SNPfiltR: an R package for interactive and reproducible SNP filtering

Article *in* Molecular Ecology Resources · April 2022

DOI: 10.1111/1755-0998.13618

1 author:

Devon DeRaad
University of Kansas
13 PUBLICATIONS **45** CITATIONS

SEE PROFILE

Received: 10 November 2021 | Revised: 5 March 2022 | Accepted: 23 March 2022

DOI: 10.1111/1755-0998.13618

MOLECULAR ECOLOGY RESOURCES WILEY

RESOURCE ARTICLE

SNPFILTR: An R package for interactive and reproducible SNP filtering

Devon A. DeRaad

Department of Ecology & Evolutionary Biology and Biodiversity Institute, University of Kansas, Lawrence, Kansas, USA

Correspondence
Devon A. DeRaad, Department of Ecology & Evolutionary Biology and Biodiversity Institute, University of Kansas, Lawrence, KS, USA.
Email: devonderaad@gmail.com

Handling Editor: Alex C. Buerkle

Abstract

Here, I describe the R package SNPFILTR and demonstrate its functionality as the backbone of a customizable, reproducible single nucleotide polymorphism (SNP) filtering pipeline implemented exclusively via the widely adopted R programming language. SNPfiltR extends existing SNP filtering functionalities by automating the visualization of key parameters such as sequencing depth, quality, and missing data proportion, allowing users to visually optimize and implement filtering thresholds within a single, cohesive work session. All SNPfiltR functions require VCFR objects as input, which can be easily generated by reading a SNP data set stored in standard variant call format (vcf) into an R working environment using the function read.vcfR() from the R package VCFR. Performance and accuracy benchmarking reveal that for moderately sized SNP data sets (up to 50 M genotypes, plus associated quality information), SNPfiltR performs filtering with comparable accuracy and efficiency to current state of the art command-line-based programs. These results indicate that for most reduced-representation genomic data sets, SNPfiltR is an ideal choice for investigating, visualizing, and filtering SNPs as part of a user friendly bioinformatic pipeline. The SNPFILTR package can be downloaded from CRAN with the command install.packages("SNPFILTR"), and the current development version is available from GitHub at: (github.com/DevonDeRaad/SNPfiltR). Thorough documentation for SNPfiltR, including multiple comprehensive vignettes detailing realistic use-cases, is available at the website: devonderaad.github.io/SNPfiltR/.

KEYWORDS
bioinfomatics/phyloinfomatics, genomics, missing data, R package, reproducibility, SNP filtering

1 | INTRODUCTION

As next-generation (i.e., massively parallel, short read) sequencing has become the ubiquitous approach for population genetic and phylogenetic investigations, SNP (single-nucleotide polymorphism) data sets have quickly become the standard input data for a wide array of phylogenetic and population genetic analyses (Toews et al., 2015). SNP data sets typically contain thousands to millions of base-calls for each individual sample (i.e., genotypes), located at thousands to millions of variable sites (i.e., SNPs) throughout the genome of the focal taxa. For storing these genotype calls along with associated quality information, the efficient formatting of the variant call format (vcf) file has become the widely accepted standard (Danecek et al., 2011). By retaining only variant sites (i.e., called SNPs), large population genomic data sets can be stored in vcf files with manageable file sizes and maximal computational tractability for downstream analyses.

As SNP data sets stored as vcf files have become standard input for population genetic and phylogenetic analyses, programs to call SNPs from next generation sequencing data have proliferated rapidly (e.g., GATK [McKenna et al., 2010], SAMTOOLS [Danecek et al., 2021], STACKS [Rochette et al., 2019], DDOCENT [Puritz et al., 2014], IPYRAD [Eaton & Overcast, 2020]). While these programs are optimized to call SNPs rapidly and accurately, called SNPs will inevitably suffer from a variety of technical issues such as sequencing error, paralagous assembly, and missing data, which must be addressed before performing downstream analyses (O'Leary et al., 2018). Further, individual samples may suffer from low sequencing coverage or contamination, necessitating their removal prior to performing downstream analyses (Cerca et al., 2021). To address these issues, some SNP calling programs contain built in functionality for filtering output SNP data sets (e.g., GATK and STACKS), but these functionalities often leave much to be desired for investigators looking to perform thorough explorations of parameter space and deeply understand the nuances of their particular SNP data set. These functional limitations result in a gap in many bioinformatic pipelines, especially for SNP data sets generated via reduced-representation genomic sequencing, where de novo assembly and rampant missing data typically necessitate careful filtering in order to maximize retained signal while minimizing systematic error (O'Leary et al., 2018).

Currently, this bioinformatic gap is often addressed via informal data visualizations implemented across multiple programs and programming languages. As reproducibility is becoming more widely acknowledged as critical for the future of science (Papin et al., 2020), effectively documenting the iterative process of investigating, visualizing, and cleaning large data sets continues to be a major challenge. Current state of the art precompiled programs for filtering SNP data sets such as GATK and VCFTOOLS (Danecek et al., 2011) are highly efficient and parallelizable, making them especially useful for massive data sets. Nonetheless, their command-line interfaces do not lend themselves easily to graphical visualization, leading many investigators to rely on custom scripts for preliminary investigation and visualization of parameter space. A common homebrewed approach is to generate files containing summary statistics describing the data set of interest using a command-line based program or Unix scripting, and then investigating and visualizing these statistics using the renowned data visualization tools of the R (R Core Team, 2019) computing language. This approach requires moving between scripting languages and writing custom code to perform visualizations, creating a steep learning curve for inexperienced users, and resulting in pipelines that may be error prone and difficult to document. The relative frequency of this homebrewed approach indicates the outstanding need within the field of evolutionary genomics for user-friendly software that automates and streamlines the process of investigating, visualizing, and filtering SNP data sets.

The R package SNPFILTRis designed to fill this bioinformatic gap with a suite of custom functions designed for visualizing and filtering reduced-representation SNP data sets within a coherent R-based framework. As input, these functions take standard vcf files stored in an R working environment as "vcfR" objects, a process which can be performed in a single step using the function read.vcfR() from the R package VCFR (Knaus & Grünwald, 2017). The suite of custom functions from SNPfiltR can then be used to generate automated visualizations of key parameters such as sequencing depth, quality, and missing data proportions. Users can then specify visually optimized filtering thresholds based on the nuances of their particular data set, for efficient filtering within the same R working environment. Finally, functions from SNPFILTR can be used in concert with import and export functions from VCFR, to generate an interactive, customizable SNP filtering pipeline, all within a single R script. Multiple realistic examples using SNPFILTR as the main component of comprehensive pipelines for filtering reduced representation genomic SNP data sets are available at the following website: (devonderaad.github.io/SNPfiltR/).

