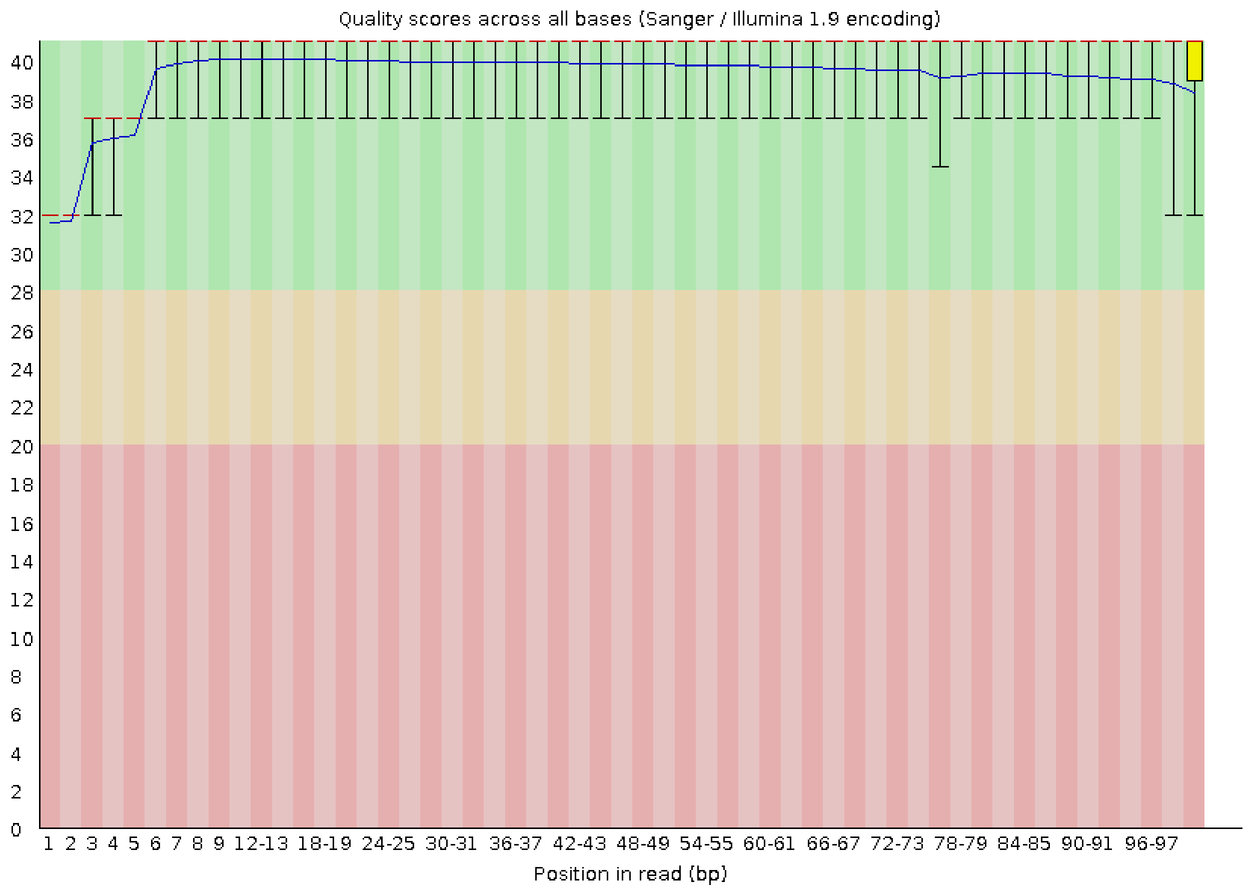
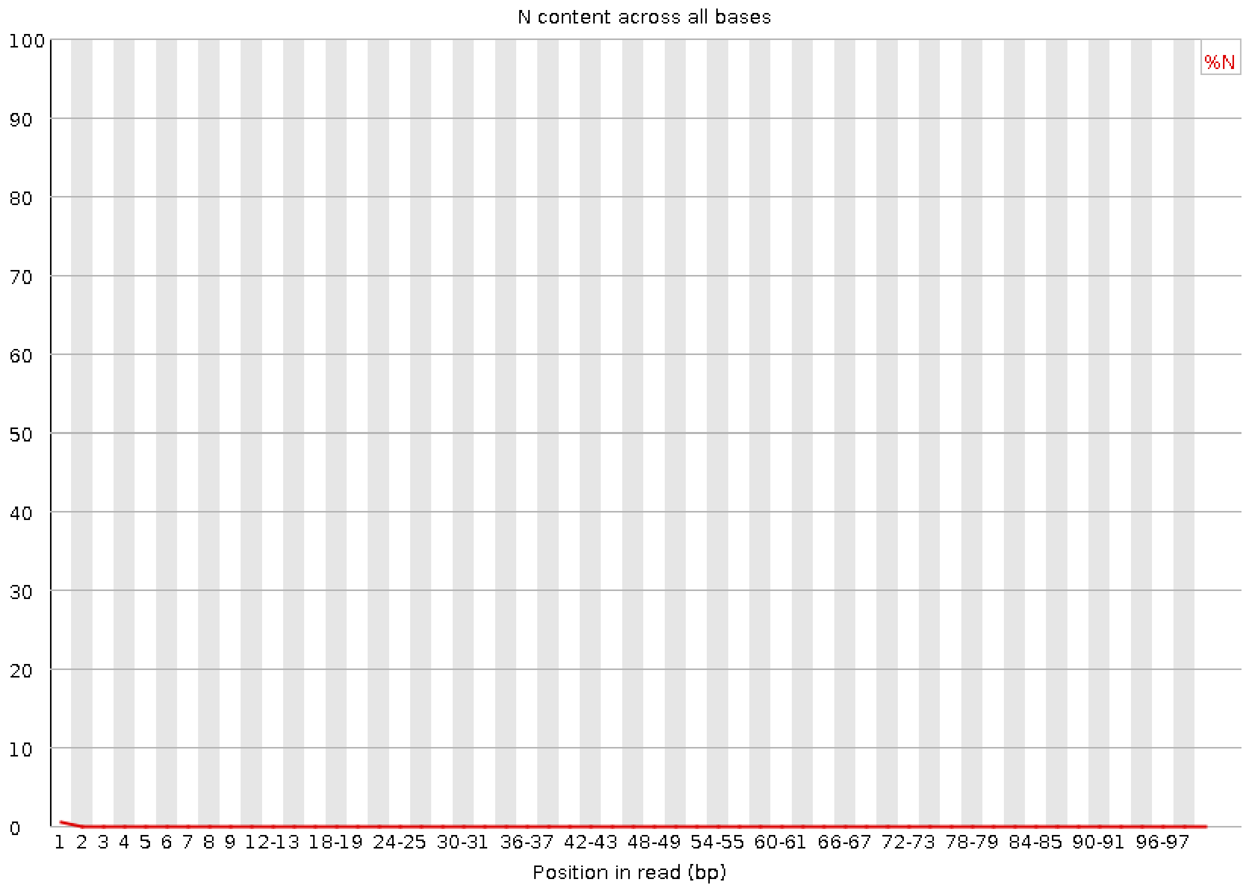
QAA

