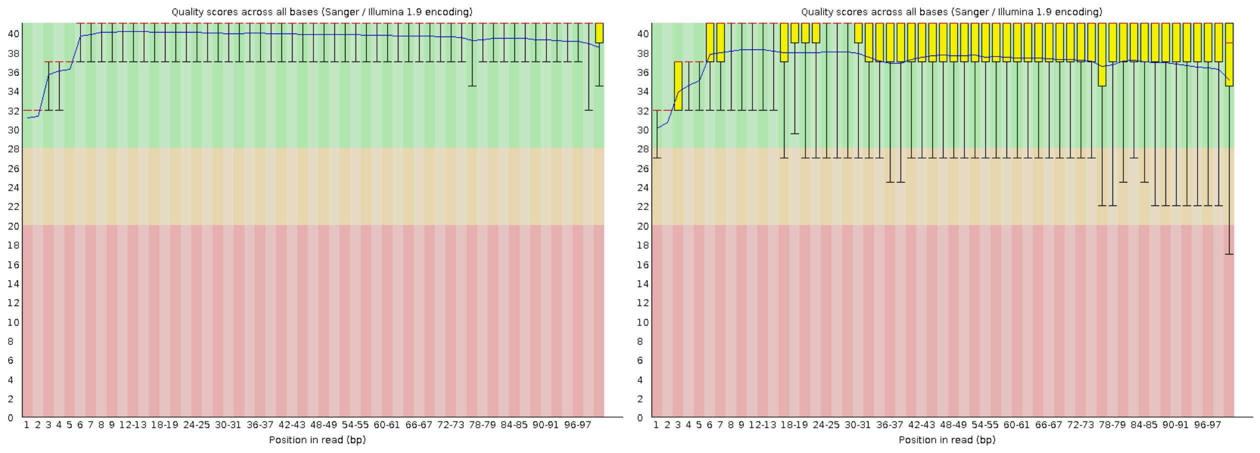
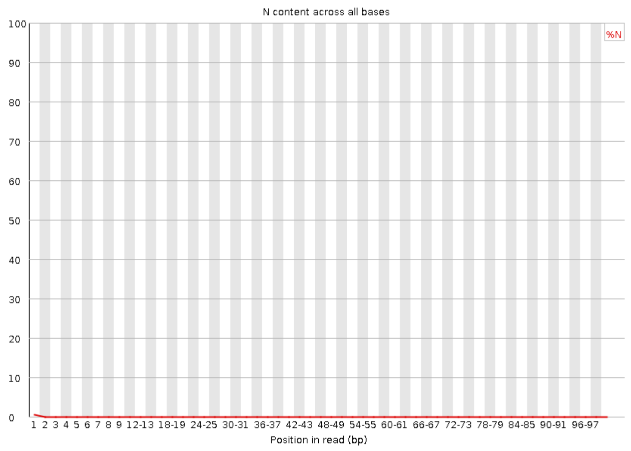


Figure 2: Per Base Sequence Quality. Distribution of average quality scores per base position for read 1 (left) and read 2 (right).



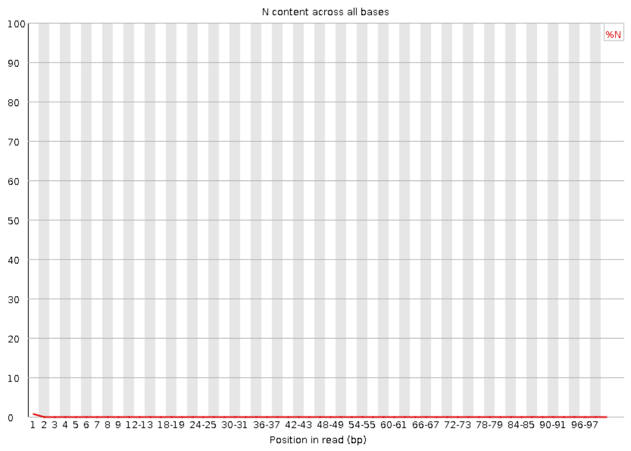
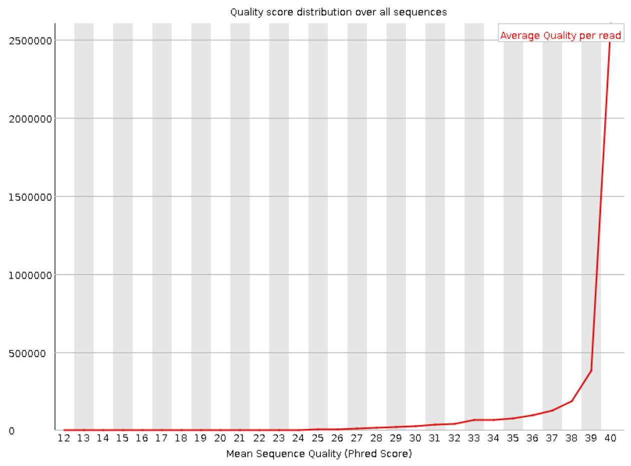


