# Basic usage of utility functions in GBScleanR

Tomoyuki Furuta

September 30, 2021

## Contents

## Introduction

The GBScleanR package has been mainly developed to conduct error correction on genotype data obtained via NGS-base genotyping methods such as RAD-seq and GBS. Nevertheless, several quality check procedure and data filtering are highly encouraged to improve error correction accuracy. Therefore, this package also provide the functions for data quality check and filtering with some data visualization functions to help filtering procedure. In this document, we walk through the utility functions implemented in GBScleanR to introduce a basic usage. An error correction procedure for GBS data of a biparental population is described in another vignette "ErrorCorrectionWithGBSR.pdf".

## Prerequisites

This package internally uses the following packages.
- ggplot2
- dplyr
- tidyr - expm - gdsfmt - biobase - GWASTools
- SeqArray

You can install GBScleanR from the Bioconductor repository with the following code.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("GBScleanR")
```

The code below let you install the package from the github repository.

```
if (!requireNamespace("devtools", quietly = TRUE))
    install.packages("devtools")
devtools::install_github("https://github.com/tomoyukif/GBScleanR.git")
```

To load the package.

```
library("GBScleanR")
```

## Data format conversion and object instantiation

The main class of the `GBScleanR` package is `gbsrGenotypData` which inherits the `GenotypeData` class in the `GWASTools` package. The `gbsrGenotypeData` class object has three slots: `data`, `snpAnnot`, and `scanAnnot`. The `data` slot holds genotype data as a `gds.class` object which is defined in the `gdsfmt` package while `snpAnnot` and `scanAnnot` contain objects storing annotation information of SNPs and samples, which are the `SnpAnnotationDataFrame` and `ScanAnnotationDataFrame` objects defined in the `GWASTools` package. See the vignette of `GWASTools` for more detail. `GBScleanR` follows the way of `GWASTools` in which a unique genotyping instance (genotyped sample) is called "scan".

GBScleanR only support a VCF file as input. As an example data, we use simulated genotype data for a simulated biparental F2 population derived from inbred founders.

```
vcf_fn <- system.file("extdata", "simpop.vcf", package = "GBScleanR")
gds_fn <- system.file("extdata", "simpop.gds", package = "GBScleanR")
```

As mentioned above, the `gbsrGenotypeData` class requires genotype data in the `gds.class` object which enable us quick access to the genotype data without loading the whole data on RAM. At the beginning of the processing, we need to convert data format of our genotype data from VCF to GDS. This conversion can be achieved using `gbsrVCF2GDS` as shown below. A compressed VCF file (.vcf.gz) also can be the input. Our sample dataset contains genotype information of 100 samples with 1000 markeres on only one chromosome.

```
gbsrVCF2GDS(vcf_fn = vcf_fn, # Path to the input VCF file.
            out_fn = gds_fn) # Path to the output GDS file.
```

Once we converted the VCF to the GDS, we can create the `gbsrGenotypeData` instance for our data.

```
gdata <- loadGDS(gds_fn)
```

If your samples have non autosomal chromosomes such as X and Y chromosomes or mitochondrial one, please pass the named list to define which chromosome is which type of non autosomal chromosome. * This argument can be specified but no effect in the current implementation. This will work in a future release.

```
# Not run.
gdata <- loadGDS(gds_fn,
                 non_autosomes =  list(X = 13,
                                       Y = 14,
                                       M = 15)) # M indicates mitochondrial chromosome.
```

Getter functions allow you to retrieve basic information of genotype data, e.g. number of SNPs and samples, chromosome names, physical position of SNPs and alleles.

```
nscan(gdata) # Number of samples
```

```
## [1] 102
```

```
nsnp(gdata) # Number of SNPs
```

```
## [1] 100
```

```
head(getChromosome(gdata)) # Indices of chromosome ID of all markers
```

```
## [1] 1 1 1 1 1 1
```

```
head(getChromosome(gdata, name = TRUE)) # Chromosome names of all markers
```

```
## [1] 1 1 1 1 1 1
## Levels: 1
```

```r
getChromosome(gdata, levels = TRUE) # Unique set of chromosome names
```

```
## [1] 1
```

```r
head(getPosition(gdata)) # Position (bp) of all markers
```

```
## [1] 1266164 1270080 2537850 2779885 2983182 3047595
```

```r
head(getAlleleA(gdata)) # Reference allele of all markers
```

```
## [1] "G" "G" "G" "G" "G" "G"
```

```r
head(getAlleleB(gdata)) # Alternative allele of all markers
```

```
## [1] "A" "A" "A" "A" "A" "A"
```

```r
head(getSnpID(gdata)) # SNP IDs
```

```
## [1] 1 2 3 4 5 6
```

```r
head(getScanID(gdata)) # sample IDs
```

```
## [1] "Founder1"     "Founder2"     "G3_1_1x1_1_1" "G3_1_1x1_1_2" "G3_1_1x1_1_3"
## [6] "G3_1_1x1_1_4"
```

The function **getGenotype** returns overall genotype data in which integer numbers 0, 1, and 2 indicate the number of reference allele.

```r
geno <- getGenotype(gdata)
```

## Calculate summary statitics

`countGenotype` and `countRead` are class methods of `gbsrGenotypeData` and they summarize genotype counts and read counts both per SNP and per sample.

```
gdata <- countGenotype(gdata)
gdata <- countRead(gdata)
```

These summary statistics can be visualized via ploting functions. With the values obtained via `countGenotype`, we can plot histgrams of missing rate (Figure 1), heterozygosity (Figure 2), reference allele frequency (Figure 3) as shown below.

```
histGBSR(gdata, stats = "missing") # Histgrams of missing rate
```

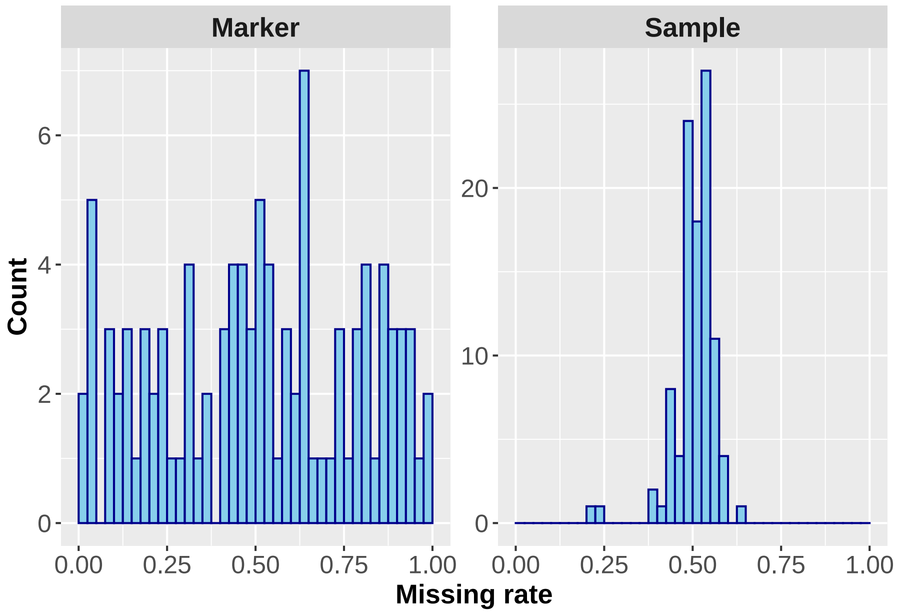


Figure 1: Missing rate per marker and per sample.

