4. Introduction to ShortRead

Maruf Ahmed Bhuiyan

7/27/2020

setwd("/Users/marufahmedbhuiyan/Desktop/DataCamp/Bioconductor")

# Install and Load ShortRead

# BiocManager::install("ShortRead")  
  
require(ShortRead)

## Loading required package: ShortRead

## Loading required package: BiocGenerics

## Loading required package: parallel

##   
## Attaching package: 'BiocGenerics'

## The following objects are masked from 'package:parallel':  
##   
## clusterApply, clusterApplyLB, clusterCall, clusterEvalQ,  
## clusterExport, clusterMap, parApply, parCapply, parLapply,  
## parLapplyLB, parRapply, parSapply, parSapplyLB

## The following objects are masked from 'package:stats':  
##   
## IQR, mad, sd, var, xtabs

## The following objects are masked from 'package:base':  
##   
## anyDuplicated, append, as.data.frame, basename, cbind, colnames,  
## dirname, do.call, duplicated, eval, evalq, Filter, Find, get, grep,  
## grepl, intersect, is.unsorted, lapply, Map, mapply, match, mget,  
## order, paste, pmax, pmax.int, pmin, pmin.int, Position, rank,  
## rbind, Reduce, rownames, sapply, setdiff, sort, table, tapply,  
## union, unique, unsplit, which, which.max, which.min

## Loading required package: BiocParallel

## Loading required package: Biostrings

## Loading required package: S4Vectors

## Loading required package: stats4

##   
## Attaching package: 'S4Vectors'

## The following object is masked from 'package:base':  
##   
## expand.grid

## Loading required package: IRanges

## Loading required package: XVector

##   
## Attaching package: 'Biostrings'

## The following object is masked from 'package:base':  
##   
## strsplit

## Loading required package: Rsamtools

## Loading required package: GenomeInfoDb

## Loading required package: GenomicRanges

## Loading required package: GenomicAlignments

## Loading required package: SummarizedExperiment

## Loading required package: Biobase

## Welcome to Bioconductor  
##   
## Vignettes contain introductory material; view with  
## 'browseVignettes()'. To cite Bioconductor, see  
## 'citation("Biobase")', and for packages 'citation("pkgname")'.

## Loading required package: DelayedArray

## Loading required package: matrixStats

##   
## Attaching package: 'matrixStats'

## The following objects are masked from 'package:Biobase':  
##   
## anyMissing, rowMedians

##   
## Attaching package: 'DelayedArray'

## The following objects are masked from 'package:matrixStats':  
##   
## colMaxs, colMins, colRanges, rowMaxs, rowMins, rowRanges

## The following objects are masked from 'package:base':  
##   
## aperm, apply, rowsum

methods(class = "ShortRead")

## [1] [ alphabetByCycle append clean   
## [5] detail dustyScore id length   
## [9] narrow pairwiseAlignment show srdistance   
## [13] srduplicated sread srorder srrank   
## [17] srsort tables trimEnds trimLRPatterns   
## [21] width writeFasta   
## see '?methods' for accessing help and source code

methods(class = "ShortReadQ")

## [1] [ [<- alphabetByCycle alphabetScore   
## [5] append clean coerce detail   
## [9] dustyScore id length narrow   
## [13] pairwiseAlignment qa reverse reverseComplement  
## [17] show srdistance srduplicated sread   
## [21] srorder srrank srsort tables   
## [25] trimEnds trimLRPatterns trimTails trimTailw   
## [29] width writeFasta writeFastq   
## see '?methods' for accessing help and source code

# Reading in files

From the video, you’ve learned the difference between fasta and fastq files and what information can be stored in those files.

You have also seen examples of reading fasta and fastq files with their respective functions. You have learned that readFasta() and readFastq() both need a location and a file pattern to read one or many files.

Which two arguments are particular to the function readFastq() from the package ShortRead?

?readFastq  
  
# qualityType and Filter

# Exploring a fastq file

Fastq files usually contain thousands/millions of reads and can become very large in size! For this exercise, you will use a small fastq sub sample of 500 reads which fits easily into memory and can be read entirely using the function readFastq().

The original sequence file comes from Arabidopsis thaliana, provided by the UC Davis Genome Center. The accession number is SRR1971253 and was downloaded from the Sequence Read Archive (SRA). It contains DNA from leaf tissues, pooled and sequenced on Illumina HiSeq 2000. These sequences are single-read sequences with 50 base pairs (bp) length.

fqsample is a ShortReadQ object and contains information about reads, quality scores, and ids. It’s your turn to explore it!

# Print fqsample  
fqsample <- readFastq("SRR1971253.fastq")  
fqsample

## class: ShortReadQ  
## length: 60836 reads; width: 50 cycles

# Check class of fqsample  
class(fqsample)

## [1] "ShortReadQ"  
## attr(,"package")  
## [1] "ShortRead"

# Check class sread fqsample  
class(sread(fqsample))

## [1] "DNAStringSet"  
## attr(,"package")  
## [1] "Biostrings"

# Check id of fqsample  
id(fqsample)

## A BStringSet instance of length 60836  
## width seq  
## [1] 57 SRR1971253.1 HS2:546:C4DUWACXX:3:1101:4923:2155 length=50  
## [2] 57 SRR1971253.2 HS2:546:C4DUWACXX:3:1101:3183:2734 length=50  
## [3] 57 SRR1971253.3 HS2:546:C4DUWACXX:3:1101:6939:2512 length=50  
## [4] 57 SRR1971253.4 HS2:546:C4DUWACXX:3:1101:9401:2515 length=50  
## [5] 58 SRR1971253.5 HS2:546:C4DUWACXX:3:1101:11225:2887 length=50  
## ... ... ...  
## [60832] 62 SRR1971253.60832 HS2:546:C4DUWACXX:3:2316:9832:99913 length=50  
## [60833] 63 SRR1971253.60833 HS2:546:C4DUWACXX:3:2316:19796:99904 length=50  
## [60834] 64 SRR1971253.60834 HS2:546:C4DUWACXX:3:2316:14204:100062 length=50  
## [60835] 64 SRR1971253.60835 HS2:546:C4DUWACXX:3:2316:17279:100218 length=50  
## [60836] 63 SRR1971253.60836 HS2:546:C4DUWACXX:3:2316:1719:100669 length=50

# Extract a sample from a fastq file

It is your turn to draw a sample piece from a sequence of many reads.