Figure 3: Per Base N Content. Distribution of average percent N content per base position for read 1 (left) and read 2 (right).

31__4F__fox__S22 FastQC Plots:



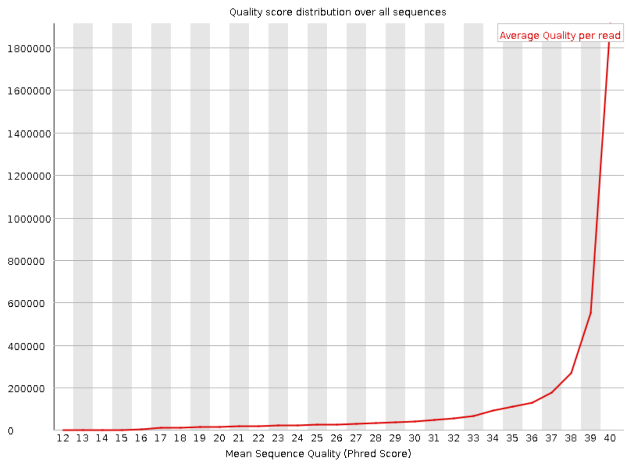


Figure 4: Per Sequence Quality Scores. Distribution of average quality scores per sequence for read 1 (left) and read 2 (right).

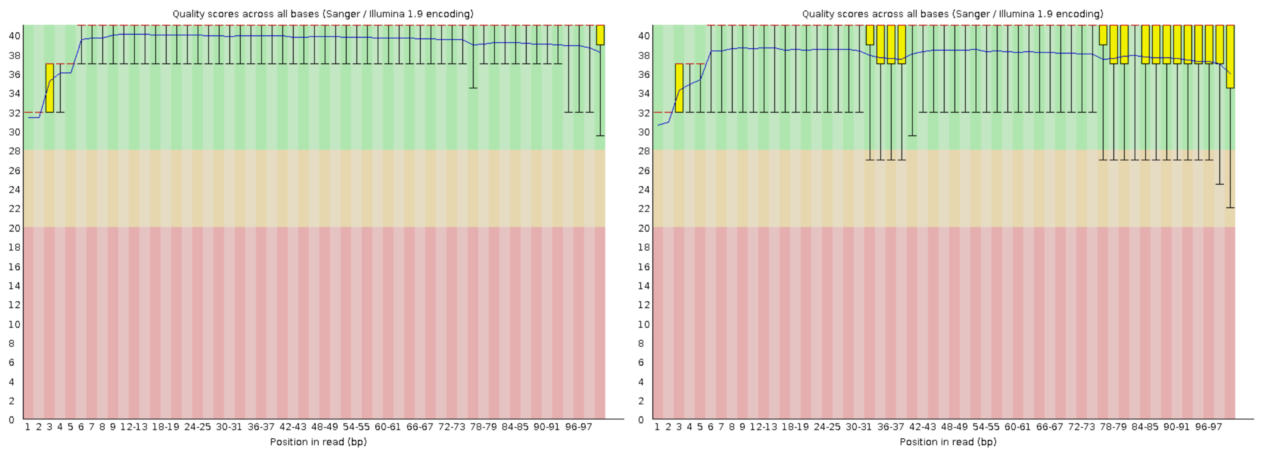


Figure 5: Per Base Sequence Quality. Distribution of average quality scores per base position for read 1 (left) and read 2 (right).