2 | MATERIALS AND METHODS

2.1 | Example data sets

SNPfiltR is distributed via the Comprehensive R Archive Network (CRAN) with a provided example data set. Users can install the package and load this example data set by calling install.packages("SNPFILTR"); data(vcfR.example). The small size of this example data set, which contains 500 SNPs from 20 individual samples (10 K unique genotypes), allows for its distribution with the SNPfiltR package without pushing the distributed package over the 1 Mb limit imposed by CRAN. Nonetheless this example data set, a subset of a real empirical SNP data set, retains sufficient variation for rapid testing and validation of SNPfiltR functions. For SNPfiltR functions that require an input "popmap" which maps individual samples in the input vcf file to putative species/populations, an example can be accessed by calling data(popmap) once SNPfiltR has been successfully installed. A fully documented SNP filtering pipeline using this example data set is publicly available at: devonderaad.github.io/SNPfiltR/articles/reproducible-vignette.html.

I used additional example data sets to provide fully worked vignettes integrating functions from SNPfiltR and vcfR into fully R-based, customizable SNP filtering pipelines for genomic data sets resulting from restriction-site associated DNA sequencing (RADseq) (Davey & Blaxter, 2010) (available at: devonderaad.github.io/SNPfiltR/articles/scrub-jay-RADseq-vignette.html) and the sequencing of ultra-conserved elements (UCEs) (Faircloth et al., 2012) (available at: devonderaad.github.io/SNPfiltR/articles/scrub-jay-UCE-vignette.html). The RADseq vignette uses as input a vcf file containing 210,336 unfiltered SNPs for 115 individuals, called using STACKSv.2.41 (Rochette et al., 2019). This empirical data set from throughout the entire distribution of Scrub-Jays (genus *Aphelocoma*) across North America is publicly available from the following source (https://doi.org/10.5281/zenodo.6382078). The UCE vignette uses as input an unfiltered vcf file containing 44,490 unfiltered SNPs for 28 samples, called using PHYLUCE (Faircloth, 2016) and GATK (McKenna et al., 2010). This data set was the focus of McCormack et al. (2016), and is publicly

available for download via the Dryad repository: (datadryad.org/stash/dataset/doi:10.5061/dryad.qh8sh).

2.2 | Novel functions for visualizing and filtering SNP data sets in R

The SNPFILTR package relies on the efficient import and export functions of the VCFR package to read vcf files into the local memory of an R working environment as vcfR objects, and to write vcfR objects to disc as gzipped vcf files. Each SNPfiltR function takes a vcfR object as input, and can be run without specified thresholds or cutoffs. For instance, hard_filter(vcfR=vcfR.object) will visualize the distribution of genotype depth and quality in the input file without performing filtering, which allows users to quickly make informed decisions about potential filtering thresholds based on patterns specific to their input data set. The same function is then capable of implementing the chosen filtering thresholds (e.g., hard_filter(vcfR=vcfR.object, depth=5, gq=30)). SNPfiltR contains a suite of commonly implemented filters for genomic data sets, including filtering based on genotype quality, minimum and maximum read depth, allele balance, number of alleles present, missing data per sample, missing data per SNP, minor allele count, and genomic distance between SNPs. While most of these filters can already be implemented in other programs (e.g., VCFTOOLSand GATK), SNPfiltR is the first program offering dedicated functions for a comprehensive suite of SNP visualization and filtering options. Each SNP filtering function can be implemented at the discretion of the user, to build an interactive SNP filtering pipeline customized to the specific needs of a given genomic data set.

Beyond simply filtering, SNPfiltR also contains functions to automate investigation of the downstream effects of missing data on a SNP data set. The SNPfiltR functions assess_missing_data_pca() and assess_missing_data_tsne() are designed to perform dimensionality reduction on highly multidimensional SNP data sets, using principal components analysis (PCA) via the R package ADEGENET (Jombart, 2008) and t-distributed stochastic neighbour embedding implemented via the R package RTSNE (Krijthe & van der Maaten, 2015). Both of these functions then visualize the similarity between input samples in two-dimensional space, across a user specified suite of per SNP missing data thresholds. Users also have the option to perform unsupervised clustering to assign samples to groups without a priori information using partitioning around medoids (PAM) implemented internally via the R package cluster (Maechler et al., 2018). Finally, each of these functions will generate an additional visualization of sample similarity in two-dimensional space with samples colour-coded by missing data proportion, allowing the user to visually assess whether missing data is driving patterns of sample clustering. These investigative functions can be used in tandem with the functions missing_by_snp() and missing_by_sample(), to ensure that user specified missing data thresholds, both per sample and per SNP, are sufficient for mitigating the potentially confounding effects of missing data.

2.3 | Performance and accuracy benchmarking

To evaluate the performance of SNPfiltR, I compared filtering runtimes and accuracy with the widely used program VCFTOOLS (Danecek et al., 2011). VCFTOOLS is a highly efficient command-line based program written in Perl and C++, which is frequently used for filtering vcf files. VCFTOOLS can parse and filter a vcf file without having to read the entire file into local memory, offering an assumed advantage in efficiency over R-based implementations such as SNPFILTR, especially for larger input files. To objectively evaluate the utility of SNPFILTR, compared to a program like VCFTOOLS, I benchmarked performance under a simple, biologically plausible filtering scenario; setting minimum sequencing depth per called genotype = 5 and minimum genotype quality per called genotype = 30. I compared runtimes across three different approaches; (1) using the SNPfiltR function hard_filter() on a vcf file that has already been read into the local memory as a vcfR object, (2) wrapping the vcfR function read.vcf() inside of a call to hard_filter(), to first read the given vcf file into the local R working environment as a vcfR object, and then perform filtering on the vcfR object, and (3) directly specifying the full path to the given vcf file to VCFtools to perform filtering and output a filtered vcf file. For each of these approaches I recorded the runtime for filtering eight separate vcf files, each containing 100 samples, and 10 K to 500 K SNPs (these input vcf files are publicly available at: https://doi.org/10.5281/zenodo.6382078), across three replicate runs. All benchmarking was performed on a 2.3 GHz Dual-Core Intel Core i5 CPU, running MacOS Big Sur 11.5.1, with 8 GB 2133 MHz LPDDR3 SDRAM (i.e., a personal laptop with reasonable computing power), and exact runtimes were recorded with a precision of 1/1000th of a second using the function microbenchmark() from the R package MICROBENCHMARK (Mersmann et al., 2015) for iterations executed in R, and the bash function 'time' for iterations executed using VCFTOOLS. A fully documented iteration of this performance benchmarking process is available at: devonderaad.github.io/SNPfiltR/articles/performance-benchmarking.html.

