SF-seq assignment

David Ho

9/27/2017

The files I am working with are:

```
David   24_4A_control   34_4H_both
```

How many paired reads are in there?

```
$ wc -l 24_4A_control_S18_L008_R1_001.fastq
42063496 24_4A_control_S18_L008_R1_001.fastq

$ wc -l 34_4H_both_S24_L008_R1_001.fastq
36162388 34_4H_both_S24_L008_R1_001.fastq
```

24_4A_control: 10515874

34_4H_both: 9040597

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.

The version of `FastQC` we are using is: `FastQC v0.11.5`

```
time fastqc -o /home/dho/Bi624/assignments/SF_seq/fastqc_output/ --noextract -f fastq /h
ome/dho/Bi624/assignments/SF_seq/seq_files/*
```

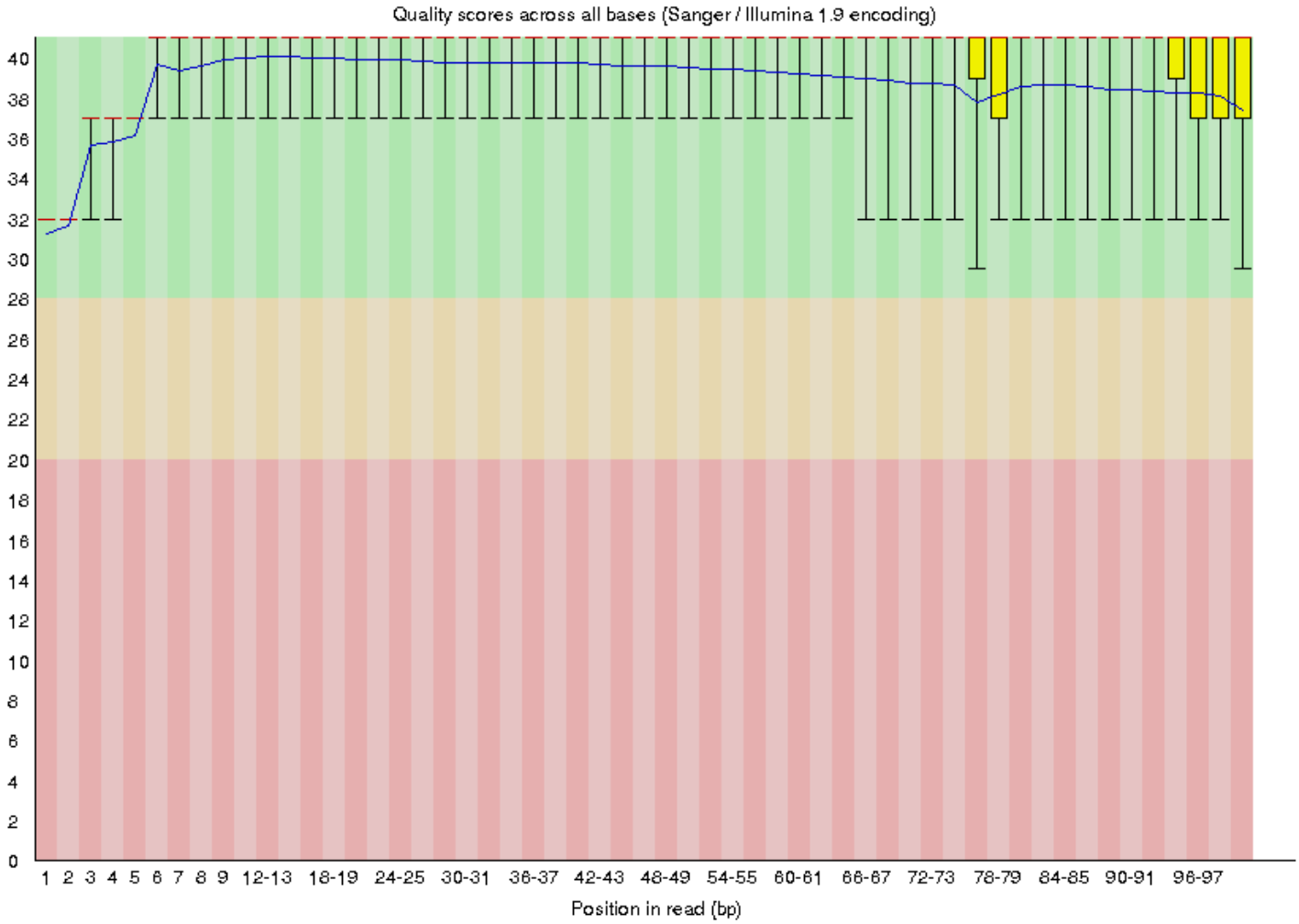
Separated by red headers, we are looking at (1) per-base quality score, (2) distribution of average quality score per read, and (3) per-base N content:

1. per-base quality score

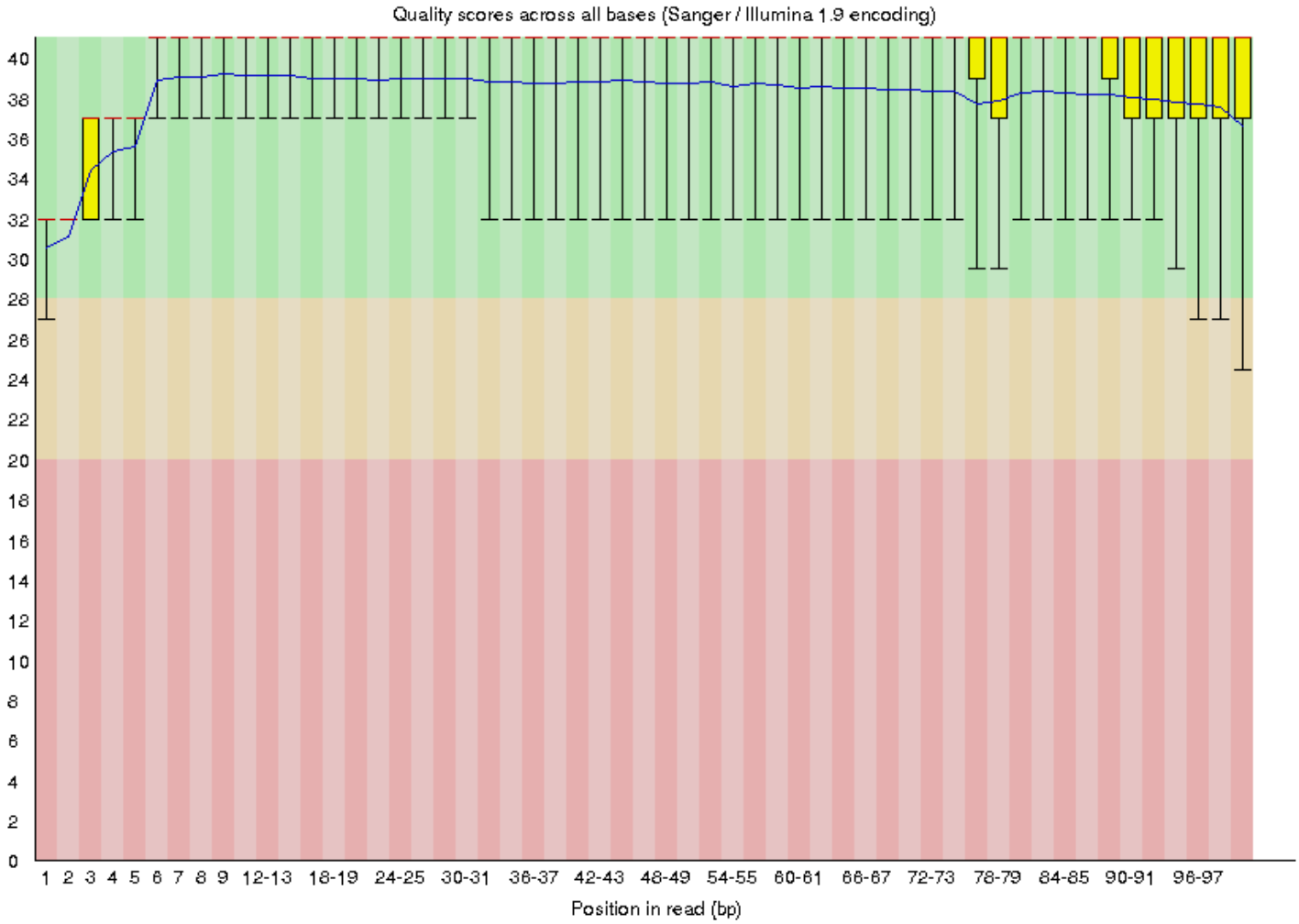
24_4A_control

Plots from `FastQC`

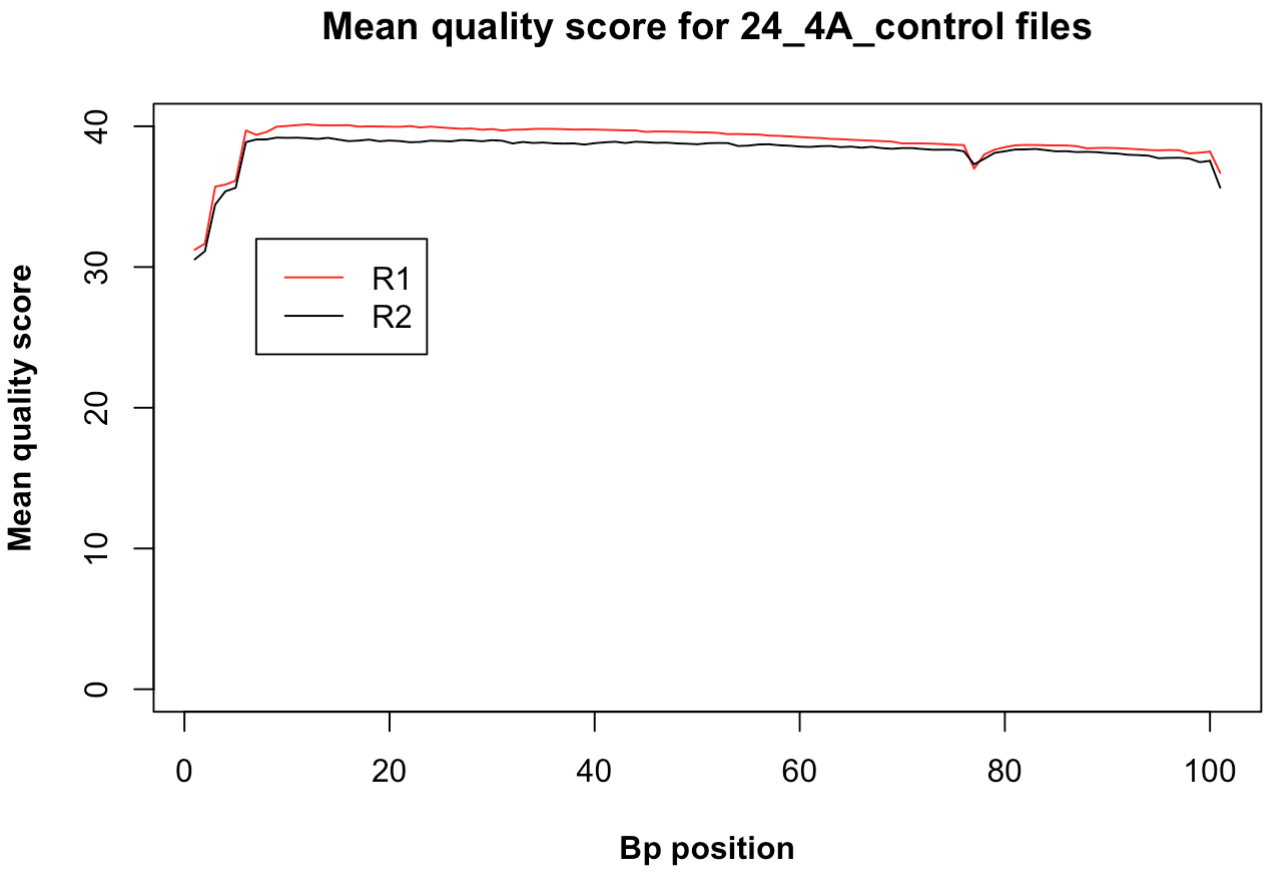
R1


Quality scores across all bases (Sanger / Illumina 1.9 encoding)
40
38
36
34
32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
1 2 3 4 5 6 7 8 9 12-13 18-19 24-25 30-31 36-37 42-43 48-49 54-55 60-61 66-67 72-73 78-79 84-85 90-91 96-97
Position in read (bp)

