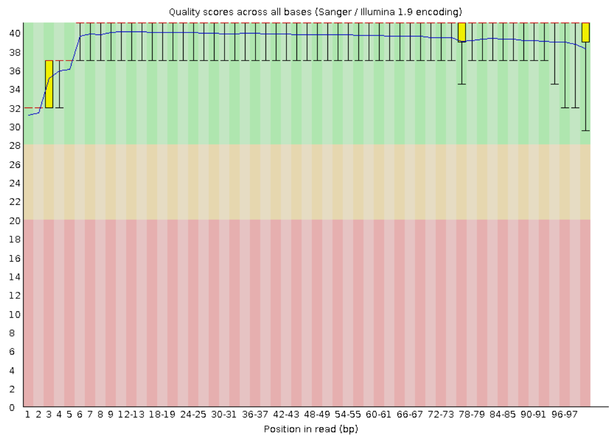
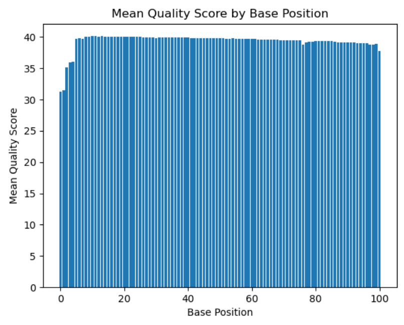
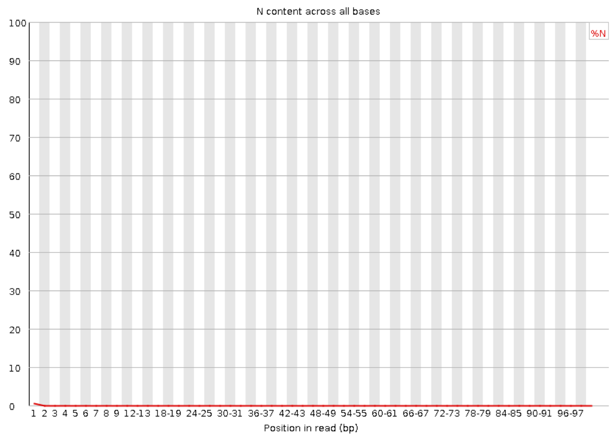
QAA_Sam_Kupp

2022-09-07

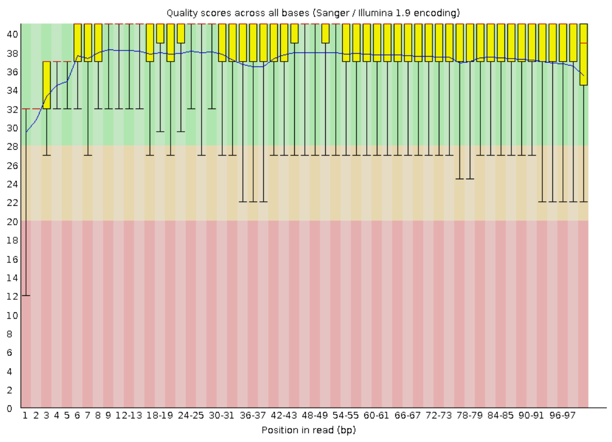
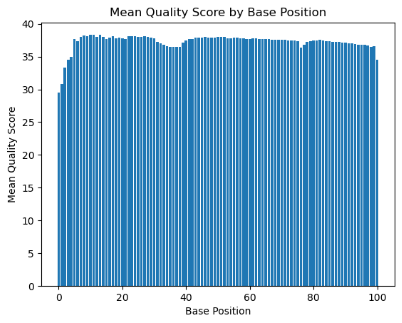
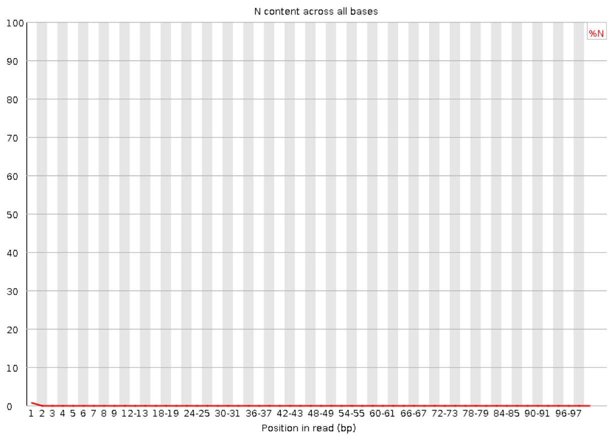
Part 1- Read Quality Score Distributions