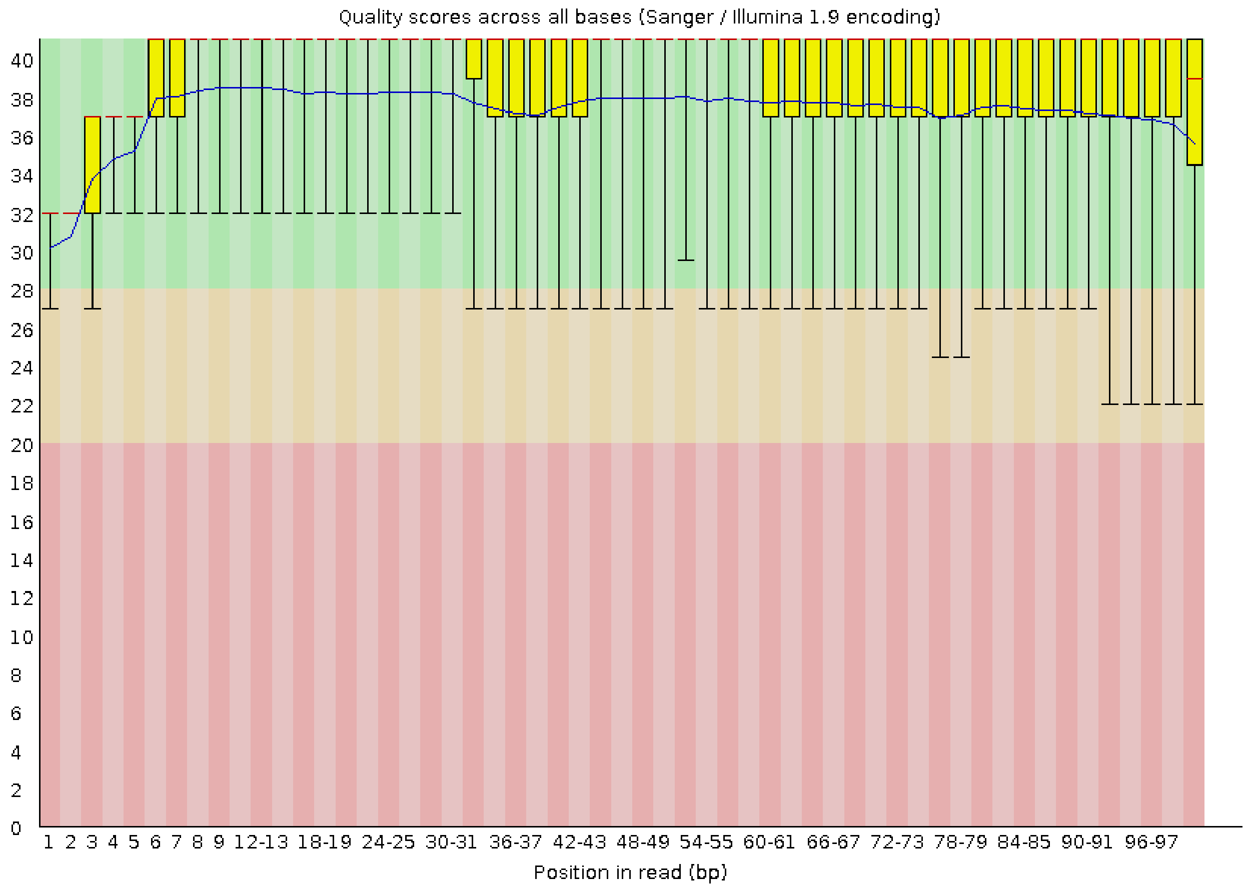
Nelson Ruth
QAA
Part 1: Read quality score distributions:
I used FASTQC (v0.11.5).



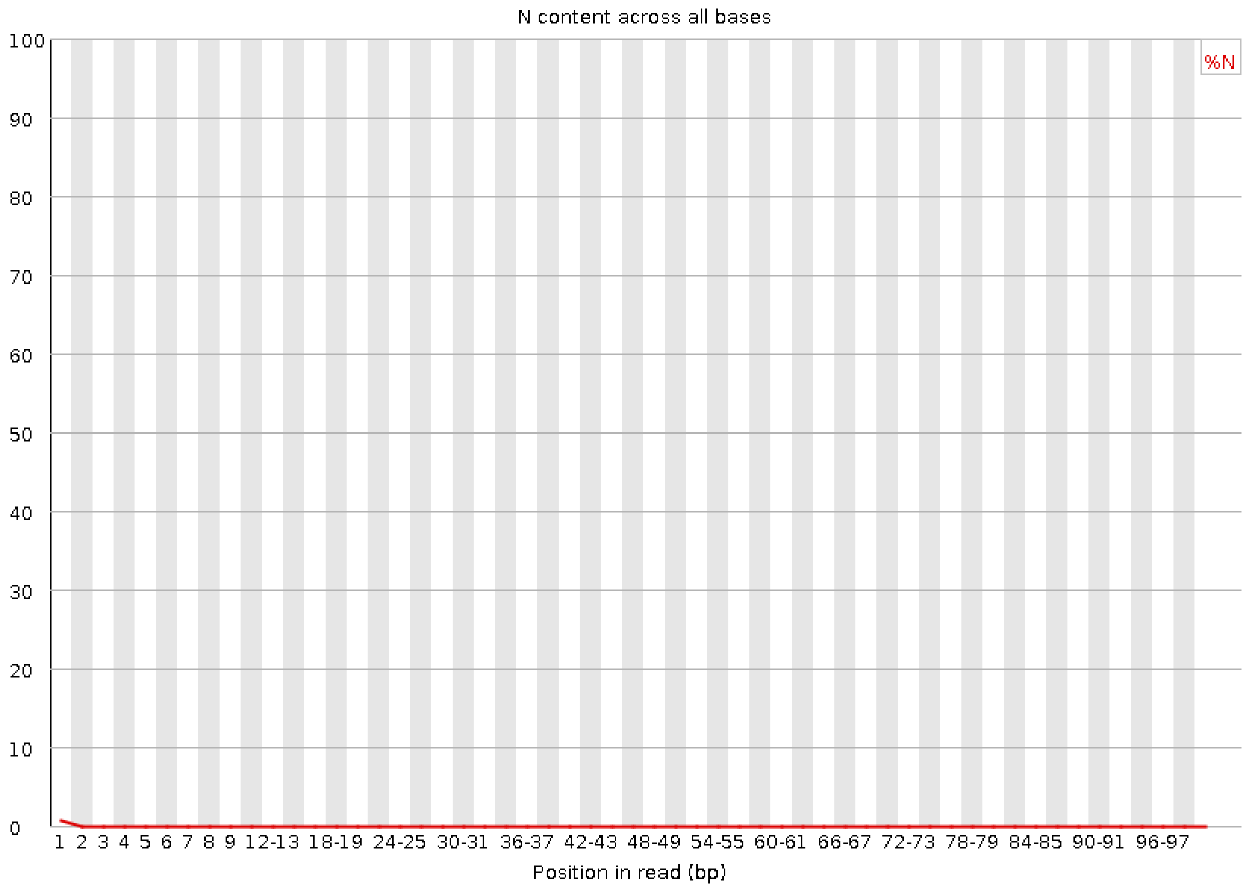
FASTQC graph showing per base sequence quality of 8_2F_fox_S7_L008_R1_001