R2



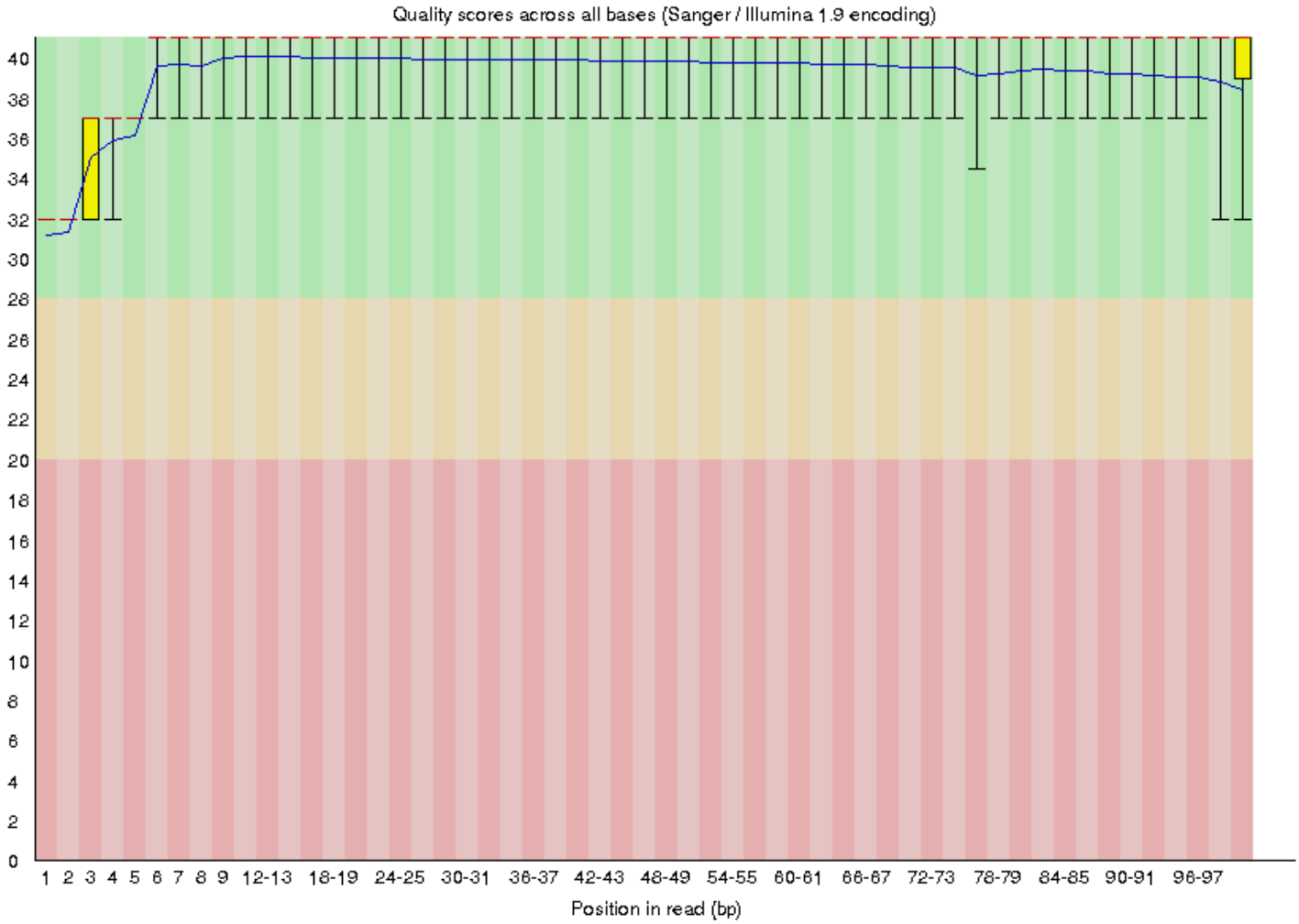
Plots from my script



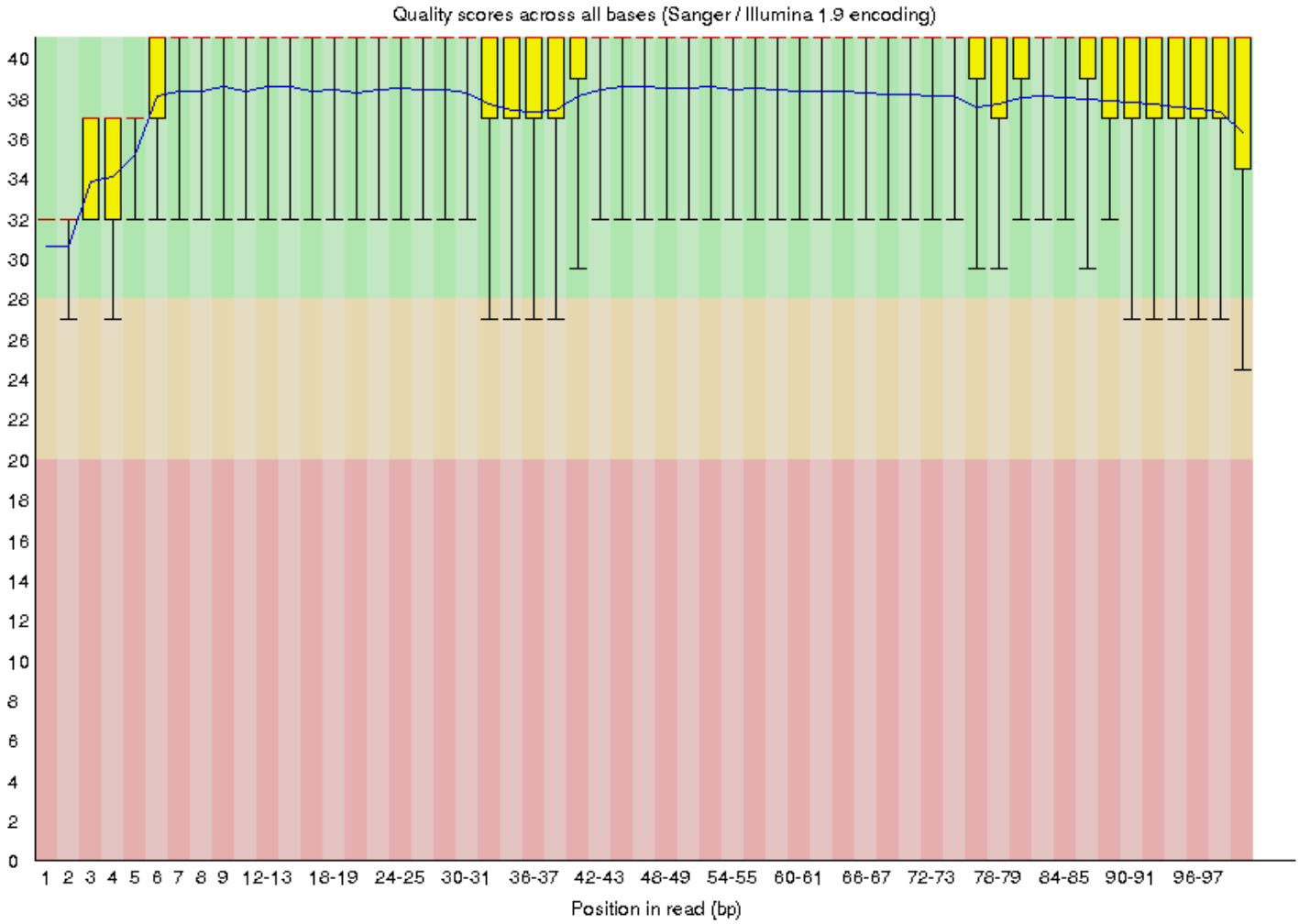
34_4H_both

Plots from `FastQC`

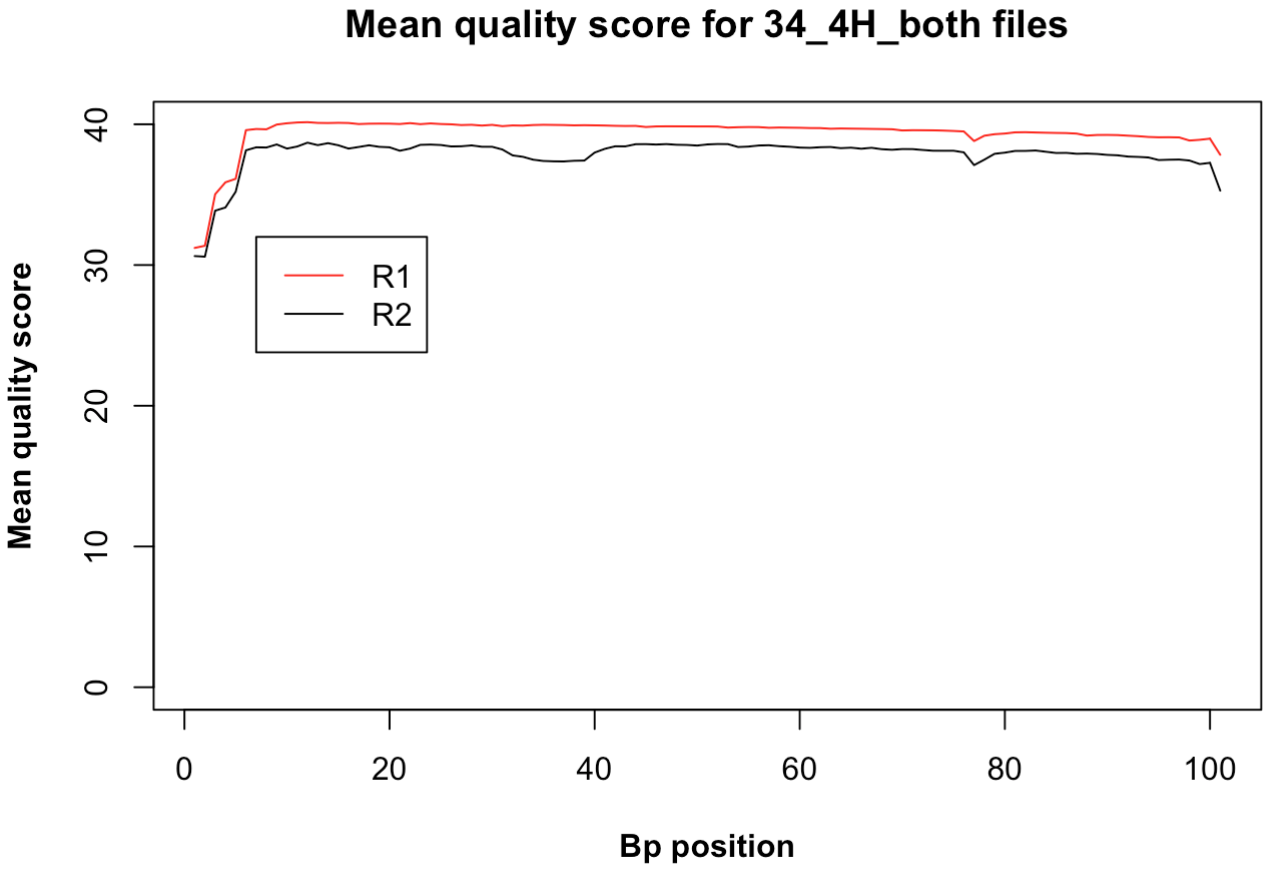
R1


Quality scores across all bases (Sanger / Illumina 1.9 encoding)
40
38
36
34
32
30
28
26
24
22
20
18
16
14
12
10
8
6
4
2
0
1 2 3 4 5 6 7 8 9 12-13 18-19 24-25 30-31 36-37 42-43 48-49 54-55 60-61 66-67 72-73 78-79 84-85 90-91 96-97
Position in read (bp)

