# RNA-seq Quality Assasment Assignment Report

Carter Alzamora

This report is in reference to the libraries: 29_4E_fox_S21_L008 - from here on referred to as S21
11_2H_both_S9_L008 - from here on referred to as S9
Both S21 and S9 have an R1 and R2 file.

## Part 1 - Read Quality Score Distributions

To begin investigating these two libraries, I first ran my own python script to parse the average quality score per base in each file. The output of this script is as follows:

R1 for both S9 and S21 (A + C) is more consistent than R2 (B + D) in quality score for both data sets, as show in in FIGURE 1 below. As expected, read quality dips at the beginning and ends of the reads. Overall, reads average over 30 in phred+33 encoding. The S21R2 (D) Library is the most variable with relatively large dips in the quality score throughout the read compared to the more stable quality score of the other libraries.

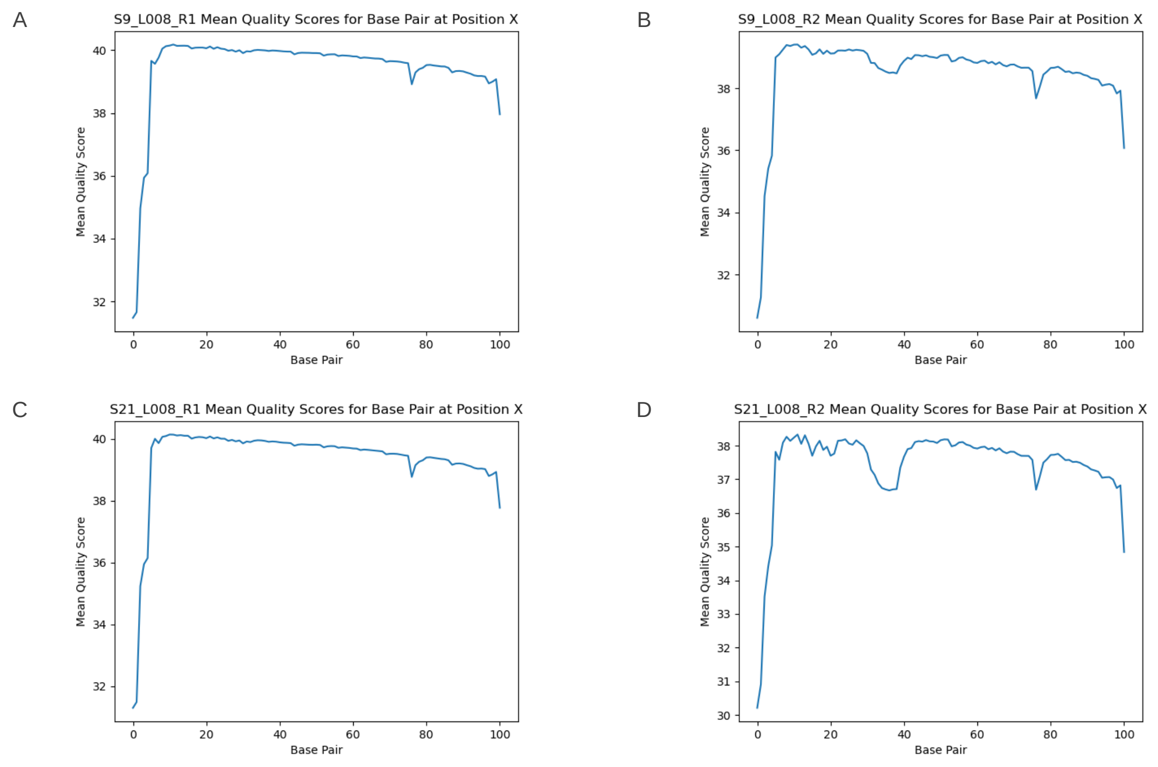

Figure 1: Per Base Quality Score Output from dist_qscore.py

**The next step was to run these libraries through FastQC in order to compare the results of the professional software to my own. FastQC output per-base mean quality score graphs as well as a report of other metrics of the data. The highlights of this output are as follows:**

The scale of the FastQC graphs show the outputs of these libraries relative to the entire scale of phred scores as seen in FIGURE 2 below. They show that the majority of these reads fall on the high end of this scale. However, this scale obscures some of the variation in quality score throughout the reads.
The same major patterns can be seen as in the graphs output from dist_qscore.py most importantly, the dips in quality at the beginning and ends of reads. The colored visualization of the score ranges as green, yellow, and red, do convey the overall quality of these libraries at a glance. The yellow boxes representing the 25th and 75th percentiles are another good visualization of the quality of the data. Again, the S21R2 (D) Library is the most variable with a much wider spread of quality throughout the entire read as shown by the wider quartile boxes and large error bars. For both S9 and S21, the R1 libraries are more consistantly high quality.