You will use the same file you’ve read in the previous exercise. This file has 500 reads, each of 50 bp. The file path is stored in an object called f.

Using FastqSampler(con = file\_path, n = length), set.seed(), and yield() you can subset 100 reads from your sequence file.

# Load ShortRead  
library(ShortRead)  
  
# Set a seed for sampling  
set.seed(1234)  
  
file\_path <- "/Users/marufahmedbhuiyan/Desktop/DataCamp/Bioconductor/SRR1971253.fastq"  
  
# Use FastqSampler with f and select 100 reads  
fs <- FastqSampler(con = file\_path, n = 100)  
  
# Generate new sample yield  
my\_sample <- yield(fs)  
  
# Print my\_sample  
my\_sample

## class: ShortReadQ  
## length: 100 reads; width: 50 cycles

# FastqSampler() draws a sub-sample from a fastq file and   
# yield() is the function used to extract the sample from the FastqSampler().

# Intro to Sequence Quality

# fqsample <- readFastq("SRR1971253.fastq")  
  
# Reads  
sread(fqsample)

## A DNAStringSet instance of length 60836  
## width seq  
## [1] 50 TGGAAAGGGGCGCCAGAGAGGGTGAGAGCCCCGTCGTGCCCGGACCCTGT  
## [2] 50 GGTGGAGATGACGCGATCCGCCTCACGACGAATCGCGGCGCCTGTCAGCG  
## [3] 50 ATTCCCGCGCTTGCATGATGCCATGGCAACGACGTTTTGTGCCTACGCTG  
## [4] 50 AAACTTCGTTTCATCAACAGTAAGAGCCAAAAGGCTTCTAACAGTCTATA  
## [5] 50 TACCCAATGCCAAATAGCTGCCAAGAAGCACAAACCAGAAAACACTATAT  
## ... ... ...  
## [60832] 50 CGTTTCCTCTGAAGCTCGTTGGAGGGAGAGATCATGCAGCCCTCGCAAAT  
## [60833] 50 GTGTTCAGGCGATCATGGGCCAAGACAGCGACACCAGCAGCGCCCAGCCA  
## [60834] 50 TGTAGGCCGACATCGCCGACCCGGACGCCGTGCAGCGCCTGTTCGACGCG  
## [60835] 50 CGGTAGACGAACTTGAGGTGGTCGTTGCGCAGGAGCCCACCGGGCAGCAG  
## [60836] 50 GAAGGTCACCGGACTCACTTTGATATAATACTTGTTGCGCGAGTCCGCCG

# Quality score of reads  
quality(fqsample)

## class: FastqQuality  
## quality:  
## A BStringSet instance of length 60836  
## width seq  
## [1] 50 CCCFFFFFHHHHHJJJIIHIJJFHIJIJGIJJJIJHHFEFFDDDDDDDDD  
## [2] 50 B@=DFFFFHHHHGJJJJJJJJJIJJJIIJIIJGIJJGI6ABD@BACDDDD  
## [3] 50 CCCFFFFFHGHHHJJJJJJIIJIIJIIIJIJJJJHIJJJHIJJJIJIHHH  
## [4] 50 CCCFFFFFHHHHHJJJJJJJIIJJJJJJJJJIJJJJIJJJJJJJIIJJJI  
## [5] 50 CCCFFFFFHHHHHJJJHIJIJJJJJJJJIJIJJJJJIIIJJJJJJJJJJJ  
## ... ... ...  
## [60832] 50 =B:=BDFFHHHHHJJJJJHJJJGIJJIJGGHGHIIJJJJJJJJJJGJJJJ  
## [60833] 50 1:+2222,+=A<AACBCB=AB;ABBBA?==4?847==AABBAAA######  
## [60834] 50 :;:+ADDDFDAADBCGHIBHGFHGFFGBAA9=<AC;;>@B?CCCBB?B57  
## [60835] 50 17+2<7>7=0<<3AA3=C=<;?2?##########################  
## [60836] 50 ##################################################

# Phred Quality score  
pq <- PhredQuality(quality(fqsample))  
qs <- as(pq, "IntegerList")  
qs

## IntegerList of length 60836  
## [[1]] 34 34 34 37 37 37 37 37 39 39 39 39 ... 37 37 35 35 35 35 35 35 35 35 35  
## [[2]] 33 31 28 35 37 37 37 37 39 39 39 39 ... 32 33 35 31 33 32 34 35 35 35 35  
## [[3]] 34 34 34 37 37 37 37 37 39 38 39 39 ... 39 40 41 41 41 40 41 40 39 39 39  
## [[4]] 34 34 34 37 37 37 37 37 39 39 39 39 ... 41 41 41 41 41 40 40 41 41 41 40  
## [[5]] 34 34 34 37 37 37 37 37 39 39 39 39 ... 41 41 41 41 41 41 41 41 41 41 41  
## [[6]] 34 33 34 37 37 37 37 37 39 39 39 39 ... 41 41 41 41 41 41 41 41 41 41 41  
## [[7]] 30 31 31 30 28 35 35 35 32 35 37 39 ... 38 37 29 27 32 39 26 34 36 34 32  
## [[8]] 34 34 34 37 37 36 37 37 39 39 39 39 ... 40 41 41 41 41 41 41 41 40 40 40  
## [[9]] 34 34 34 37 37 37 37 35 39 37 39 37 ... 37 35 36 35 35 35 35 35 35 35 35  
## [[10]] 34 34 34 37 37 37 37 37 37 39 39 39 ... 39 37 37 35 35 35 35 28 32 30 33  
## ...  
## <60826 more elements>