```
histGBSR(gdata, stats = "het") # Histgrams of heterozygosity
```

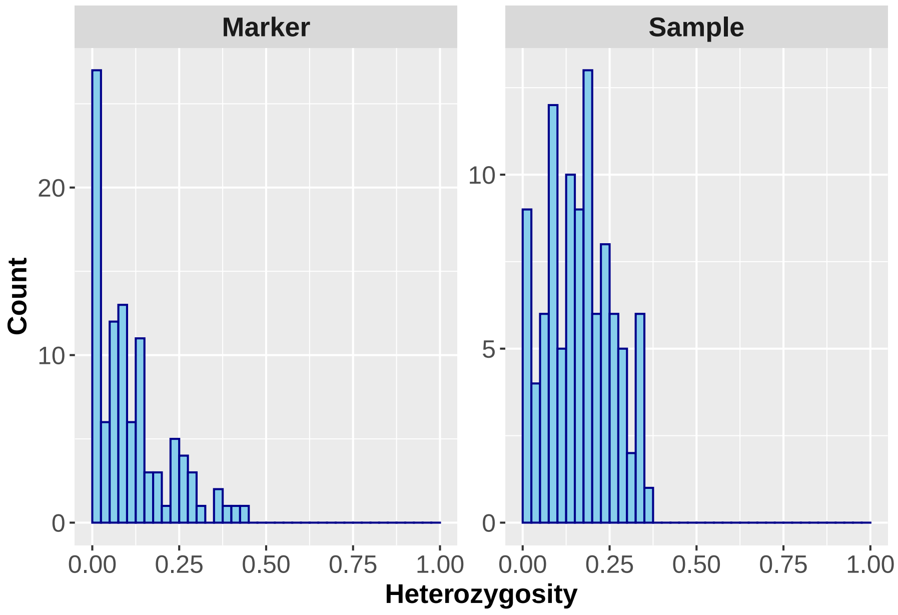


Figure 2: Heterozygosity per marker and per sample.

```r
histGBSR(gdata, stats = "raf") # Histgrams of reference allele frequency
```

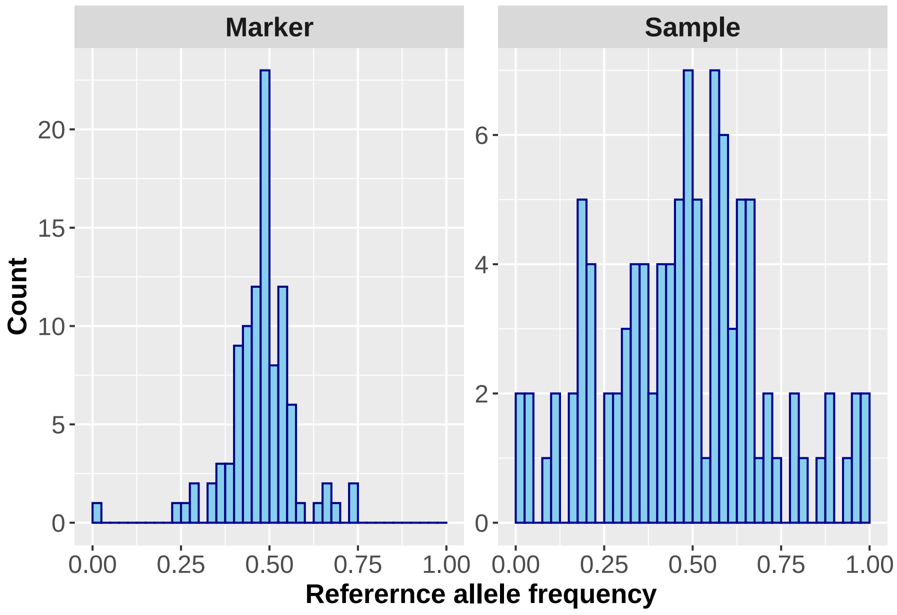


Figure 3: Reference allele frequency per marker and per sample.

With the values obtained via `countRead`, we can plot histgrams of total read depth (Figure 4), allelic read depth (Figure 5), reference read frequency (Figure 6) as shown below.

```r
histGBSR(gdata, stats = "dp") # Histgrams of total read depth
```

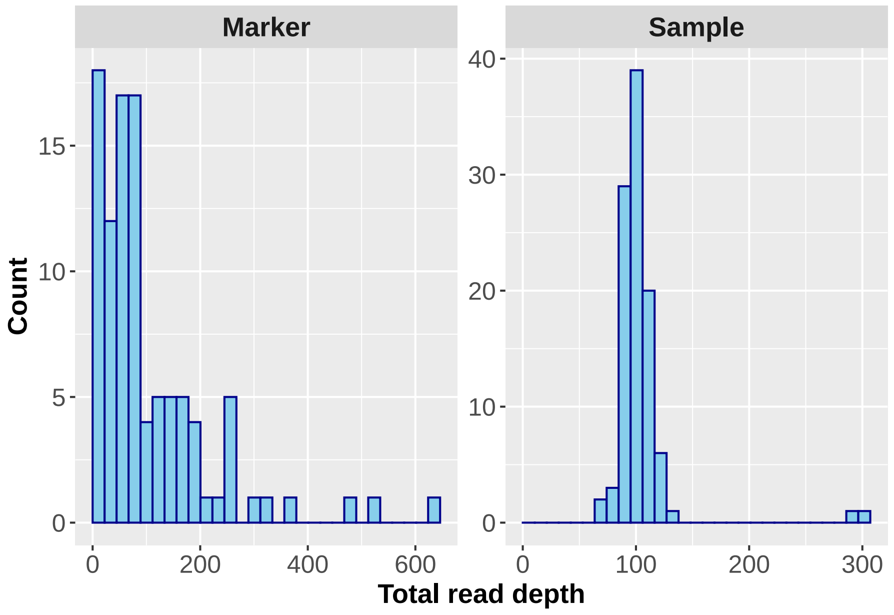


Figure 4: Total read depth per marker and per sample.

```
histGBSR(gdata, stats = "ad_ref") # Histgrams of allelic read depth
```

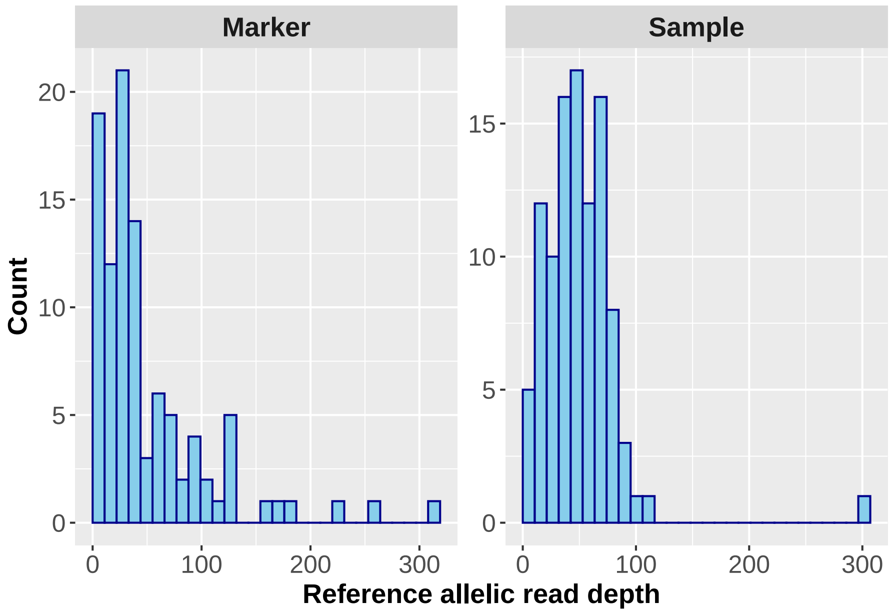


Figure 5: Reference read depth per marker and per sample.

