Introduction to qckitfastq

August Guang and Wenyue Xing

Introduction

`qckitfastq` is part of the planned `qckit` suite of packages from the Computational Biology Core at Brown University. The long-term goal of the qckit suite is to not only provide comprehensive quality control metrics for common genomics sequencing workflows, but to 1) also provide quality control visualizations for multiple samples through updated modules in multiQC (Ewels et al. 2016), and 2) provide a public quality control database which allows benchmarking of QC metrics for an experiment against other similar experiments. Users of `qckitfastq` will have the option to enter SRA metadata and create visualizations from our database as well as submit their qc results (if submitting an SRA archive) to our database once these features have been implemented. The purpose of this particular package is to run quality control on FASTQ files from genome sequencing.

Why use qckitfastq?

Indeed there are many other quality control packages for FASTQ files existing already, including ShortReads (Morgan et al. 2009) and seqTools (Kaisers 2017) for R and the popular FASTQC (Andrews 2010) Java-based program. qckitfastq offers a few advantages compared to these 3 programs for users who need such features:

1. access to raw sequence and quality data
2. access to data frames of the quality control results in addition to plots
3. quality control analyses of the entire FASTQ file
4. fast file processing

To break it down further, seqTools and ShortReads do not offer as comprehensive set of quality control metrics as qckitfastq and FASTQC. seqTools further provides limited access to raw data and intermediate analysis results. ShortRead provides users with access to the raw sequencing data and intermedite analysis results, but is inefficient on datasets exceeding 10 million reads. FASTQC meanwhile truncates any reads longer than 75bp as well as estimates overall quality only based on the first 100,000 reads of any FASTQ file. `qckitfastq` does not contain any of these limitations.

Running qckitfastq

`qckitfastq` provides the following metrics:

- Read length distribution
- Per base read quality
- Per read read quality
- GC content
- Nucleotide read content
- Kmer count
- Overrepresented reads
- Overrepresented kmers

- Adapter content

The simplest way to run `qckitfastq` is by executing `run_all`, a single command that will produce a report of all of the included metrics in a user-provided directory with some default parameters. However, each metric can also be run separately for closer examination.

Individual metrics

For each individual metric that `qckitfastq` provides, the user can choose to save either the data frame for the metric, the associated plot, or both.

Our example in this vignette has 25,000 reads, each 100bp long. The majority of metrics are run on the path to the FASTQ file. Some functions for quality control in this package are simply wrappers around `seqTools` due to the fact that their functions are fast. We provide these wrappers for the sake of completeness in quality control metrics. These wrappers require processing the FASTQ data through the `seqTools::fastqq` command first:

```
library(qckitfastq)
infile <- system.file("extdata", "10^5_reads_test.fq.gz", package = "qckitfastq")
fseq <- seqTools::fastqq(infile)
```

```
## [fastqq] File ( 1/1) '/tmp/RtmpwXAdlH/Rinst4db5ad04162/qckitfastq/extdata/10^5_reads_test.fq.gz' don
```