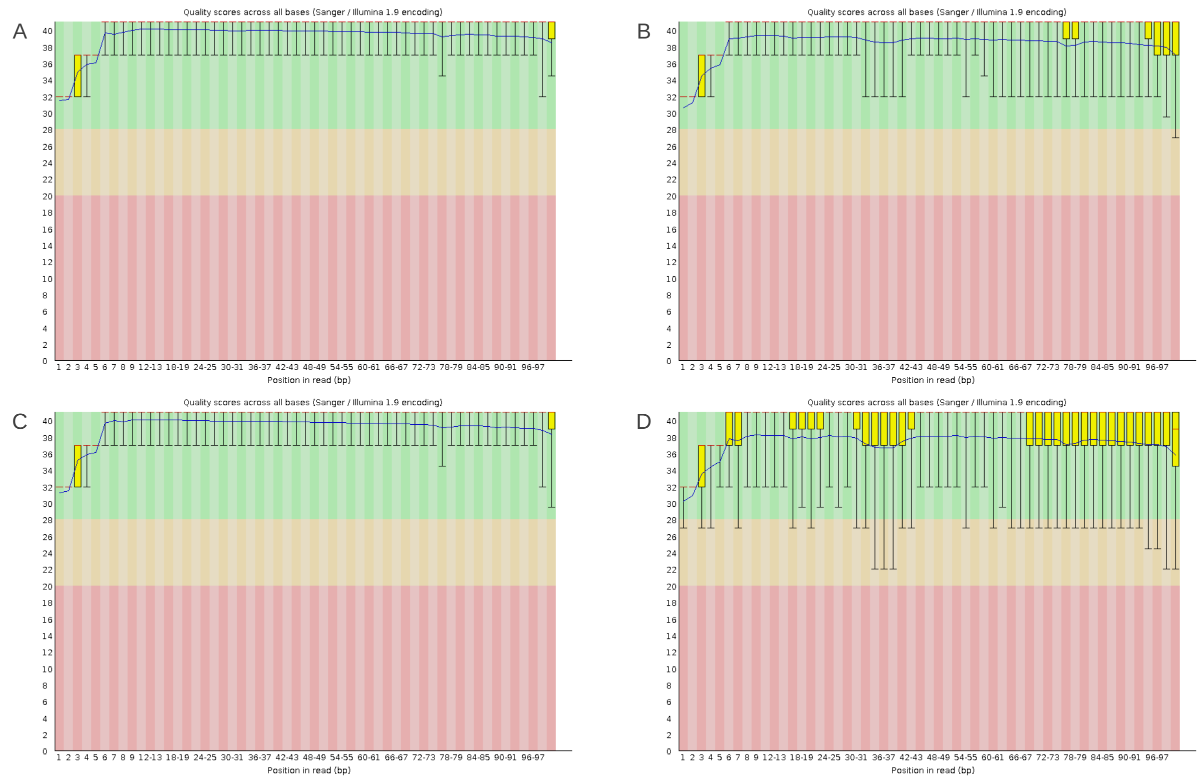

Figure 2: FastQC quality per base S9R1 (A), S9R2(B), S21R1(C), and S21R2(D)

**FastQC More info:**
The per sequence quality score of these libraries is show in FIGURE 3. All these libraries are consistently averaging over 30 relatively consistently. S21R2(D) has the most spread of mean sequence quality score. However, the majority of it's sequences are still well over 30 in phred score. This is a pattern that is reflected in the per-base quality score distribution so it is expected that the sequence quality scores would be more dispersed in this metric as well.