Quality Score Distributions from FASTQC

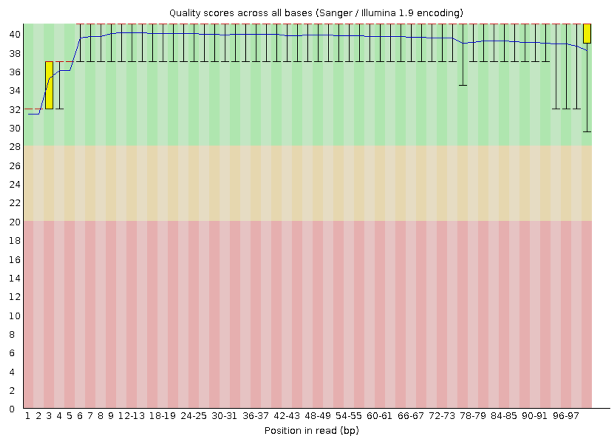
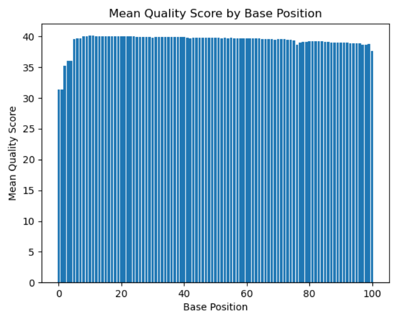
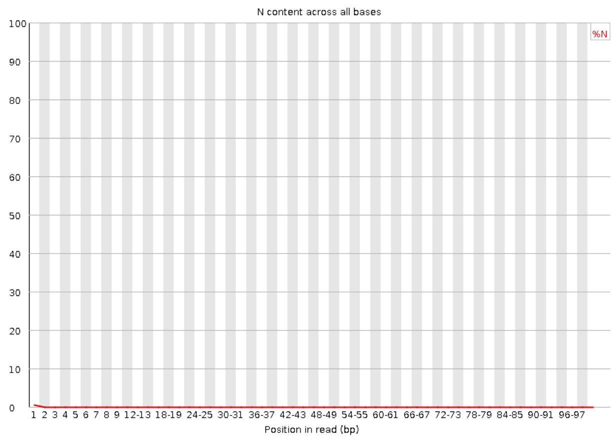
23_4A_control_S17_L008 Read 1