To validate the accuracy of the SNPfiltR filtering functions, I performed filtering procedures using SNPfiltR functions hard_filter(), missing_by_snp(), distance_thin(), and min_mac() on three identical input vcf files, followed by identical filtering procedures implemented via VCFTOOLS. I then tested whether the value for each genotype matched between the output files generated using SNPFILTR and VCFTOOLS for each function, using R. This accuracy benchmarking process is fully documented and available at: devonderaad.github.io/SNPfiltR/articles/accuracy-benchmarking.html.

3 | RESULTS

3.1 | Performance and accuracy benchmarking

To compare runtimes between VCFtools and the SNPfiltR function hard_filter(), I performed an identical filtering protocol using identical input files for each program (minimum genotype depth = 5,

minimum genotype quality = 30). The SNPfiltR function hard_filter() consistently performed an identical filtering protocol more rapidly than VCFtools if the input vcf was already stored the local memory of an R working environment (Figure 1). Conversely, if the amount of time taken to read the vcf file into local memory as a vcfR object before filtering is counted against SNPfiltR, the approach takes longer than performing the identical filtering operation using VCFtools. This additional step of reading the vcf file into R as a vcfR object appears to increase the slope, rather than the intercept, of the line (Figure 1), indicating that this step scales poorly as the number of SNPs in the input vcf file increases. Accuracy benchmarking revealed that the functions hard_filter(), missing_by_snp(), and min_mac() perfectly recapitulated the filtering results from VCFtools with 100% genotype concordance, while output files filtered using the SNPfiltR function distance_thin() were generally less than 60% concordant with VCFtools results (Figure 2a). Direct investigation of this discrepancy reveals that SNPfiltR consistently retained more SNPs in the output vcf file than VCFtools (Figure 2b) and that despite the low congruence, both programs accurately filtered each data set according to the specified minimum distance between SNPs (100 bp, Figure 2c).

3.2 | Quality filtering a RADseq SNP data set

To provide an example of SNPfiltR and vcfR functioning as a cohesive R-based SNP filtering pipeline, I used an empirical SNP data set derived from restriction-enzyme associated DNA (RAD) sequencing

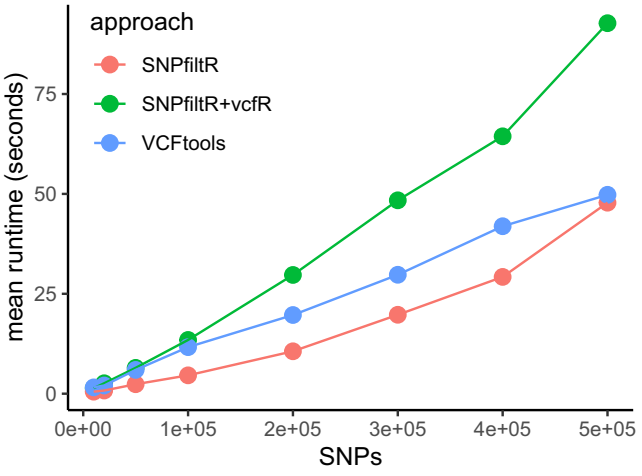


FIGURE 1 Dotplot showing mean runtimes for filtering vcf files according to the following thresholds: minimum depth per genotype = 5, and minimum genotype quality per genotype = 30. Each input vcf file contained between 10 K and 500 K SNPs for 100 individual samples. The three approaches to performing this filtering were as follows; SNPfiltR – using the function hard_filter() on a vcf file that has already been read into the local memory as a vcfR object; SNPfiltR+vcfR - wrapping the vcfR function read.vcf() inside of a call to hard_filter(), in order to read in the given vcf file as a vcfR object, and then perform filtering on the vcfR object; and VCFtools – directly specifying the full path to the given vcf file to VCFtools to filter and output a new, filtered vcf file

from throughout the range of the scrub-jays (genus *Aphelocoma*). I began by reading an unfiltered vcf file into my R working environment using the function read.vcfR() from the VCFRpackage, resulting in a vcfR object containing 210,336 SNPs for 115 samples (Figure 3). I then used the SNPfiltR function hard_filter() to set minimum thresholds of five sequencing reads per called genotype, and a minimum genotype quality of 30, which resulted in the removal of 32.92 and 2.01% of called genotypes, respectively (Figure 3). Filtering to retain only biallelic loci using the function filter_biallelic() confirmed there were no called SNPs with more than two alleles. I then filtered heterozygous genotypes based on allele balance, or the ratio of reads for each allele in called heterozygous genotypes, using the function filter_allele_balance(). I followed the recommendations of DDOCENT(Puritz et al., 2014), removing called heterozygous genotypes for which the allele balance fell outside of the range 0.25–0.75, which removed of 7.56% of heterozygous genotypes (0.39% of all called genotypes) from our SNP data set (Figure 3). Next, I implemented a maximum depth of coverage filter to remove assembled loci containing paralogues, which can result in problematic and misleading downstream inferences (O'Leary et al., 2018). I used the SNPfiltR function max_depth() to visualize the distribution of mean read depth per sample for all called SNPs in this particular data set, and then set a maximum mean depth cutoff of 100 reads, which removed 12.85% of all SNPs from the data set (Figure 3). I then implemented a minor allele count (MAC) filter using the SNPfiltR function min_mac(), because the aforementioned genotype quality filtering steps can generate invariant SNPs if all minor allele genotypes are converted to "NA" for a given SNP. This MAC filter, requiring one minor allele in each SNP, removed 55.87% of SNPs, resulting in 80,885 quality filtered SNPs remaining in the filtered vcfR object (Figure 3).

