Learn: vcfR

From: vcfR Documentation

Contents

Setup **1**

A quick introduction **1**
Preliminaries 1
Data input 1
Creating chromR objects 2
Processing chromR objects 2

These tutorials are based on the official `vcfR` tutorials from the vcfR documentation

Setup

```
library(vcfR)
```

```
##
##    *****       ***   vcfR   ***       *****
##    This is vcfR 1.8.0
##      browseVignettes('vcfR') # Documentation
##      citation('vcfR') # Citation
##    *****       *****      *****       *****
```

A quick introduction

vcfR tutorial: A quick introduction

Preliminaries

Since R reads all data into memory, sometimes it's best to split VCF data up into chromosomes.

Data input

vcfR works with VCF files and you can also add FASTA and GFF files for context, but these aren't required.

```
library(pinfsc50)
pkg = "pinfsc50"
vcf_file = system.file("extdata", "pinf_sc50.vcf.gz", package = pkg)
dna_file = system.file("extdata", "pinf_sc50.fasta", package = pkg)
gff_file = system.file("extdata", "pinf_sc50.gff", package = pkg)
```

Read in the VCF file:

```
vcf = read.vcfR(vcf_file, verbose=F)
```

The file is stored as a vcfR object (S4 class) with 3 slots, one each for metadata, fixed data, and genotype data.

Read in FASTA files with the `ape` package:

```
library("ape")
dna = read.dna(dna_file, format = "fasta")
```

Annotation files contain coordinates for genomic annotations. GFF is currently supported. Read in a GFF file with `read.table()`:

```
gff = read.table(gff_file, sep = "\t", quote = "")
```

vcfR was designed to work with individual chromosomes as reading an entire genome into memory is a technical challenge.

Creating chromR objects

Once data is in memory, use `create.chromR()` to create a chromR object and populate it with VCF data.

```
chrom = create.chromR(name="Supercontig", vcf = vcf, seq = dna, ann = gff)
```

```
## Names in vcf:

##   Supercontig_1.50

## Names of sequences:

##   Supercontig_1.50 of Phytophthora infestans T30-4

## Warning in create.chromR(name = "Supercontig", vcf = vcf, seq = dna, ann = gff):
##         Names in variant data and sequence data do not match perfectly.
##         If you choose to proceed, we'll do our best to match the data.
##         But prepare yourself for unexpected results.

## Names in annotation:

##   Supercontig_1.50

## Initializing var.info slot.

## var.info slot initialized.
```

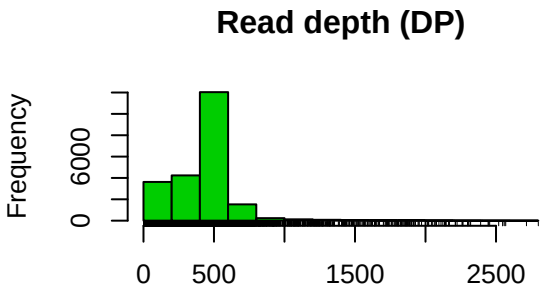
Notice the warning that the contig names are not exactly the same bc of different sources. In this case the warning can be ignored because they refer to the same data.

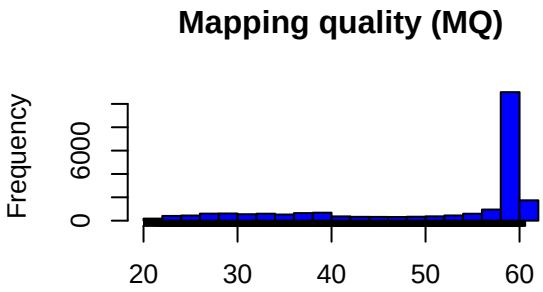
The `name` parameter is the name of the chromR object and used when plotting it.