	FASTQC Output	Python Output
Mean Quality Distribution		
Per Base N Content		

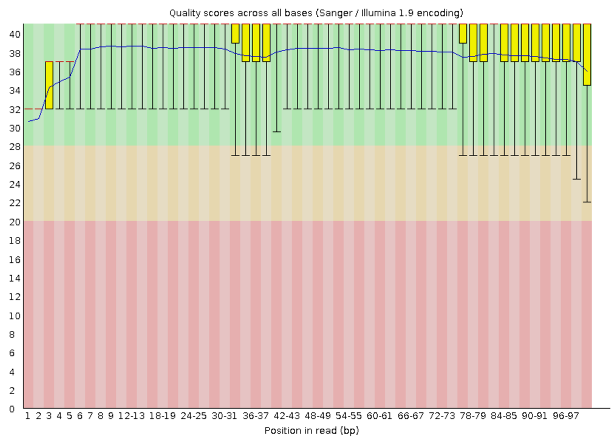
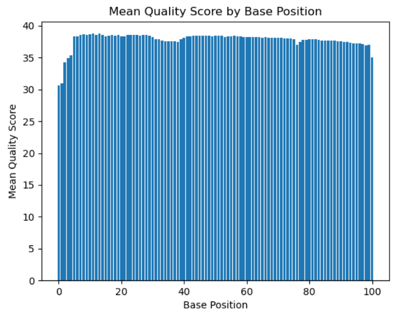
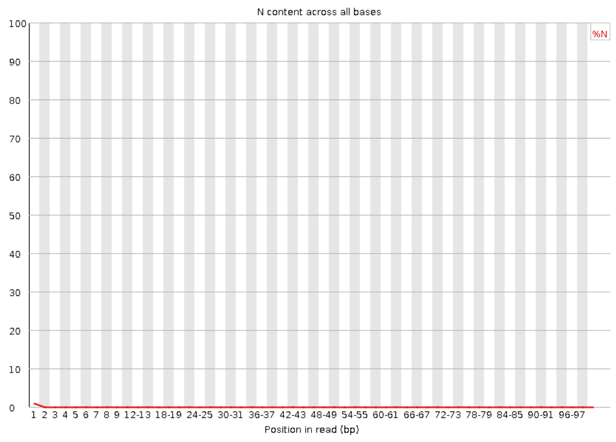
23_4A_control_S17_L008 Read 2

	FASTQC Output	Python Output
Mean Quality Distribution		
Per Base N Content		

31_4F_fox_S22_L008 Read 1

	FASTQC Output	Python Output
Mean Quality Distribution		
Per Base N Content		

31_4F_fox_S22_L008 Read 2

	FASTQC Output	Python Output
Mean Quality Distribution		
Per Base N Content		

The per base n content plots are consistent with the per base quality plots. In each of the 4 files, the only base position with a significant proportion of reads that are 'N', is the first. This is also the position with the lowest average quality.

As expected, the score distributions from FASTQC and the Python script are identical, with any visual discrepancies likely due to the plotting geometry of bar vs line graph.

Run Times for Quality Distribution Plots

FASTQ File	Python Script Run Time (s)	FASTQC Run Time (s)
23_4A_control_S17_L008 Read 1	1376.91	166.47
23_4A_control_S17_L008 Read 2	1408.47	166.57
31_4F_fox_S22_L008 Read 1	115.9	16.56
31_4F_fox_S22_L008 Read 2	117.31	17.21

As can be seen from the table, FASTQC is much faster than my Python Script. This is likely at least in part because Python is an extremely flexible, but slow language. It is also possible that unzipping protocol that FASTQC uses is more efficient than gzip in Python.

The library qualities are as generally expected. Each of the first reads is relatively low quality, the later ~80 reads are between 37-40 quality, but taper towards the later reads in the file. Additionally, the reverse reads in each file set are lower quality as the DNA has likely degraded before these reads were sequenced.