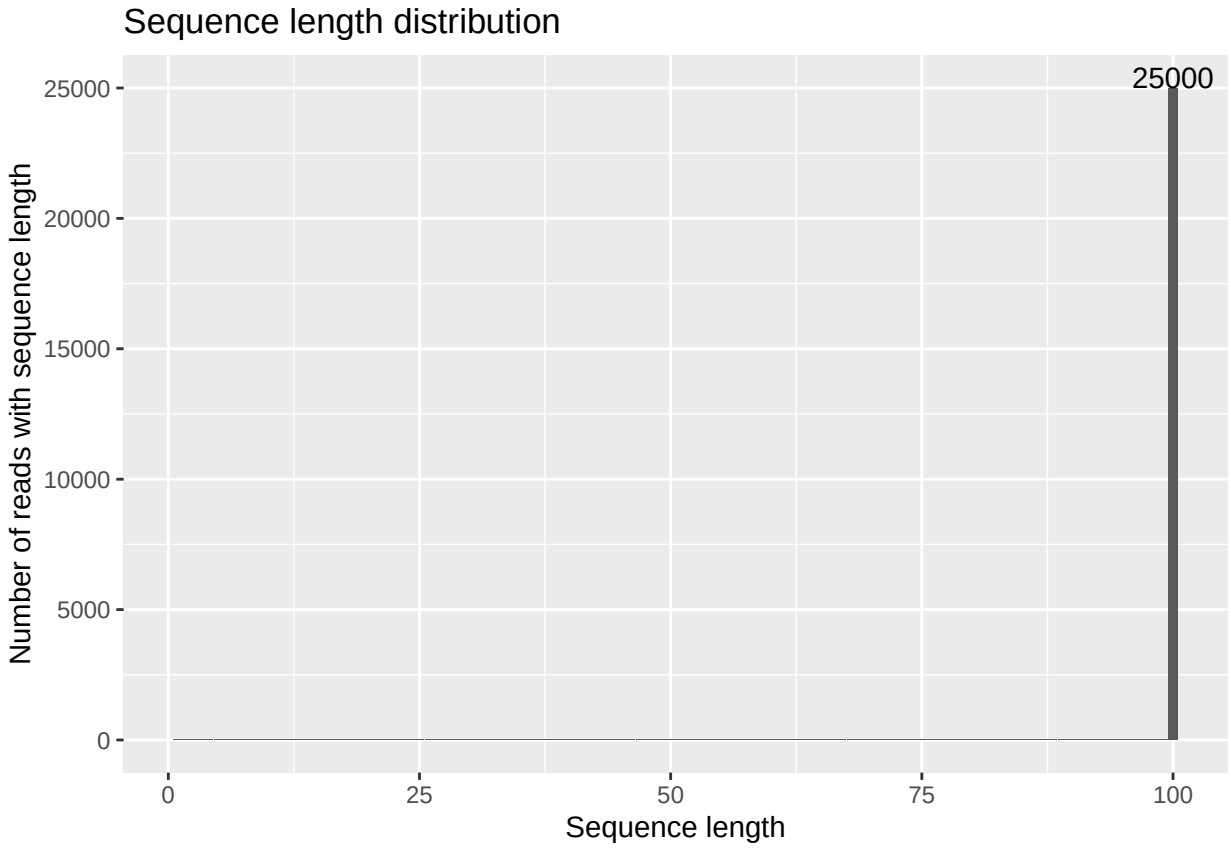
Read length distribution

read_length generates a data frame of read lengths and the number of reads with that particular read length. *plot_read_length* generates a distribution plot of the length of all reads. The generated plot would show the sequence length of all the sequences throughout the file. The plot is considered an indication of good data quality is all sequences have roughly the same sequence length with minimal deviations. The following plot shows that all reads in the file have sequence length of 100.

```
read_len <- read_length(fseq)
kable(head(read_len)) %>% kable_styling()
```

read_length	num_reads
1	0
2	0
3	0
4	0
5	0
6	0

```
plot_read_length(read_len)
```