```
histGBSR(gdata, stats = "ad_ref") # Histgrams of allelic read depth
```

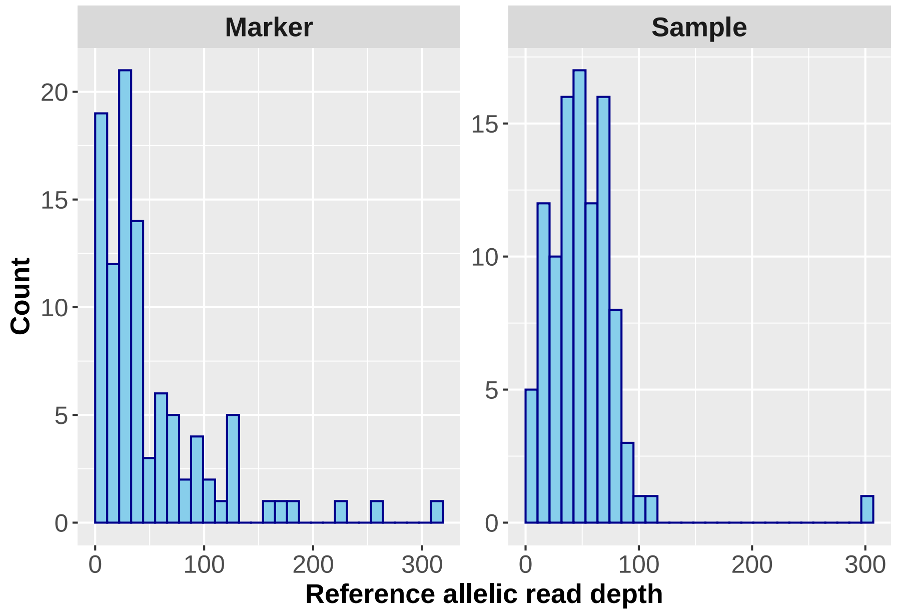


Figure 6: Alternative read depth per marker and per sample.

```r
histGBSR(gdata, stats = "rrf") # Histgrams of reference allele frequency
```

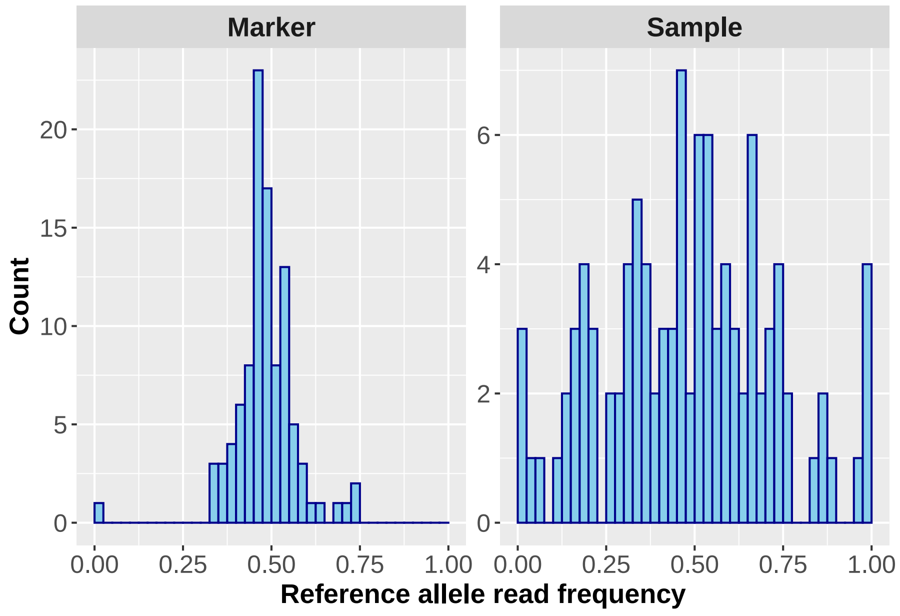


Figure 7: Reference read per marker and per sample.

In addition to `countGenotype` and `countRead`, we can get mean, sd, and quantile of read counts per marker and per sample. Unlike `countRead`, this function first normalize read counts by dividing each read count of both alleles at a marker in a sample by the total read count of the sample followed by multiplying it by 10ˆ6 to be read counts per million. This normalization allow us to compare read data distributions obtained for the samples without concern for absolute differences in total read counts between samples. This calculation takes a longer time than those by `countGenotype` and `countRead`.

```
gdata <- calcReadStats(gdata, q = 0.5)
```

The values specified for the “q” argument are passed to the “quantile” function internally to get quantiles. The “q” argument accepts a numeric vector and has `NULL` as default which let the function return no quantile.

To plot those statistics, we can also use `hist`.

```
histGBSR(gdata, stats = "mean_ref") # Histgrams of mean allelic read depth
```

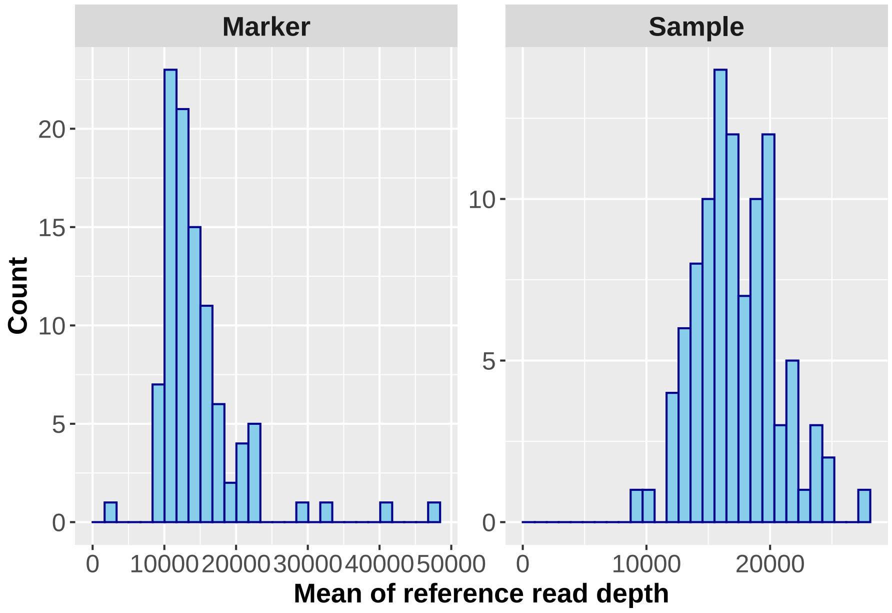


Figure 8: Mean of reference read depth per marker and per sample.

```
histGBSR(gdata, stats = "mean_ref") # Histgrams of mean allelic read depth
```

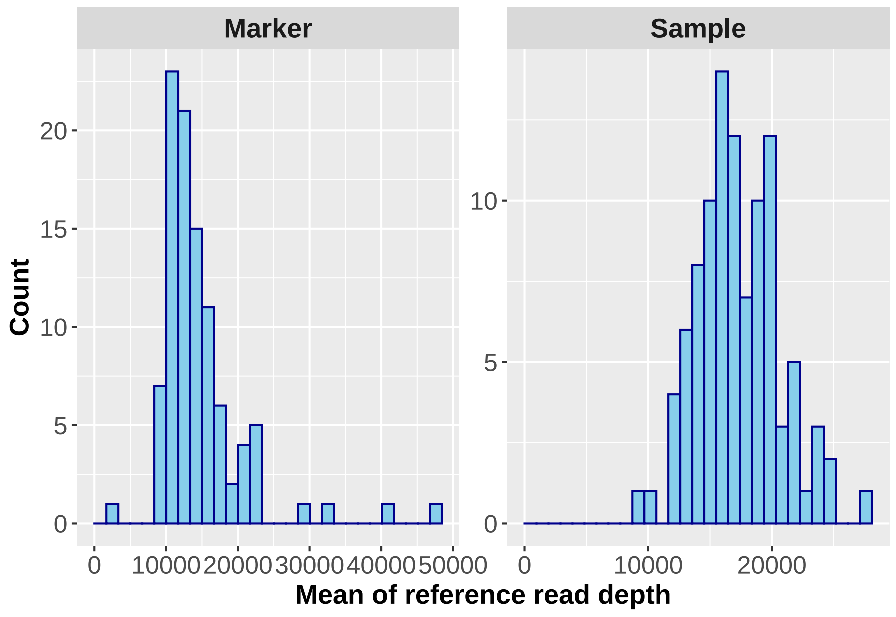


Figure 9: Mean of alternative read depth per marker and per sample.