Part 2- Adapter trimming comparison

Cut Adapt

CutAdapt Percent of Reads Trimmed

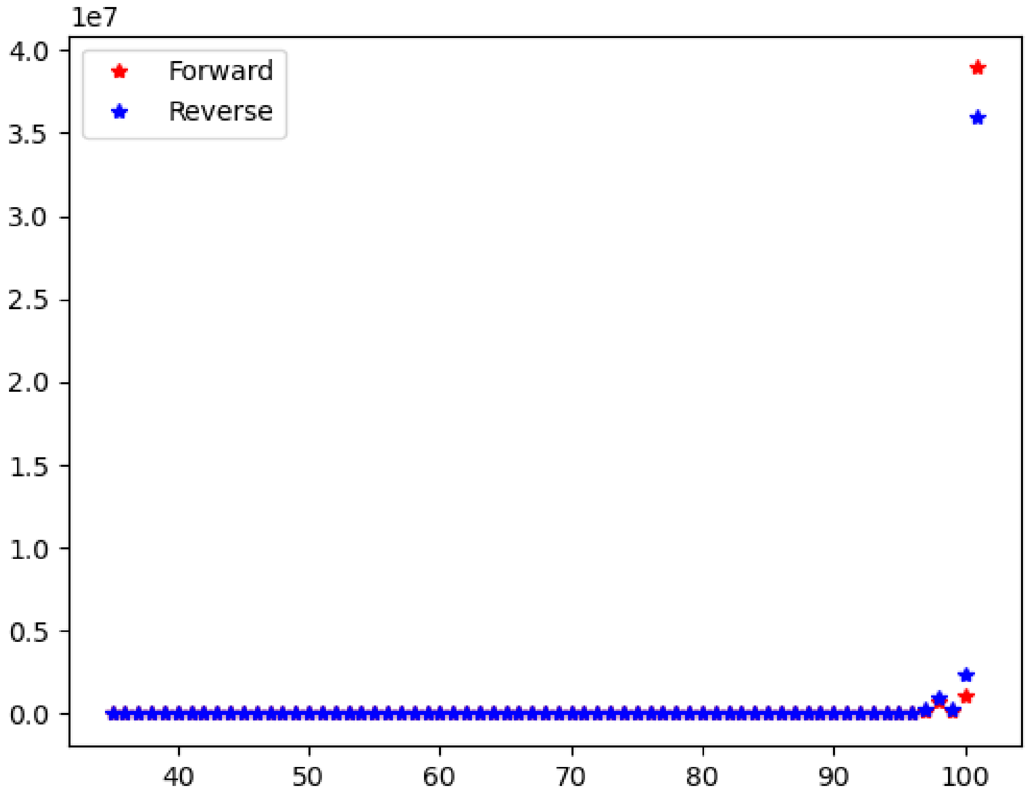
FASTQ File Pair	Percent of Filtered Base Pairs
23_4A_control_S17_L008	96.8
31_4F_fox_S22_L008	99.7

```
Adapter 1 Check
zcat 23_4A_control_S17_L008_R1_001.fastq.gz | head -100000 |grep -c "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
29
zcat 23_4A_control_S17_L008_R2_001.fastq.gz | head -100000 |grep -c "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
0

zcat 31_4F_fox_S22_L008_R1_001.fastq.gz | head -100000|grep -c "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
629
zcat 31_4F_fox_S22_L008_R2_001.fastq.gz | head -100000|grep -c "AGATCGGAAGAGCACACGTCTGAACTCCAGTCA"
0

Adapter 2 Check
zcat 23_4A_control_S17_L008_R1_001.fastq.gz | head -100000 |grep -c "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
0
zcat 23_4A_control_S17_L008_R2_001.fastq.gz | head -100000 |grep -c "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
29

zcat 31_4F_fox_S22_L008_R1_001.fastq.gz | head -100000| grep -c "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
0
zcat 31_4F_fox_S22_L008_R2_001.fastq.gz | head -100000| grep -c "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT"
630
```