# Quality Assessment  
qa\_summary <- qa(fqsample, type = "fastq", lane = 1)  
qa\_summary

## class: ShortReadQQA(10)  
## QA elements (access with qa[["elt"]]):  
## readCounts: data.frame(1 3)  
## baseCalls: data.frame(1 5)  
## readQualityScore: data.frame(512 4)  
## baseQuality: data.frame(95 3)  
## alignQuality: data.frame(1 3)  
## frequentSequences: data.frame(50 4)  
## sequenceDistribution: data.frame(5 4)  
## perCycle: list(2)  
## baseCall: data.frame(226 4)  
## quality: data.frame(1752 5)  
## perTile: list(2)  
## readCounts: data.frame(0 4)  
## medianReadQualityScore: data.frame(0 4)  
## adapterContamination: data.frame(1 1)

names(qa\_summary)

## [1] "readCounts" "baseCalls" "readQualityScore"   
## [4] "baseQuality" "alignQuality" "frequentSequences"   
## [7] "sequenceDistribution" "perCycle" "perTile"   
## [10] "adapterContamination"

# Quality report summary  
browseURL(report(qa\_summary))  
  
# ASCII encoding for the Quality Scores  
encoding(quality(fqsample))

## ! " # $ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 :   
## 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25   
## ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T   
## 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51   
## U V W X Y Z [ \\ ] ^ \_ ` a b c d e f g h i j k l m n   
## 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77   
## o p q r s t u v w x y z { | } ~   
## 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93

alphabet(sread(fqsample))

## [1] "A" "C" "G" "T" "M" "R" "W" "S" "Y" "K" "V" "H" "D" "B" "N" "-" "+" "."

# abc <- alphabetByCycle(alphabet(sread(fqsample)))  
  
# nuc\_by\_cycle <- t(abc[1:4,])

# Exploring sequence quality

It is your turn to perform the quality control check on the fqsample. This is an important step, done before starting any further analyses, to quickly find out whether the data is OK or if it has any problems.

To check the encoding values for each letter in quality(), use encoding():

encoding(quality(fqsample)) For a quality assessment (QA) summary use qa():

qaSummary <- qa(fqsample, type = “fastq”, lane = 1) This qaSummary has already been created for you. QA elements can be accessed with qaSummary[[“nameElement”]] where nameElement is the name of the element you wish to inspect.

# Check quality  
quality(fqsample)

## class: FastqQuality  
## quality:  
## A BStringSet instance of length 60836  
## width seq  
## [1] 50 CCCFFFFFHHHHHJJJIIHIJJFHIJIJGIJJJIJHHFEFFDDDDDDDDD  
## [2] 50 B@=DFFFFHHHHGJJJJJJJJJIJJJIIJIIJGIJJGI6ABD@BACDDDD  
## [3] 50 CCCFFFFFHGHHHJJJJJJIIJIIJIIIJIJJJJHIJJJHIJJJIJIHHH  
## [4] 50 CCCFFFFFHHHHHJJJJJJJIIJJJJJJJJJIJJJJIJJJJJJJIIJJJI  
## [5] 50 CCCFFFFFHHHHHJJJHIJIJJJJJJJJIJIJJJJJIIIJJJJJJJJJJJ  
## ... ... ...  
## [60832] 50 =B:=BDFFHHHHHJJJJJHJJJGIJJIJGGHGHIIJJJJJJJJJJGJJJJ  
## [60833] 50 1:+2222,+=A<AACBCB=AB;ABBBA?==4?847==AABBAAA######  
## [60834] 50 :;:+ADDDFDAADBCGHIBHGFHGFFGBAA9=<AC;;>@B?CCCBB?B57  
## [60835] 50 17+2<7>7=0<<3AA3=C=<;?2?##########################  
## [60836] 50 ##################################################

# Check encoding of quality  
encoding(quality(fqsample))

## ! " # $ % & ' ( ) \* + , - . / 0 1 2 3 4 5 6 7 8 9 :   
## 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25   
## ; < = > ? @ A B C D E F G H I J K L M N O P Q R S T   
## 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51   
## U V W X Y Z [ \\ ] ^ \_ ` a b c d e f g h i j k l m n   
## 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77   
## o p q r s t u v w x y z { | } ~   
## 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93

# Check baseQuality  
qa\_summary[["baseQuality"]]

## score count lane  
## 1 0 1  
## 2 ! 0 1  
## 3 " 0 1  
## 4 # 187191 1  
## 5 $ 0 1  
## 6 % 0 1  
## 7 & 106 1  
## 8 ' 1181 1  
## 9 ( 2539 1  
## 10 ) 7582 1  
## 11 \* 3760 1  
## 12 + 8105 1  
## 13 , 1776 1  
## 14 - 1237 1  
## 15 . 1344 1  
## 16 / 1358 1  
## 17 0 6176 1  
## 18 1 8301 1  
## 19 2 6484 1  
## 20 3 7108 1  
## 21 4 5981 1  
## 22 5 4107 1  
## 23 6 6134 1  
## 24 7 8276 1  
## 25 8 12491 1  
## 26 9 11484 1  
## 27 : 18777 1  
## 28 ; 19662 1  
## 29 < 31542 1  
## 30 = 22919 1  
## 31 > 17224 1  
## 32 ? 62198 1  
## 33 @ 83790 1  
## 34 A 59840 1  
## 35 B 83007 1  
## 36 C 147690 1  
## 37 D 190664 1  
## 38 E 78410 1  
## 39 F 310879 1  
## 40 G 174978 1  
## 41 H 338268 1  
## 42 I 347755 1  
## 43 J 761476 1  
## 44 K 0 1  
## 45 L 0 1  
## 46 M 0 1  
## 47 N 0 1  
## 48 O 0 1  
## 49 P 0 1  
## 50 Q 0 1  
## 51 R 0 1  
## 52 S 0 1  
## 53 T 0 1  
## 54 U 0 1  
## 55 V 0 1  
## 56 W 0 1  
## 57 X 0 1  
## 58 Y 0 1  
## 59 Z 0 1  
## 60 [ 0 1  
## 61 \\ 0 1  
## 62 ] 0 1  
## 63 ^ 0 1  
## 64 \_ 0 1  
## 65 ` 0 1  
## 66 a 0 1  
## 67 b 0 1  
## 68 c 0 1  
## 69 d 0 1  
## 70 e 0 1  
## 71 f 0 1  
## 72 g 0 1  
## 73 h 0 1  
## 74 i 0 1  
## 75 j 0 1  
## 76 k 0 1  
## 77 l 0 1  
## 78 m 0 1  
## 79 n 0 1  
## 80 o 0 1  
## 81 p 0 1  
## 82 q 0 1  
## 83 r 0 1  
## 84 s 0 1  
## 85 t 0 1  
## 86 u 0 1  
## 87 v 0 1  
## 88 w 0 1  
## 89 x 0 1  
## 90 y 0 1  
## 91 z 0 1  
## 92 { 0 1  
## 93 | 0 1  
## 94 } 0 1  
## 95 ~ 0 1

# Create a html report  
browseURL(report(qa\_summary))

# Try your own nucleotide frequency plot

Now it’s time to take a closer look at the frequency of nucleotides per cycle. The best way to do this is by making a visualization. Usually, the first cycles are a bit random and then the frequency of nucleotides should stabilize with the coming cycles.