```
histGBSR(gdata, stats = "sd_ref") # Histgrams of standard deviation of read depth
```

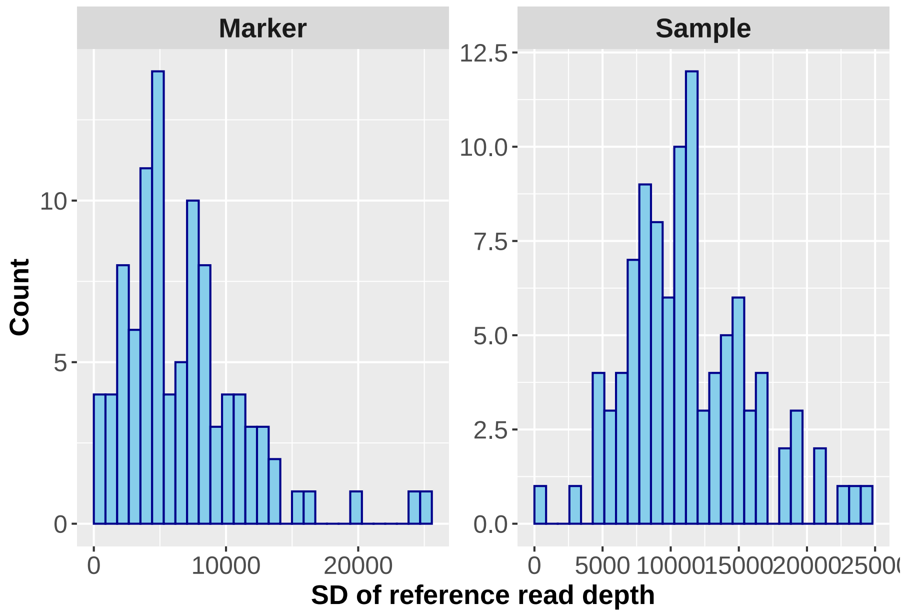


Figure 10: SD of reference read depth per marker and per sample.

```
histGBSR(gdata, stats = "sd_ref") # Histgrams of standard deviation of read depth
```

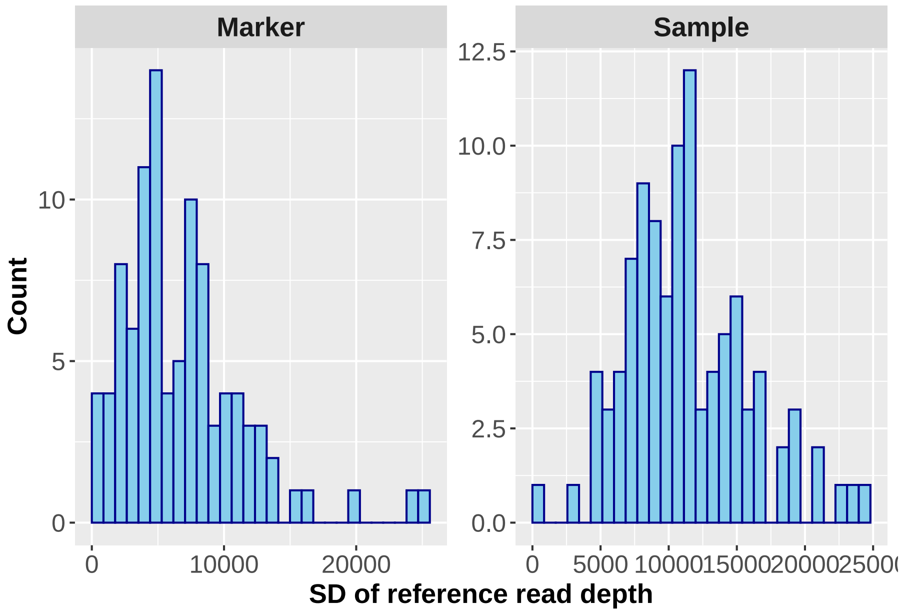


Figure 11: SD of alternative read depth per marker and per sample.

```
histGBSR(gdata, stats = "qtile_ref", q = 0.5) # Histgrams of quantile of read depth
```

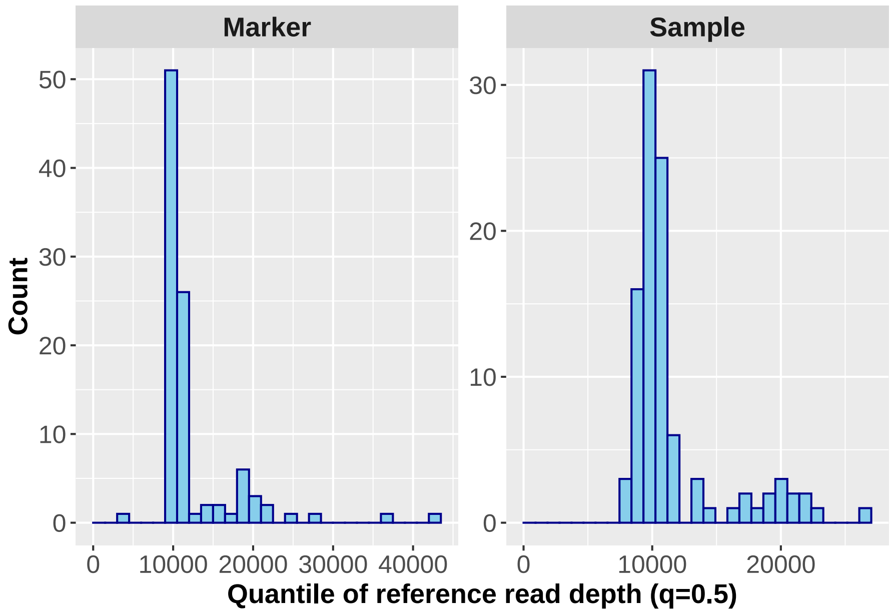


Figure 12: Quantile of reference read depth per marker and per sample.