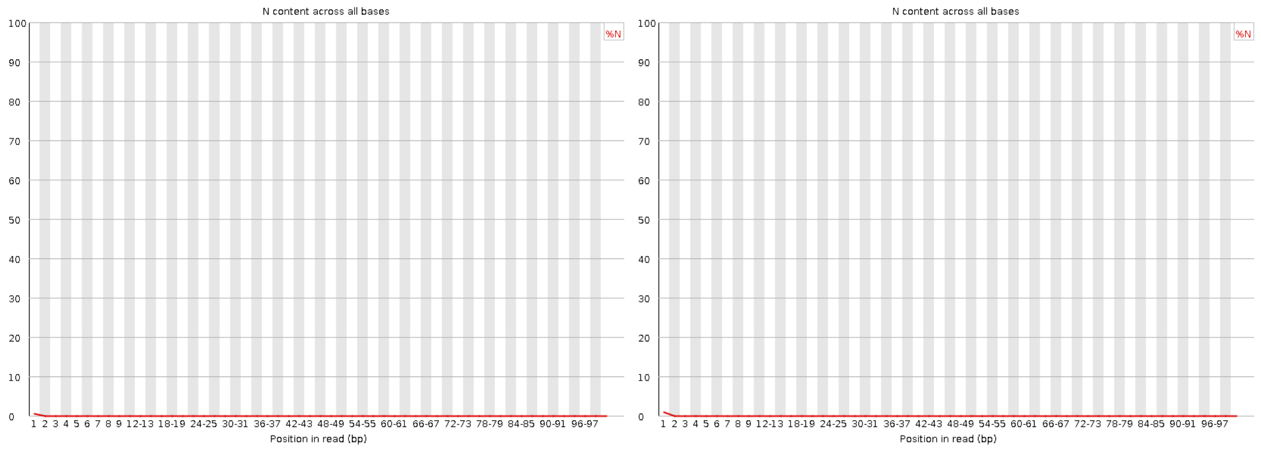


Figure 6: Per Base N Content. Distribution of average percent N content per base position for read 1 (left) and read 2 (right).

According to the per base N content plots (for both R1 and R2), there is consistently very low average N content across base positions. The first few base positions do have slighter higher N content compared to the rest of the base positions, but it appears to be at very minimal levels (<1%). According to the per base quality score plots, this increased N content in the first few bases is to be expected. The per base quality score plots show a trend toward lower quality scores in the first few base positions, which consequently could result in more uncertainty in the actual base calls and therefore a higher proportion of N calls.

2. Quality score distribution plots generated by script from demultiplexing assignment

10_2G_both_S8 my plots:

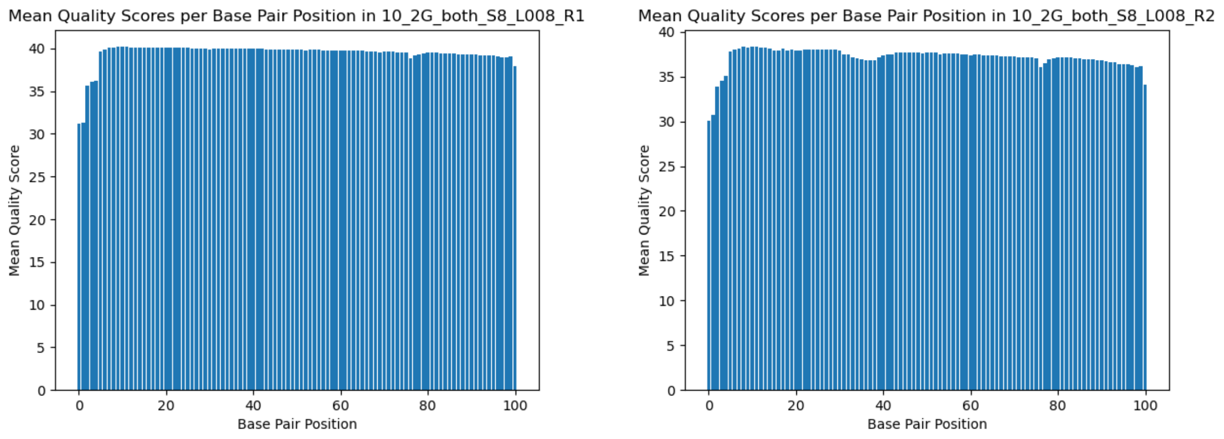


Figure 7: Per Sequence Quality Scores. Distribution of average quality scores per sequence for read 1 (left) and read 2 (right).