This exercise uses the complete fastq file SRR1971253 with some pre-processing done for you:

library(ShortRead) fqsample <- readFastq(dirPath = “data”, pattern = “SRR1971253.fastq”) # extract reads  
abc <- alphabetByCycle(sread(fqsample))

# Transpose nucleotides A, C, G, T per column

nucByCycle <- t(abc[1:4,])

# Tidy dataset

nucByCycle <- nucByCycle %>% as.tibble() %>% # convert to tibble mutate(cycle = 1:50) # add cycle numbers Your task is to make a Nucleotide Frequency by Cycle plot using tidyverse functions!

fqsample <- readFastq(dirPath = getwd(),   
 pattern = "SRR1971253.fastq")  
fqsample

## class: ShortReadQ  
## length: 60836 reads; width: 50 cycles

# extract reads   
abc <- alphabetByCycle(sread(fqsample))  
abc

## cycle  
## alphabet [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]  
## A 16839 13056 13666 14723 16316 15524 15562 15708 14793 15190 15503  
## C 16335 13327 15617 15439 14082 15195 14212 15014 15488 15165 14844  
## G 16740 12064 13198 14239 14445 15146 16181 14712 14859 15246 14894  
## T 10878 22389 18355 16435 15990 14971 14881 15402 15696 15235 15594  
## M 0 0 0 0 0 0 0 0 0 0 0  
## R 0 0 0 0 0 0 0 0 0 0 0  
## W 0 0 0 0 0 0 0 0 0 0 0  
## S 0 0 0 0 0 0 0 0 0 0 0  
## Y 0 0 0 0 0 0 0 0 0 0 0  
## K 0 0 0 0 0 0 0 0 0 0 0  
## V 0 0 0 0 0 0 0 0 0 0 0  
## H 0 0 0 0 0 0 0 0 0 0 0  
## D 0 0 0 0 0 0 0 0 0 0 0  
## B 0 0 0 0 0 0 0 0 0 0 0  
## N 44 0 0 0 3 0 0 0 0 0 1  
## - 0 0 0 0 0 0 0 0 0 0 0  
## + 0 0 0 0 0 0 0 0 0 0 0  
## . 0 0 0 0 0 0 0 0 0 0 0  
## cycle  
## alphabet [,12] [,13] [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21] [,22]  
## A 15441 15552 15773 15558 15389 15441 15508 15480 15449 15406 15409  
## C 15277 15099 14919 15279 15275 14965 15067 15076 15239 15056 14983  
## G 14914 15271 14887 14737 15180 14881 14846 15325 14689 15081 15263  
## T 15204 14911 15257 15262 14988 15545 15410 14950 15452 15292 15181  
## M 0 0 0 0 0 0 0 0 0 0 0  
## R 0 0 0 0 0 0 0 0 0 0 0  
## W 0 0 0 0 0 0 0 0 0 0 0  
## S 0 0 0 0 0 0 0 0 0 0 0  
## Y 0 0 0 0 0 0 0 0 0 0 0  
## K 0 0 0 0 0 0 0 0 0 0 0  
## V 0 0 0 0 0 0 0 0 0 0 0  
## H 0 0 0 0 0 0 0 0 0 0 0  
## D 0 0 0 0 0 0 0 0 0 0 0  
## B 0 0 0 0 0 0 0 0 0 0 0  
## N 0 3 0 0 4 4 5 5 7 1 0  
## - 0 0 0 0 0 0 0 0 0 0 0  
## + 0 0 0 0 0 0 0 0 0 0 0  
## . 0 0 0 0 0 0 0 0 0 0 0  
## cycle  
## alphabet [,23] [,24] [,25] [,26] [,27] [,28] [,29] [,30] [,31] [,32] [,33]  
## A 15683 15397 15282 15377 15349 15088 15708 15475 15311 15430 15372  
## C 15041 15313 15144 15204 15244 15272 14882 15222 14979 15186 15413  
## G 14801 14702 15181 14988 14744 15144 14837 14701 15499 14727 14771  
## T 15311 15424 15229 15266 15499 15331 15409 15438 15047 15485 15277  
## M 0 0 0 0 0 0 0 0 0 0 0  
## R 0 0 0 0 0 0 0 0 0 0 0  
## W 0 0 0 0 0 0 0 0 0 0 0  
## S 0 0 0 0 0 0 0 0 0 0 0  
## Y 0 0 0 0 0 0 0 0 0 0 0  
## K 0 0 0 0 0 0 0 0 0 0 0  
## V 0 0 0 0 0 0 0 0 0 0 0  
## H 0 0 0 0 0 0 0 0 0 0 0  
## D 0 0 0 0 0 0 0 0 0 0 0  
## B 0 0 0 0 0 0 0 0 0 0 0  
## N 0 0 0 1 0 1 0 0 0 8 3  
## - 0 0 0 0 0 0 0 0 0 0 0  
## + 0 0 0 0 0 0 0 0 0 0 0  
## . 0 0 0 0 0 0 0 0 0 0 0  
## cycle  
## alphabet [,34] [,35] [,36] [,37] [,38] [,39] [,40] [,41] [,42] [,43] [,44]  
## A 15356 15627 15348 15391 15779 15257 15402 15709 15397 15454 15587  
## C 15127 14917 15211 15299 14847 15027 15104 14945 15041 15174 14932  
## G 15257 14917 14702 15221 14887 14979 14935 14911 15109 15114 14876  
## T 15083 15373 15570 14925 15323 15569 15395 15271 15289 15089 15438  
## M 0 0 0 0 0 0 0 0 0 0 0  
## R 0 0 0 0 0 0 0 0 0 0 0  
## W 0 0 0 0 0 0 0 0 0 0 0  
## S 0 0 0 0 0 0 0 0 0 0 0  
## Y 0 0 0 0 0 0 0 0 0 0 0  
## K 0 0 0 0 0 0 0 0 0 0 0  
## V 0 0 0 0 0 0 0 0 0 0 0  
## H 0 0 0 0 0 0 0 0 0 0 0  
## D 0 0 0 0 0 0 0 0 0 0 0  
## B 0 0 0 0 0 0 0 0 0 0 0  
## N 13 2 5 0 0 4 0 0 0 5 3  
## - 0 0 0 0 0 0 0 0 0 0 0  
## + 0 0 0 0 0 0 0 0 0 0 0  
## . 0 0 0 0 0 0 0 0 0 0 0  
## cycle  
## alphabet [,45] [,46] [,47] [,48] [,49] [,50]  
## A 15351 15409 15606 15504 15552 15539  
## C 15125 15159 14945 14997 14909 14851  
## G 14848 14938 14902 14847 15149 15015  
## T 15510 15322 15378 15478 15220 15430  
## M 0 0 0 0 0 0  
## R 0 0 0 0 0 0  
## W 0 0 0 0 0 0  
## S 0 0 0 0 0 0  
## Y 0 0 0 0 0 0  
## K 0 0 0 0 0 0  
## V 0 0 0 0 0 0  
## H 0 0 0 0 0 0  
## D 0 0 0 0 0 0  
## B 0 0 0 0 0 0  
## N 2 8 5 10 6 1  
## - 0 0 0 0 0 0  
## + 0 0 0 0 0 0  
## . 0 0 0 0 0 0