3.3 | Missing data filtering a RADseq SNP data set

I then used the dedicated visualization tools of SNPfiltR to investigate patterns of missing data per sample and per SNP for this quality filtered SNP data set (Figure 4). I used the function missing_by_sample() to visualize missing data per sample, which revealed an even distribution of missing across a priori identified species groups. Visualization of the proportion of missing genotype calls in each sample shows a relatively continuous distribution from samples missing less than 20% of genotype calls to samples missing nearly 100% of genotype calls (Figure 4). I then used the missing_by_sample() function to filter out samples missing calls at more than 81% of their geonotypes, which resulted in the removal of 20 samples. Because SNPs can become invariant if all minor allele genotypes for a given SNP are removed when samples are dropped from the data set, I implemented a MAC filter to remove invariant sites, again requiring a minimum of one minor allele genotype per SNP, which resulted in 0.61% of remaining SNPs being dropped (Figure 4). I then used the SNPfiltR function missing_by_snp() to visualize the proportion of missing data in each

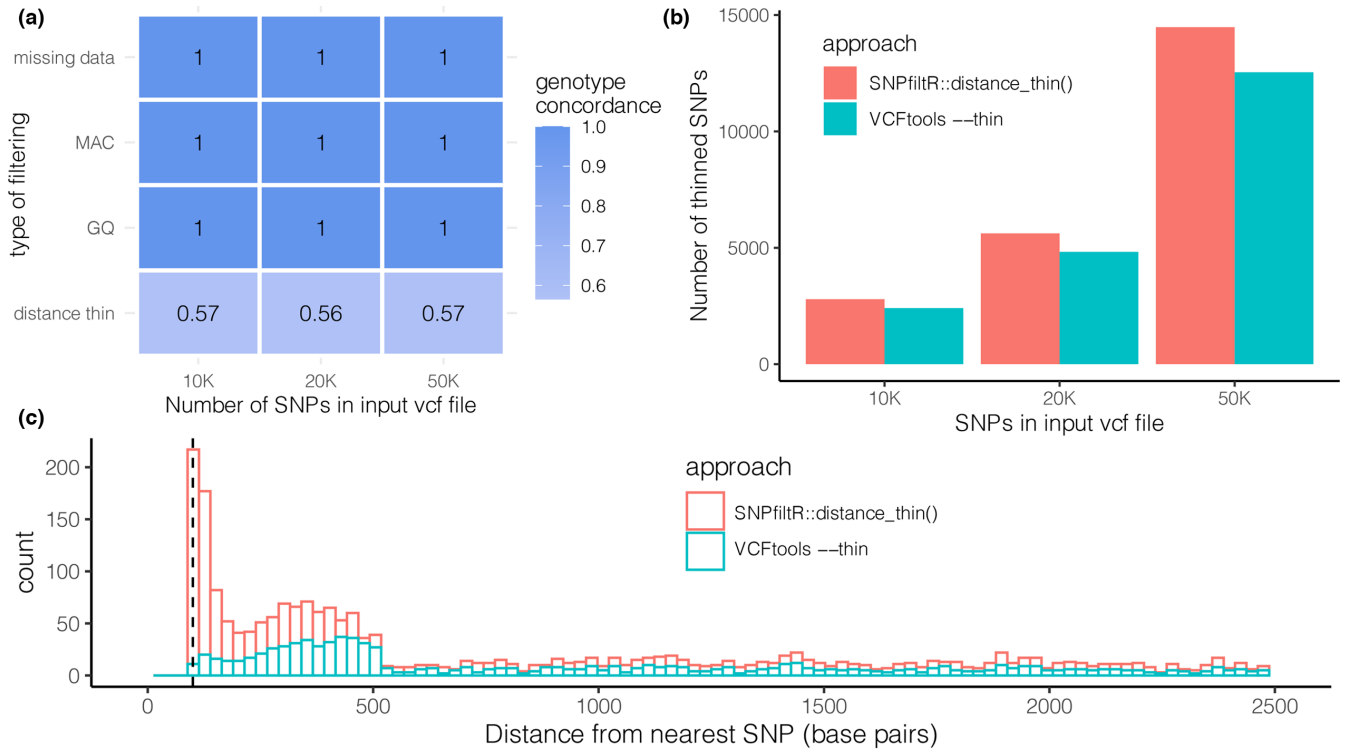


FIGURE 2 Accuracy benchmarking comparing filtering results between identical approaches implemented in SNPfiltR and VCFtools. (a) Heatmap showing concordance (i.e., proportion of identical genotypes) between output vcf files filtered by SNPfiltR and VCFtools. Implemented filters removed SNPs based on missing data, minor allele count (MAC), and physical distance thresholds, and genotypes based on a genotype quality (GQ) threshold. (b) Grouped bar plots show the difference in number of retained SNPs between SNPfiltR and VCFtools when filtering an identical input file to a minimum distance of 100 base pairs (bp) between SNPs. (c) Histogram showing the distance between SNPs in output vcf files that have been filtered to a minimum distance of 100 bp between SNPs using SNPfiltR and VCFtools

sample across a reasonable set of potential per-SNP completeness thresholds (Figure 4). This visualization shows a continuous distribution of missing data within retained samples and no visible outlier samples, suggesting that samples with obviously problematic missing data profiles have been removed. Dotplots show a strong negative correlation between total proportion missing data and the total number of SNPs retained in the data set, across potential per-SNP filtering thresholds.

To assess the potential effects of a range of potential per SNP completeness thresholds (75%, 85%, and 95%), I used the function assess_missing_data_pca() to visualize sample clustering (Figure 5). Across all thresholds, samples cluster visually according to a priori species assignment. Within species groups, samples with the most missing data are clustered the least tightly and increasing the per SNP completeness threshold reduces the clustering uncertainty of the most problematic samples (Figure 5). Sample clustering using t-SNE reveals population substructure within species *woodhouseii* and shows that species level clustering is not affected by missing data proportion in individual samples (Figure 6). Based on sample clustering results, I implemented a per-SNP completeness cutoff of 85% using the function missing_by_snp(), resulting in a final quality and missing data filtered SNP data set containing 95 samples, 16,307 SNPs, and 5.7% total missing genotypes (Figure 6). A final filter for physical linkage, using the function distance_thin(), removed all SNPs separated by less than 500 base pairs, resulting in an unlinked SNP data set of 2,803 SNPs ready for input in downstream analyses.