31_4F_fox_S22 my plots:

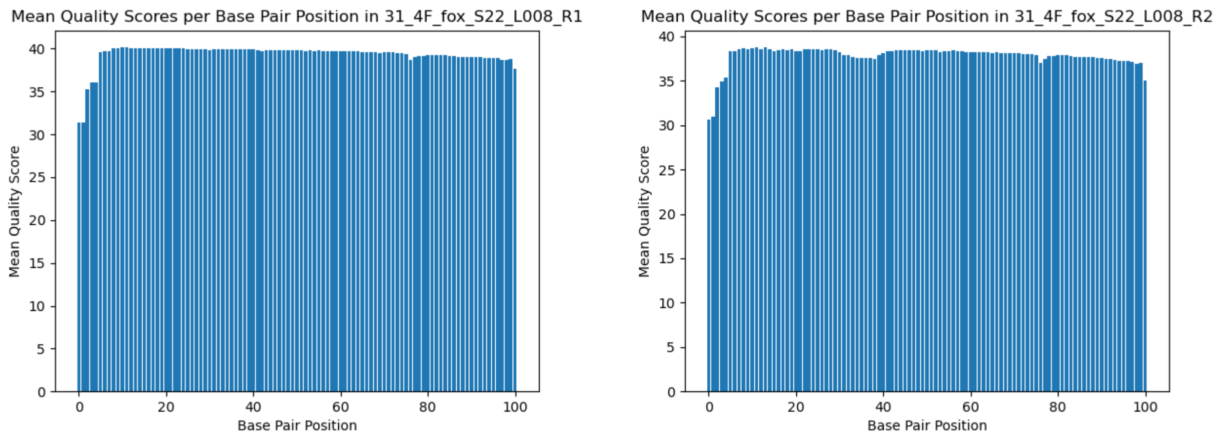


Figure 8: Per Base Sequence Quality. Distribution of average quality scores per base position for read 1 (left) and read 2 (right).

The per base quality score plots generated by FastQC are very similar to the quality score plots generated by my quality score plotting python script. Both methods generated plots that clearly show a trend of reduced quality scores in the first few base positions and are otherwise followed by consistenly high quality scores for the rest of the base positions. There is a slight reduction in quality score at the very last base position for both plots as well. For read 1 and read 2 quality score distributions, both plotting methods demonstrate overall lower quality score for read 2. This is expected due to the increased chance for degradation to occur during the time the DNA is spent on the sequencer before read 2 sequences get read.

Table 1: Run times for generating plots by fastQC and my own script

Sample	fastQC Runtime (mm:ss)	My Script Runtime (mm:ss)
10_2G_both_S8_L008_R1	8:23.42	34:57.18
10_2G_both_S8_L008_R2	8:41.50	35:24.24
31_4F_fox_S22_L008_R1	0:26.16	1:40.80
31_4F_fox_S22_L008_R2	0:26.16	1:40.44

The runtimes to generate plots by fastQC are ~3-4-fold faster compared to my quality score plot generating python script. This is likely because the code used by fastQC is better optimized than my code. Also, my code is only generating a single plot while fastQC is generating a whole report with multiple plots. One notable difference between the two is that fastQC is written in Java while my code is written in python.

3. Overall data quality of the two libraries

Based on the fastQC generated plots and the plots generated from my own quality score plotting script, the quality of both library, 10_2G_both_S8 and 31_4F_fox_S22, appears to be of high quality. The per base quality score distributions, according to both the fastQC generated plots and my own plots, show consistently high average quality scores across bases with the expected lows and the very beginning and end base positions (as mentioned above). Read 2 quality is slightly lower on average than read 1, but overall the quality of both reads is of relatively high quality. The average per sequence quality score plots from fastQC show similar results as the per base distributions and confirm that the majority of sequences have high average quality scores. Likewise, the N content distribution plots show minimal N content across base positions with the exception of the very first few bases, which is to be expected. All in all, I would say that both libraries are of high quality and can be used to proceed with trimming and further analysis.

SBATCH script used to run fastQC (fastQC.sh)

SBATCH script used to run my quality score plotting script (my_qc.sh)

Python script for generating quality score plots (qs_dist.py)

Part 2 – Adapter trimming comparison

4. Installation of cutadapt and trimmomatic in new conda environment called QAA

Creating a new conda environment:

```
conda create --name QAA python=3.9
conda activate QAA
```

Installation of cutadapt and trimmomatic:

```
conda install cutadapt
conda install trimmomatic
```

Version confirmations:

```
cutadapt --version
3.4
trimmomatic -version
0.39
```

5. Trimming of adapter sequences using cutadapt

Adapter sequences:

```
Read 1 - AGATCGGAAGAGCACACGTCTGAACTCCAGTCA
Read 2 - AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT
```