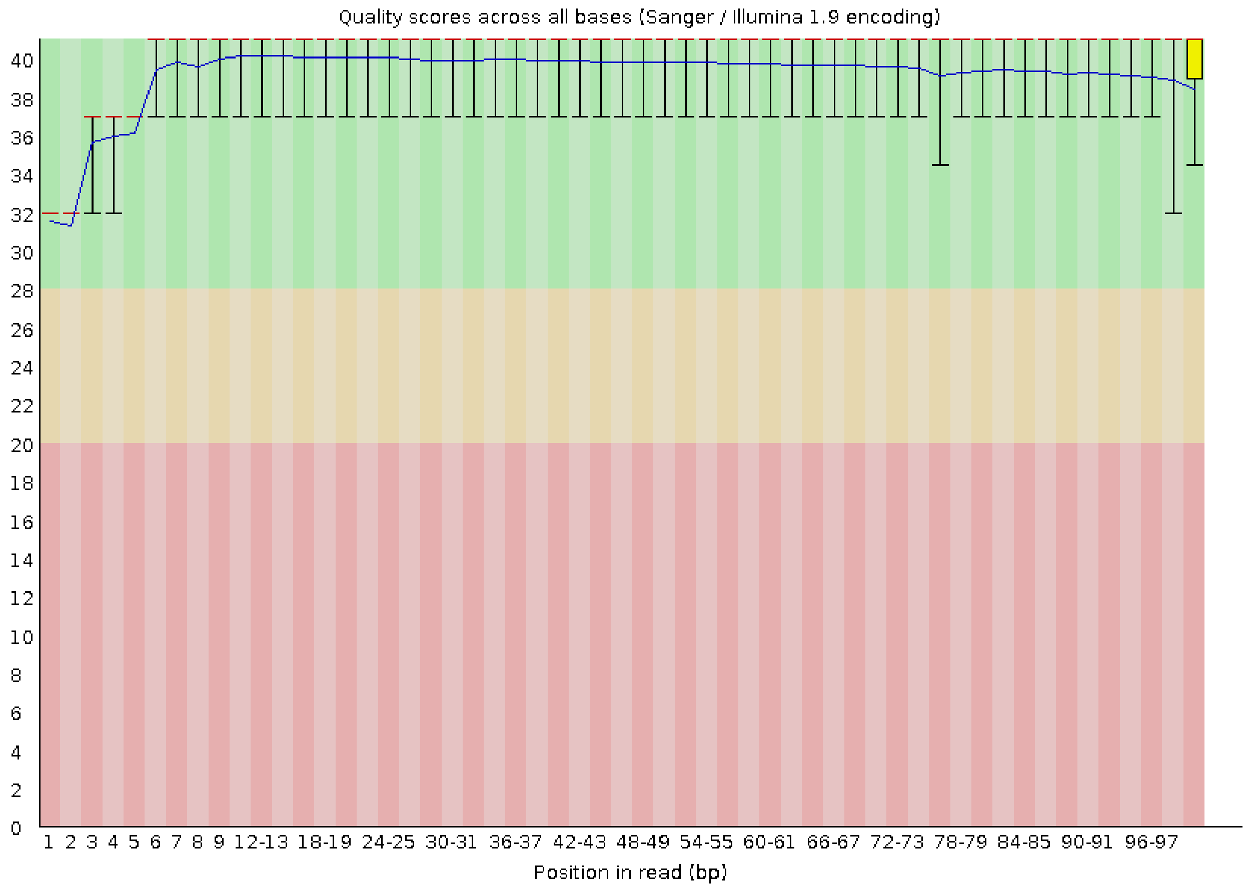
FASTQC graph showing N distribution of 8_2F_fox_S7_L008_R1_001