```
histGBSR(gdata, stats = "qtile_ref", q = 0.5) # Histgrams of quantile of read depth
```

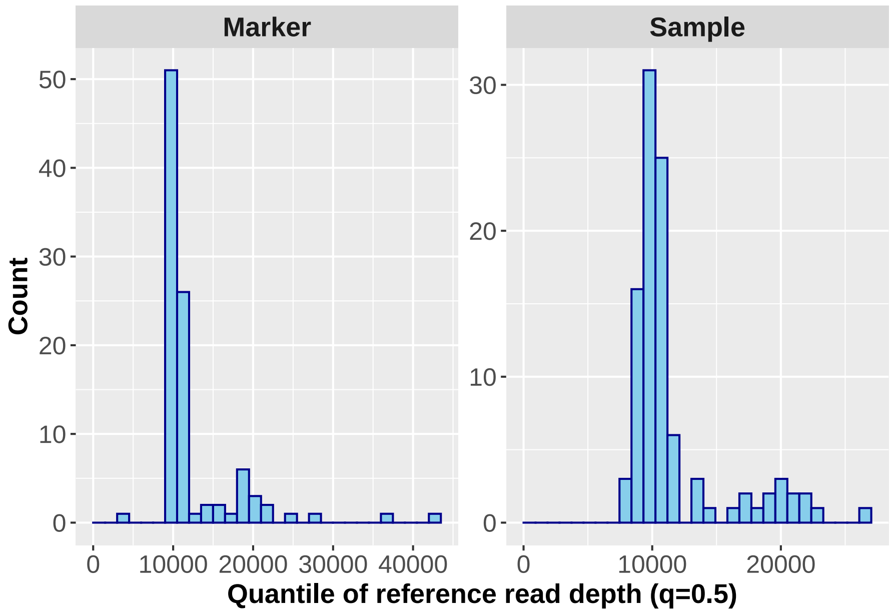
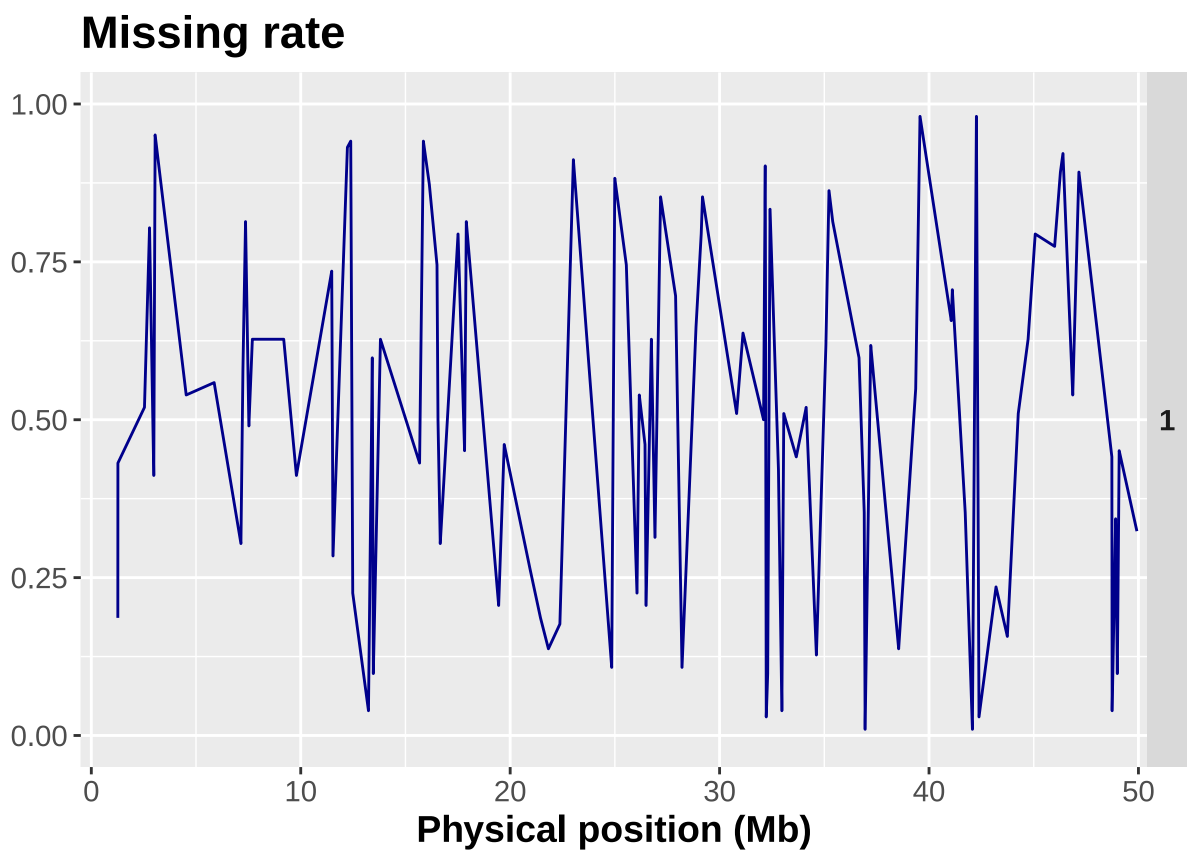


Figure 13: Quantile of alternative read depth per marker and per sample.

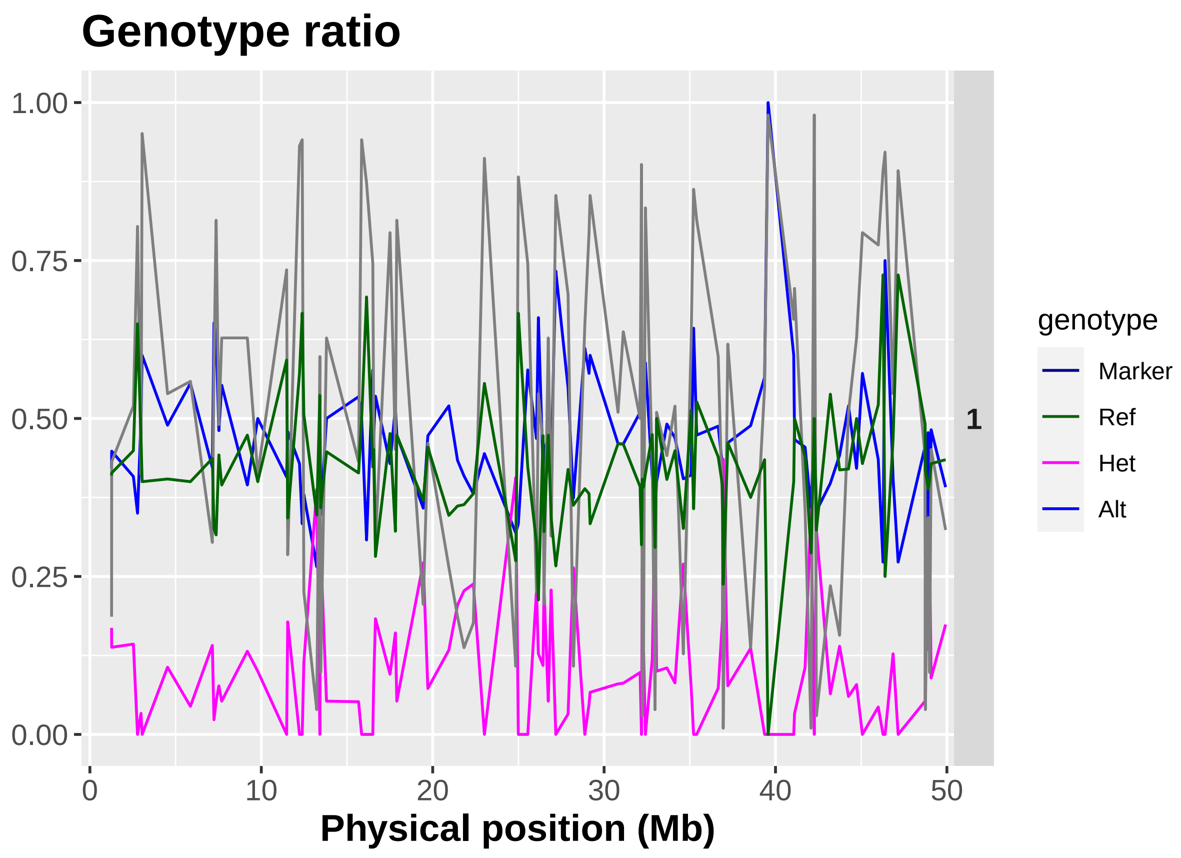
`plot()` and `pairs()` provide other ways to visualize statistics. `plot()` draws a line plot of a specified statistics per marker along each chromosome. `pairs()` give us a two-dimensional scatter plot to visualize relationship between statistics.