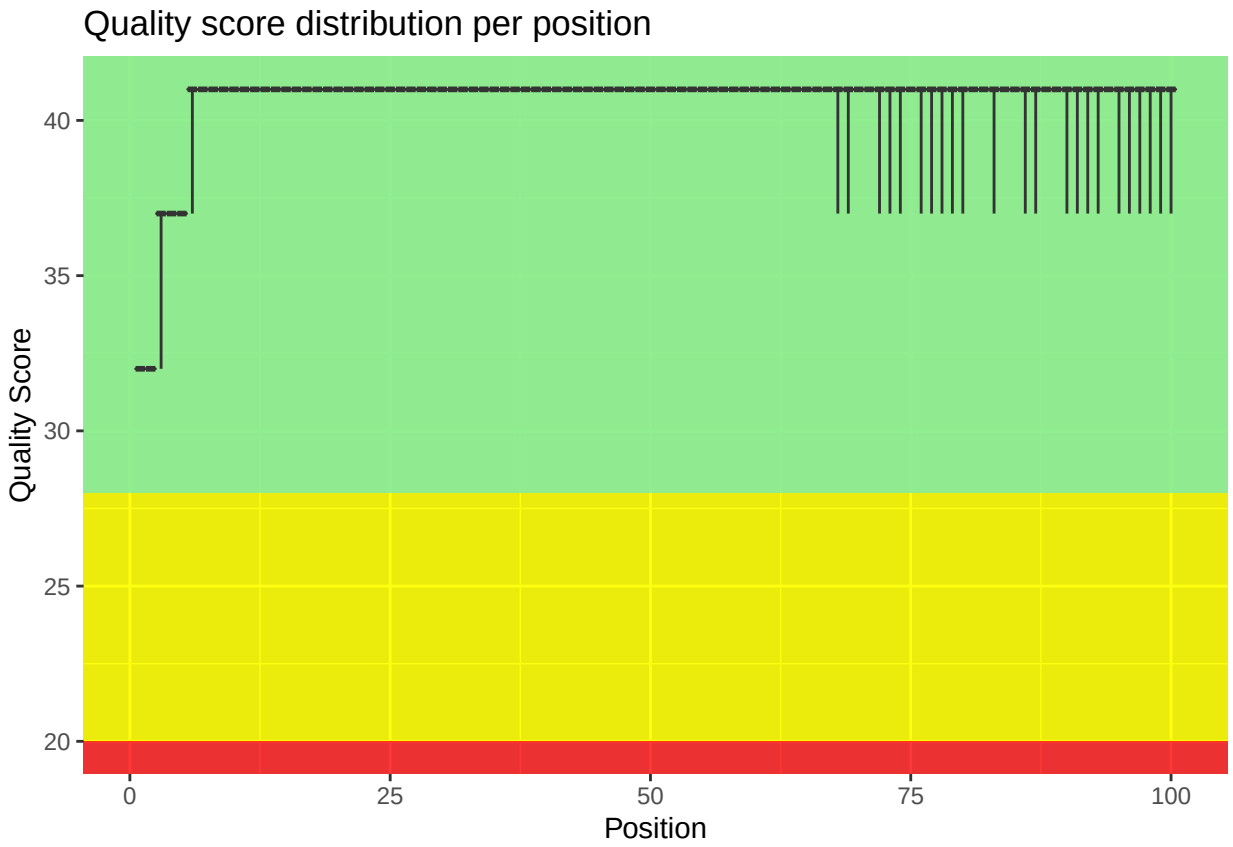
Per base quality

The *per_base_quality* function calculates quality score per base summary statistics of 10th, 25th, median, 75th and 90th quantiles across sequences. Currently it treats all quality score encodings as Phred+33. We can use the result to create a quality score distribution per position plot using *plot_per_base_quality*. As a basic heuristic, quality scores above 28 can be categorized as good (green), those from 20 to 28 can be categorized as medium (yellow), and under 20 is bad (red).

```
bs <- per_base_quality(infile)
kable(head(bs)) %>% kable_styling()
```

position	q10	q25	median	q75	q90
1	32	32	32	32	32
2	32	32	32	32	32
3	32	37	37	37	37
4	37	37	37	37	37
5	37	37	37	37	37
6	37	41	41	41	41

```
plot_per_base_quality(bs)
```



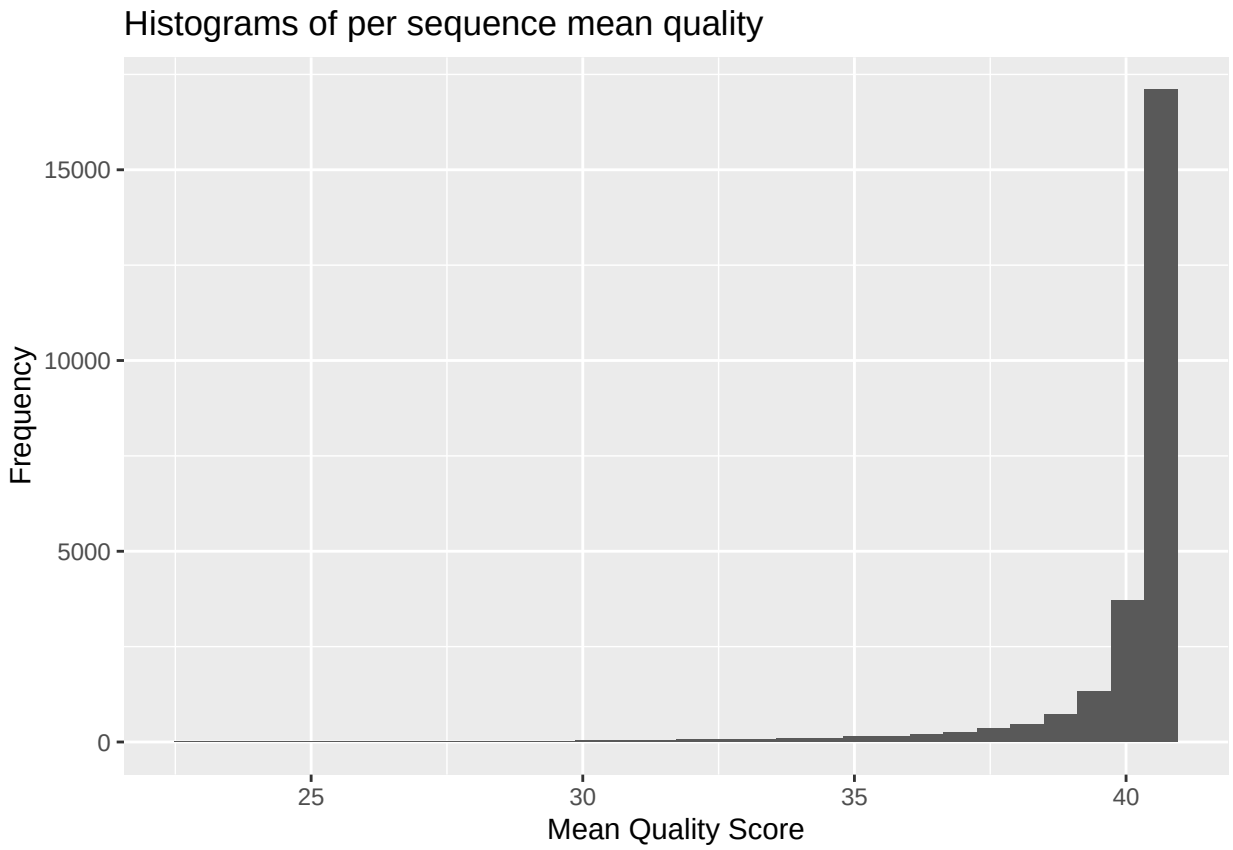
Per read quality score

The *per_read_quality* function compute the mean quality score per read. We can then use *plot_per_read_quality* to generate a histogram of this statistics. The histogram is considered an indication of good data quality if the majority of reads have mean quality scores greater than 30. If a significant portion of reads have quality scores less than 30, then the data most likely has issues that need to be examined.

```
prq <- per_read_quality(infile)
kable(head(prq)) %>% kable_styling()
```

read	sequence_mean
1	39.11
2	38.15
3	40.35
4	40.35
5	39.83
6	40.40

```
plot_per_read_quality(prq)
```