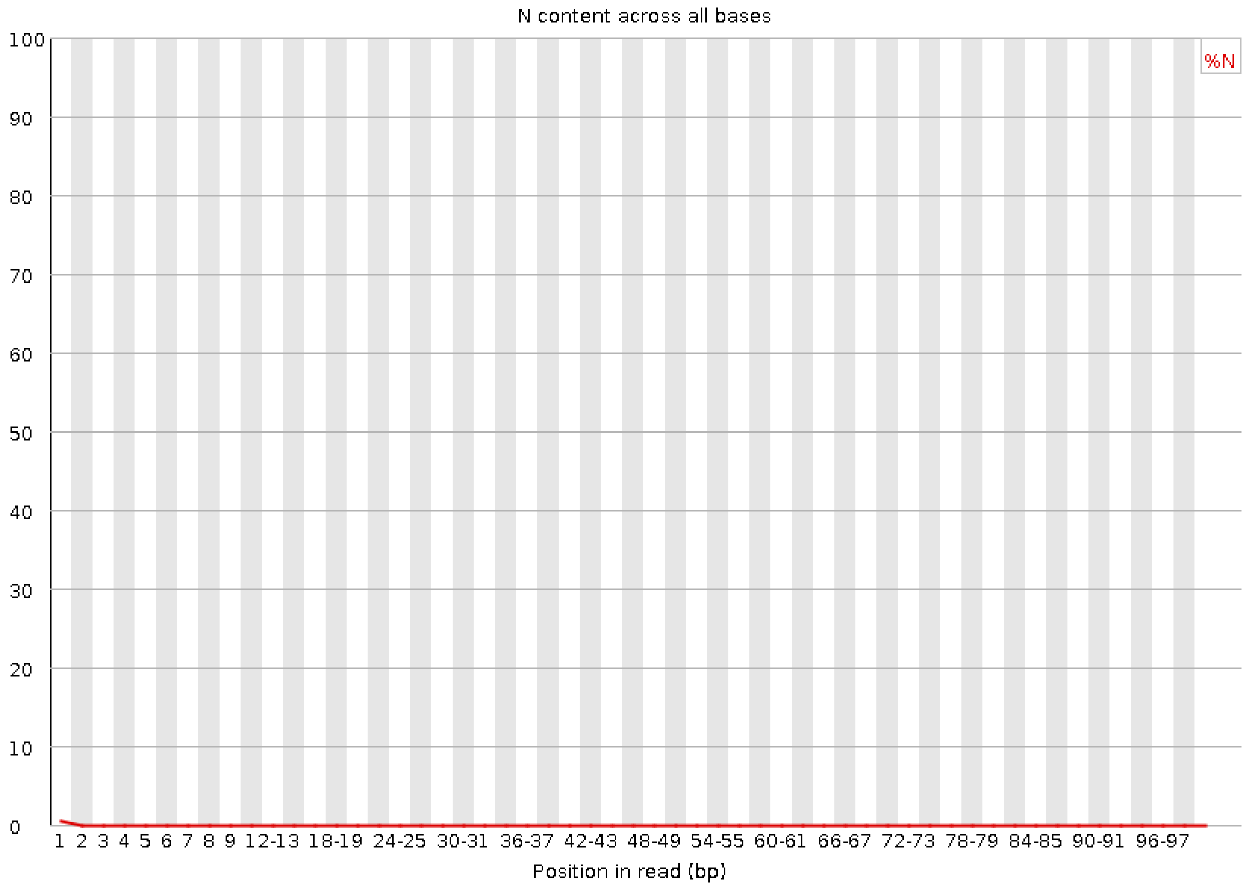
FASTQC graph showing per base sequence quality of 8_2F_fox_S7_L008_R2_001



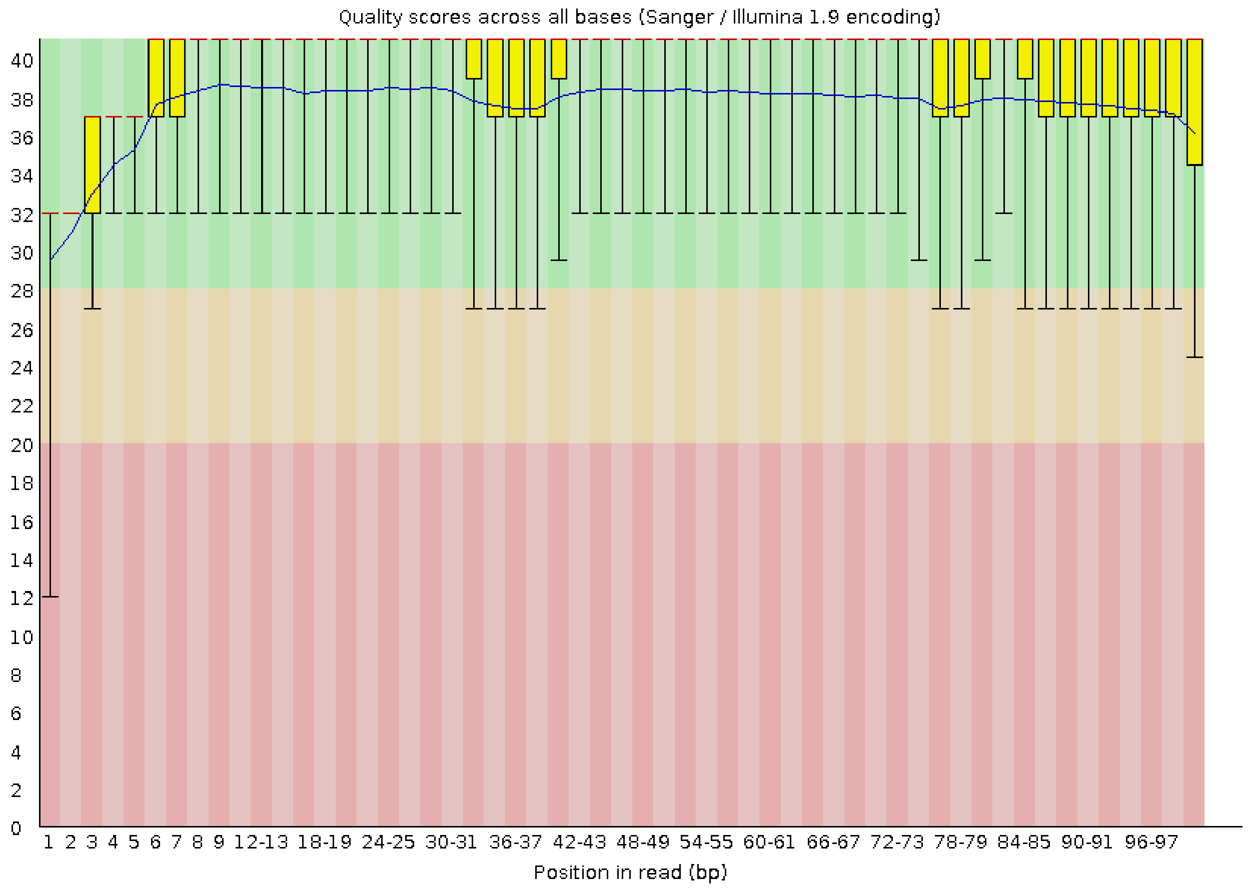
FASTQC graph showing N distribution of 8_2F_fox_S7_L008_R2_001