```
plotGBSR(gdata, stats = "missing", coord = c(6, 2))
```



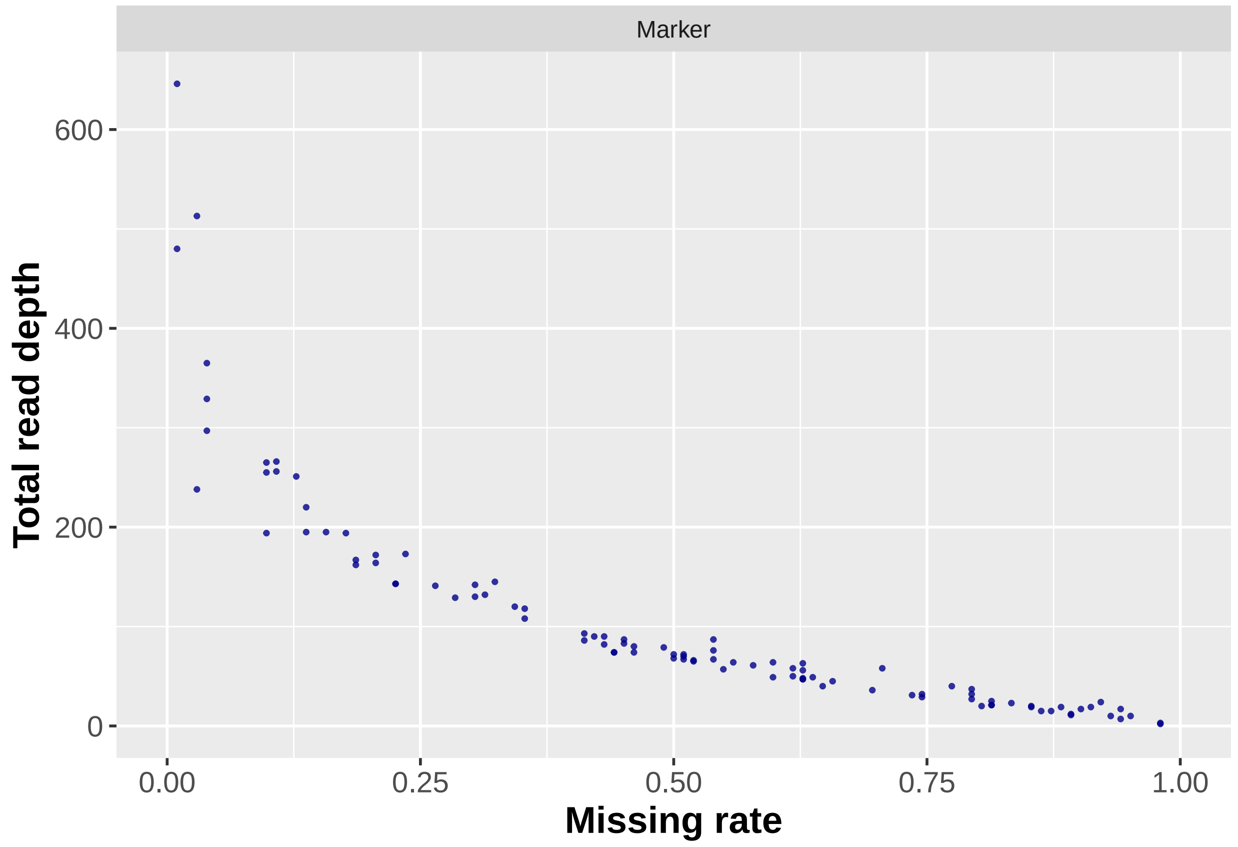
```r
# coord controls the number of rows and columns of facets.
```

```r
plotGBSR(gdata, stats = "geno", coord = c(6, 2))
```



```
# coord controls the number of rows and columns of facets.
```

```
pairsGBSR(gdata, stats1 = "missing", stats2 = "dp")
```



The statistics obtained via `countGenotype`, `countReat`, and `calcReadStats` are sotred in the `snpAnnot` and `scanAnnot` slots. They can be retrieved using getter functions as follows.

```
head(getCountGenoRef(gdata, target = "snp")) # Reference genotype count per marker
```

```
## [1] 34 24 22 13 28  2
```

```
head(getCountGenoRef(gdata, target = "scan")) # Reference genotype count per sample
```

```
## [1] 79  0 24 35 28  6
```

```
head(getCountGenoHet(gdata, target = "snp")) # Heterozygote count per marker
```

```
## [1] 14  8  7  0  2  0
```

```
head(getCountGenoHet(gdata, target = "scan")) # Heterozygote count per sample
```

```
## [1] 0 0 9 5 9 9
```

```
head(getCountGenoAlt(gdata, target = "snp")) # Alternative genotype count per marker
```

```
## [1] 35 26 20  7 30  3
```

```
head(getCountGenoAlt(gdata, target = "scan")) # Alternative genotype count per sample
```

```
## [1]  0 76 16 13 13 32
```

```
head(getCountGenoMissing(gdata, target = "snp")) # Missing count per marker
```

```
## [1] 19 44 53 82 42 97
```

```
head(getCountGenoMissing(gdata, target = "scan")) # Missing count per sample
```

```
## [1] 21 24 51 47 50 53
```

```
head(getCountAlleleRef(gdata, target = "snp")) # Reference allele count per marker
```

```
## [1] 82 56 51 26 58  4
```

```
head(getCountAlleleRef(gdata, target = "scan")) # Reference allele count per sample
```

```
## [1] 158   0  57  75  65  21
```

```
head(getCountAlleleAlt(gdata, target = "snp")) # Alternative allele count per marker
```

```
## [1] 84 60 47 14 62  6
```

```
head(getCountAlleleAlt(gdata, target = "scan")) # Alternative allele count per sample
```

```
## [1]   0 152  41  31  35  73
```

```
head(getCountAlleleMissing(gdata, target = "snp")) # Missing allele count per marker
```

```
## [1]  38  88 106 164  84 194
```

```
head(getCountAlleleMissing(gdata, target = "scan")) # Missing allele count per sample
```

```
## [1]  42  48 102  94 100 106
```

```
head(getCountReadRef(gdata, target = "snp")) # Reference read count per marker
```

```
## [1] 84 38 33 13 44  6
```

```
head(getCountReadRef(gdata, target = "scan")) # Reference read count per sample
```

```
## [1] 307   0  50  74  67  19
```

```
head(getCountReadAlt(gdata, target = "snp")) # Alternative read count per marker
```

```
## [1] 78 44 32  7 49  4
```

```
head(getCountReadAlt(gdata, target = "scan")) # Alternative read count per sample
```

```
## [1]   0 293  45  22  32  82
```

```
head(getCountRead(gdata, target = "snp"))
```

```
## [1] 162  82  65  20  93  10
```

```
# Sum of reference and alternative read counts per marker
head(getCountRead(gdata, target = "scan"))
```

```
## [1] 307 293  95  96  99 101
```

```
# Sum of reference and alternative read counts per sample
```

```
head(getMeanReadRef(gdata, target = "snp"))
```

```
## [1] 16870.262 11790.714 10970.672  9519.221 13575.372 13193.827
```

```
# Mean of reference allele read count per marker
head(getMeanReadRef(gdata, target = "scan"))
```

```
## [1] 12658.23      NaN 15948.96 19270.83 18291.02 12541.25
```

```
# Mean of reference allele read count per sample
```

```
head(getMeanReadAlt(gdata, target = "snp"))
```

```
## [1] 15220.400 12838.317 11356.384  9619.598 14993.132  8281.562
```

```
# Mean of Alternative allele read count per marker
head(getMeanReadAlt(gdata, target = "scan"))
```