R2



Plots from my script

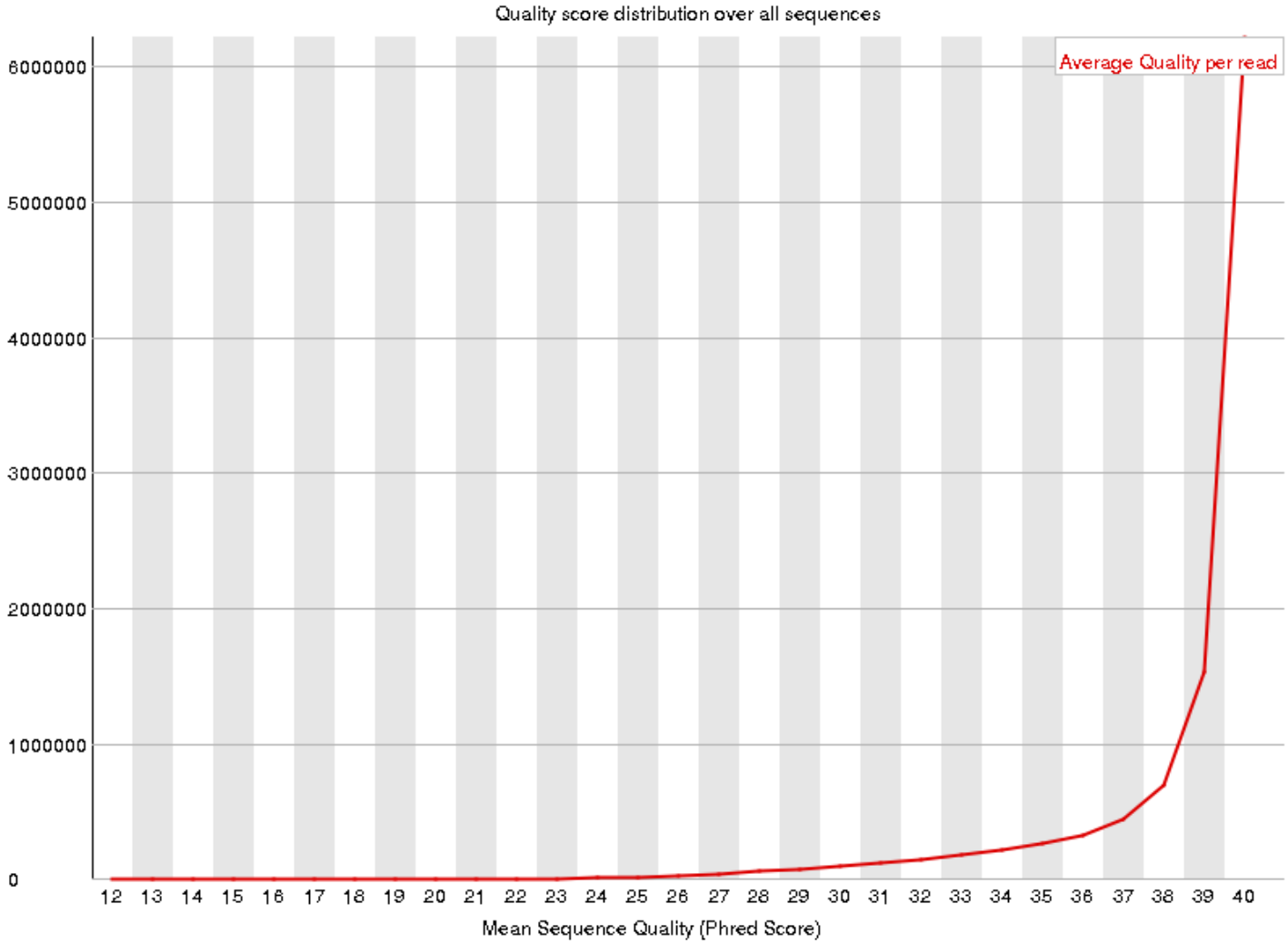


2. distribution of average quality score per read

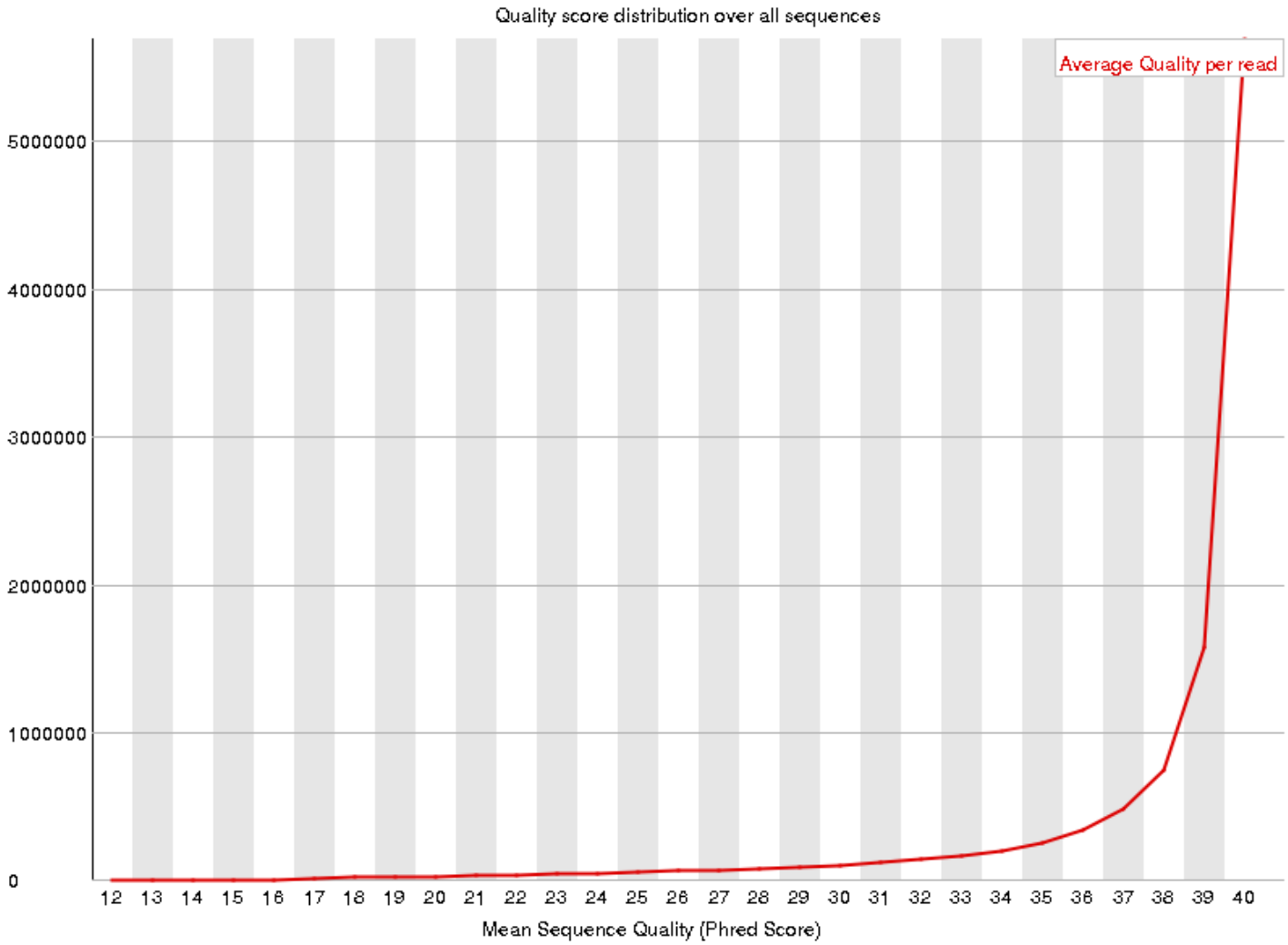
24_4A_control

Plots from `FastQC`

R1

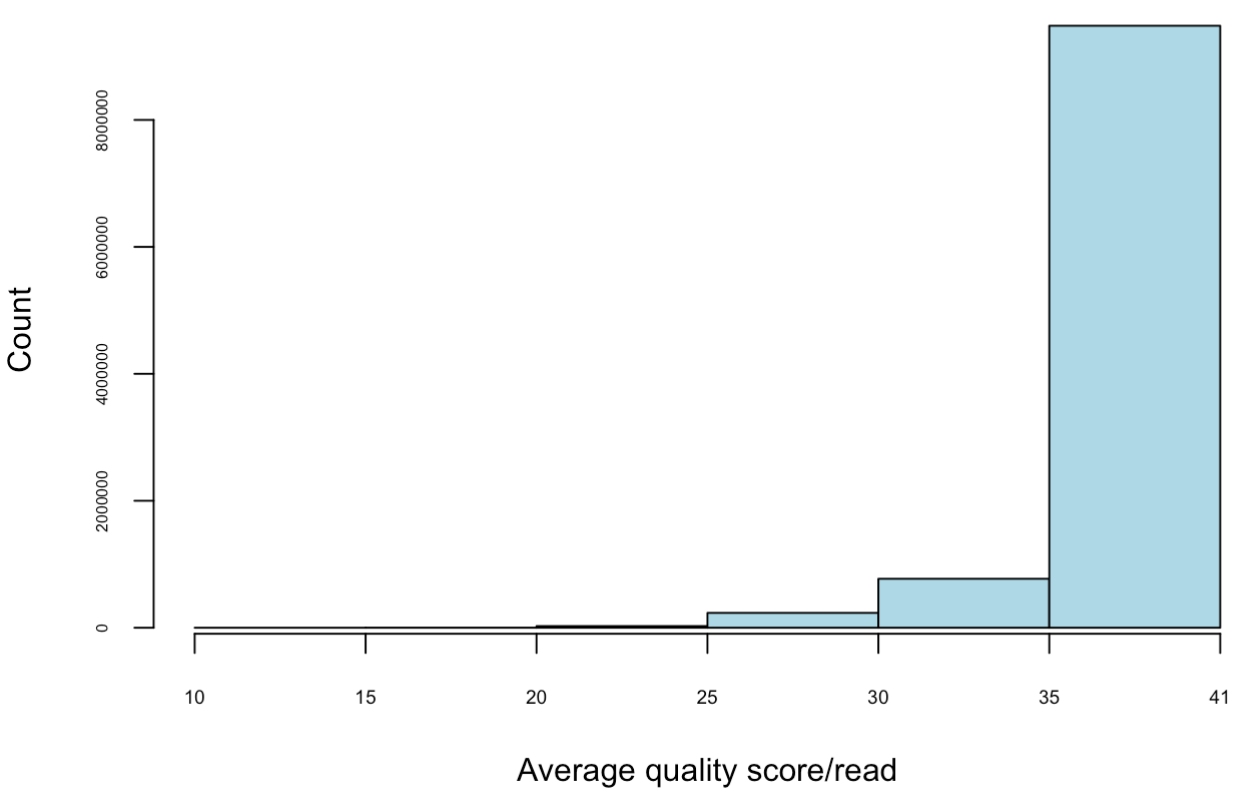

Quality score distribution over all sequences
Average Quality per read
6000000
5000000
4000000
3000000
2000000
1000000
0
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
Mean Sequence Quality (Phred Score)