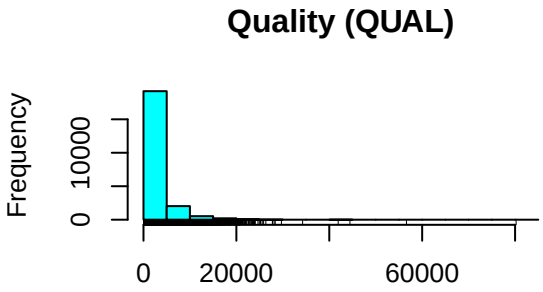
Processing chromR objects

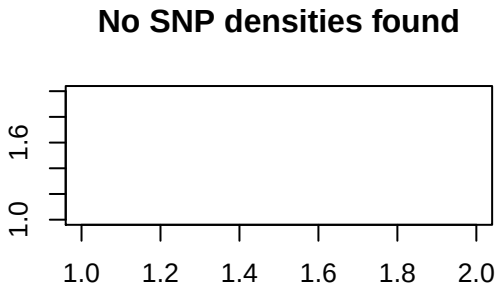
Get a quick look at the chromR object data by plotting.

```
plot(chrom)
```



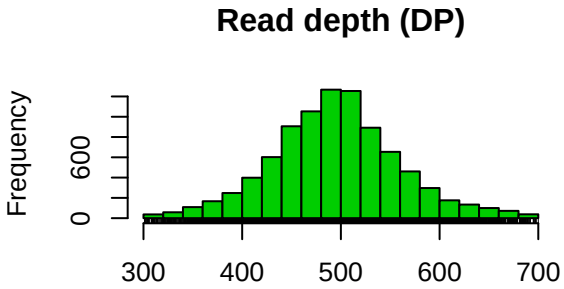


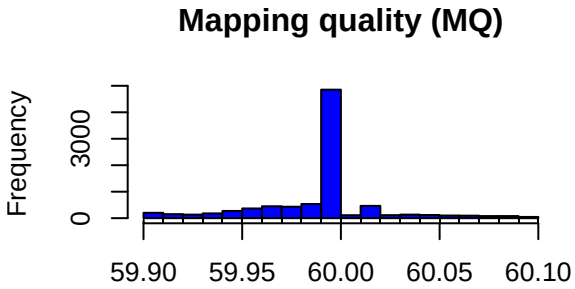


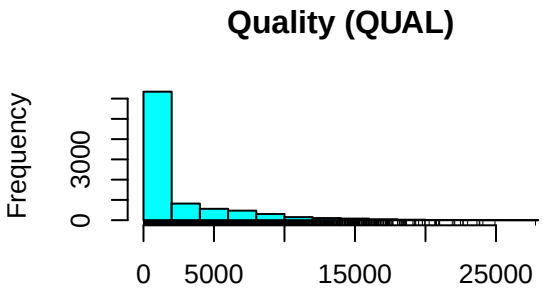


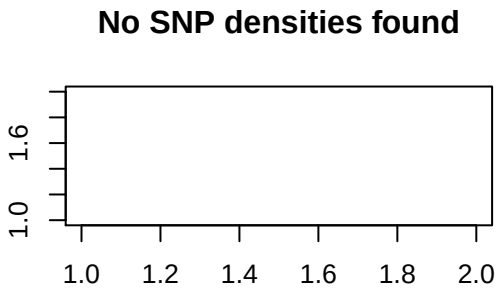
Use the `masker()` function to filter out low-confidence data. It uses quality, depth, and mapping quality to try and select high quality variants. Low quality variants are not deleted, but instead a logical vector is made to indicate which variants have been filtered.

```
chrom = masker(chrom, min_QUAL = 1, min_DP = 300, max_DP = 700, min_MQ = 59.9, max_MQ = 60.1)
plot(chrom)
```









Once satisfied with filtering and the resulting set of high-quality variants, process the chromR object with `proc.chromR()`. This calls several helper functions to process the variant, sequence, and annotation data for

viz.

```
chrom = proc.chromR(chrom, verbose = T)
```

```
## Nucleotide regions complete.
##   elapsed time:  0.264
## N regions complete.
##   elapsed time:  0.243
## Population summary complete.
##   elapsed time:  0.251
## window_init complete.
##   elapsed time:  0
## windowize_fasta complete.
##   elapsed time:  0.136
## windowize_annotations complete.
##   elapsed time:  0.018
## windowize_variants complete.
##   elapsed time:  0.001
```

```
plot(chrom)
```