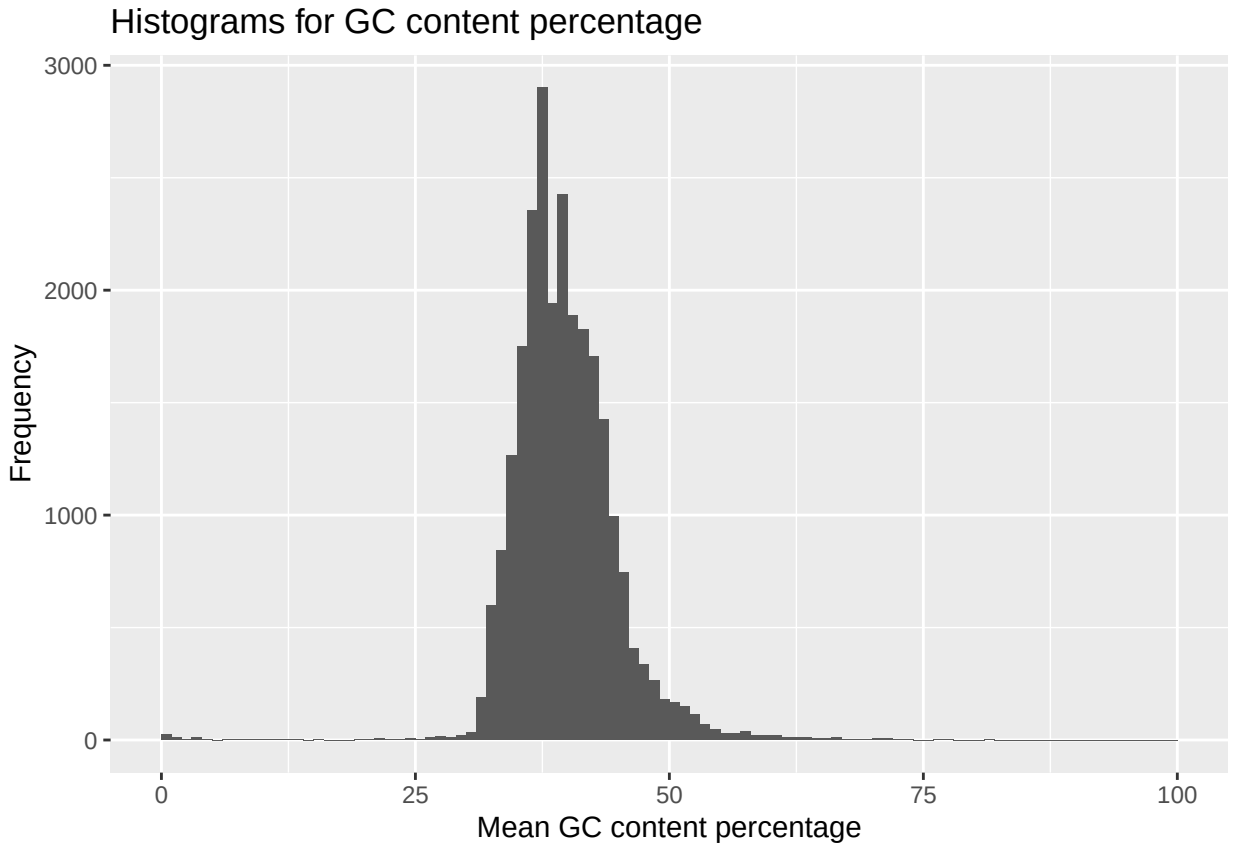
Per read GC content

The *GC_content* function computes the GC nucleotide content percentage per read, and *plot_GC_content* plots the distribution of GC content. As a general rule, an indication of good data quality is when the GC content percentage in each read is between 30 and 50% and roughly follows a normal distribution.

```
gc_df <- GC_content(infile)
kable(head(gc_df)) %>% kable_styling()
```

read	mean_GC
1	43
2	37
3	39
4	40
5	42
6	38

```
plot_GC_content(gc_df)
```



Per position nucleotide read content

read_content calculates the total numbers across reads of each nucleotide base by position. *plot_read_content* plots the percentage of nucleotide content per position. We also provide an additional function *read_base_content* that allows the user to get the nucleotide base content by position across reads for a specific nucleotide (options are a,c,t,g,n). As a general rule, the plot would be considered an indication of good data quality when the percentage of each nucleotide sequence content is about evenly distributed across all bases. However, there are some types of analyses for which this will not be the case. For example, RNA-Seq will have an uneven sequence content distribution in the first 10 bases, and RRBS will have almost no cytosines and very high thymine content because the library prep protocol converts most C to T (Meissner et al. 2005). Knowledge of the library prep protocol is thus important for evaluating quality in terms of nucleotide sequence content.