R2

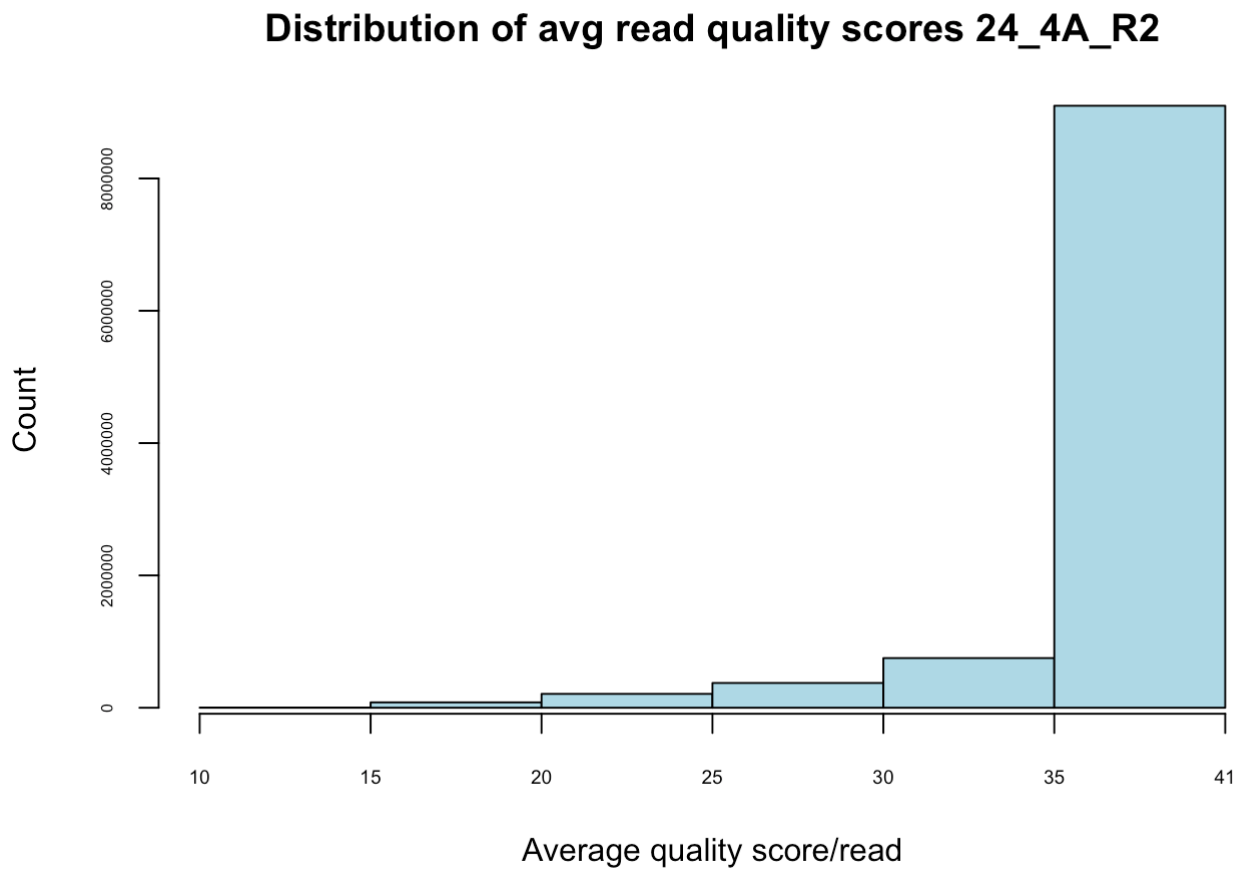

Quality score distribution over all sequences
Average Quality per read
5000000
4000000
3000000
2000000
1000000
0
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
Mean Sequence Quality (Phred Score)

My script

R1


Distribution of avg read quality scores 24_4A_R1
Count
0
2000000
4000000
6000000
8000000
10
15
20
25
30
35
41
Average quality score/read

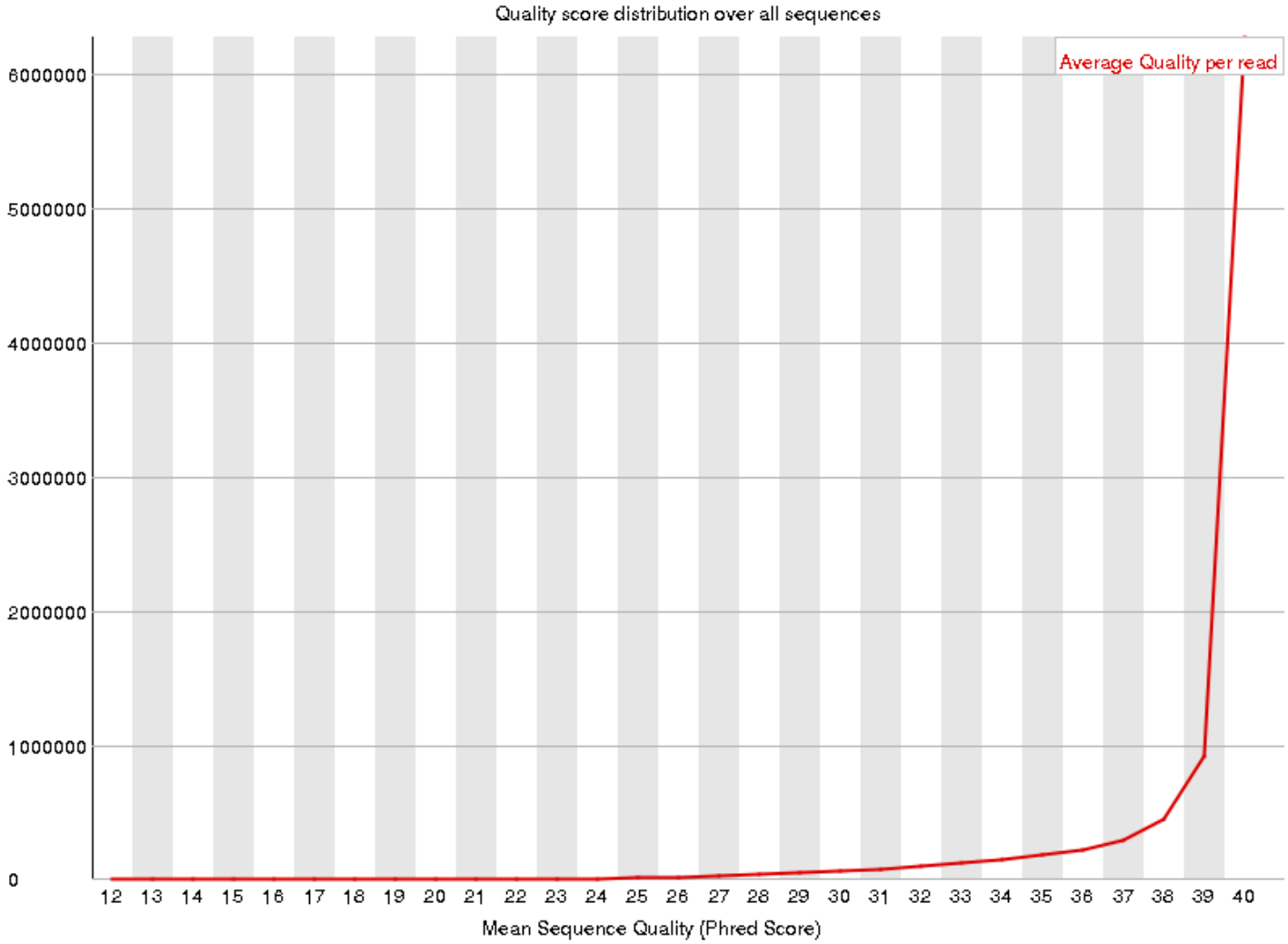
R2



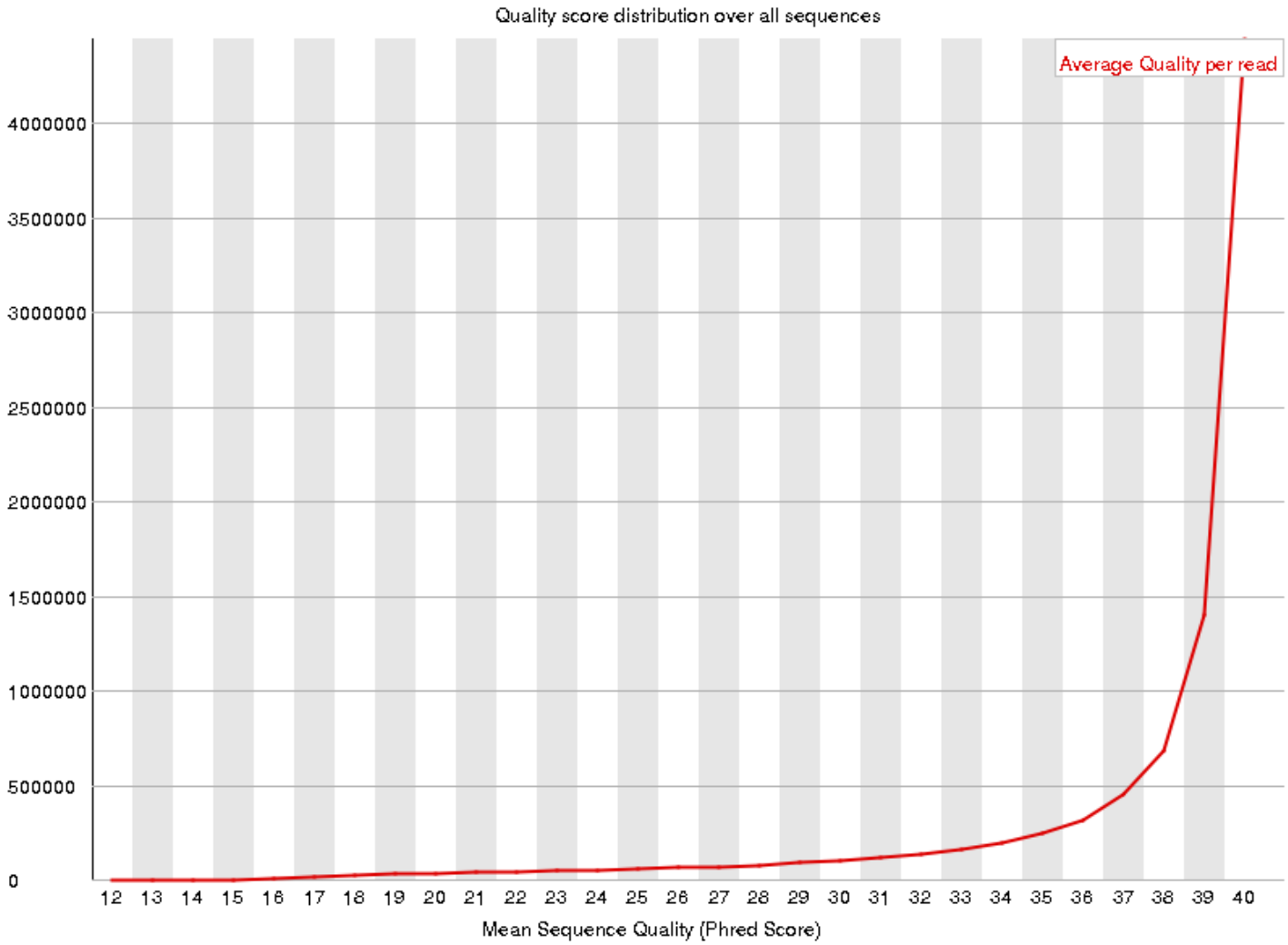
34_4H_both

Plots from `FastQC`

R1

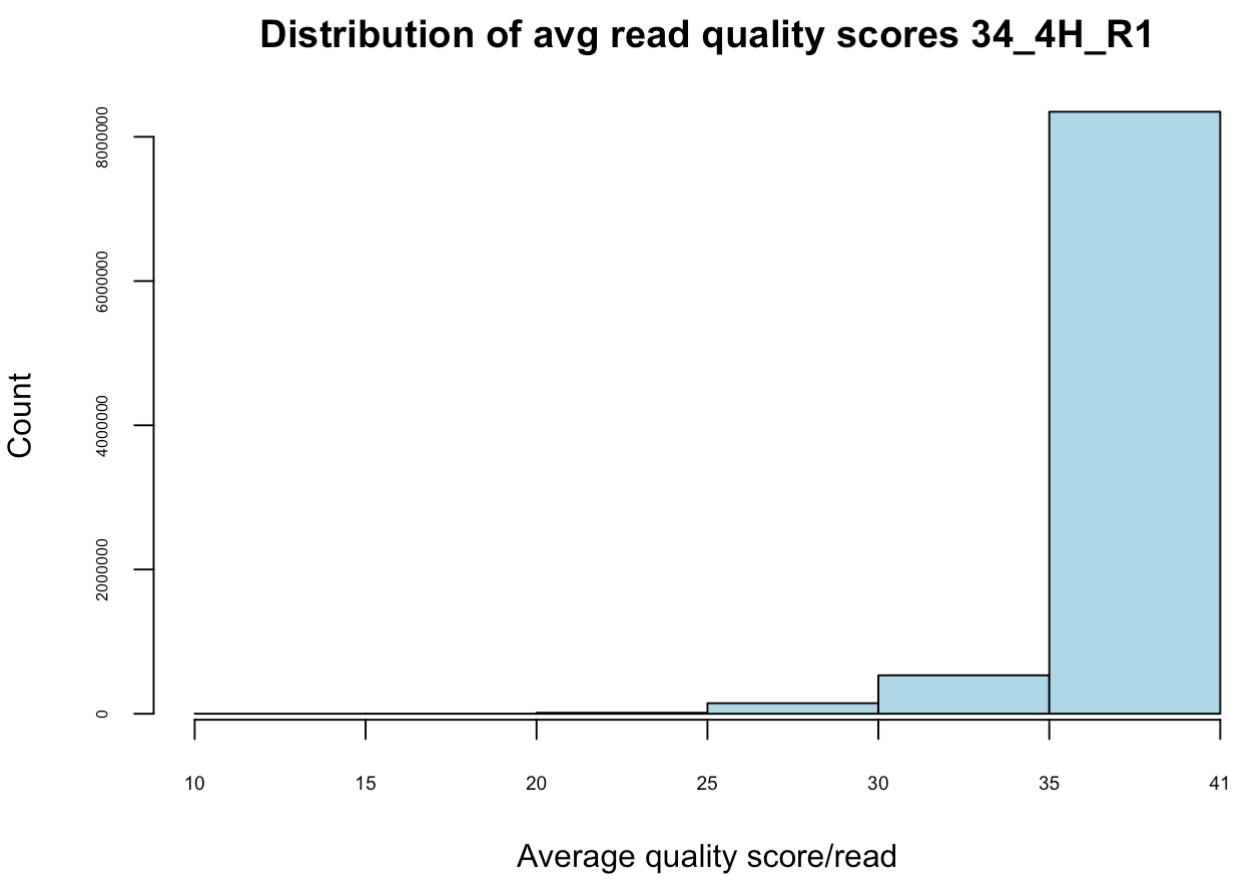

Quality score distribution over all sequences
Average Quality per read
6000000
5000000
4000000
3000000
2000000
1000000
0
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
Mean Sequence Quality (Phred Score)