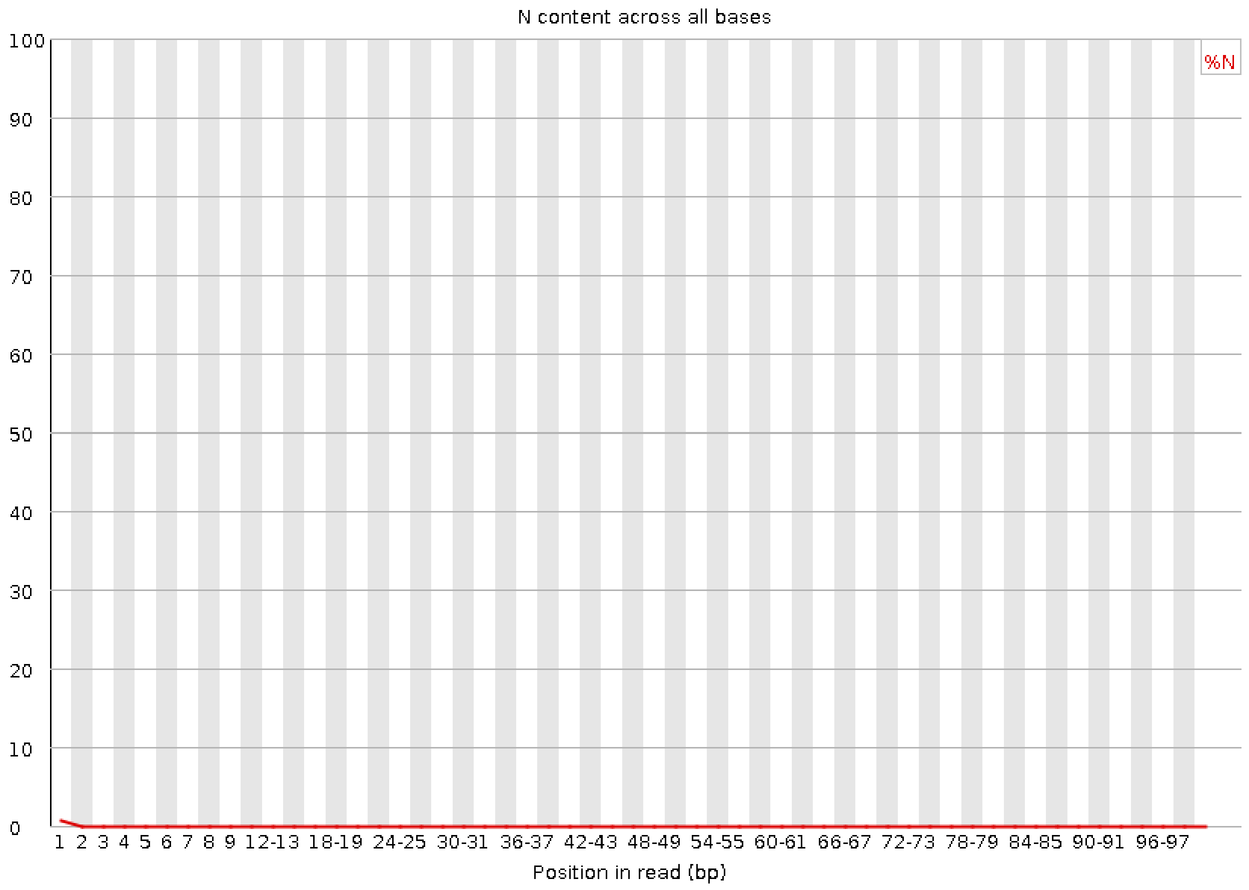
FASTQC graph showing per base sequence quality of 14_3B_control_S10_L008_R1_001



FASTQC graph showing N distribution of 14_3B_control_S10_L008_R1_001



FASTQC graph showing per base sequence quality of 14_3B_control_S10_L008_R2_001



FASTQC graph showing N distribution of 14_3B_control_S10_L008_R2_001

Answers:
1) For all the per base N content, we see a slight increase at the very start, then they stay at 0 for the rest of the sequence. The R1 files have slightly lower level of N's at the start, likely due to their overall higher per base sequence quality. The R2 files have a lower per base sequence quality so I would expect them to have a slightly higher amount of N's, but this is not the case.
2) The distributions look the same. To run my DM script on both R1 and R2 it took 47 minutes and the FASTQC 1 min 36 sec. This massive difference could be due to inherent differences in Python vs Java, professional programmers vs students, or slow code written by me.
3) The overall quality looks good, as expected R1 files are higher quality than R2 due to R2 being on the sequencer for longer and having more time to degrade. The higher amount of N at the starting position of each read could be caused by mismatches between the primer or calibration steps for the sequencing machine.

Part 2: Adapter trimming comparison

Cutadapt: -Below are my commands using cutadapt (Version 3.4, Build py39h38f01e4_1) to trim adapter sequences.

```
#for 8_2F_fox_S7_L008 files
cutadapt -b AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -B AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -o /home/ndr/bgmp/bi
```

=== Summary ===

Total read pairs processed: 36,482,601
Read 1 with adapter: 2,778,535 (7.6%)
Read 2 with adapter: 2,877,524 (7.9%)
Pairs written (passing filters): 36,482,601 (100.0%)

Total basepairs processed: 7,369,485,402 bp
Read 1: 3,684,742,701 bp
Read 2: 3,684,742,701 bp
Total written (filtered): 7,311,697,532 bp (99.2%)
Read 1: 3,656,033,391 bp
Read 2: 3,655,664,141 bp

```
#for 14_3B_control_S10_L008 files
cutadapt -b AGATCGGAAGAGCACACGTCTGAACTCCAGTCA -B AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT -o /home/ndr/bgmp/bi
```

=== Summary ===

Total read pairs processed: 4,440,378
Read 1 with adapter: 336,435 (7.6%)
Read 2 with adapter: 357,683 (8.1%)
Pairs written (passing filters): 4,440,378 (100.0%)

Total basepairs processed: 896,956,356 bp
Read 1: 448,478,178 bp
Read 2: 448,478,178 bp
Total written (filtered): 888,894,806 bp (99.1%)
Read 1: 444,481,143 bp
Read 2: 444,413,663 bp

Sanity Check: I used `zgrep --color adapter filename` to confirm the orientations of R1 and R2 in my files. To start, I looked for R1 in the R1 files and saw them in every line towards the 3' end. I repeated this for R2 in the R2 files and saw the adapter in every line towards the 3' end once again. I then swapped and looked for R2 in the R1 files and didn't see the adapter, and didn't see R1 in the R2 files either. Next, I checked for the adapters in the cutadapt output files to ensure I used cutadapt properly, and saw no adapter sequences, confirming the adapters were properly trimmed.