4 | DISCUSSION

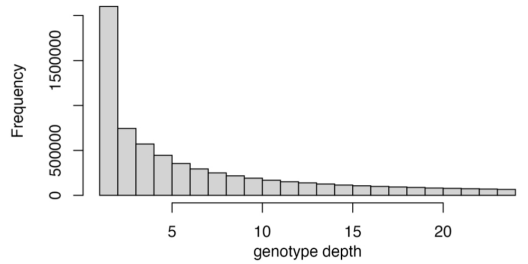
Historically, programs designed for performing computationally intensive bioinformatic processes have rarely been implemented in the R language because the requirement that input data sets be read into local memory generates computational bottlenecks for large input files. Here, I showed that the R package SNPFILTR can be used to filter moderately sized reduced-representation SNP data sets with efficiency and accuracy comparable to precompiled programs implemented directly via command-line interface. Specifically, I demonstrated that the VCFR and SNPFILTR packages can be used in tandem to read and quality filter a SNP data set containing 50 M genotypes (500 K genotypes for 100 samples) and associated quality information in less than 2 min on a personal laptop. These results indicate that the computational power of the R language has been overlooked for the purposes of processing and filtering reduced-representation genomic SNP data sets. SNPFILTR stands apart from currently available programs by combining efficient filtering functions with built in data

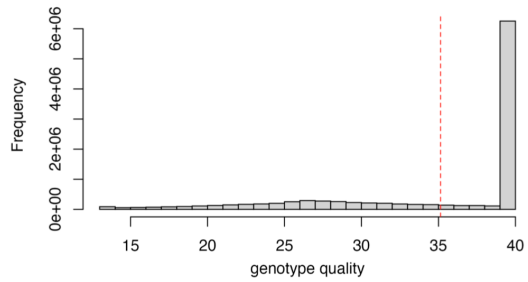
Quality filtering using SNPfiltR

```
library(SNPfiltR)
library(vcfR)
#read in vcf as vcfR
vcfR <- read.vcfR("~/Desktop/aph.data/populations.snps.vcf")
### check the metadata present in your vcf
vcfR
#> ***** Object of Class vcfR *****
#> 115 samples
#> 87 CHROMs
#> 210,336 variants
#> Object size: 685.5 Mb
#> 57.1 percent missing data
#> *****        *****        *****
```

start by visualizing the distributions of depth of sequencing and genotype quality among called genotypes, then set appropriate cutoffs for both values for this dataset.

```
#visualize distributions
hard_filter(vcfR=vcfR)
#> no depth cutoff provided, exploratory visualization will be generated.
```

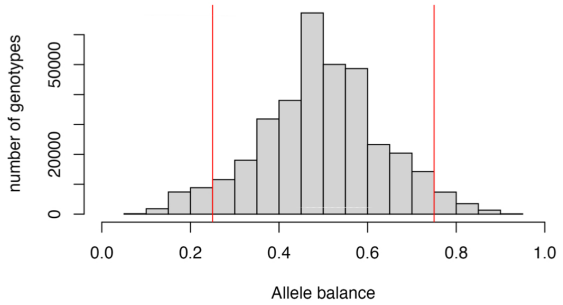




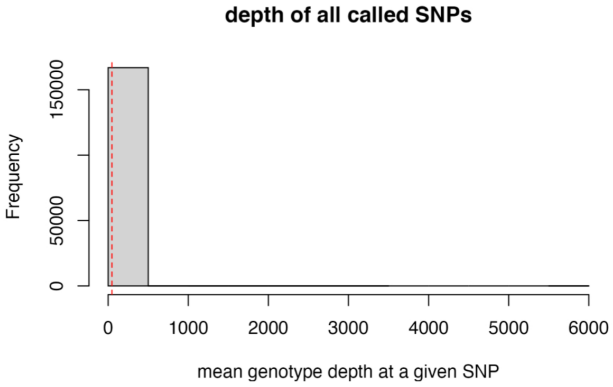

```
#hard filter to minimum depth of 5, and minimum genotype quality of 30
vcfR<-hard_filter(vcfR=vcfR, depth = 5, gq = 30)
#> 32.92% of genotypes fall below a read depth of 5 and were converted to NA
#> 2.01% of genotypes fall below a genotype quality of 30 and were converted to NA

#remove loci with > 2 alleles
vcfR<-filter_biallelic(vcfR)
#> 0 SNPs, 0% of all input SNPs, contained more than 2 alleles, and were removed
```

```
#execute allele balance filter
vcfR<-filter_allele_balance(vcfR)
#> 7.56% of het genotypes (0.39% of all genotypes) fall outside of .25 - .75 allele
    balance and were converted to NA
```

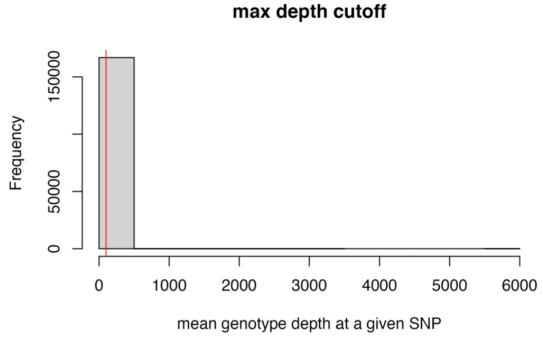



```
#visualize and pick appropriate max depth cutoff
max_depth(vcfR)
#> cutoff is not specified, exploratory visualization will be generated.
```

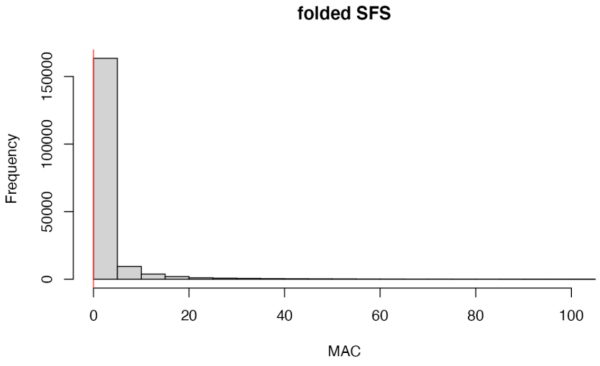



```
#> dashed line indicates a mean depth across all SNPs of 46.7
```

```
#filter vcf by the max depth cutoff you chose
vcfR<-max_depth(vcfR, maxdepth = 100)
#> maxdepth cutoff is specified, filtered vcfR object will be returned
#> 12.85% of SNPs were above a mean depth of 100 and were removed from the vcf
```




```
#remove invariant SNPs generated during the genotype filtering steps
vcfR<-min_mac(vcfR, min.mac = 1)
#> 55.87% of SNPs fell below a minor allele count of 1 and were removed from the VCF
```




```
#check vcfR to see how many SNPs we have left
vcfR
#> ***** Object of Class vcfR *****
#> 115 samples
#> 70 CHROMs
#> 80,885 variants
#> Object size: 283.2 Mb
#> 60.85 percent missing data
#> *****        *****        *****
```