# Transpose nucleotides A, C, G, T per column  
nucByCycle <- t(abc[1:4,])   
nucByCycle

## alphabet  
## cycle A C G T  
## [1,] 16839 16335 16740 10878  
## [2,] 13056 13327 12064 22389  
## [3,] 13666 15617 13198 18355  
## [4,] 14723 15439 14239 16435  
## [5,] 16316 14082 14445 15990  
## [6,] 15524 15195 15146 14971  
## [7,] 15562 14212 16181 14881  
## [8,] 15708 15014 14712 15402  
## [9,] 14793 15488 14859 15696  
## [10,] 15190 15165 15246 15235  
## [11,] 15503 14844 14894 15594  
## [12,] 15441 15277 14914 15204  
## [13,] 15552 15099 15271 14911  
## [14,] 15773 14919 14887 15257  
## [15,] 15558 15279 14737 15262  
## [16,] 15389 15275 15180 14988  
## [17,] 15441 14965 14881 15545  
## [18,] 15508 15067 14846 15410  
## [19,] 15480 15076 15325 14950  
## [20,] 15449 15239 14689 15452  
## [21,] 15406 15056 15081 15292  
## [22,] 15409 14983 15263 15181  
## [23,] 15683 15041 14801 15311  
## [24,] 15397 15313 14702 15424  
## [25,] 15282 15144 15181 15229  
## [26,] 15377 15204 14988 15266  
## [27,] 15349 15244 14744 15499  
## [28,] 15088 15272 15144 15331  
## [29,] 15708 14882 14837 15409  
## [30,] 15475 15222 14701 15438  
## [31,] 15311 14979 15499 15047  
## [32,] 15430 15186 14727 15485  
## [33,] 15372 15413 14771 15277  
## [34,] 15356 15127 15257 15083  
## [35,] 15627 14917 14917 15373  
## [36,] 15348 15211 14702 15570  
## [37,] 15391 15299 15221 14925  
## [38,] 15779 14847 14887 15323  
## [39,] 15257 15027 14979 15569  
## [40,] 15402 15104 14935 15395  
## [41,] 15709 14945 14911 15271  
## [42,] 15397 15041 15109 15289  
## [43,] 15454 15174 15114 15089  
## [44,] 15587 14932 14876 15438  
## [45,] 15351 15125 14848 15510  
## [46,] 15409 15159 14938 15322  
## [47,] 15606 14945 14902 15378  
## [48,] 15504 14997 14847 15478  
## [49,] 15552 14909 15149 15220  
## [50,] 15539 14851 15015 15430

library(tidyverse)

## ── Attaching packages ──────────────────

## ✓ ggplot2 3.3.2 ✓ purrr 0.3.4  
## ✓ tibble 3.0.3 ✓ dplyr 1.0.0  
## ✓ tidyr 1.1.0 ✓ stringr 1.4.0  
## ✓ readr 1.3.1 ✓ forcats 0.5.0

## ── Conflicts ── tidyverse\_conflicts() ──  
## x dplyr::collapse() masks Biostrings::collapse(), IRanges::collapse()  
## x dplyr::combine() masks Biobase::combine(), BiocGenerics::combine()  
## x purrr::compact() masks XVector::compact()  
## x purrr::compose() masks ShortRead::compose()  
## x dplyr::count() masks matrixStats::count()  
## x dplyr::desc() masks IRanges::desc()  
## x tidyr::expand() masks S4Vectors::expand()  
## x dplyr::filter() masks stats::filter()  
## x dplyr::first() masks GenomicAlignments::first(), S4Vectors::first()  
## x dplyr::id() masks ShortRead::id()  
## x dplyr::lag() masks stats::lag()  
## x dplyr::last() masks GenomicAlignments::last()  
## x ggplot2::Position() masks BiocGenerics::Position(), base::Position()  
## x purrr::reduce() masks GenomicRanges::reduce(), IRanges::reduce()  
## x dplyr::rename() masks S4Vectors::rename()  
## x purrr::simplify() masks DelayedArray::simplify()  
## x dplyr::slice() masks XVector::slice(), IRanges::slice()  
## x tibble::view() masks ShortRead::view()