```
## [1]      NaN 13157.89 18947.37 12731.48 14692.38 19801.98
```

```
# Mean of Alternative allele read count per sample
```

```
head(getSDReadRef(gdata, target = "snp"))
```

```
## [1] 9382.715 4449.872 3078.132 2046.770 5260.520 4373.904
```

```
# SD of reference allele read count per marker
head(getSDReadRef(gdata, target = "scan"))
```

```
## [1] 11973.729        NA 11804.388 13038.183 18262.360  7908.997
```

```
# SD of reference allele read count per sample
```

```
head(getSDReadAlt(gdata, target = "snp"))
```

```
## [1] 7325.900 5564.233 4302.412 3131.060 6031.649 1398.706
```

```
# SD of Alternative allele read count per marker
head(getSDReadAlt(gdata, target = "scan"))
```

```
## [1]        NA 12656.598 12892.051  4456.173  6777.593 18654.895
```

```
# SD of Alternative allele read count per sample
```

```
head(getQtileReadRef(gdata, target = "snp", q = 0.5))
```

```
## [1] 11579.65 10471.49 10204.08 10000.00 10752.69 13193.83
```

```
# Quantile of reference allele read count per marker
head(getQtileReadRef(gdata, target = "scan", q = 0.5))
```

```
## [1]  9771.987        NA 10526.315 10416.667 10101.010  9900.990
```

```
# Quantile of reference allele read count per sample
```

```
head(getQtileReadAlt(gdata, target = "snp", q = 0.5))
```

```
## [1] 11111.111 10204.082 10204.082 10309.278 13247.863  8403.361
```

```
# Quantile of Alternative allele read count per marker
head(getQtileReadAlt(gdata, target = "scan", q = 0.5))
```

```
## [1]       NA 10238.91 10526.32 10416.67 10101.01  9900.99
```

```
# Quantile of Alternative allele read count per sample
```

```
head(getMAF(gdata, target = "snp")) # Minor allele frequency per marker
```

```
## [1] 0.4939759 0.4827586 0.4795918 0.3500000 0.4833333 0.4000000
```

```
head(getMAF(gdata, target = "scan")) # Minor allele frequency per sample
```

```
## [1] 0.0000000 0.0000000 0.4183673 0.2924528 0.3500000 0.2234043
```

```r
head(getMAC(gdata, target = "snp")) # Minor allele count per marker
```

```
## [1] 82 56 47 14 58  4
```

```r
head(getMAC(gdata, target = "scan")) # Minor allele count per sample
```

```
## [1]  0  0 41 31 35 21
```

You can get the proportion of each genotype call with `prop = TRUE`.

```r
head(getCountGenoRef(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.4096386 0.4137931 0.4489796 0.6500000 0.4666667 0.4000000
```

```r
head(getCountGenoHet(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.16867470 0.13793103 0.14285714 0.00000000 0.03333333 0.00000000
```

```r
head(getCountGenoAlt(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.4216867 0.4482759 0.4081633 0.3500000 0.5000000 0.6000000
```

```r
head(getCountGenoMissing(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.1862745 0.4313725 0.5196078 0.8039216 0.4117647 0.9509804
```

The proportion of each allele counts.

```r
head(getCountAlleleRef(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.4939759 0.4827586 0.5204082 0.6500000 0.4833333 0.4000000
```

```r
head(getCountAlleleAlt(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.5060241 0.5172414 0.4795918 0.3500000 0.5166667 0.6000000
```

```r
head(getCountAlleleMissing(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.1862745 0.4313725 0.5196078 0.8039216 0.4117647 0.9509804
```

The proportion of each allele read counts.

```r
head(getCountReadRef(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.5185185 0.4634146 0.5076923 0.6500000 0.4731183 0.6000000
```

```r
head(getCountReadAlt(gdata, target = "snp", prop = TRUE))
```

```
## [1] 0.4814815 0.5365854 0.4923077 0.3500000 0.5268817 0.4000000
```

## Filtering and subsetting data

Based on the statistics we obtained, we can filter out less reliable markers and samples using `setSnpFilter` and `setScanFilter`.

```
# Not run
gdata <- setSnpFilter(
  id,   # Specify a character vector of snpID to be removed.
  missing = 1,   # Specify an upper limit of missing rate.
  het = c(0, 1),   # Specify a lower and an upper limit of heterozygosity rate.
  mac = 0,   # Specify a lower limit of minor allele count.
  maf = 0.05,   # Specify a lower limit of minor allele frequency.
  ad_ref = c(0, Inf), # Specify a lower and an upper limit of reference allele count.
  ad_alt = c(0, Inf), # Specify a lower and an upper limit of alternative allele count.
  dp = c(0, Inf), # Specify a lower and an upper limit of total read count.
  mean_ref = c(0, Inf),
  # Specify a lower and an upper limit of mean reference allele count.
  mean_alt = c(0, Inf),
  # Specify a lower and an upper limit of mean alternative allele count.
  sd_ref = Inf, # Specify a lower and an upper limit of SD of reference allele count.
  sd_alt = Inf # Specify a lower and an upper limit of SD of alternative allele count.
)

gdata <- setScanFilter(
  id,   # Specify a character vector of snpID to be removed.
  missing = 1,   # Specify an upper limit of missing rate.
  het = c(0, 1),   # Specify a lower and an upper limit of heterozygosity rate.
  mac = 0,   # Specify a lower limit of minor allele count.
  maf = 0,   # Specify a lower limit of minor allele frequency.
  ad_ref = c(0, Inf),   # Specify a lower and an upper limit of reference allele count.
  ad_alt = c(0, Inf),   # Specify a lower and an upper limit of alternative allele count.
  dp = c(0, Inf),   # Specify a lower and an upper limit of total read count.
  mean_ref = c(0, Inf),
  # Specify a lower and an upper limit of mean reference allele count.
  mean_alt = c(0, Inf),
  # Specify a lower and an upper limit of mean alternative allele count.
  sd_ref = Inf,   # Specify a lower and an upper limit of SD of reference allele count.
  sd_alt = Inf   # Specify a lower and an upper limit of SD of alternative allele count.
)
```

`setCallFilter()` is another type of filtering which works on each genotype call. We can replace some genotype calls with missing. If you would like to filter out less reliable genotype calls supported by less than 5 reads, set the arguments as below.

```
gdata <- setCallFilter(gdata, dp_count = c(5, Inf))
```

If need to remove genotype calls supported by too many reads, which might be the results of mismapping from repetitive sequences, set as follows.

```
gdata <- setCallFilter(gdata, norm_dp_count = c(0, 1000))
gdata <- setCallFilter(gdata, norm_ref_count = c(0, 1000),
                       norm_alt_count = c(0, 800))
```

Usually reference reads and alternative reads show different data distributions. Thus, we can set the different thresholds for them via `norm_ref_count` and `norm_alt_count`. `setCallFilter()` also has arguments `scan_ref_qtile`, `scan_alt_qtile`, `snp_ref_qtile`, and `snp_alt_qtile` to filter out genotype calls based on quantiles of read counts per marker and per sample.

Here, let's filter out calls supported by less than 5 reads and then filter out markers having more than 10% of missing rate.

```
gdata <- setCallFilter(gdata, dp_count = c(5, Inf))
gdata <- setSnpFilter(gdata, missing = 0.1)
```

In addition to those statistics based filtering functions, `GBScleanR` provides filtering function based on relative marker positions. Markers locating too close each other usually have redundant information, especially if those markers are closer each other than the read length, in which case the markers are supported by completely (or almost) the same set of reads. To select only one marker from those markers, we can sue `thinMarker`. This function selects one marker having the least missing rate from each stretch with the specified length. If some markers have the least missing rate, select the first marker in the stretch.