R2

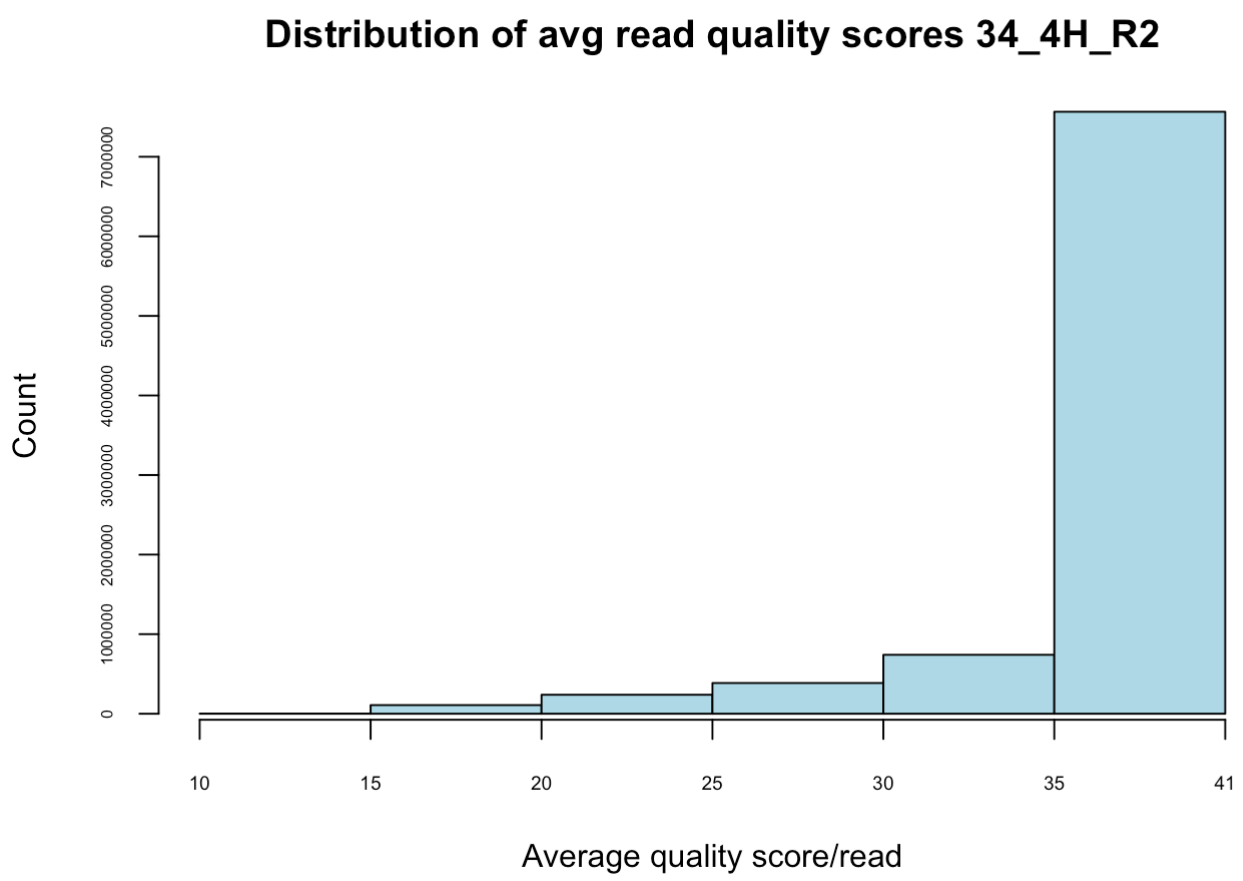

Quality score distribution over all sequences
Average Quality per read
4000000
3500000
3000000
2500000
2000000
1500000
1000000
500000
0
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
Mean Sequence Quality (Phred Score)

My script

R1


Distribution of avg read quality scores 34_4H_R1
Count
0
2000000
4000000
6000000
8000000
10
15
20
25
30
35
41
Average quality score/read