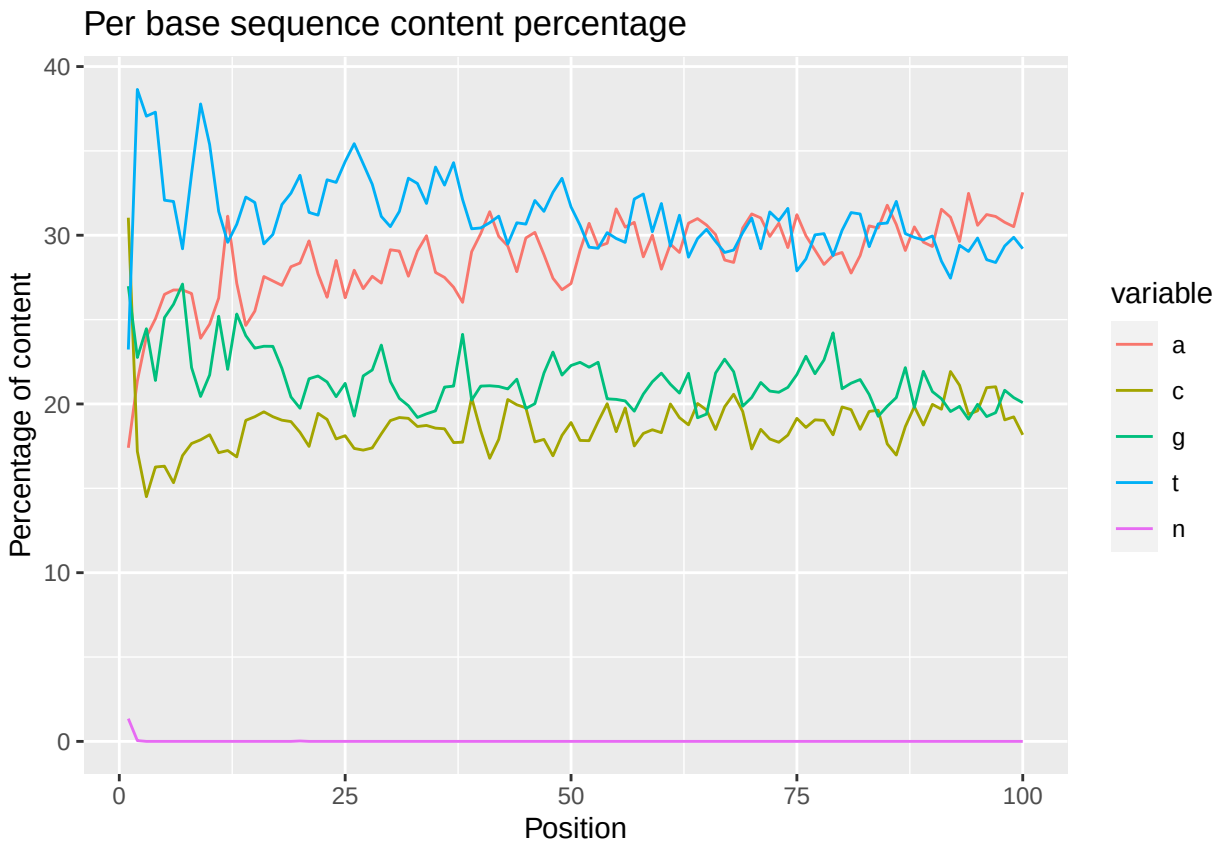
```
scA <- read_base_content(fseq, content = "A")
kable(head(scA)) %>% kable_styling()
```

x
4351
5341
5995
6262
6624
6689

```
rc <- read_content(fseq)
kable(head(rc)) %>% kable_styling()
```

position	a	c	t	g	n
1	4351	7760	5807	6745	337
2	5341	4297	9661	5689	12
3	5995	3626	9264	6115	0
4	6262	4066	9324	5348	0
5	6624	4078	8019	6279	0
6	6689	3833	8001	6477	0

```
plot_read_content(rc)
```



Per position kmer count

kmer_count function produces the per position kmer count with given path to the FASTQ file and the kmer length specified.

```
km <- kmer_count(infile,k=6)
```

```
## [fastq_Klocs] File ( 1/1) '/tmp/RtmpwXAdlH/Rinst4db5ad04162/qckitfastq/extdata/10^5_reads_test.fq.gz
```

```
kable(head(km)) %>% kable_styling()
```

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
AAAAAA	14	9	44	18	33	42	69	163	41	34	15	37	28	49	63	28	29	19	50	30	30
AAAAAC	8	4	1	24	2	9	21	14	142	5	7	1	12	13	18	32	15	4	6	3	11
AAAAAG	2	2	0	3	2	3	2	2	1	1	1	0	1	2	1	1	0	1	4	1	1
AAAAAT	8	28	29	34	44	34	33	65	28	58	43	36	36	21	39	41	75	74	42	99	28
AAAACA	0	2	5	4	4	3	4	6	3	3	3	6	8	3	4	6	5	1	11	8	5
AAAACC	2	8	4	1	25	2	9	21	15	143	4	8	1	11	13	15	32	14	4	6	3