FIGURE 3 An example using SNPfiltR to filter a vcf file based on genotype quality, genotype depth, number of alleles, allele balance, SNP depth, and minor allele count. This quality filtered vcf is now ready for filtering based on missing data thresholds

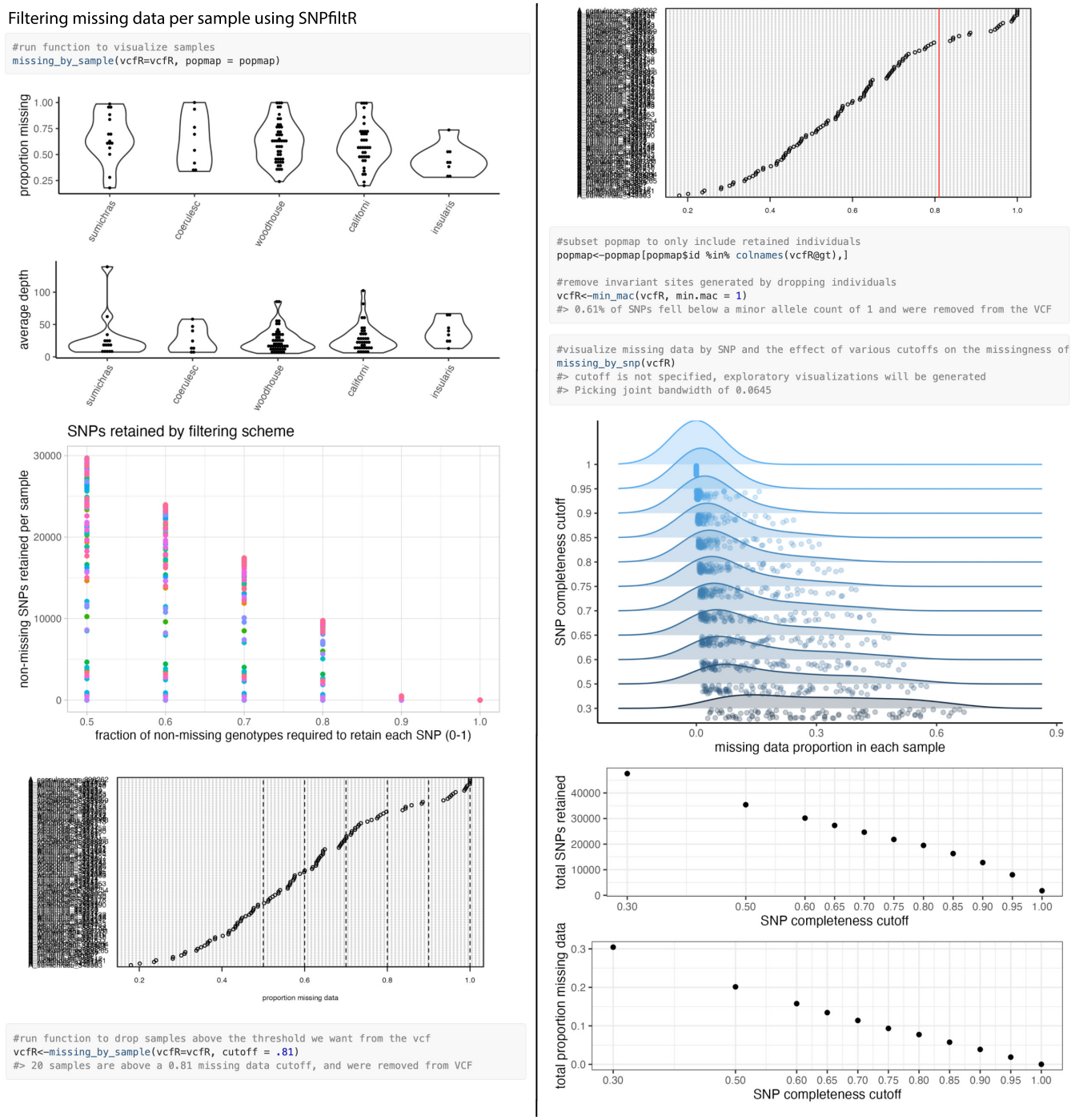


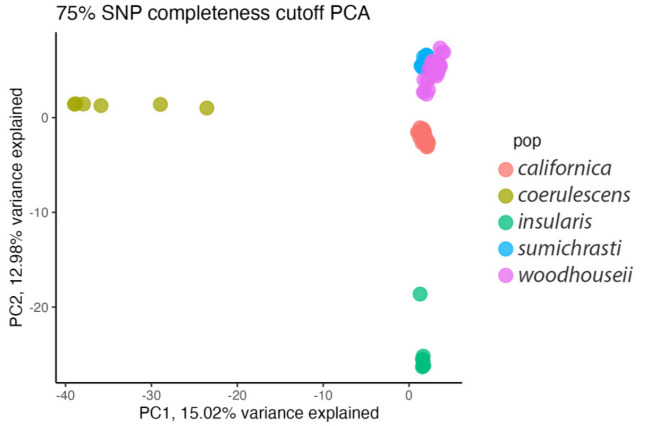
FIGURE 4 Visualization of missing data per sample and implementation of a filter based on a user specified per SNP missing data threshold, using the function missing_by_sample(). Then, visualization of the proportion of missing data across an array of per SNP completeness thresholds using missing_by_snp()

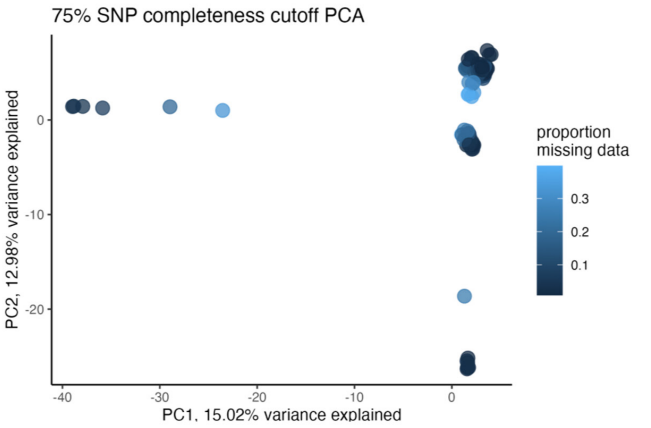
visualization capabilities, allowing users to easily design a customized SNP filtering pipeline within a single R script.