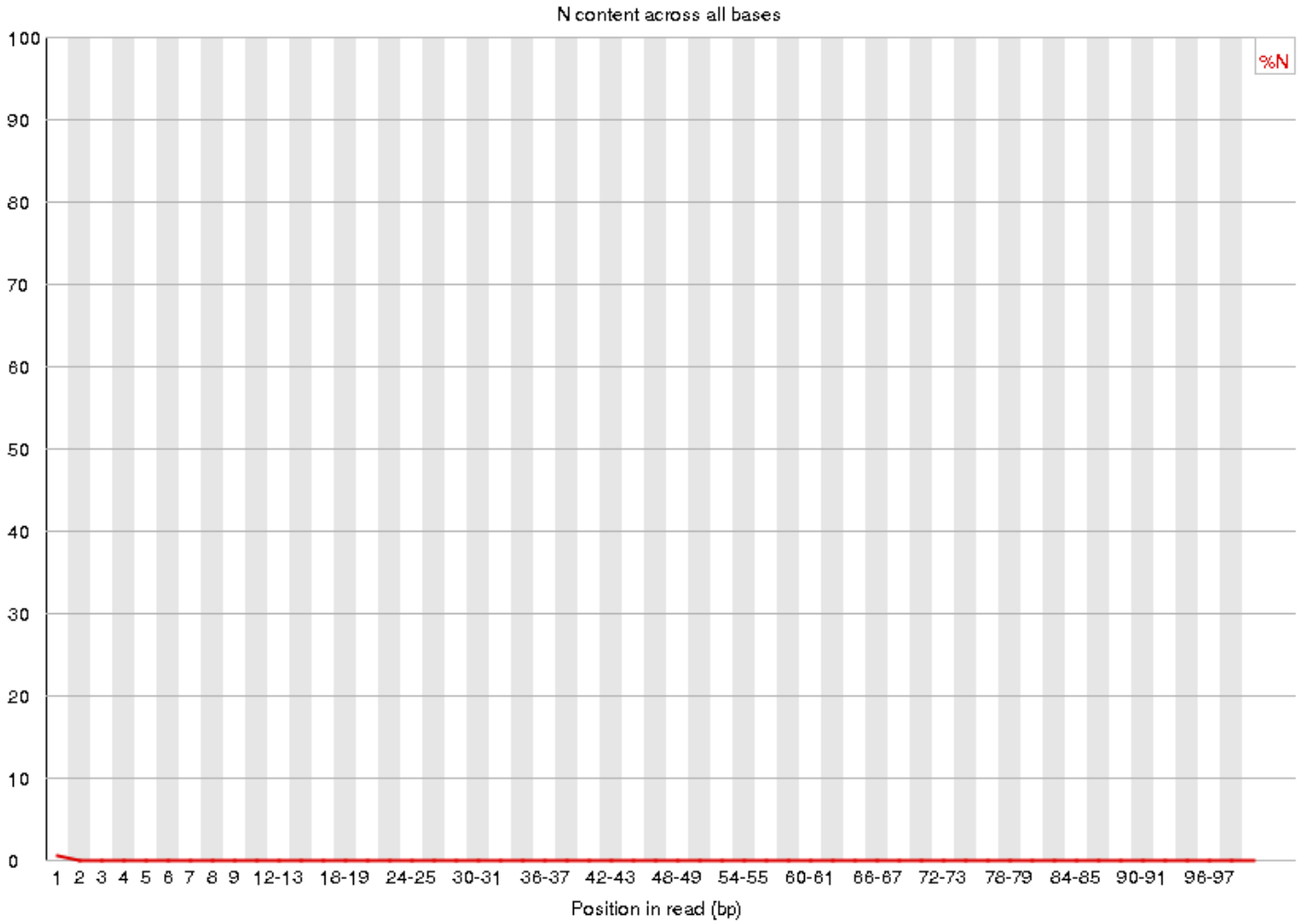
R2



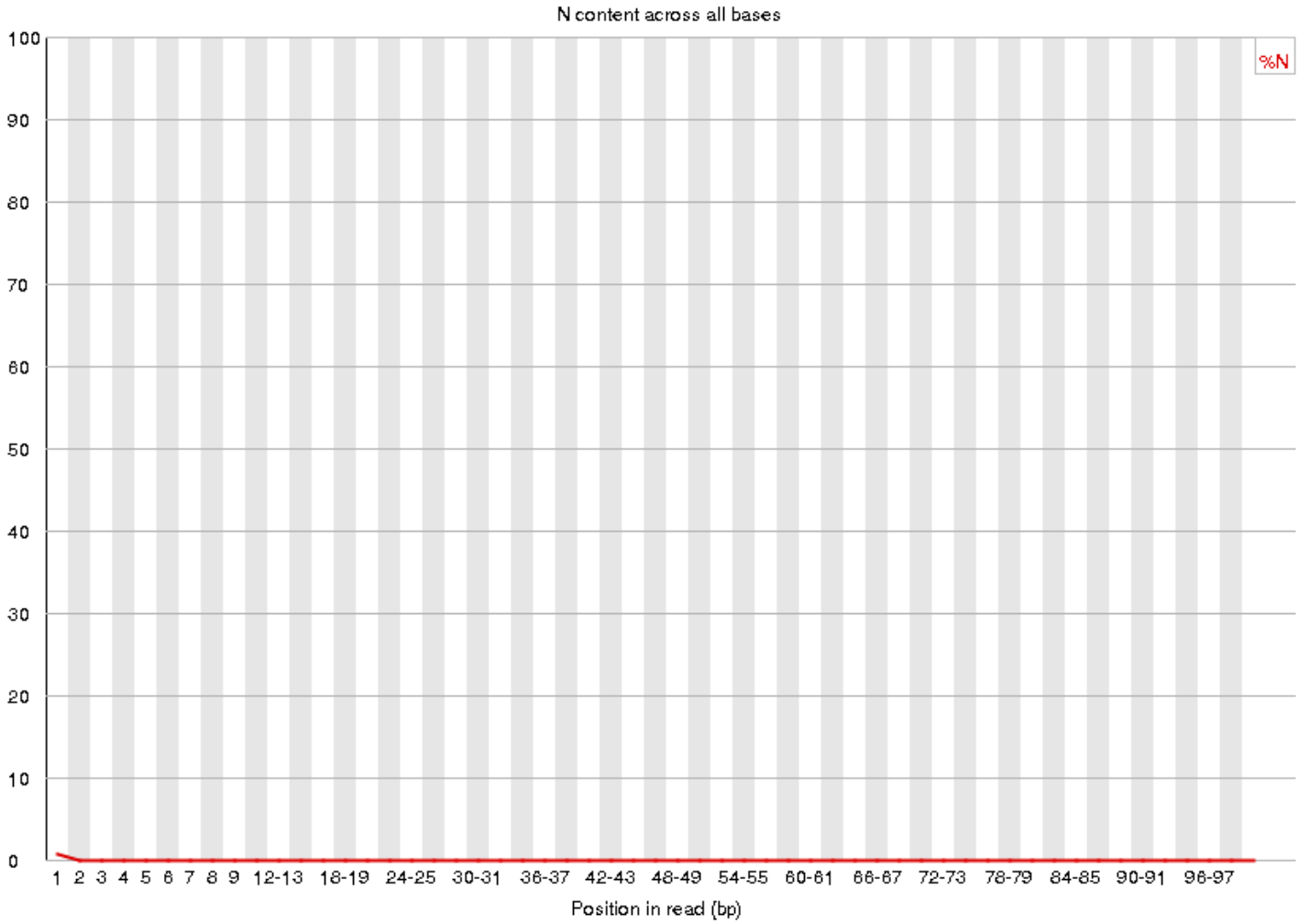
3. per-base N content

Plots from FastQC

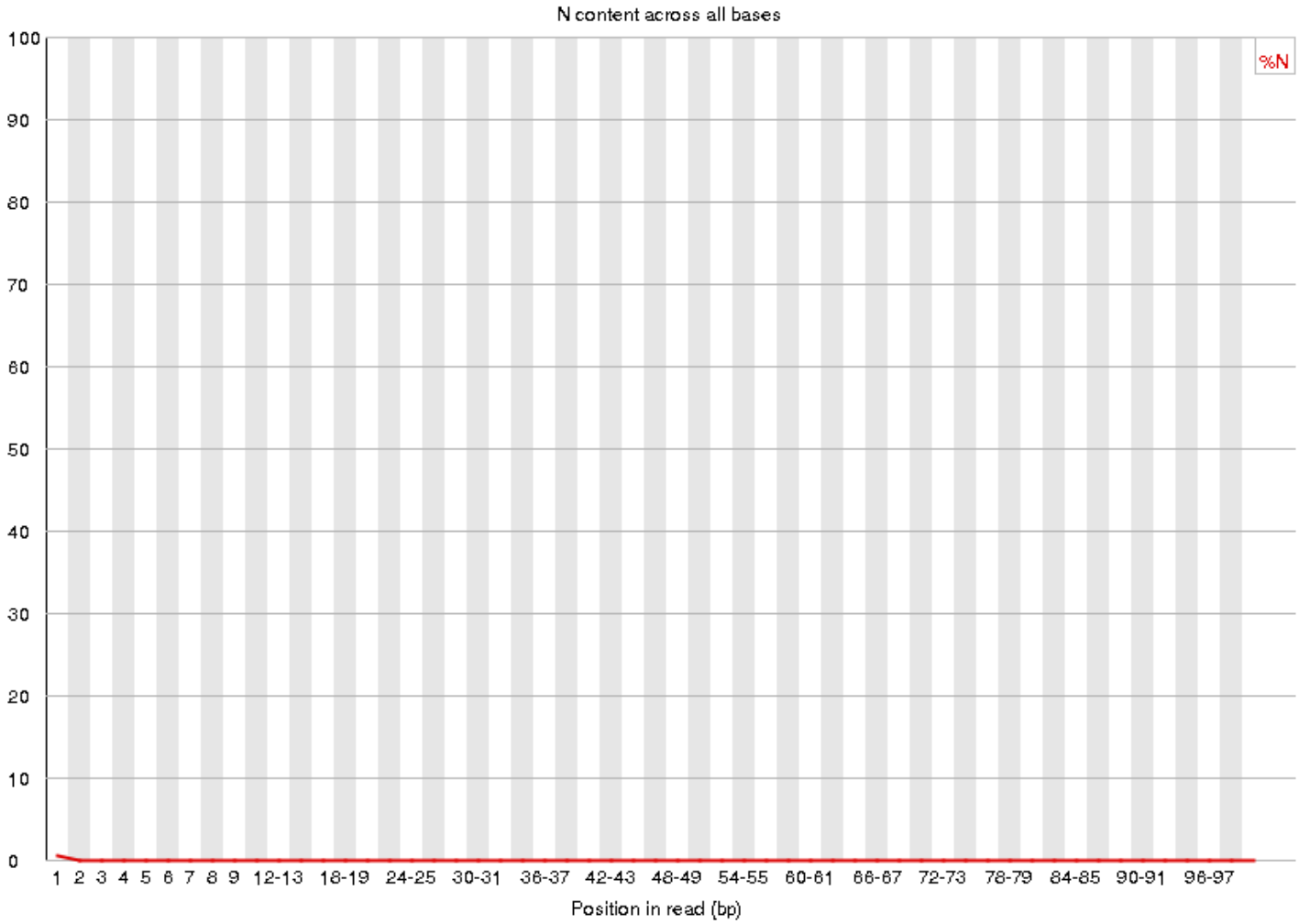
24_4A_R1


N content across all bases
%N
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 12-13 18-19 24-25 30-31 36-37 42-43 48-49 54-55 60-61 66-67 72-73 78-79 84-85 90-91 96-97
Position in read (bp)

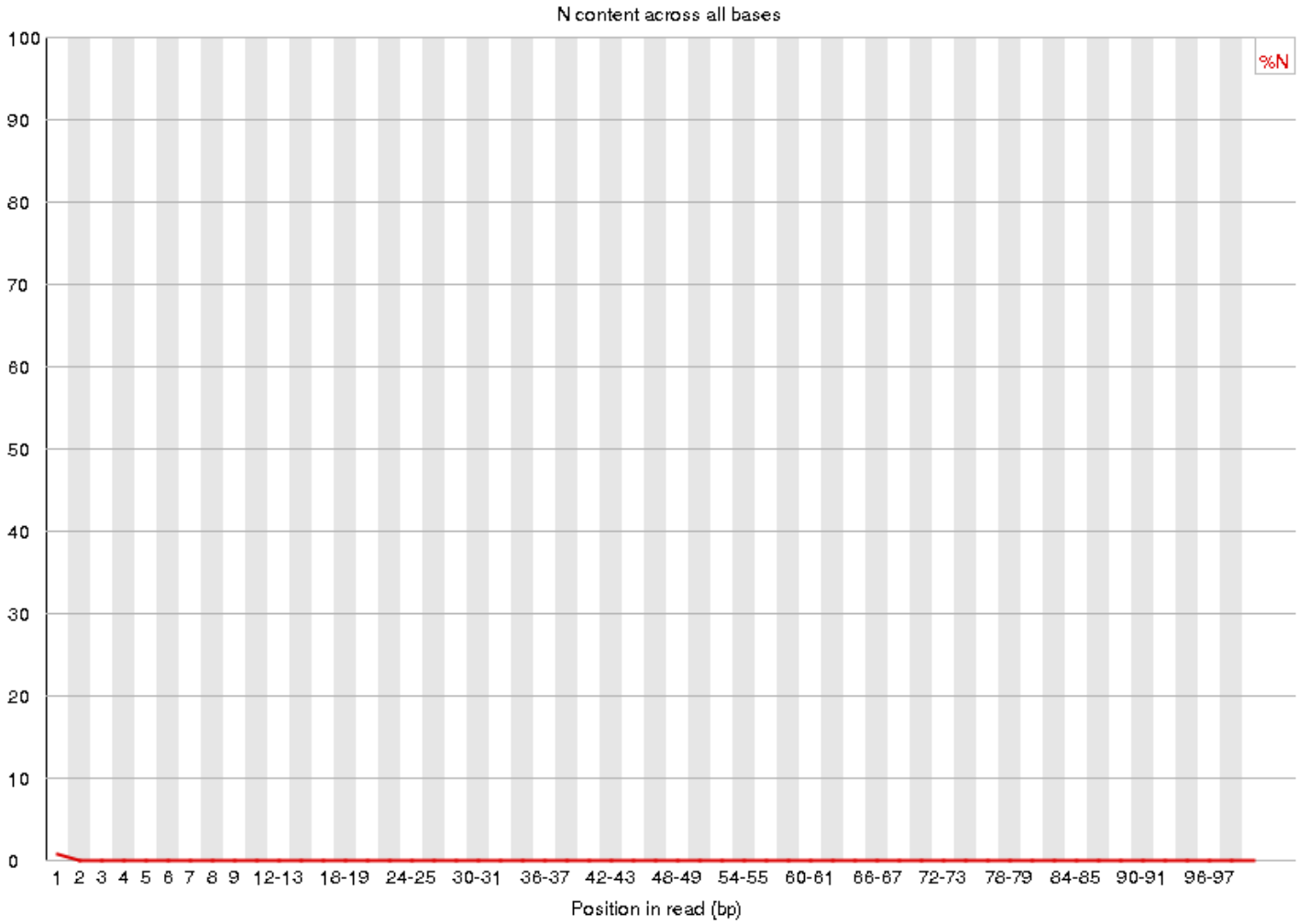
24_4A_R2


N content across all bases
%N
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 12-13 18-19 24-25 30-31 36-37 42-43 48-49 54-55 60-61 66-67 72-73 78-79 84-85 90-91 96-97
Position in read (bp)

34_4H_R1


N content across all bases
%N
100
90
80
70
60
50
40
30
20
10
0
1 2 3 4 5 6 7 8 9 12-13 18-19 24-25 30-31 36-37 42-43 48-49 54-55 60-61 66-67 72-73 78-79 84-85 90-91 96-97
Position in read (bp)

34_4H_R2



The time it took to run four files with `FastQC` and my own script:

	FastQC	**My script**
real	5m31.513s	90m52.311s
user	5m19.604s	90m49.511s
sys	0m22.280s	0m4.631s

The plots generated from `FastQC` and my own script for the mean quality score per bp position look similar. I changed the axes of my own plots from 0-40 so that it matched what `FastQC` outputs. Similarly, when comparing the distribution of avg quality scores among all of the reads, there were similar results between `FastQC` and my own script. I binned the quality scores in my plots, but the overall trend in that the majority of the reads have an average high quality score (35-40).

In all 4 files, there is a small percentage reads that had Ns at the beginning. This is consistent with the lower quality scores at those positions as well. Overall, `FastQC` was much quicker to run (5:31 v 90:52) and not only did it generate mean scores/bp position, N% content, but also GC content, sequence length distribution, and much more, in the same amount of time (along with plots). My script only generates values in which I needed to

import into R in order to plot. One reason `FastQC` is quicker is because it utilizes Java instead of Python and can be multi-threaded. My script probably wasn't written to output the data I want efficiently either (see `my_qual_script.py` in `scripts/`).

Part 2 – Adaptor trimming comparison

3. Look into the adaptor trimming options for `cutadapt`, `process_shortreads`, and `Trimmomatic` (all on Talapas), and briefly describe the differences. Pick one of these to properly trim adapter sequences. Use default settings. What proportion of reads (both forward and reverse) was trimmed?

What are the three programs to compare?

cutadapt

```
Homepage: http://opensource.scilifelab.se/projects/cutadapt/

Cutadapt finds and removes adapter sequences, primers, poly-A tails and
       other types of unwanted sequence from your high-throughput sequencing reads.
```

Example of `cutadapt` with our data. `-A` and `-p` arguments indicates that the input files are paired:

```
ml easybuild  icc/2017.1.132-GCC-6.3.0-2.27  impi/2017.1.132  cutadapt

dir=/home/dho/Bi624/assignments/SF_seq/seq_files

cutadapt -a ADAPTER1 -A ADAPTER2 -o R1_24.fastq -p R2_24.fastq $dir/24_4A_control_S18_L0
08_R1_001.fastq.gz $dir/24_4A_control_S18_L008_R2_001.fastq.gz
```

To load on Talapas:

```
$ ml easybuild  icc/2017.1.132-GCC-6.3.0-2.27  impi/2017.1.132  cutadapt
```

The version on Talapas is: `1.14`

Trimmomatic

```
Description:
      Trimmomatic performs a variety of useful trimming tasks for illumina
      paired-end and single ended data.The selection of trimming steps and
      their associated parameters are supplied on the command line.
```

To load on Talapas:

```
$ ml easybuild Trimmomatic
```

To execute:

```
To execute Trimmomatic run: java -jar $EBROOTTRIMMOMATIC/trimmomatic-0.36.jar
```

The version on Talapas: `0.36`

process_shortreads

To load on Talapas:

```
$ ml slurm easybuild intel/2017a Stacks/1.46
```

The version on Talapas is: `1.46`

One of the biggest difference between the three programs are the languages they are written in. `cutadapt` uses Python, `process_shortreads` uses C++ & Perl, and `Trimmomatic` uses Java. `process_shortreads` also dumps low quality reads into a file (orphaned). From reading various documentations and articles, it seems like between `Trimmomatic` and `cutadapt` , the former is quicker at performs its task. The input/arguments varies between the programs. For example, `Trimmomatic` requires adapter sequences to be input as `fasta` files, while the other two has the user input the queried sequence right in the command line. All programs have the ability to adjust the tolerance for mismatches.