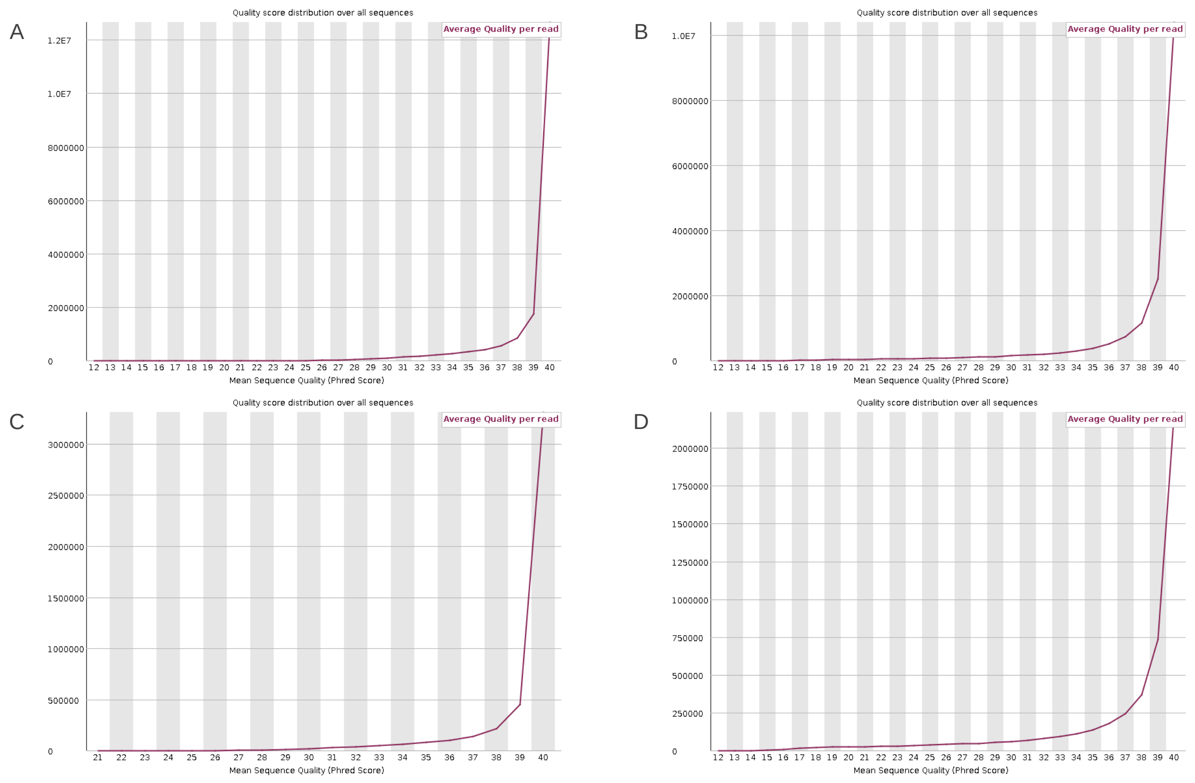

Figure 3: Per Sequence Quality Score Graphs output by FastQC for S9R1(A), S9R2(B), S21R1(C), and S21R2(D)

The overall n-content is very low in every library as seen in FIGURE 4. Each of them has a small increase of n's at the very beginning of the reads around base 1-2. This is expected due to the quality score dips at the beginning of the libraries. This is not concerning to me as it is expected for Illumina reads to have these lower quality reads at each end. Even at these early bases, the N-content is low and will not obscure the actual reads.

FastQC consistently required less time and CPU to process than my dist_qscore.py script to do signficantly more analysis. It also ran the R1 and R2 reads simultaneously while my software had to break these up individually. The comparative metrics of these runs are referenced in Tables 1-4 below.