While many existing R packages are capable of working with SNP data, no existing R package contains comparable functions for automated visualization and filtering of SNP data. A few existing packages focus directly on reading and manipulating SNP data (e.g., vcfR [Knaus & Grünwald, 2017] and dartR [Gruber et al., 2018]), but largely require custom scripting to filter and visualize SNP data sets. snpfiltr is complementary to these packages, extending their functionalities with modular functions that automate key visualization and filtering steps. Notably, functions from the snpfiltr package rely on vcfR objects as input, which can be generated by reading vcf files into an R session using the vcfR function read.vcfR(). snpfiltr is also complementary to the growing suite of R packages designed for phylogenetic and population genetic analysis (e.g., ape [Paradis & Schliep, 2019], stampp [Pembleton et al.,

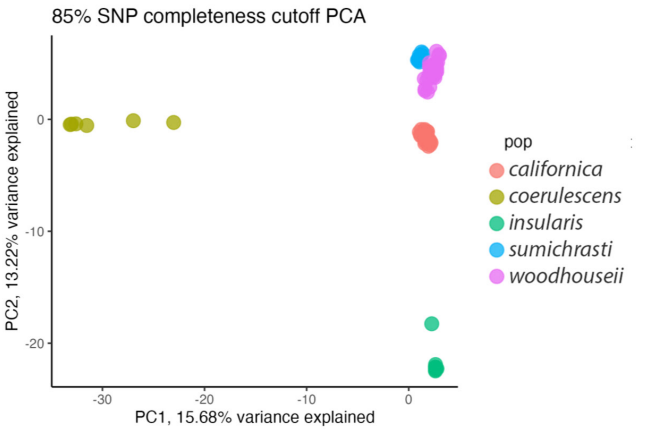
Assessing the effect of missing data using SNPfiltR

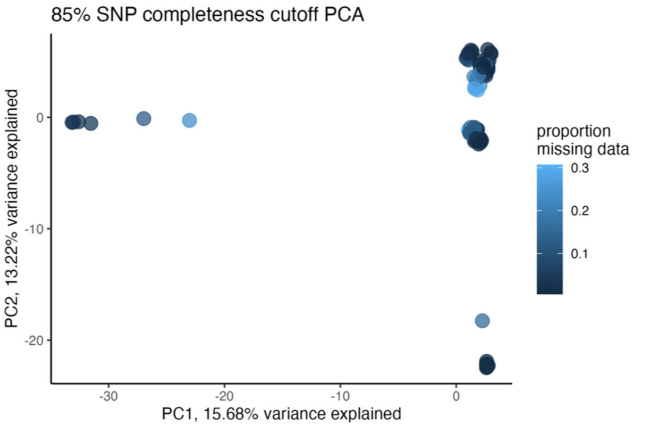
```
#check if these cutoffs prevent samples from clustering
#by patterns of missing data in a PCA
miss<-assess_missing_data_pca(vcfR=vcfR, popmap = popmap,
                              thresholds = c(.75,.85), clustering = FALSE)
#> cutoff is specified, filtered vcfR object will be returned
#> 72.84% of SNPs fell below a completeness cutoff of 0.75 and were removed
```



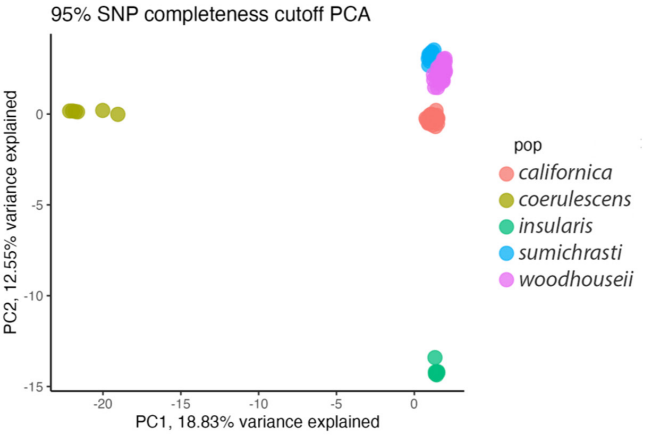



```
#> 79.72% of SNPs fell below a completeness cutoff of 0.85 and were removed
```






```
#> 90.03% of SNPs fell below a completeness cutoff of 0.95 and were removed
```



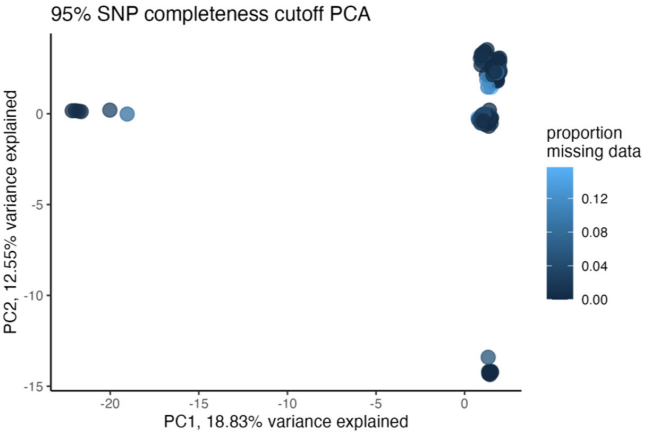


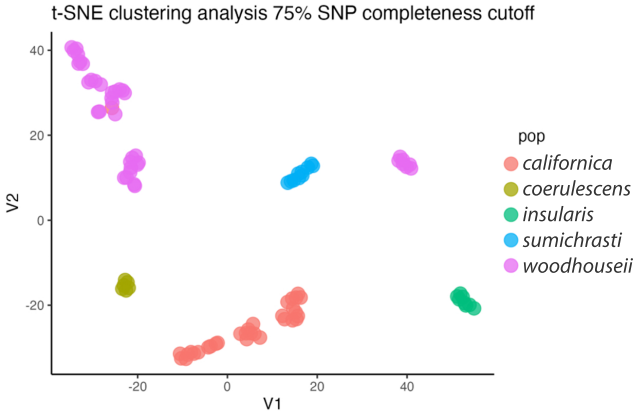
FIGURE 5 Visualizations comparing sample clustering patterns across potential per SNP completeness thresholds using principal component analysis (PCA)