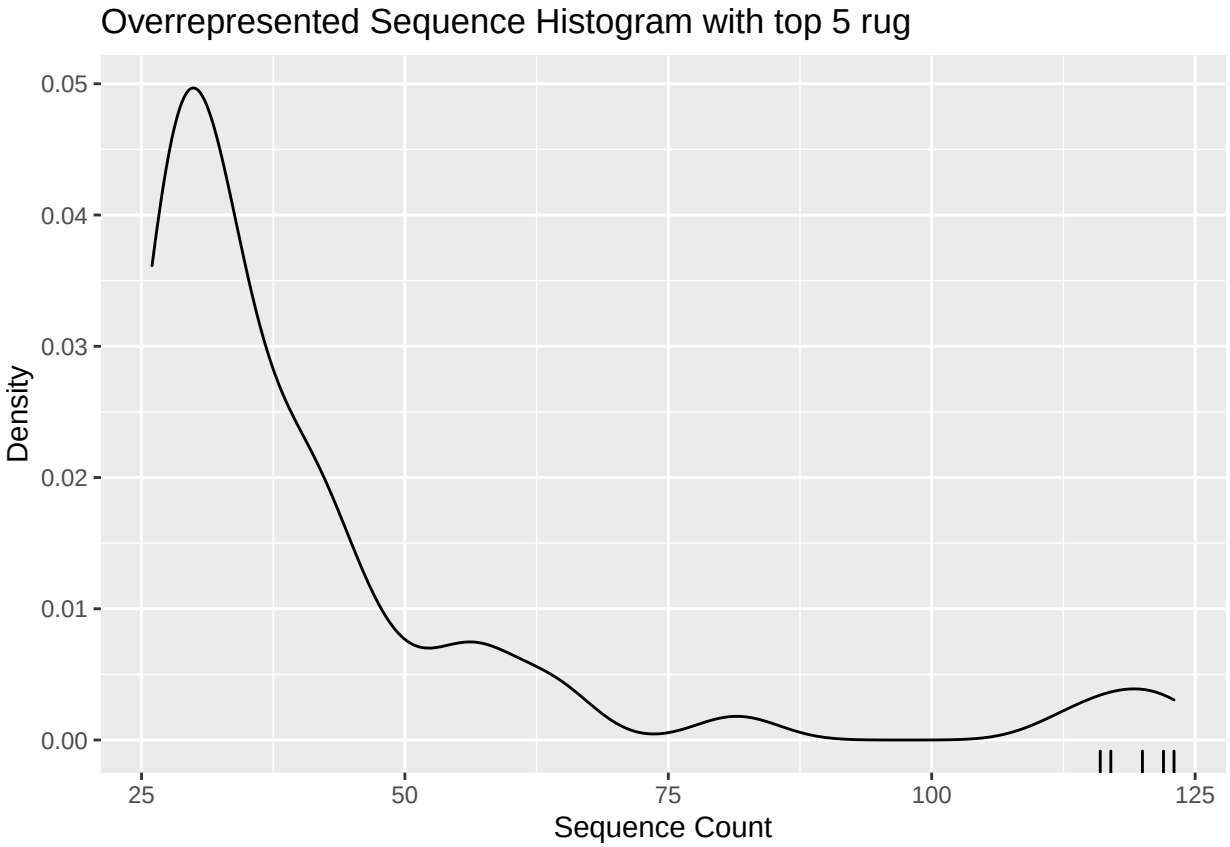
Overrepresented reads

overrep_reads produces a data frame consisting of overrepresented reads and their counts in decreasing order of counts. Here overrepresented is defined as unique reads that have counts larger than 0.1% of the total reads in the file. *plot_overrep_reads* produces a density plot of the counts and marks the top 5 overrepresented reads in red.

```
overrep_reads<-overrep_reads(infile)
knitr::kable(head(overrep_reads,n = 5)) %>% kable_styling()
```

read_sequence
TGGGTGTGAGGAGTTCAGTTATATGTTTGGGATTTTTTAGGTAGTGGGTGTTGAGCTTGAACGCTTTCTTAAT
CCCCAAACCCACTCCACCTTACTACCAGACAACCTTAGCCAAACCATTTACCCAAATAAAGTATAGGCGATAGAA
CACTAGGAAAAAACCTTGTAGAGAGAGTAAAAAATTTAACACCCATAGTAGGCCTAAAAGCAGCCACCAATTAA
CTAAACCTAGCCCCAAACCCACTCCACCTTACTACCAGACAACCTTAGCCAAACCATTTACCCAAATAAAGTATA
TAAACCTAGCCCCAAACCCACTCCACCTTACTACCAGACAACCTTAGCCAAACCATTTACCCAAATAAAGTATAG

```
plot_overrep_reads(overrep_reads)
```



Overrepresented kmers

overrep_kmer generates a data frame of overrepresented kmers with its maximum log2(observed/expected) ratio and the position of the maximum obs/exp ratio in descending order. Only those kmers with a ratio greater than 2 are returned in the data frame. We can also create a boxplot of the obs/exp ratio thatn includes plotting the top 2 (or n) kmer outliers.