Proportion of reads trimmed:

Sample	Total read-pairs	# Read 1 Trimmed	% Read 1 Trimmed	# Read 2 Trimmed	% Read 2 Trimmed
10_2G_both_S8	81477069	2131954	2.6%	2770901	3.4%
31_4F_fox_S22	3788343	456168	12.0%	482503	12.7%

Proportion of basepairs trimmed:

Sample	# basepairs processed	# basepairs written	% basepairs written	# Read 1 basepairs	# Read 2 basepairs
10_2G_both_S8	16458367938	16434424025	99.9%	8218682322	8215741703
31_4F_fox_S22	765245286	740901113	96.8%	370502712	370398401

Running cutadapt on the 2 two samples, resulted in trimming of 2.6% of read 1 sequences and 3.4% of read 2 sequences for the 10_2G sample. For the 31_4F sample, there was 12.0% trimming of read 1 sequences and 12.7% trimming of read 2 sequences. In terms of basepairs, 99.9% of basepairs were retained for 10_2G and 96.8% of basepairs for 31_4F. In both cases, read 2 sequences were trimmed at slightly higher levels compared to read 1, but this difference is quite small (<1%).

Confirmation of adapter sequences: I used grep / wc -l to search for and determine the frequency of the provided adapter sequences in both the read 1 and read 2 files. I used the 31_4F_fox_S22 for doing this confirmation since it is the smaller of the two files. Assuming the provided adapter sequences were correct, we expect to see the R1 sequence present only in read 1 files and the read 2 sequence only in read 2 files. As show below, R1 adapter seq was only found in read 1 files (99875 times) and R2 adapter seq was only found in read 2 files (100174 times). Searching the reverse orientation of both sequences was not present at all in either read 1 or read 2 files (not show below). This is expected and confirms that the R1 adapter sequence is the sequence found at the 5' of the read 2 primer binding site. For read 1 sequences, we would want to trim off any part of the sequence that includes or comes after the R1 adapter sequence since this is where the read 2 primer binding site begins. Similarly, the R2 adapter sequence is the sequence found at the 5' of the read 1 primer binding site and we would want to trim any part of read 2 sequences that include or come after the R2 adapter sequence since this is where the read 1 primer binding site begins.

```
# searching R1 seq in read 1
zcat "/projects/bgmp/shared/2017_sequencing/demultiplexed/31_4F_fox_S22_L008_R1_001.fastq.gz" |
grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA" | wc -l
99875
# searching R1 seq in read 2
zcat "/projects/bgmp/shared/2017_sequencing/demultiplexed/31_4F_fox_S22_L008_R2_001.fastq.gz" |
grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA" | wc -l
0
# searching R2 seq in read 1
zcat "/projects/bgmp/shared/2017_sequencing/demultiplexed/31_4F_fox_S22_L008_R1_001.fastq.gz" |
grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc -l
0
# searching R2 seq in read 2
zcat "/projects/bgmp/shared/2017_sequencing/demultiplexed/31_4F_fox_S22_L008_R2_001.fastq.gz" |
grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" | wc -l
100174
```

6. Quality trimming of reads using Trimmomatic

Trimmomatic Results:

Category	# Read Pairs (10_2G)	% Input Read Pairs (10_2G)	# Read Pairs (31_4F)	% Input Read Pairs (31_4F)
Input Read Pairs	81477069	100%	3788343	100%
Both Surviving	77520904	95.14%	3597908	94.97%
Forward Only Surviving	3865386	4.74%	151048	3.99%
Reverse Only Surviving	53289	0.07%	2965	0.08%
Dropped	37490	0.05%	36422	0.96%