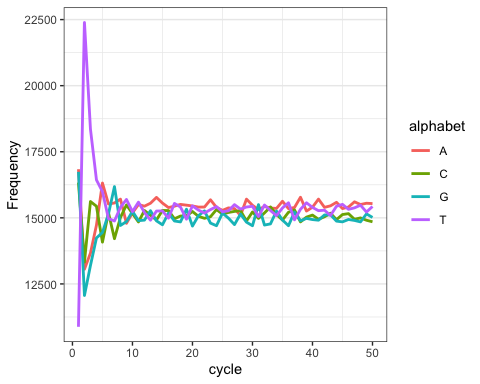
# Tidy dataset  
nucByCycle <- nucByCycle %>%   
 as\_tibble() %>% # convert to tibble  
 mutate(cycle = 1:50) # add cycle numbers  
  
nucByCycle

## # A tibble: 50 x 5  
## A C G T cycle  
## <int> <int> <int> <int> <int>  
## 1 16839 16335 16740 10878 1  
## 2 13056 13327 12064 22389 2  
## 3 13666 15617 13198 18355 3  
## 4 14723 15439 14239 16435 4  
## 5 16316 14082 14445 15990 5  
## 6 15524 15195 15146 14971 6  
## 7 15562 14212 16181 14881 7  
## 8 15708 15014 14712 15402 8  
## 9 14793 15488 14859 15696 9  
## 10 15190 15165 15246 15235 10  
## # … with 40 more rows

# ?gather  
  
# Glimpse nucByCycle  
glimpse(nucByCycle)

## Rows: 50  
## Columns: 5  
## $ A <int> 16839, 13056, 13666, 14723, 16316, 15524, 15562, 15708, 14793, …  
## $ C <int> 16335, 13327, 15617, 15439, 14082, 15195, 14212, 15014, 15488, …  
## $ G <int> 16740, 12064, 13198, 14239, 14445, 15146, 16181, 14712, 14859, …  
## $ T <int> 10878, 22389, 18355, 16435, 15990, 14971, 14881, 15402, 15696, …  
## $ cycle <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, …

# Create a line plot of cycle vs count  
nucByCycle %>%   
 # Gather the nucleotide letters in alphabet and get a new count column  
 pivot\_longer(-cycle, names\_to = "alphabet", values\_to = "count") %>%  
 # gather(key = alphabet, value = count , -cycle) %>%   
 ggplot(aes(x = cycle, y = count, color = alphabet)) +  
 geom\_line(size = 1 ) +  
 labs(y = "Frequency") +  
 theme\_bw() +  
 theme(panel.grid.major.x = element\_blank())



# vignette("pivot")

# Math & Filter

Filtering reads on the go! What if, from all of the reads in a file, you are only interested in some of those reads? You can use a filter!

Let’s say that you are interested only in those reads that start with the pattern “ATGCA”. A tiny filtering function can do the job, making use of the srFilter() function:

myStartFilter <- srFilter(function(x) substr(sread(x), 1, 5) == “ATGCA”) This function, which has been created for you, takes a ShortRead derived object as an input and outputs the reads starting with the pattern “ATGCA”. Let’s put this function to use!

# Check class of fqsample  
class(fqsample)

## [1] "ShortReadQ"  
## attr(,"package")  
## [1] "ShortRead"

myStartFilter <- srFilter(function(x)   
 substr(sread(x), 1, 5) == "ATGCA")  
  
# Filter reads into selectedReads using myStartFilter  
selectedReads <- fqsample[myStartFilter(fqsample)]  
  
# Check class of selectedReads  
class(selectedReads)

## [1] "ShortReadQ"  
## attr(,"package")  
## [1] "ShortRead"

# Check detail of selectedReads  
detail(selectedReads)

## class: ShortReadQ   
##   
## sread:  
## A DNAStringSet instance of length 91  
## width seq  
## [1] 50 ATGCAGGGCCTCGGCGAACTGGGCCCAGCGATTGGTGAACTGCGCGACAC  
## [2] 50 ATGCAGCTAAAGAAAACCTCAAATGCTTCATGCTCGGATTTAGTCACTTT  
## [3] 50 ATGCATGACCGAAGGCGACAAGGCCGTCGCGAATGCGGCGAGCGCGAGGA  
## [4] 50 ATGCAAGGCCTCGGCGTAGACCTGCCCACCGCCGATCACCATCAGCCCGG  
## [5] 50 ATGCAAAAAATCCTCGCCAATCGGCTGGCTGTGCTACCGATAAACTACCT  
## ... ... ...  
## [87] 50 ATGCAATCGTATATATTCTTTTTTGAAGACGTGATAAACCATTCGCCGTG  
## [88] 50 ATGCAATGCTCACTACTCGCCGACGGATTCAAATCCTCGGTCCCGGAAGG  
## [89] 50 ATGCATGTGCCTCACTTAACCAAGTAGTCTCAATCAGAACCGTTCATTTA  
## [90] 50 ATGCATAATTTATATGAGGGTTTGTAAAAATATTCTATTTTCTTAGGGTT  
## [91] 50 ATGCAAGTATTGTGGGGAGATTCATCCGATCGGTGGGCCGCACGTATGTA  
##   
## id:  
## A BStringSet instance of length 91  
## width seq  
## [1] 61 SRR1971253.299 HS2:546:C4DUWACXX:3:1101:18454:65432 length=50  
## [2] 62 SRR1971253.1171 HS2:546:C4DUWACXX:3:1103:10297:40727 length=50  
## [3] 62 SRR1971253.1903 HS2:546:C4DUWACXX:3:1104:18590:77349 length=50  
## [4] 61 SRR1971253.2145 HS2:546:C4DUWACXX:3:1105:5257:24605 length=50  
## [5] 61 SRR1971253.3211 HS2:546:C4DUWACXX:3:1107:3706:29356 length=50  
## ... ... ...  
## [87] 62 SRR1971253.56661 HS2:546:C4DUWACXX:3:2311:4091:79748 length=50  
## [88] 63 SRR1971253.56705 HS2:546:C4DUWACXX:3:2311:13163:83719 length=50  
## [89] 63 SRR1971253.58564 HS2:546:C4DUWACXX:3:2314:14979:21853 length=50  
## [90] 63 SRR1971253.58892 HS2:546:C4DUWACXX:3:2314:18176:69143 length=50  
## [91] 63 SRR1971253.59077 HS2:546:C4DUWACXX:3:2314:17354:90285 length=50  
## class: FastqQuality  
## quality:  
## A BStringSet instance of length 91  
## width seq  
## [1] 50 CCCFFFFFGHHHHJJJJJJJJJJIJJIIIIGHHJJHIJJJJJHHHEDDDD  
## [2] 50 CCCFFFFFHHHHHJJJJJJJJJJJJJJJJJJJJJJJJJJJJJJHJJJJJJ  
## [3] 50 @C@DFDDEFHDHHEFEGIIGG@FFDEHIIGGHF==@<?>=?8B@BDD@B8  
## [4] 50 @@@DDD?DBDHHHIG>GGDBC?FDGFHGE66?DFHA@BGEE@==7)?;99  
## [5] 50 CCCFFFFFHHHHHJJJJJJJJJJJJFII=D=BFEGGGHHHIGGICHHEHF  
## ... ... ...  
## [87] 50 CCCFFFFFHHHHHJJJJIJJJJJJJJJJIJJIHJJJJJJJJJJJJIIJJJ  
## [88] 50 C@CFFFFFHHHHHJJJJJJIJJIIJJHIJJJJJJJJJJIIIJJJJHFFFD  
## [89] 50 CCCFFFFFHHHHHJJJJJJJJJJJHJJIGJJJJJJJIJIJJJIJJJJJJJ  
## [90] 50 B@C1DDFFHHHGHIJIIJJJGHJJIJJJJJJJJJJIGJJJJIJJJJJIGH  
## [91] 50 @@@<DDFDDFFFHHGEH=@HIGIEBFHGGHGHIHEG>EDF@AADF9?D;>