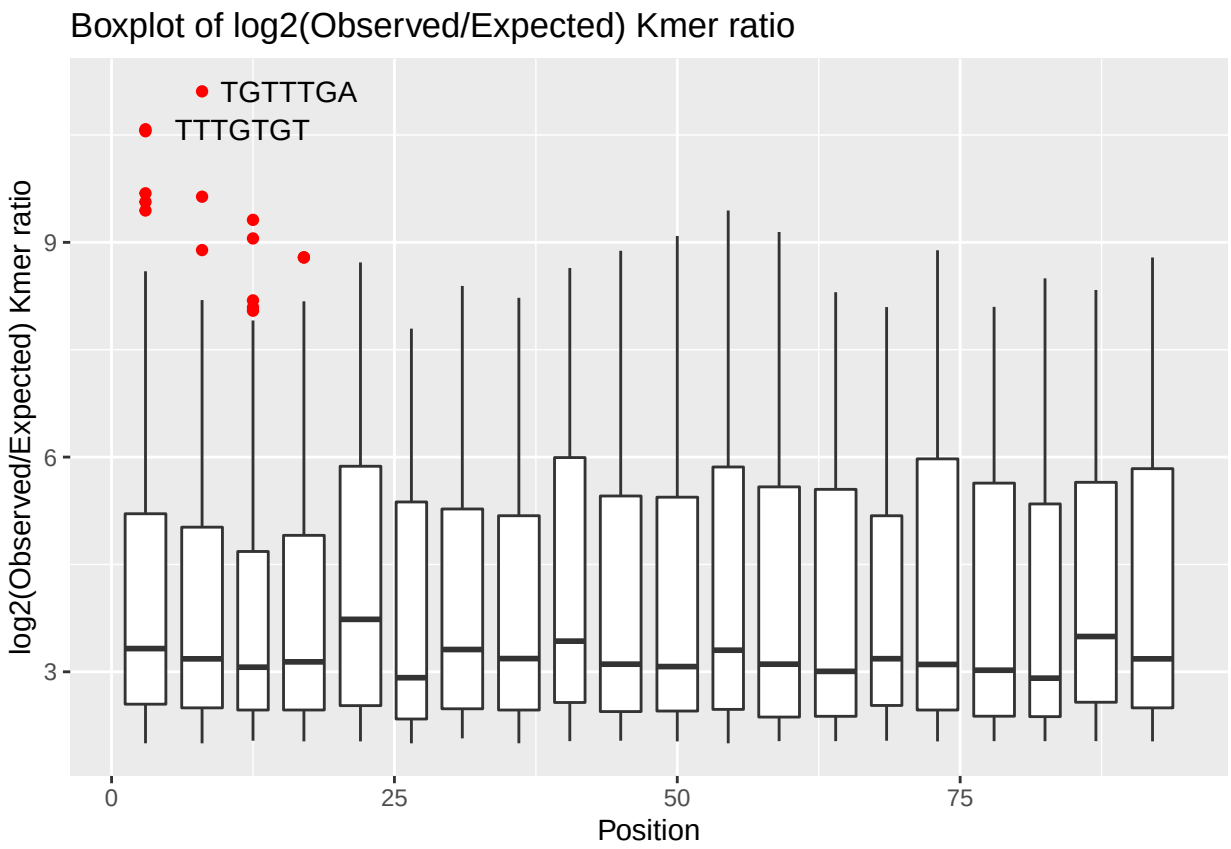
```
overkm <-overrep_kmer(infile,7)
```

```
## [fastqq] File ( 1/1) '/tmp/RtmpwXAdlH/Rinst4db5ad04162/qckitfastq/extdata/10^5_reads_test.fq.gz' don
## [fastq_Klocs] File ( 1/1) '/tmp/RtmpwXAdlH/Rinst4db5ad04162/qckitfastq/extdata/10^5_reads_test.fq.gz
```

```
knitr::kable(head(overkm,n=10)) %>% kable_styling()
```

row	position	obsexp_ratio	kmer
15353	6	11.109619	TGTTTGA
16316	2	10.580131	TTTGTGT
8175	1	10.552171	CTTTGTG
12031	5	9.685497	GTGTTTG
12259	7	9.638035	GTTTGAG
15296	4	9.565904	TGTGTTT
8066	56	9.445229	CTTGAAC
16112	3	9.444925	TTGTGTT
12108	13	9.314122	GTTCAGT
2553	54	9.278962	AGCTTGA