7. Generating plots for trimmed read length distributions

Bash commands to get read lengths after trimming to be used for plotting:

```
zcat 10_2G_both_S8_L008_R1_001_AT_Trk.fastq.gz | sed -n 2~4p | awk '{print length($0)}'
zcat 10_2G_both_S8_L008_R2_001_AT_Trk.fastq.gz | sed -n 2~4p | awk '{print length($0)}'
zcat 31_4F_fox_S22_L008_R1_001_AT_Trk.fastq.gz | sed -n 2~4p | awk '{print length($0)}'
zcat 31_4F_fox_S22_L008_R2_001_AT_Trk.fastq.gz | sed -n 2~4p | awk '{print length($0)}'
```

```r
# read in the list of read lengths
s10_R1 = read.table("./read_lengths/10_2G_both_S8_L008_R1_001_AT_Trk.txt",
                    col.names = c("Read_Length"))
s10_R2 = read.table("./read_lengths/10_2G_both_S8_L008_R2_001_AT_Trk.txt",
                    col.names = c("Read_Length"))
s31_R1 = read.table("./read_lengths/31_4F_fox_S22_L008_R1_001_AT_Trk.txt",
                    col.names = c("Read_Length"))
s31_R2 = read.table("./read_lengths/31_4F_fox_S22_L008_R2_001_AT_Trk.txt",
                    col.names = c("Read_Length"))
```

```r
# generate count distributions for each
c10_R1 = count(s10_R1, `Read_Length`)
c10_R2 = count(s10_R2, `Read_Length`)
c31_R1 = count(s31_R1, `Read_Length`)
c31_R2 = count(s31_R2, `Read_Length`)
```

```r
ggplot() +
  geom_line(data = c10_R1, aes(x = `Read_Length`, y = n, color = "Read 1")) +
  geom_line(data = c10_R2, aes(x = `Read_Length`, y = n, color = "Read 2")) +
  labs(x= "Read Length", y="Frequency (log10)",
       title="Read Length Distribution for 10_2G_both_S8", color= "") +
  scale_y_log10()

ggplot() +
  geom_line(data = c31_R1, aes(x = `Read_Length`, y = n, color = "Read 1")) +
  geom_line(data = c31_R2, aes(x = `Read_Length`, y = n, color = "Read 2")) +
  labs(x= "Read Length", y="Frequency (log10)",
       title="Read Length Distribution for 31_4F_fox_S22", color= "") +
  scale_y_log10()
```

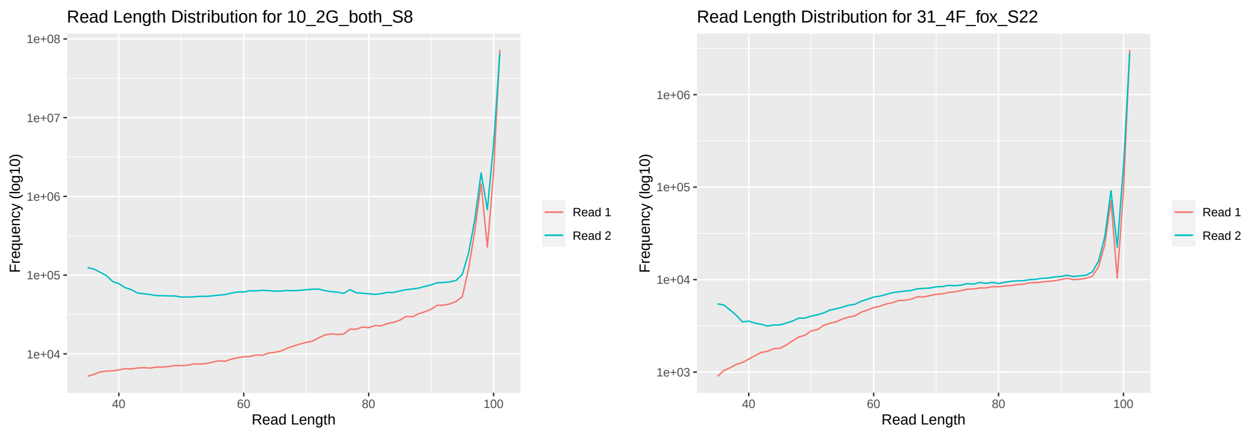


Figure 9: Read Length Distributions. Distribution of read lengths after cutadapt and trimmomatic trimming.

I would not expect there to be different levels of adapter-trimming between read 1 sequences and read 2 sequences, however I do expect there to be relatively higher levels of quality trimming for read 2 sequences compared to read 1, which is confirmed by the 2 plots above where we can see a greater number of read 2 sequences that have shorter read lengths after trimming. This is because read 2 sequencing is performed after read 1 sequencing and therefore the DNA is more likely to have degraded during this waiting period in the sequencer and would have overall lower quality compared to read 1. I don't think read 1 and read 2 should differ in their levels of adapter trimming since the main reason for adapter trimming is if the read overflows into the primer binding site of the next read. This would happen if the insert length is shorter than the read length, but both of these parameters are the same for both reads, and hence should not differ.