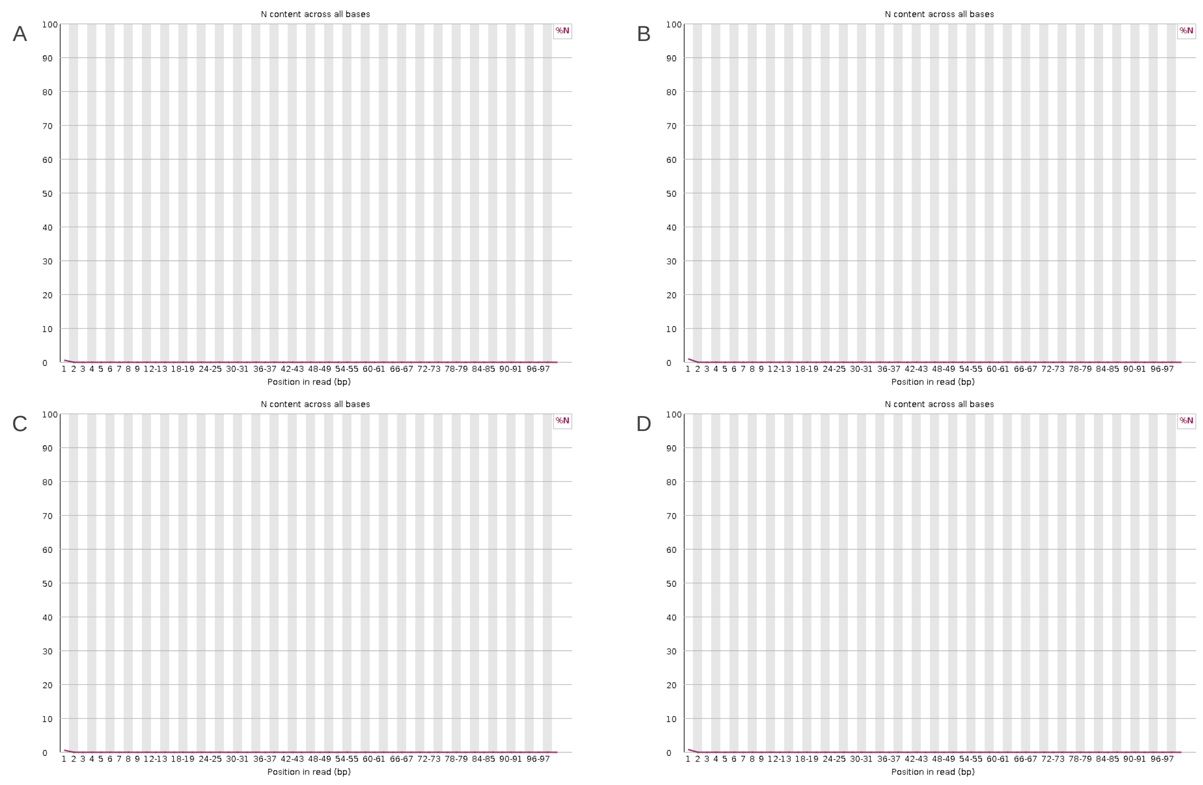

Figure 4: Per Base N Content Graphs output by FastQC for S9R1(A), S9R2(B), S21R1(C), and S21R2(D)

Table 1: Run Metrics for S21R1 in dist_qscore.py vs FastQC for R1 + R2

| NA | S21_L008_R1_001 | My Code | FastQC (R1 + R2) | NA |
|---|---|---|---|---|
| NA | Runtime user(sec) | 87.64 | 152.83 | NA |
| NA | Mem/CPU | 100% | 98% | NA |

Table 2: Run Metrics for S21R2 in dist_qscore.py vs FastQC for R1 + R2

| NA | S21_L008_R2_001 | My Code | FastQC (R1 + R2) | NA |
|---|---|---|---|---|
| NA | Runtime user(sec) | 313.13 | 152.83 | NA |
| NA | Mem/CPU | 97% | 98% | NA |

Table 3: Run Metrics for S9R1 in dist_qscore.py vs FastQC for R1 + R2

| NA | S9_L008_R1_001 | My Code | FastQC (R1 + R2) | NA |
|---|---|---|---|---|
| NA | Runtime user(sec) | 313.13 | 152.83 | NA |
| NA | Mem/CPU | 97% | 98% | NA |

Table 4: Run Metrics for S9R2 in dist_qscore.py vs FastQC for R1 + R2

| NA | S9_L008_R2_001 | My Code | FastQC (R1 + R2) | NA |
|---|---|---|---|---|
| NA | Runtime user(sec) | 312.84 | 152.83 | NA |
| NA | Mem/CPU | 99% | 98% | NA |

**Are these libraries high quality?**

Yes - I would consider these libraries to be of high enough qulaity to continue. Their per base(fig.1 + 2) and per sequence quality(fig.3) scores are all consistantly high with the exception of the S21R2(fig.1 + 2 + 3 (D)). Even this library has an overall high quality score even if it is less consistant throughout the read. The n content is low throughout each of the reads suggesting that there are a minimal number of missing reads. This all combines to give me confidence in these libraries even before the processing that follows.