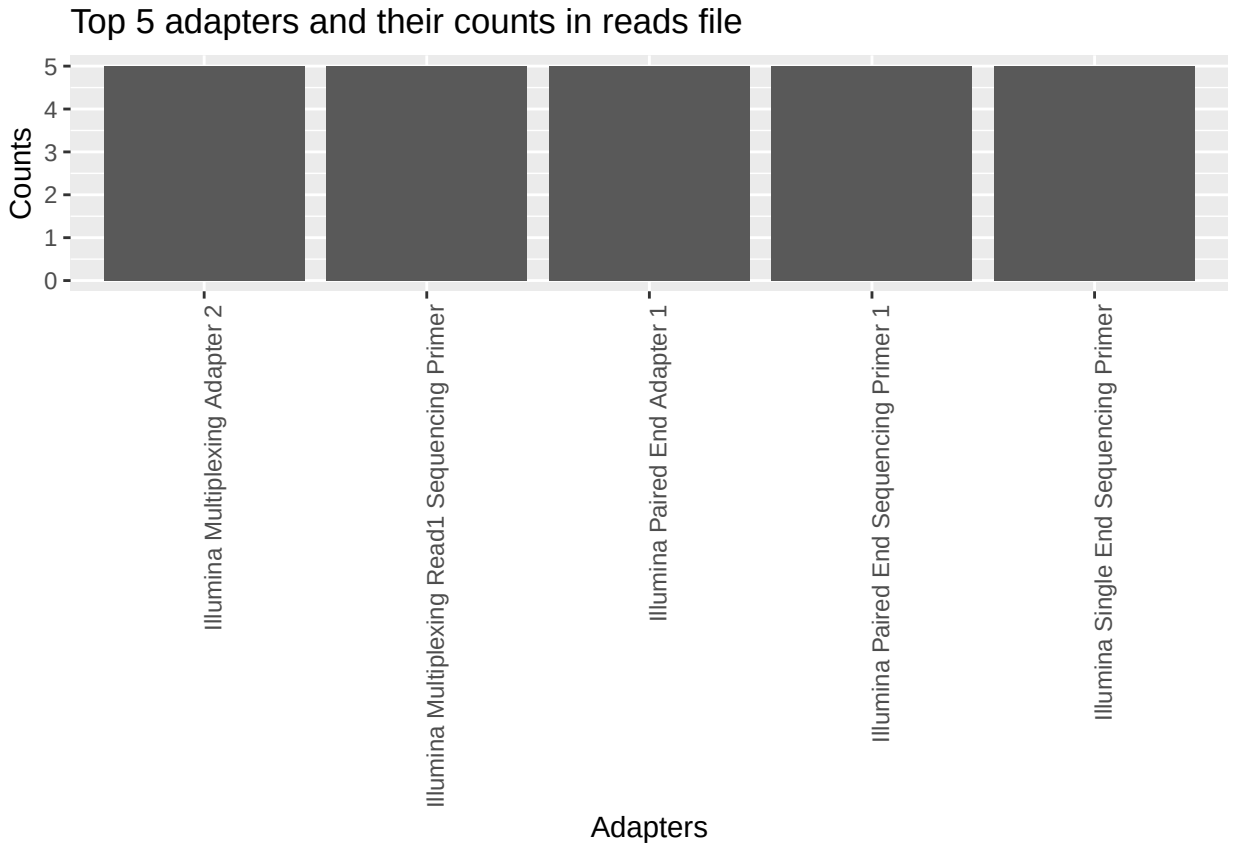
```
plot_overrep_kmer(overkm)
```



Adapter content

adapter_content computes the counts of a pre-determined set of adapter sequences and reports back any exceeding 0.1% of the total reads, sorted from most abundant to least abundant. *plot_adapter_content* creates a bar plot of the top 5 most common adapters. In this instance we will use a different file to compute adapter content, as the first example has no adapter contamination. This function is only available for macOS/Linux due to a dependency on RSeqAn/C++14, which is not supported on current Bioconductor Windows build machines.

```
if(.Platform$OS.type != "windows") {
    infile2 <- system.file("extdata", "test.fq.gz", package = "qckitfastq")
    ac_sorted <- adapter_content(infile2)
    kable(head(ac_sorted)) %>% kable_styling()
    plot_adapter_content(ac_sorted)
}
```



References

Andrews, Simon. 2010. "FastQC: A quality control tool for high throughput sequence data." https://doi.org/citeulike-article-id:11583827.

Ewels, Philip, Måns Magnusson, Sverker Lundin, and Max Käller. 2016. "MultiQC: Summarize analysis results for multiple tools and samples in a single report." *Bioinformatics*. https://doi.org/10.1093/bioinformatics/btw354.

Kaisers, Wolfgang. 2017. *SeqTools: Analysis of Nucleotide, Sequence and Quality Content on Fastq Files*.

Meissner, Alexander, Andreas Gnirke, George W. Bell, Bernard Ramsahoye, Eric S. Lander, and Rudolf Jaenisch. 2005. "Reduced representation bisulfite sequencing for comparative high-resolution DNA methylation analysis." *Nucleic Acids Research*. https://doi.org/10.1093/nar/gki901.

Morgan, Martin, Simon Anders, Michael Lawrence, Patrick Aboyoun, Hervé Pagès, and Robert Gentleman. 2009. "ShortRead: A Bioconductor Package for Input, Quality Assessment and Exploration of High-Throughput Sequence Data." *Bioinformatics* 25:2607–8. https://doi.org/10.1093/bioinformatics/btp450.