# Removing duplicates

It is always a good practice to check that your sequence reads don’t contain too many duplicates.

# Sample with duplicates of class: ShortReadQ

dfqsample

# Get the reads from dfqsample

mydReads <- sread(dfqsample)

# Counting duplicates

table(srduplicated(mydReads)) How would you go about filtering duplicated reads in a file? Pay attention to what the condition should be in this filter.

# mydReads[srduplicated(mydReads) == FALSE]  
  
# To keep those reads that are not duplicated   
# mydReads[srduplicated(mydReads) == FALSE]

# More filtering!

Awesome! Now that you’ve had some practice with filtering reads, let’s use the function polynFilter(). This function selects reads that contain less than a given number of duplicate nucleotides. For example, polynFilter(threshold = 20, nuc = c(“A”)) will select all reads that contain less than 20 A’s. The parameter nuc is a character vector containing IUPAC symbols for nucleotides or the value “other” for all non-nucleotide symbols.

The fqsample object is available in your workspace.

# Check reads of fqsample  
sread(fqsample)

## A DNAStringSet instance of length 60836  
## width seq  
## [1] 50 TGGAAAGGGGCGCCAGAGAGGGTGAGAGCCCCGTCGTGCCCGGACCCTGT  
## [2] 50 GGTGGAGATGACGCGATCCGCCTCACGACGAATCGCGGCGCCTGTCAGCG  
## [3] 50 ATTCCCGCGCTTGCATGATGCCATGGCAACGACGTTTTGTGCCTACGCTG  
## [4] 50 AAACTTCGTTTCATCAACAGTAAGAGCCAAAAGGCTTCTAACAGTCTATA  
## [5] 50 TACCCAATGCCAAATAGCTGCCAAGAAGCACAAACCAGAAAACACTATAT  
## ... ... ...  
## [60832] 50 CGTTTCCTCTGAAGCTCGTTGGAGGGAGAGATCATGCAGCCCTCGCAAAT  
## [60833] 50 GTGTTCAGGCGATCATGGGCCAAGACAGCGACACCAGCAGCGCCCAGCCA  
## [60834] 50 TGTAGGCCGACATCGCCGACCCGGACGCCGTGCAGCGCCTGTTCGACGCG  
## [60835] 50 CGGTAGACGAACTTGAGGTGGTCGTTGCGCAGGAGCCCACCGGGCAGCAG  
## [60836] 50 GAAGGTCACCGGACTCACTTTGATATAATACTTGTTGCGCGAGTCCGCCG

# Create myFil using polynFilter  
myFil <- polynFilter(threshold = 3, nuc = c("A"))  
  
# Check myFil  
myFil

## class: SRFilter   
## name: PolyNFilter   
## use srFilter(object) to see filter

# Check reads of fqsample  
sread(fqsample)

## A DNAStringSet instance of length 60836  
## width seq  
## [1] 50 TGGAAAGGGGCGCCAGAGAGGGTGAGAGCCCCGTCGTGCCCGGACCCTGT  
## [2] 50 GGTGGAGATGACGCGATCCGCCTCACGACGAATCGCGGCGCCTGTCAGCG  
## [3] 50 ATTCCCGCGCTTGCATGATGCCATGGCAACGACGTTTTGTGCCTACGCTG  
## [4] 50 AAACTTCGTTTCATCAACAGTAAGAGCCAAAAGGCTTCTAACAGTCTATA  
## [5] 50 TACCCAATGCCAAATAGCTGCCAAGAAGCACAAACCAGAAAACACTATAT  
## ... ... ...  
## [60832] 50 CGTTTCCTCTGAAGCTCGTTGGAGGGAGAGATCATGCAGCCCTCGCAAAT  
## [60833] 50 GTGTTCAGGCGATCATGGGCCAAGACAGCGACACCAGCAGCGCCCAGCCA  
## [60834] 50 TGTAGGCCGACATCGCCGACCCGGACGCCGTGCAGCGCCTGTTCGACGCG  
## [60835] 50 CGGTAGACGAACTTGAGGTGGTCGTTGCGCAGGAGCCCACCGGGCAGCAG  
## [60836] 50 GAAGGTCACCGGACTCACTTTGATATAATACTTGTTGCGCGAGTCCGCCG