Trimmomatic:
-Below are my commands using Trimmomatic (Version 0.39, Build hdfd78af_2) to quality trim the reads.
Settings:

LEADING: quality of 3
TRAILING: quality of 3
SLIDING WINDOW: window size of 5 and required quality of 15
MINLENGTH: 35 bases

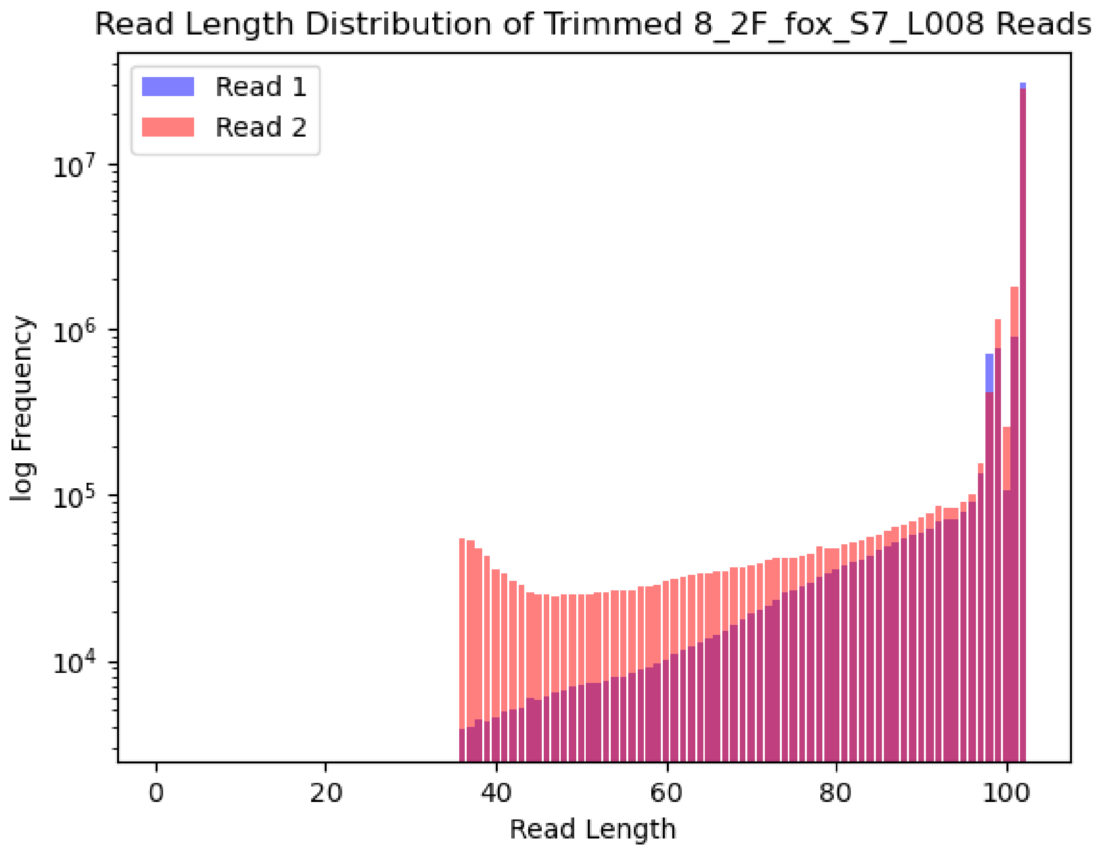
```
#for 8_2F_fox_S7_L008 samples
trimmomatic PE -phred33 cut_8_2F_fox_S7_L008_R1_001_fastqc.fq cut_8_2F_fox_S7_L008_R2_001_fastqc.fq -ba
TrimmomaticPE: Started with arguments:
 -phred33 cut_8_2F_fox_S7_L008_R1_001_fastqc.fq cut_8_2F_fox_S7_L008_R2_001_fastqc.fq -baseout 8_2F_fox

Using templated Output files: 8_2F_fox_S7_L008_trimmed_1P.fq 8_2F_fox_S7_L008_trimmed_1U.fq 8_2F_fox_S7
Input Read Pairs: 36482601 Both Surviving: 34798153 (95.38%) Forward Only Surviving: 1631451 (4.47%) Re
TrimmomaticPE: Completed successfully

#For cut_14_3B_control_S10_L008
trimmomatic PE -phred33 cut_14_3B_control_S10_L008_R1_001_fastqc.fq cut_14_3B_control_S10_L008_R2_001_f

TrimmomaticPE: Started with arguments:
 -phred33 cut_14_3B_control_S10_L008_R1_001_fastqc.fq cut_14_3B_control_S10_L008_R2_001_fastqc.fq -base

Using templated Output files: 14_3B_control_S10_L008_trimmed_1P.fq 14_3B_control_S10_L008_trimmed_1U.fq
```

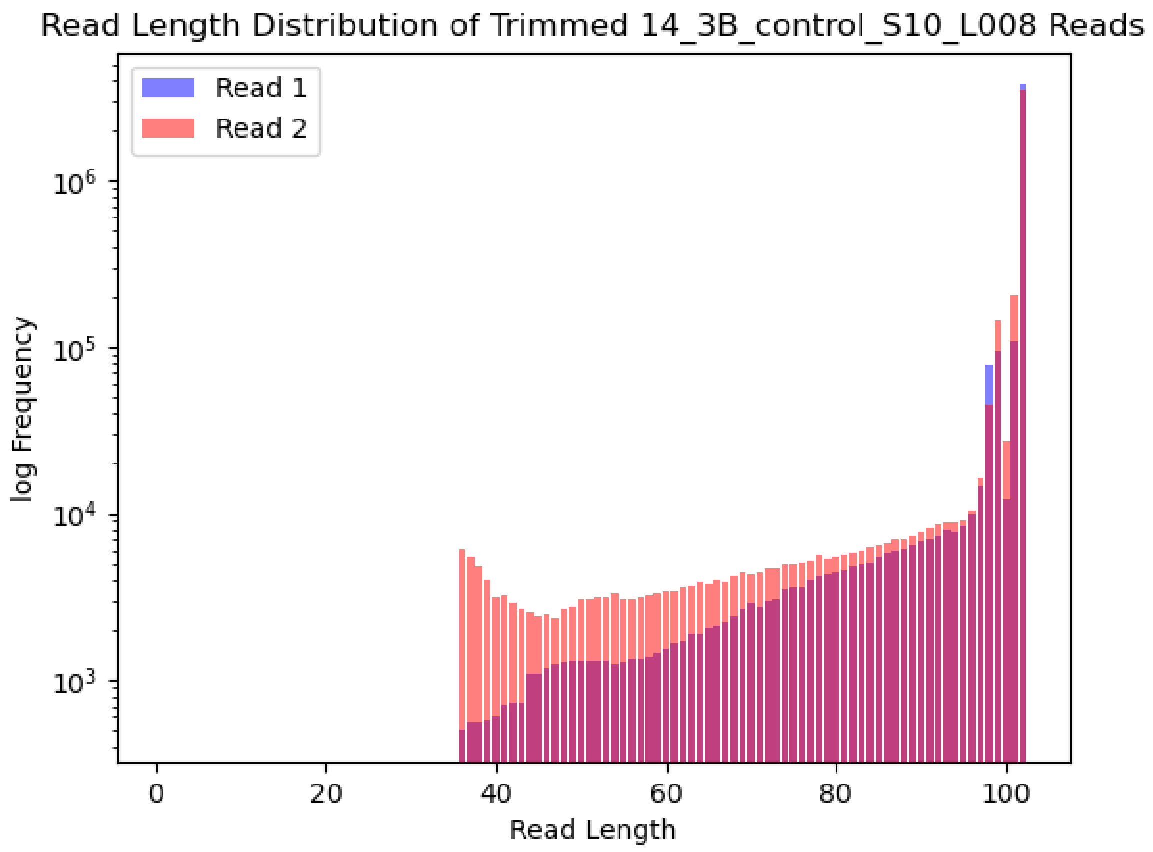
```
Input Read Pairs: 4440378 Both Surviving: 4251493 (95.75%) Forward Only Surviving: 182469 (4.11%) Revers
TrimmomaticPE: Completed successfully
```