## Part 2 - Adaptor Trimming Comparison

After using cutadapt to remove adapter sequences and trimmomatic to quality trim the data, I plotted the length distribution of reads in the trimmed data - this is shown in FIGURE 5 below. In both the S9 and S21 data R2(blue) was trimmed more extensively than R1(pink). This is shown in the higher concentration of short sequences in the reverse(R2) library when compared to the forward(R1) library. This makes sense as the initial data were higher quality for both R1 libraries.

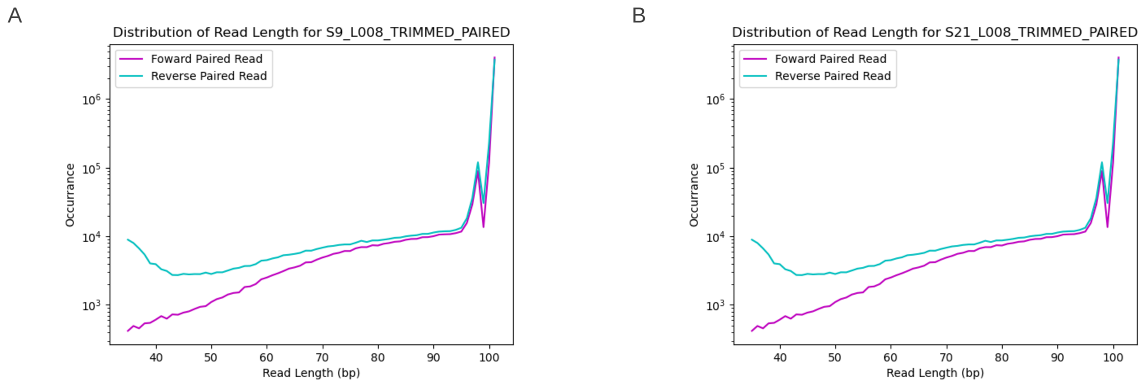

Figure 5: Read Length Distribution of Trimmed Libraries S9(A), and S21(B)

When FastQC was run on the trimmed data as shown in FIGURE 6 below, the resulting quality per base graphs showed a mugh higher average when compared to the raw data in FIGURE 2. This shows how removing the adaptor sequences and low quality reads raises the overall quality score of the library even while it leaves them with shorter reads.

Again, the R2 reverse files (FIGURE 6. (B + D)) are overall less consistent in mean quality score throughout the read. The S21R2 (D) library is still less consistant than the others, however it's mean quality scores have a less dramatic spread and are averaging higher than in the initial library.

I would expect the R1 reads to be adapter-trimmed at a higher rate because the read qualities are higher overall, there will be more adapter sequences on the reads to be trimmed. This will not necessarily result in shorter read lengths as the R1 will not be quality trimmed as harshly as R2.