For the following adapter trimming, I used `process_shortreads` .

The adapters used were:

R1: AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC

R2: AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT

See `process_sreads_24_34.srun` in `scripts/` for the commands.

How many records were retained after trimmed (see files in `part2_log_files/`):

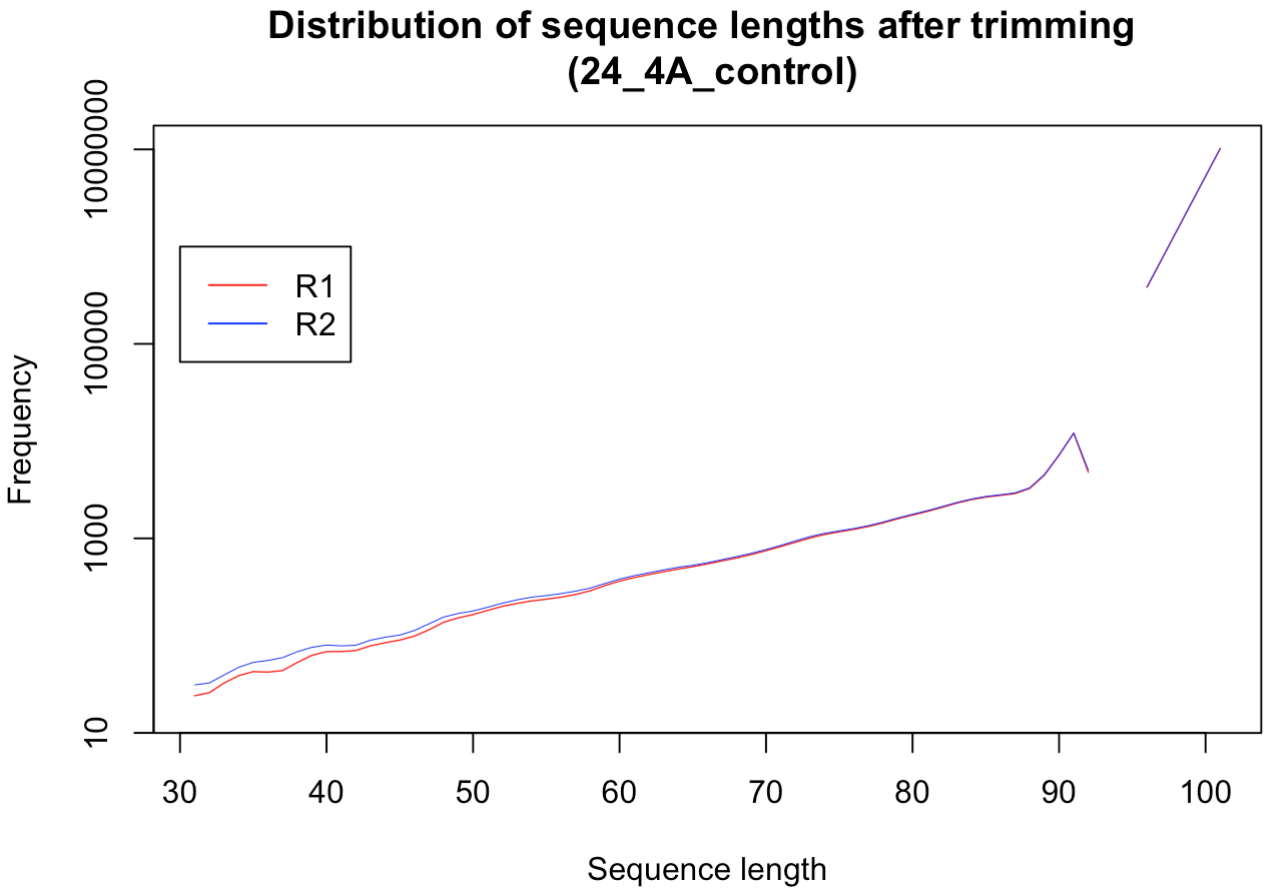
Library	Pre-trim reads (R1+R2)	Trimmed reads	% trimmed
24_4A_control	21031748	228960	1.09
34_4H_both	18081194	1177277	6.51

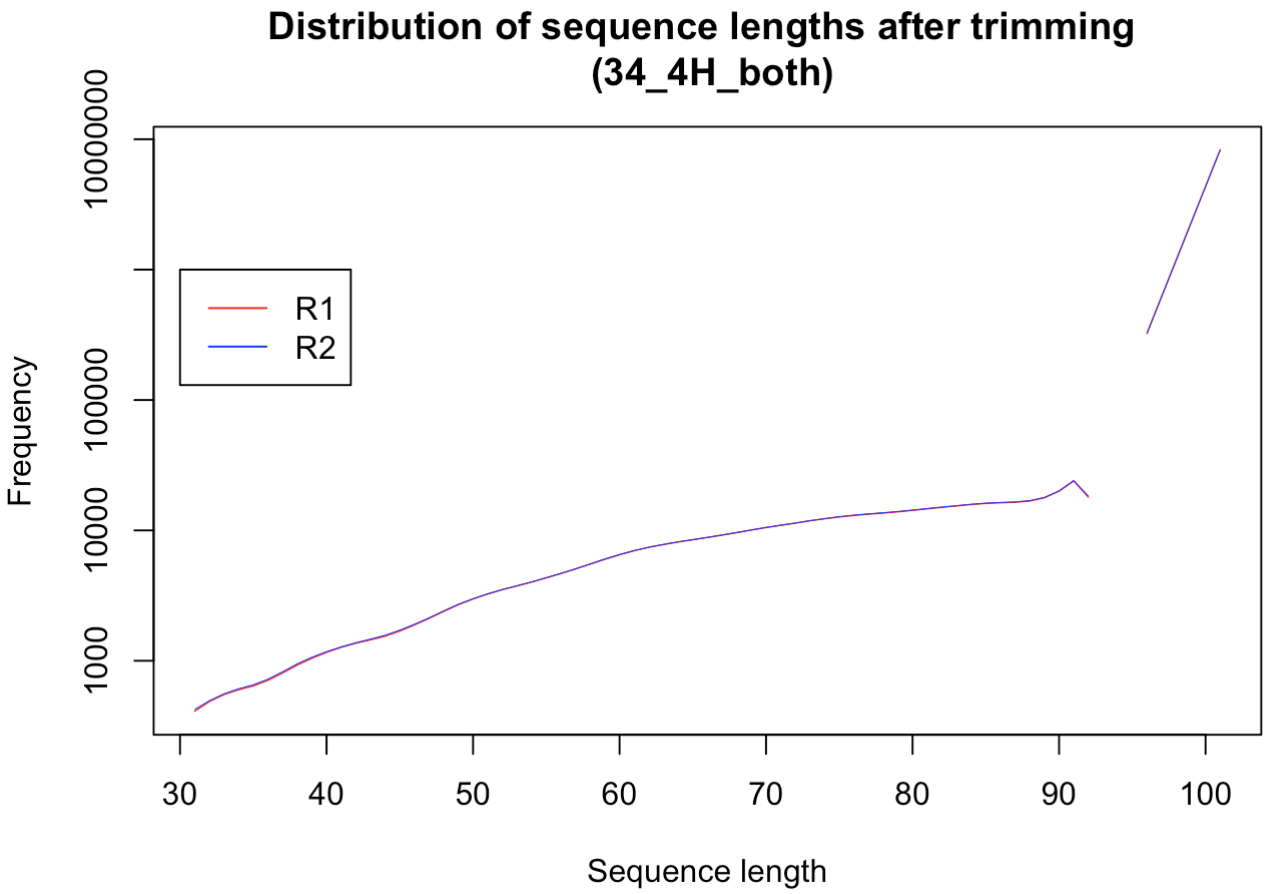
Sanity check: Use your Unix skills to search for the adapter sequences in your datasets and confirm the expected sequence orientations.

```
$ grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC" 24_4A_control_S18_L008_R1_001.fastq | wc -l
7417

$ grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" 24_4A_control_S18_L008_R2_001.fastq | wc -l
8484

$ grep "AGATCGGAAGAGCACACGTCTGAACTCCAGTCAC" 34_4H_both_S24_L008_R1_001.fastq | wc -l
129475

$ grep "AGATCGGAAGAGCGTCGTGTAGGGAAAGAGTGT" 34_4H_both_S24_L008_R2_001.fastq | wc -l
137879
```

When using `grep` for the whole adapter sequence, I saw that for the most part, the adapters were found towards the end of the sequence reads. Because I looked for the whole adapter sequence and not just parts of it, I got fewer hits than the amount `process_shortreads` trimmed.

4. Plot the trimmed read length distributions for both forward and reverse reads (on the same plot). If necessary, consult Assignment 5 (Block 1) from Bi 623 to refresh your memory.



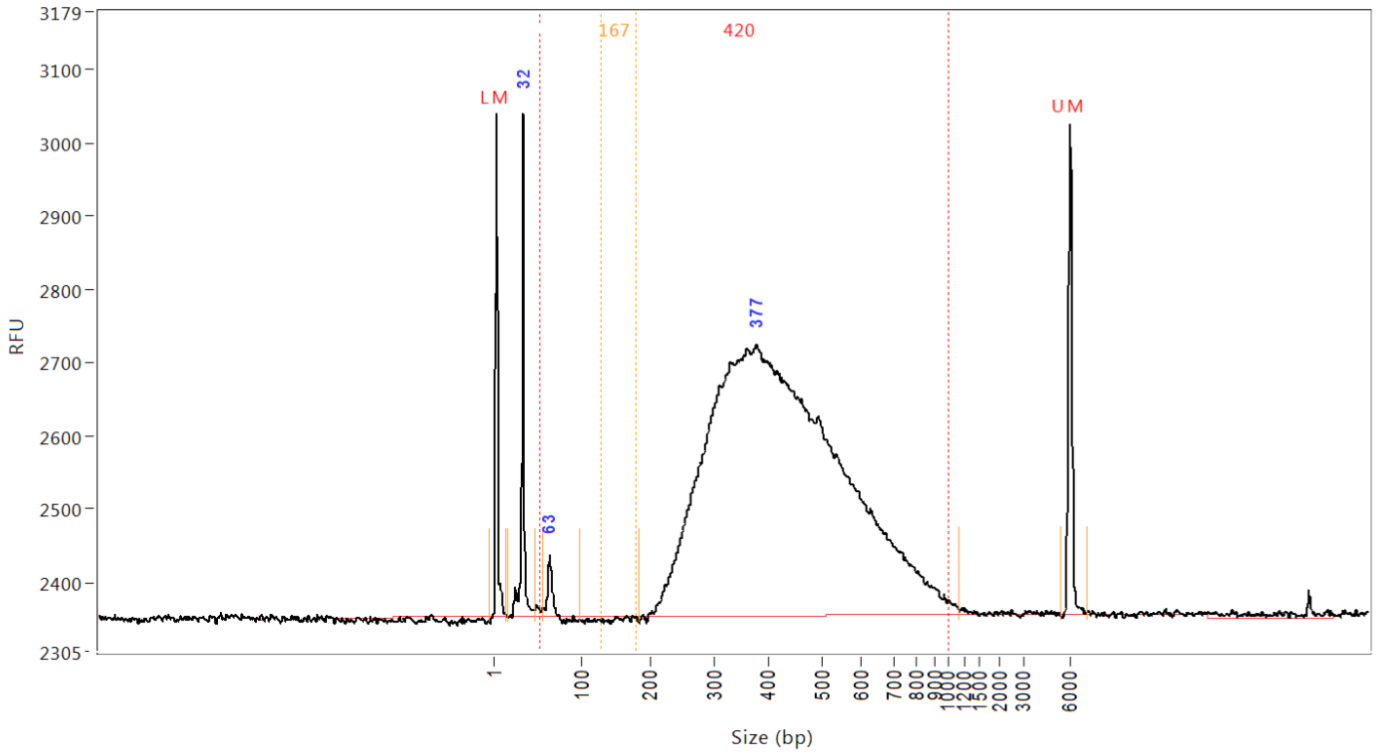


The distribution for the 4 files range from about 30 to 101, indicating that some of the reads had the whole adapter in the read (a very small propotion) and some of the reads had parts of the adapters.