```
thinMarker(gdata, range = 150) # Here we select only one marker from each 150 bp stretch.
```

```
## File: /home/ftom/hdd2/softDevel/GBScleanR/inst/extdata/simpop.gds (28.6K)
## +    [  ] *
## |--+ sample.id   { Str8 102 LZMA_ra(16.9%), 245B }
## |--+ snp.id   { Int32 100 LZMA_ra(48.5%), 201B }
## |--+ snp.rs.id   { Str8 100 LZMA_ra(77.4%), 233B }
## |--+ snp.position   { Int32 100 LZMA_ra(104.5%), 425B }
## |--+ snp.allele   { Str8 100 LZMA_ra(22.5%), 97B }
## |--+ genotype   { Bit2 102x100 LZMA_ra(87.6%), 2.2K } *
## |--+ annotation   [  ]
## |  |--+ info   [  ]
## |  \--+ format   [  ]
## |     |--+ AD   [  ] *
## |     |  |--+ data   { VL_Int 102x200 LZMA_ra(20.7%), 4.1K } *
## |     |  |--+ norm   { Float32 200x102 LZMA_ra(8.18%), 6.5K }
## |     |  |--+ filt.scan   { Bit1 100x102 LZMA_ra(34.0%), 441B }
## |     |  \--+ filt.data   { VL_Int 102x200 LZMA_ra(4.56%), 937B }
## |     \--+ DP   [  ] *
## |        \--+ data   { VL_Int 102x100 LZMA_ra(28.6%), 3.0K } *
## |--+ snp.chromosome.name   { Str8 100 LZMA_ra(43.0%), 93B }
## |--+ snp.chromosome   { Int8 100 LZMA_ra(82.0%), 89B }
## \--+ filt.genotype   { Bit2 102x100 LZMA_ra(18.1%), 469B }
## An object of class 'SnpAnnotationDataFrame'
##   snps: 1 2 ... 100 (100 total)
##   varLabels: snpID chromosome ... qtileReadAlt0.5 (23 total)
##   varMetadata: labelDescription
## An object of class 'ScanAnnotationDataFrame'
##   scans: 1 2 ... 102 (102 total)
##   varLabels: scanID validScan
##   varMetadata: labelDescription
```

We can obtain the summary statistics using `countGenotype()`, `countRead()`, and `calcReadStats()` for only the SNPs and samples retained after filtering with the same codes we used before.

```
gdata <- countGenotype(gdata)
gdata <- countRead(gdata)
gdata <- calcReadStats(gdata)
```

`calcReadStats()` never calculate the normalized read counts again for the filtered data but gets mean, sd, and quantiles from the normalized values of the retained markers of samples.

We can check which markers and samples are retained after the filtering using `getValidSnp()` and `getValidScan()`.

```
head(getValidSnp(gdata))
```

```
## [1] FALSE FALSE FALSE FALSE FALSE FALSE
```

```
head(getValidScan(gdata))
```

```
## [1] TRUE TRUE TRUE TRUE TRUE TRUE
```

The class methods of `gbsrGenotypeData` basically work with only the markers and samples having `TRUE` in the returned values of `getValidSnp()` and `getValidScan()`, if you don't explicitly specify `valid = FALSE` as an argument of the class methods.

```
nsnp(gdata)
```

```
## [1] 10
```

```
nsnp(gdata, valid = FALSE)
```

```
## snps
##  100
```

We can reset filtering as following.

```
gdata <- resetSnpFilters(gdata) # Reset the filter on markers
gdata <- resetScanFilters(gdata) # Reset the filter on samples
gdata <- resetCallFilters(gdata) # Reset the filter on calls
gdata <- resetFilters(gdata) # Reset all filters
```

To save the filtered data, we can create the subset GDS file containing only the retained data.

```
subset_gdata <- subsetGDS(gdata,
                          out_fn = "simpop_subset.gds",
                          snp_incl = getValidSnp(gdata),
                          scan_incl = getValidScan(gdata))
```

`out_fn` is the file path of the output GDS file storing the subset data. Users need to specify, for `snp_incl` and `scan_incl`, a logical vector indicating which markers and samples should be included in the subset. The functions `getValidSnp()` and `getValidScan` return a logical vector indicating which markers and samples are retained by `setSnpFilter()` and `setScanFilter()`. `subsetGDS` returns a new `gbsrGenotypeData` object for the subset.

```
closeGDS(gdata)
```

## Session information

```r
sessionInfo()
```

```
## R version 4.1.1 (2021-08-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=C
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
## [1] GBScleanR_0.99.0    GWASTools_1.38.0    Biobase_2.52.0
## [4] BiocGenerics_0.38.0
##
## loaded via a namespace (and not attached):
##   [1] nlme_3.1-152           bitops_1.0-7           matrixStats_0.61.0
##   [4] fs_1.5.0               usethis_2.0.1          devtools_2.4.2
##   [7] bit64_4.0.5            rprojroot_2.0.2        GenomeInfoDb_1.28.4
##  [10] tools_4.1.1            backports_1.2.1        utf8_1.2.2
##  [13] R6_2.5.1               DBI_1.1.1              mgcv_1.8-37
##  [16] colorspace_2.0-2       DNAcopy_1.66.0         withr_2.4.2
##  [19] tidyselect_1.1.1       prettyunits_1.1.1      processx_3.5.2
##  [22] bit_4.0.4              compiler_4.1.1         cli_3.0.1
##  [25] quantreg_5.86          expm_0.999-6           mice_3.13.0
##  [28] SparseM_1.81           xml2_1.3.2             desc_1.4.0
##  [31] sandwich_3.0-1         labeling_0.4.2         scales_1.1.1
##  [34] lmtest_0.9-38          quantsmooth_1.58.0     callr_3.7.0
##  [37] digest_0.6.28          commonmark_1.7         stringr_1.4.0
##  [40] GWASExactHW_1.01       rmarkdown_2.11         XVector_0.32.0
##  [43] htmltools_0.5.2        pkgconfig_2.0.3        sessioninfo_1.1.1
##  [46] fastmap_1.1.0          rlang_0.4.11           rstudioapi_0.13
##  [49] RSQLite_2.2.8          farver_2.1.0           generics_0.1.0
##  [52] zoo_1.8-9              dplyr_1.0.7            RCurl_1.98-1.5
##  [55] magrittr_2.0.1         GenomeInfoDbData_1.2.6 Matrix_1.3-4
##  [58] Rcpp_1.0.7             munsell_0.5.0          S4Vectors_0.30.1
##  [61] fansi_0.5.0            lifecycle_1.0.1        yaml_2.2.1
##  [64] stringi_1.7.4          zlibbioc_1.38.0        pkgbuild_1.2.0
##  [67] grid_4.1.1             formula.tools_1.7.1    blob_1.2.2
##  [70] crayon_1.4.1           lattice_0.20-44        Biostrings_2.60.2
##  [73] splines_4.1.1          knitr_1.36             ps_1.6.0
##  [76] pillar_1.6.3         GenomicRanges_1.44.0   logistf_1.24
##  [79] gdsfmt_1.28.1        stats4_4.1.1           pkgload_1.2.2
##  [82] glue_1.4.2           evaluate_0.14          RcppParallel_5.1.4
##  [85] data.table_1.14.2    remotes_2.4.0          operator.tools_1.6.3
##  [88] vctrs_0.3.8          testthat_3.0.4         MatrixModels_0.5-0
##  [91] gtable_0.3.0         purrr_0.3.4            tidyr_1.1.4
##  [94] SeqArray_1.32.0      cachem_1.0.6           ggplot2_3.3.5
##  [97] xfun_0.26            broom_0.7.9            roxygen2_7.1.2
## [100] survival_3.2-13      tibble_3.1.5           conquer_1.0.2
## [103] memoise_2.0.0        IRanges_2.26.0         ellipsis_0.3.2
```