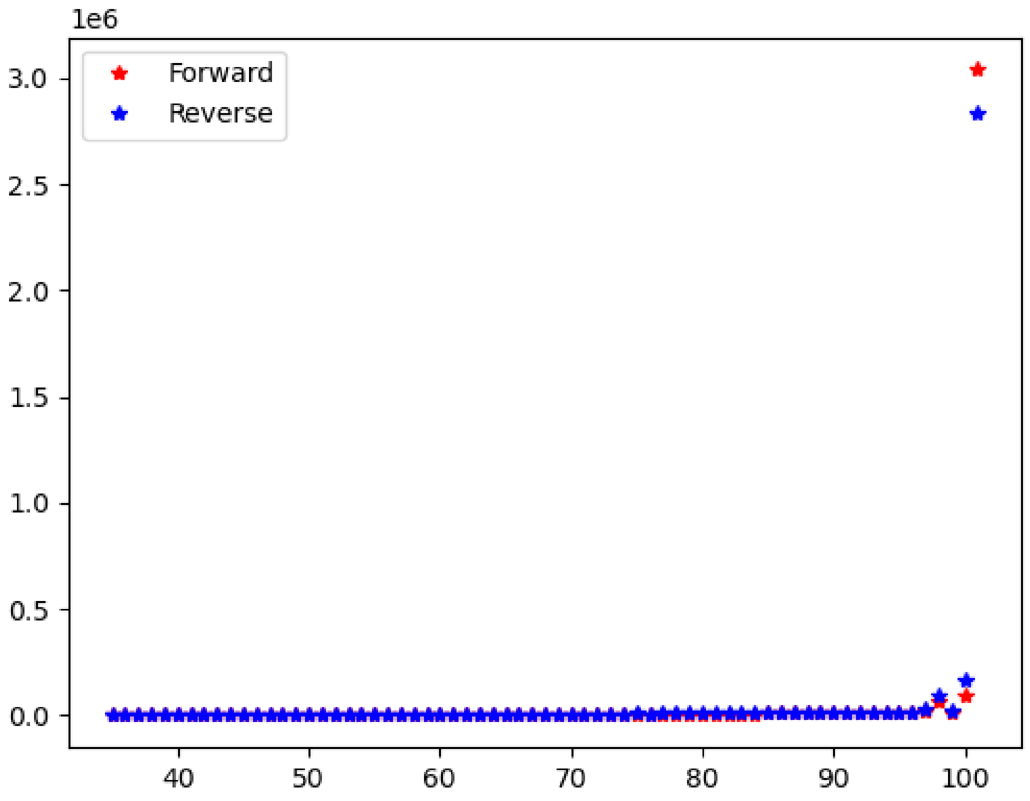
The above commands were used to search for each adapter sequence in each of the read files. Here the first 25000 sequences were examined. As expected adapter sequence 1 was found only in read 1 and adapter sequence 2 was found only in read 2. This confirms that the reads were properly oriented.

Trimmomatic

23_4A_control_S17_L008 Trimmed Sequence Lengths



31_4F_fox_S22_L008 Trimmed Sequence Lengths



As expected, the forward reads exhibit more reads at 101 base pairs than the reverse read. This is likely due to degradation of the DNA on the sequencer and the quality trimming done during Trimmomatic. Both distributions are heavily peaked at 101 base pairs, indicating that both reads are high quality.

Part 3- Alignment and strand-specificity

After building a STAR database and aligning the trimmed reads to it, the results are as shown below:

Mapped Reads from Python Script

Read File	Number of Reads Mapped	Number of Reads Not Mapped
23_4A_control_S17_L008	79473028	4640124
31_4F_fox_S22_L008	6969880	225936

HTSeq Counts Output

Mapped Reads from HTSeq

Count File	Number of Reads Mapped	Total Number of Reads	Percentage of Reads Mapped
23_4A_control_S17_L008 stranded=reverse	65634193	81286685	80.7441
23_4A_control_S17_L008 stranded=yes	2261967	81286685	2.7827
31_4F_fox_S22_L008 stranded=reverse	5691153	6695862	84.9951
31_4F_fox_S22_L008 stranded=yes	291122	6695862	4.34779

Reads Mapped, Total Number of Reads and Percentage of Reads Mapped Calculated with:

```
for file in `ls ~/Documents/BI_622/QAA/*.tsv`; do echo $file ; awk '{sum1 += $2}\
$1~"ENSMU"{sum2 += $2} END {print  sum2 "\t" sum1 "\t" sum2/sum1*100}' $file ; done
```

```
## /Users/sam/Documents/BI_622/QAA/23_4A_control_S17_L008.reverse.count.tsv
## 65634193 81286685    80.7441
## /Users/sam/Documents/BI_622/QAA/23_4A_control_S17_L008.stranded.count.tsv
## 2261967  81286685    2.7827
## /Users/sam/Documents/BI_622/QAA/31_4F_fox_S22_L008.reverse.count.tsv
## 5691153  6695862 84.9951
## /Users/sam/Documents/BI_622/QAA/31_4F_fox_S22_L008.stranded.count.tsv
## 291122   6695862 4.34779
```

From the percentage of reads mapped, we can determine that the library is strand specific. Because roughly 80% of reads are mapped when stranded=reverse in the HTSeq command but only less than 5% are mapped when stranded=yes. This means that a specific strand was sequenced, not both as in an unstranded library prep.