SBATCH script used to run cutadapt (cutadapt.sh)

SBATCH script used to run trimmomatic (trimmomatic.sh)

Part 3 – Alignment and strand-specificity

8. Conda installations in QAA environment

```
conda install star
conda install numpy
conda install pysam
conda install matplotlib
pip install HTSeq
```

9. Performing alignment of reads to mouse genomic database using STAR aligner.

Downloaded the following fasta and gtf files from ensembl for mouse:

```
http://ftp.ensembl.org/pub/release-104/fasta/mus_musculus/dna/Mus_musculus.GRCm39.dna.primary_assembly.
http://ftp.ensembl.org/pub/release-104/gtf/mus_musculus/Mus_musculus.GRCm39.104.gtf.gz
```

Steps:

1. Build a reference database with STAR using the above ensembl files

2. Align RNA-seq reads to the reference database
3. Perform counts for mapped and unmapped reads

Make a directory to contain the STAR database:

```
Name of assembly: Mus_musculus.GRCm39.dna
Release of Ensembl: ens104
Version of STAR: 2.7.9a
$ mkdir Mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a
```

SBATCH script used to build STAR database (buildSTAR.sh)

SBATCH script used perform alignment of trimmed RNA-seq reads to STAR database (align-STAR_10_2G.sh)

SBATCH script used perform alignment of trimmed RNA-seq reads to STAR database (align-STAR_31_4F.sh)

10. Number of mapped and unmapped reads in SAM files using my python script from PS8

Sample	Mapped	Unmapped
10_2G_both_S8	152719556	2322252
31_4F_fox_S22	6969878	225938

Python script used to count mapped and unmapped reads in SAM file (count_mapped.py)

11. Counting reads that map to features using htseq-count

```
htseq-count --version
0.13.5
```

Commands to get total # reads and # that mapped to a feature:

```
$ awk '{sum+=$2} END {print sum}' 31_4F_stranded.genecount
$ grep -v '^__' 31_4F_stranded.genecount | awk '{sum+=$2} END {print sum}'
$ grep -v '^__' 31_4F_unstranded.genecount | awk '{sum+=$2} END {print sum}'

$ awk '{sum+=$2} END {print sum}' 10_2G_stranded.genecount
$ grep -v '^__' 10_2G_stranded.genecount | awk '{sum+=$2} END {print sum}'
$ grep -v '^__' 10_2G_unstranded.genecount | awk '{sum+=$2} END {print sum}'
```

Category	Count [10_2G stranded]	Count [10_2G unstranded]	Count [31_4F stranded]	Count [31_4F unstranded]
mapped	2900936	65025604	180499	2922418
no_feature	69853236	3771976	3142921	237215
ambiguous	48198	4004790	3343	167130
too_low_aQual	136867	136867	5281	5281
not_aligned	1088394	1088394	110170	110170
alignment_not_unique	3493273	3493273	155694	155694

Category	% [10_2G stranded]	% [10_2G unstranded]	% [31_4F stranded]	% [31_4F unstranded]
mapped	3.74	83.88	5.02	81.23
no_feature	90.11	4.87	87.35	6.59
ambiguous	0.06	5.17	0.09	4.65
too_low_aQual	0.18	0.18	0.15	0.15
not_aligned	1.40	1.40	3.06	3.06
alignment_not_unique	4.51	4.51	4.33	4.33

12. Strand-specificity of RNA-Seq libraries According to the htseq-count data presented above, I propose that the two libraries (10_2G and 31_4F) are strand-specific. When running htseq-count with the stranded parameter set to yes, we only see 3.74% of reads mapping to features for sample 10_2G and 5.02% of reads for sample 31_4F. On the other hand, running htseq-count with the stranded parameter set to no, we see 83.88% reads mapping to features for 10_2G and 81.23% for 31_4F. Although we do see a higher percentage of mapping when stranded=no, this does not necessarily mean that the libraries are unstranded. If the libraries were actually unstranded, we would expect to see about half the amount of mapping with stranded=yes compared to stranded=no because we would expect an equal amount of mapping to each strand. However, we see a very low proportion of reads mapping with stranded=yes. This leads me to believe that the libraries are actually stranded, but the opposite strand is being read with stranded=yes. To confirm this, htseq-count can be run with stranded=reverse, which should should show similar mapping counts to that of unstranded.

SBATCH script used to run htseq-count (htseq_10_2G.sh)

SBATCH script used to run htseq-count (htseq_31_4F.sh)