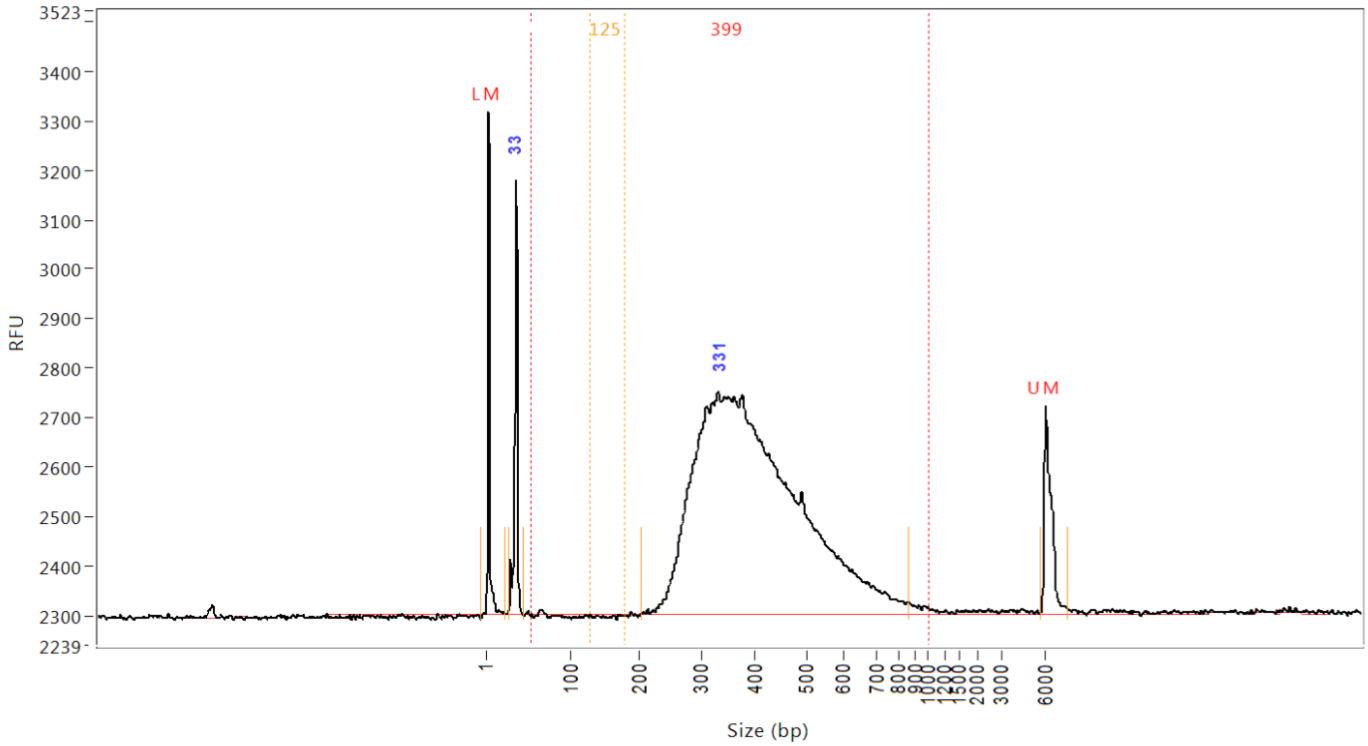
5. Briefly describe whether the adaptor trimming results are consistent with the insert size distributions for your libraries. The size distribution information is in the Fragment Analyzer trace file on Github.

The following are electropherograms of the two libraries:

24_4A_control



34_4H_both



We sized selected our inserts during the preparation for 400-500bp inserts, but from these two electropherograms, we can see that the majority of the molecules were around 331-377 bp. The distribution of the molecule size includes both the adapters & inserts.

Our adapters are 140bp in total and we are sequencing 101 bp, so anything library molecule under 241bp should contain adapters. For both electropherograms, we see that there is an area under the curve that is less than 241bp. I'm surprised that `process_shortreads` only trimmed 1.09% and 6.51% for the 24_4A_control and

34_4H_both libraries, respectively. I do not think the trimming results is consistent with what is seen from the fragment analyzer. I also only used the default settings so adding more arguments to make the trimming more stringent might increase the amount of trimming.

Part 3 – rRNA reads and strand-specificity

6. Find publicly available mouse rRNA sequences and generate a gsnap database from them. Align the SF-Seq reads to your mouse rRNA database and report the proportion of reads that likely came from rRNAs.

Download non-coding RNA sequences from Ensembl and filter for rRNA sequences:

```
$ awk '!/^>/ { printf "%s", $0; n = "\n" } /^>/ { print n $0; n = "" } END { printf "%
s", n }' Mus_musculus.GRCm38.ncrna.fa | grep "rRNA" -A 1 | grep -v "^--" > mouse_rRNA.fa

$ grep "^>" mouse_rRNA.fa | wc -l
     358
```

There are 358 rRNA sequences.

I used this mouse-rRNA fasta file to generate a database and did some alignment with `gsnap` (see `mouse_rRNA.srun` in `scripts/`)

Look at `.nomapping` files: how many reads did NOT align to rRNA:

```
$ cat gsnap_24.nomapping | grep -v "^@" | cut -d "       " -f 1| sort | uniq -c | wc -l
10084174
```

```
$ cat gsnap_34.nomapping | grep -v "^@" | cut -d "       " -f 1| sort | uniq -c | wc -l
8987301
```

Library	Total reads	% Aligned to rRNA
24_4A_control	10515874	4.11
34_4H_both	9040597	0.59

Interestingly, there were more reads that aligned to one of the libraries. One explanation for rRNA contamination could be during the library prep. We selected for poly-A tails but pipetting errors could have led to some rRNA being passed onto the next step as well.

7. Demonstrate convincingly that the SF-Seq data are from “strand-specific” RNA-Seq libraries. There are a number of possible strategies to address this problem, but you need only implement one. Report your evidence in numeric and graphical (e.g. a plot) forms.

Because these are mRNA-seq libraries and our selection for the mRNA was an enrichment for poly-A tails, we would assume one of the files would have more poly-A than poly-Ts, and vice-versa.

In this case, I considered any string of 15 or more As & Ts to be searched for:

```
$ for letter in A T; do echo $letter; grep -E -e "$letter{15,}" 24_4A_control_S18_L008_R
1_001.fastq| wc -l; done
A
11124
T
15124

$ for letter in A T; do echo $letter; grep -E -e "$letter{15,}" 24_4A_control_S18_L008_R
2_001.fastq| wc -l; done
A
10647
T
31116

$ for letter in A T; do echo $letter; grep -E -e "$letter{15,}" 34_4H_both_S24_L008_R1_0
01.fastq | wc -l; done
A
13693
T
25955

$ for letter in A T; do echo $letter; grep -E -e "$letter{15,}" 34_4H_both_S24_L008_R2_0
01.fastq | wc -l; done
A
19221
T
24915
```

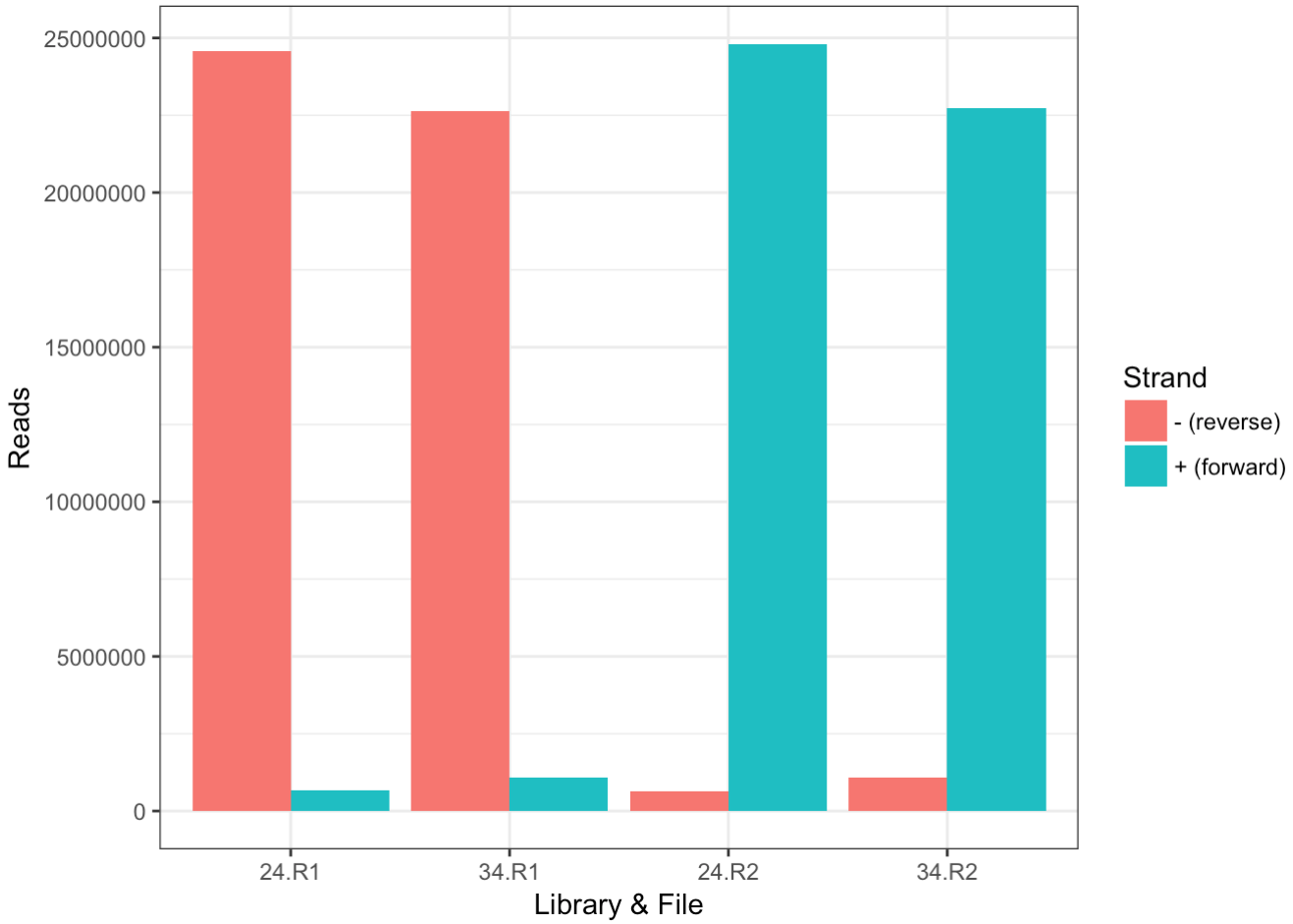
From all of those files, there were more reads that contain poly-Ts than poly-As.

Another approach I used was creating a `gmap` of the mouse cDNA and aligning the reads so that we get a `gff` file (see `cDNA_24.srun` & `cDNA_34.srun` in `scripts/`). The GFF file has a field that contains information about the “stranded-ness,” whether it is a `+` or `-`. If the libraries are stranded, we should see that a majority of reads from one file containing `+` and the other `-`.

```
$ sed 1,2d 24_R1_GFF.gff3 | grep -v "##" | cut -f 7 | sort | uniq -c
24581781 -
 658155 +

$ sed 1,2d 24_R2_GFF.gff3 | grep -v "##" | cut -f 7 | sort | uniq -c
 637093 -
24808586 +

$ sed 1,2d 34_R1_GFF.gff | grep -v "##" | cut -f 7 | sort | uniq -c
22635802 -
1077677 +

$ sed 1,2d 34_R2_GFF.gff | grep -v "##" | cut -f 7 | sort | uniq -c
1071286 -
22733866 +
```

Graphical representation of the reads in a file determined `+` or `-`:

```
## Scale for 'fill' is already present. Adding another scale for 'fill',
## which will replace the existing scale.
```



For both libraries (24 & 34), the R1 file contained more reads assigned to the reverse strand, while the R2 files have the majority of their reads assigned to the forward strand. Because of this, these libraries are strand-specific.