# Create myFil using polynFilter  
myFil <- polynFilter(threshold = 3, nuc = c("A"))  
myFil

## class: SRFilter   
## name: PolyNFilter   
## use srFilter(object) to see filter

# Apply your filter to fqsample  
filterCondition <- myFil(fqsample)  
filterCondition

## class: SRFilterResult   
## name: PolyNFilter   
## output: FALSE FALSE ... FALSE FALSE   
## stats:  
## Name Input Passing Op  
## 1 PolyNFilter 60836 717 <NA>

# Use myFil with fqsample  
filteredSequences <- fqsample[filterCondition]  
filteredSequences

## class: ShortReadQ  
## length: 717 reads; width: 50 cycles

# Check reads of filteredSequences  
sread(filteredSequences)

## A DNAStringSet instance of length 717  
## width seq  
## [1] 50 GCGCCGCTGAGCCCACGTTCCACCTGCGGCCGCCTGCGTCGGCGGTCGCC  
## [2] 50 CCTGCGCGCGCATCGGTGCCATCCATTCGGTGGTCTTCGGCGGCTTCTCG  
## [3] 50 GTGCCCGGTGCTCCCGGCGTGCCTGCTGCTCCTGGCTACTCTGGCACTCC  
## [4] 50 CCACCAGGTCGCCCTGGCCCGGTTCTTCCATTTCCTTGTTGGCCTTGCCG  
## [5] 50 GTCCGCAGCGGCGGGGGGGGGCGGGGCGGCTTTGTTAGGGGGGGGGGGGG  
## ... ... ...  
## [713] 50 CCGGCTTGCTGTCGCTGGCTGCCGTGTTCGCGGTCGCTCTTATCAGCCCC  
## [714] 50 TCGGGGGCGTCTTCGGCTGCTGGGTTAGCCCCAGTATTTGGGTCCTCTGC  
## [715] 50 AGTGGGTTCGGCGCTGGGGCCGTGCCGGGTGCGGTGGCGTTCGAAGCGGC  
## [716] 50 TGTGGCTCTTTTCGGTATTTTGTGCGGCTATTTTCGCTGGCTTGGCTTTA  
## [717] 50 CGACCGCGTCACGCTCGCCGGCCGTCTGGCGCGCGGCGATGCGCGCCTCG

# Compare before(fqsample) & after(filteredSeqeuences)  
sread(fqsample)

## A DNAStringSet instance of length 60836  
## width seq  
## [1] 50 TGGAAAGGGGCGCCAGAGAGGGTGAGAGCCCCGTCGTGCCCGGACCCTGT  
## [2] 50 GGTGGAGATGACGCGATCCGCCTCACGACGAATCGCGGCGCCTGTCAGCG  
## [3] 50 ATTCCCGCGCTTGCATGATGCCATGGCAACGACGTTTTGTGCCTACGCTG  
## [4] 50 AAACTTCGTTTCATCAACAGTAAGAGCCAAAAGGCTTCTAACAGTCTATA  
## [5] 50 TACCCAATGCCAAATAGCTGCCAAGAAGCACAAACCAGAAAACACTATAT  
## ... ... ...  
## [60832] 50 CGTTTCCTCTGAAGCTCGTTGGAGGGAGAGATCATGCAGCCCTCGCAAAT  
## [60833] 50 GTGTTCAGGCGATCATGGGCCAAGACAGCGACACCAGCAGCGCCCAGCCA  
## [60834] 50 TGTAGGCCGACATCGCCGACCCGGACGCCGTGCAGCGCCTGTTCGACGCG  
## [60835] 50 CGGTAGACGAACTTGAGGTGGTCGTTGCGCAGGAGCCCACCGGGCAGCAG  
## [60836] 50 GAAGGTCACCGGACTCACTTTGATATAATACTTGTTGCGCGAGTCCGCCG

# 717 reads is what we have left after applying the   
# polynFilter() and filtering using the filterCondition  
# on fqsample. Once you are comfortable using conditions  
# directly as subsets, you can create the object filteredSequences  
# in one line of code, like this:   
# objectToFilter[customFilter(objectToFilter)]  
  
oneline\_filter <- fqsample[myFil(fqsample)]  
sread(oneline\_filter)

## A DNAStringSet instance of length 717  
## width seq  
## [1] 50 GCGCCGCTGAGCCCACGTTCCACCTGCGGCCGCCTGCGTCGGCGGTCGCC  
## [2] 50 CCTGCGCGCGCATCGGTGCCATCCATTCGGTGGTCTTCGGCGGCTTCTCG  
## [3] 50 GTGCCCGGTGCTCCCGGCGTGCCTGCTGCTCCTGGCTACTCTGGCACTCC  
## [4] 50 CCACCAGGTCGCCCTGGCCCGGTTCTTCCATTTCCTTGTTGGCCTTGCCG  
## [5] 50 GTCCGCAGCGGCGGGGGGGGGCGGGGCGGCTTTGTTAGGGGGGGGGGGGG  
## ... ... ...  
## [713] 50 CCGGCTTGCTGTCGCTGGCTGCCGTGTTCGCGGTCGCTCTTATCAGCCCC  
## [714] 50 TCGGGGGCGTCTTCGGCTGCTGGGTTAGCCCCAGTATTTGGGTCCTCTGC  
## [715] 50 AGTGGGTTCGGCGCTGGGGCCGTGCCGGGTGCGGTGGCGTTCGAAGCGGC  
## [716] 50 TGTGGCTCTTTTCGGTATTTTGTGCGGCTATTTTCGCTGGCTTGGCTTTA  
## [717] 50 CGACCGCGTCACGCTCGCCGGCCGTCTGGCGCGCGGCGATGCGCGCCTCG

# Rqc

# BiocManager::install("Rqc")  
  
require(Rqc)

## Loading required package: Rqc

# Plotting cycle average quality

Here you are! To challenge yourself, are you ready to test a bit of all of what you have learned by making three quick plots? You will see that now it all pays off!

It is your turn to use the new package Rqc and check multiple files at once.

# a <- rqcCycleAverageQualityPlot(qa)  
  
# Average per cycle quality plot with white background  
# a + theme\_minimal()  
  
# Read quality plot with white background  
# a + theme\_minimal()