Plotting read length distributions:
-Below is the Python script used to generate read length distribution graphs from adapter-trimmed files. I expect Read 2 to be trimmed more extensively than Read 1 because it will be slightly more degraded and have more bases with low quality score because it is on the sequencing machine for more time than Read 1. In short, the Python script below creates dictionaries with numbers 1-103 as keys and 0 as every value for each read file. It loops through the read files, and each time the length of the sequence being read matches the dictionary key, the respective value increases by 1. File is named trimmed_distributions.py.

```
#!/usr/bin/env python

import gzip, matplotlib.pyplot as plt

#test files with first 50 lines from their respective parent files
fox_test1 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/fox_r1_test.fq.gz"
#I dramatically shortened the reads in fox_test2 to be sure the colors and legend of my graphs properly
fox_test2 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/fox_r2_test.fq.gz"
control_test1 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/control_r1_test.fq.gz"
control_test2 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/control_r2_test.fq.gz"

fox_r1 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/8_2F_fox_S7_L008_trimmed_1P.fq.gz"
fox_r2 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/8_2F_fox_S7_L008_trimmed_2P.fq.gz"
control_r1 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/14_3B_control_S10_L008_trimmed_1P.fq.gz"
control_r2 = "/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/14_3B_control_S10_L008_trimmed_2P.fq.gz"

#initialize dictionary where keys = length of read and values = freequency of reads of key length
fox_r1_len_dict =  {}
#create keys for range of read lengths
for x in range(1, 103):
    fox_r1_len_dict[x] = 0

fox_r2_len_dict = {}
for x in range(1, 103):
    fox_r2_len_dict[x] = 0

control_r1_len_dict = {}
for x in range(1, 103):
    control_r1_len_dict[x] = 0

control_r2_len_dict = {}
for x in range(1, 103):
    control_r2_len_dict[x] = 0

#create length:frequency dictionary from 8_2F_fox_S7_L008_trimmed_1P.fq.gz
with gzip.open (fox_r1, "r") as fh:
    num_lines = 0
    for line in fh:
        num_lines += 1
        if num_lines % 4 == 2:
            line = line.decode('UTF-8')
            for key, value in fox_r1_len_dict.items():
                if len(line) == key:
                    fox_r1_len_dict[key] += 1

#create length:frequency dictionary from 8_2F_fox_S7_L008_trimmed_2P.fq.gz
with gzip.open (fox_r2, "r") as fh:
    num_lines = 0
    for line in fh:
        num_lines += 1
        if num_lines % 4 == 2:
            line = line.decode('UTF-8')
            for key, value in fox_r2_len_dict.items():
                if len(line) == key:
                    fox_r2_len_dict[key] += 1

#create length:frequency dictionary from 14_3B_control_S10_L008_trimmed_1P.fq.gz
with gzip.open (control_r1, "r") as fh:
    num_lines = 0
    for line in fh:
        num_lines += 1
        if num_lines % 4 == 2:
            line = line.decode('UTF-8')
            for key, value in control_r1_len_dict.items():
                if len(line) == key:
                    control_r1_len_dict[key] += 1

#create length:frequency dictionary from 14_3B_control_S10_L008_trimmed_2P.fq.gz
with gzip.open (control_r2, "r") as fh:
    num_lines = 0
    for line in fh:
        num_lines += 1
        if num_lines % 4 == 2:
            line = line.decode('UTF-8')
            for key, value in control_r2_len_dict.items():
                if len(line) == key:
                    control_r2_len_dict[key] += 1

#graphing 8_2F_fox_S7_L008 reads
plt.figure()
plt.bar(*zip(*fox_r1_len_dict.items()), color='blue', alpha=0.5)
plt.bar(*zip(*fox_r2_len_dict.items()), color='red', alpha=0.5)
plt.title("Read Length Distribution of Trimmed 8_2F_fox_S7_L008 Reads")
plt.xlabel("Read Length")
plt.yscale("log")
plt.ylabel("log Frequency")
plt.legend(["Read 1", "Read 2"])
plt.savefig("8_2F_fox_S7_L008.png")

#graphing 14_3B_control_S10_L008 reads
plt.figure()
plt.bar(*zip(*control_r1_len_dict.items()), color='blue', alpha=0.5)
plt.bar(*zip(*control_r2_len_dict.items()), color='red', alpha=0.5)
plt.title("Read Length Distribution of Trimmed 14_3B_control_S10_L008 Reads")
plt.xlabel("Read Length")
plt.yscale("log")
plt.ylabel("log Frequency")
plt.legend(["Read 1", "Read 2"])
plt.savefig("14_3B_control_S10_L008.png")
```

Below are the plots for both samples, and it appears Read 2 is trimmed more than Read 1 in both sets of reads.





Part 3: Alignment and strand-specificity

In my QAA environment, I installed:

-star (Version 2.7.9a, Build h9ee0642_0)
-numpy (Version 1.21.2, Build py39hdbf815f_0)
-pysam (Version 0.16.0.1, Build py39h051187c_3)
-matplotlib (Version 3.4.3, Build py39hf3d152e_0)

Then used pip to install HTseq (Version 0.13.5, Build pypi_0)

My mouse genome fasta file from Ensemble release 104: Mus_musculus.GRCm39.dna.primary_assembly.fa
My mouse gtf file: Mus_musculus.GRCm39.104.gtf

Below is the Talapus batch script to create an alignment database. File named QAA_database.sh.

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --job-name=QAA_DB
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=10:00:00

conda activate QAA
```

```
genome_dir="/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a"
genome_ff="/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a/Mus_m
genome_gtf="/home/ndr/bgmp/bioinfo/Bi623/Assignments/QAA/mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a/Mus_

/usr/bin/time -v STAR --runMode genomeGenerate --runThreadN 8 --genomeDir $genome_dir --genomeFastaFiles
```

Below is the Talapus batch script used to align my 14_3B_control_S10_L008 reads to the mouse genomic database. I only changed read1, read2, and outFileNamePrefix for my fox_S7_L008 samples so I didn't include them here. File named QAA_align.sh.