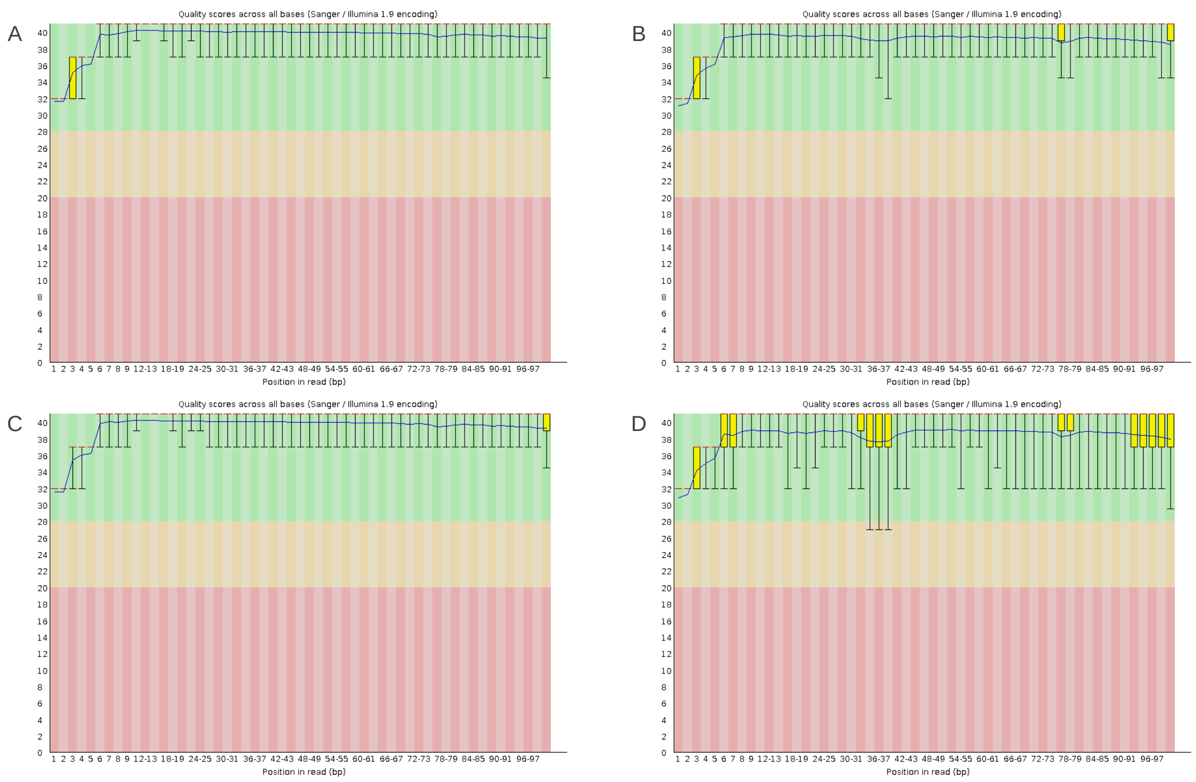

Figure 6: FastQC quality per base on trimmed libraries S9R1 (A), S9R2(B), S21R1(C), and S21R2(D)

## Part 3 - Alignment and Strand Specificity

### Are these libraries stranded?

Yes, these libraries are both stranded. The results of the R2 reverse HTseq runs are 79.12% and 84.42% for the 11_2H_both_S9_L008 (Table 6) and 29_4E_fox_S21_L008 (Table 7) libraries respectively. Comparatively, the results of the R1 forward stranded HTseq run only have 3.19% and 3.93% for the 11_2H_both_S9_L008 and 29_4E_fox_S21_L008 libraries respectively.
The higher alignment numbers for the reverse run indicate that the reverse stranded R2 file aligned better that the Forward R1 file.
This can also be seen quantitatively - the mapped and unmapped read count output from my mapped_counts.py software are equal to 2x the total reads in HTseq:

**count of mapped reads + unmapped reads = (sum of all the htseq values)x2**

**S9:** mapped_counts.py: 33,637,699(mapped) + 1,293,527(unmapped) = 34,931,226(total)
ht_seq: 17,465,613(output) * 2 = 34,931,226

**S21:**
my software: 8,883,008(mapped) + 260,800(unmapped) = 9,143,808(total)
ht_seq: 4,571,904 * 2 = 9,143,808

This indicates that when we run HT_seq with the strand flags, it is only taking in half the data - aka one strand. In contrast, my mapped_counts.py (Table 5) software does not have the ability to distinguish between the two strands and therefore is using both - double the data as HT_Seq. This means that the results of mapped_counts.py cannot speak to stranded-ness.

Table 5: Counts output by my mapped_counts.py script

| NA | mapped_counts.py Results | S9 | S21 | NA |
|---|---|---|---|---|
| NA | Total Reads | 34,931,226 | 9,143,808 | NA |
| NA | Mapped Reads | 33,637,699 | 8,883,008 | NA |
| NA | Percent of Reads. Mapped | 96.3% | 97.15% | NA |

Table 6: HTseq results for S9

| NA | S9 HTseq results | Foward | Reverse | NA |
|---|---|---|---|---|
| NA | Total Reads | 17,465,613 | 17,465,613 | NA |
| NA | Mapped Reads | 557,737 | 13,817,984 | NA |
| NA | Percent of Reads. Mapped | 3.19% | 79.12% | NA |

Table 7: HTseq results for S21

| NA | S21 HTseq results | Foward | Reverse | NA |
|---|---|---|---|---|
| NA | Total Reads | 4,571,904 | 4,571,904 | NA |
| NA | Mapped Reads | 179,976 | 3,859,630 | NA |
| NA | Percent of Reads. Mapped | 3.93% | 84.42% | NA |