```
#!/bin/bash

#SBATCH --account=bgmp
#SBATCH --partition=bgmp
#SBATCH --job-name=QAA_align_14_3B_control_S10
#SBATCH --cpus-per-task=8
#SBATCH --nodes=1
#SBATCH --time=01:00:00

conda activate QAA

genome_file="/projects/bgmp/ndr/bioinfo/Bi623/Assignments/QAA/mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a
genome_dir="/projects/bgmp/ndr/bioinfo/Bi623/Assignments/QAA/mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a
genome_gtf="/projects/bgmp/ndr/bioinfo/Bi623/Assignments/QAA/mus_musculus.GRCm39.dna.ens104.STAR_2.7.9a/
read1="/projects/bgmp/ndr/bioinfo/Bi623/Assignments/QAA/14_3B_control_S10_L008_trimmed_1P.fq.gz"
read2="/projects/bgmp/ndr/bioinfo/Bi623/Assignments/QAA/14_3B_control_S10_L008_trimmed_2P.fq.gz"

/usr/bin/time -v STAR --runThreadN 8 --runMode alignReads --outFilterMultimapNmax 3 --alignIntronMax 100
--outSAMunmapped Within KeepPairs \
--alignMatesGapMax 1000000 --readFilesCommand zcat \
--readFilesIn $read1 $read2 \
--genomeDir $genome_dir \
--outFileNamePrefix 14_3B_control_S10_L008
```

To count mapped and unmapped reads, I updated my script from PS8, shown below.

```
#!/usr/bin/env python

import argparse

def get_args():
    parser = argparse.ArgumentParser(description="A program to normalize kmer data")
    parser.add_argument("-f", "--input_filename", help="your filename", required = True)
    parser.add_argument("-o", "--output_filename", help="your filename", required = True)

    return parser.parse_args()

args = get_args()
input_file = args.input_filename
output_file = args.output_filename

mapped_count = 0
unmapped_count = 0
```

```
with open (input_file, "r") as fh:
    while True:
        new_line = fh.readline()
        if new_line == "":
            break
        if new_line[0] == "@":
            continue
        else:
            name = new_line.split()[0]      #gives us QNAME in SAM file
            flag = new_line.split()[1]      #gives us FLAG in SAM file
        if(int(flag) & 4) != 4 and (int(flag) & 256) == 0:     #need 256 bit == 0
            mapped_count += 1
        elif (int(flag) & 4) == 4 and (int(flag) & 256) == 0:
            unmapped_count += 1      #ticks up unmapped_count for ALL unmapped alignments

total = mapped_count + unmapped_count

with open (output_file, "w") as fout:
    fout.write(f'Input file: {input_file}\n')
    fout.write(f'Mapped: {mapped_count}\nUnmapped: {unmapped_count}\nTotal: {total}')
```

Results:

8_2F_fox_S7_L008 contained:
-Mapped: 67,048,088
-Unmapped: 2,548,218
-Total: 69,596,306

14_3B_control_S10_L008 contained:
-Mapped: 8,310,327
-Unmapped: 192,659
-Total: 8,502,986

I then used HTSeq to count reads that mapped to features. The HTSeq documentation said either GTF or GFF files containing the features could be used, so I used my previously mentioned mouse GTF file. The HTSeq documentation also suggested we use samtools to sort the file by name before counting, code below.

```
samtools sort -n 8_2F_fox_S7_L008Aligned.out.sam > sorted_8_2F_fox_S7_L008Aligned.out.sam
samtools sort -n 14_3B_control_S10_L008Aligned.out.sam > sorted_14_3B_control_S10_L008Aligned.out.sam
```

I ran HTSeq-count on the sorted files, changing only the stranded setting and keeping everything else on default.

```
#8_2F_fox_S7_L008, stranded=yes
htseq-count --stranded=yes sorted_14_3B_control_S10_L008Aligned.out.sam/home/ndr/bgmp/bioinfo/Bi623/Assi

#8_2F_fox_S7_L008, stranded=no
htseq-count --stranded=no sorted_14_3B_control_S10_L008Aligned.out.sam/home/ndr/bgmp/bioinfo/Bi623/Assig

#14_3B_control_S10_L008, stranded=yes
htseq-count --stranded=yes sorted_14_3B_control_S10_L008Aligned.out.sam/home/ndr/bgmp/bioinfo/Bi623/Assi

#14_3B_control_S10_L008, stranded=no
htseq-count --stranded=no sorted_14_3B_control_S10_L008Aligned.out.sam/home/ndr/bgmp/bioinfo/Bi623/Assig
```

Summarized Results:

```
8_2F_fox_S7_L008, stranded=yes
___no_feature 30,648,212
___ambiguous 26,177
___too_low_aQual 61,546
___not_aligned 1,241,080
___alignment_not_unique 1,587,953
___34,798,153 alignment pairs processed

8_2F_fox_S7_L008, stranded=no
___no_feature 3,148,132
___ambiguous 1,554,869
___too_low_aQual 61,546
___not_aligned 1,241,080
___alignment_not_unique 1,587,953
___34,798,153 alignment pairs processed

14_3B_control_S10_L008, stranded=yes
___no_feature 3,806,291
___ambiguous 3,326
___too_low_aQual 6,265
___not_aligned 92,990
___alignment_not_unique 181,967
___4,251,493 alignment pairs processed

14_3B_control_S10_L008, stranded=no
___no_feature 207,859
___ambiguous 209,168
___too_low_aQual 6,265
___not_aligned 92,990
___alignment_not_unique 181,967
___4,251,493 alignment pairs processed
```

Discussion:
Fox: 0.17% low quality, 3.6% not aligned, 4.6% alignment not unique, 96.3% mapped
Control: 0.15% low quality, 2.2% not aligned, 4.3% alignment not unique, 97.8% mapped

Fox stranded=yes: 88% no_feature, 0.08% ambiguous
Fox stranded=no: 9.0% no_feature, 4.5% ambiguous

Control stranded=yes: 90% no_feature, 0.08% ambiguous
Control stranded=no: 4.9% no_feature, 4.9% ambiguous

These data are strand specific. If our data were unstranded, we would see a 50/50 distribution of reads mapping to a feature when running stranded=yes, but instead we see both the Fox and Control having an overwhelming majority (88% and 90%, respectively) not mapping to a feature. Stranded=yes is synonymous to stranded=fwd (Batzel 2021), so we can assume if we ran stranded=reverse, ~12% Fox and ~10% Control wouldn't map to a feature, leading us to believe this was a strand specific RNASeq favoring the reverse orientation.