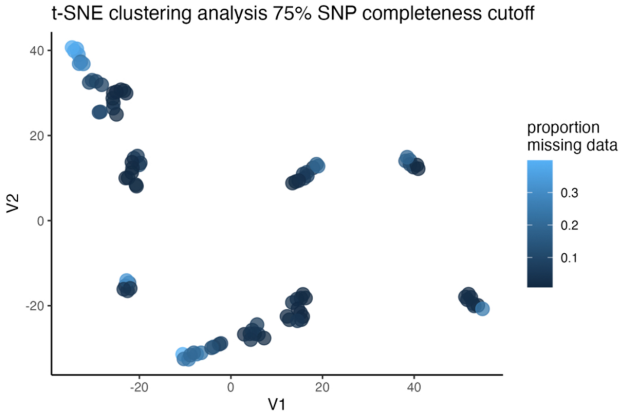
2013], SNPRELATE [Zheng et al., 2012], ADEGENET [Jombart, 2008], SAMBAR [de Jong et al., 2021], and INTROGRESS [Gompert & Buerkle, 2010]), as the VCFR objects read and returned by SNPFILTR can be easily converted into the appropriate input object class for each of these specialized packages.

It is widely accepted that the universe of elegant, open-source R based tools such as RSTUDIO and RMARKDOWN facilitate exceptional interactivity and reproducibility (Gandrud, 2018). SNPFILTR allows users to build SNP filtering pipelines entirely within this R universe, which run with comparable efficiency and accuracy to existing tools

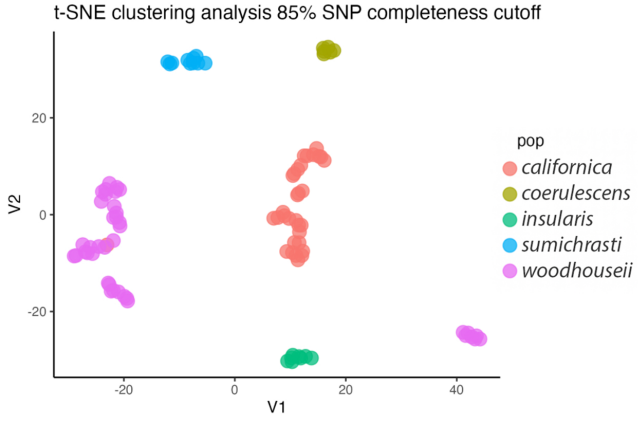
Filtering missing data per SNP using SNPfiltR

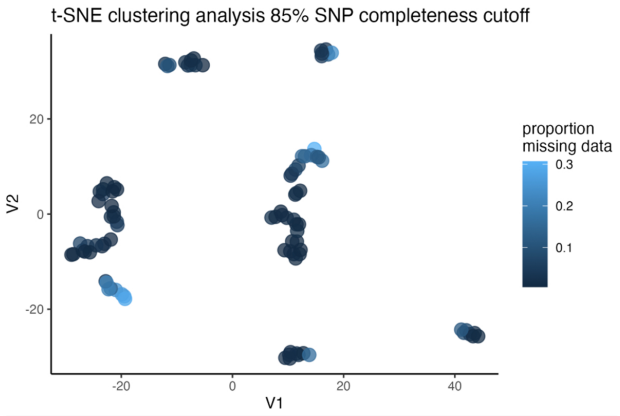
```
#check if these cutoffs prevent samples from clustering
#by patterns of missing data in a PCA
miss<-assess_missing_data_tsne(vcfR=vcfR, popmap = popmap,
                               thresholds = c(.75,.85), clustering = FALSE)
#> cutoff is specified, filtered vcfR object will be returned
#> 72.84% of SNPs fell below a completeness cutoff of 0.75 and were removed
```



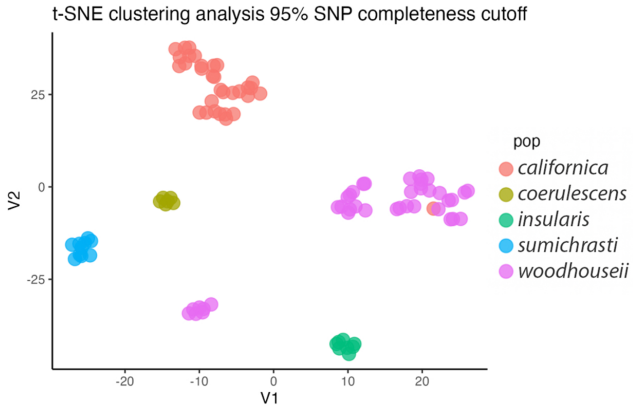


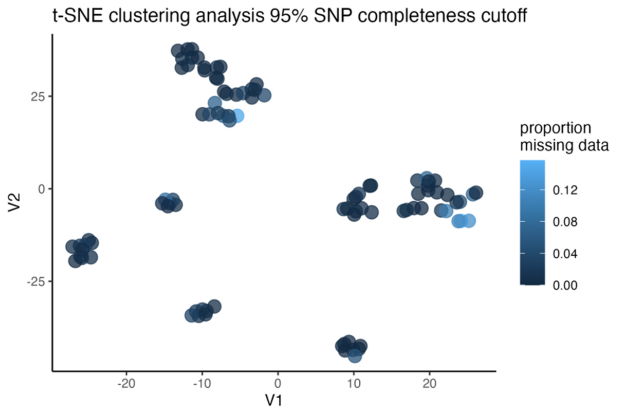

```
#> 79.72% of SNPs fell below a completeness cutoff of 0.85 and were removed
```






```
#> 90.03% of SNPs fell below a completeness cutoff of 0.95 and were removed
```






```
#choose a value that retains an acceptable amount of missing data in each sample
vcfR<-missing_by_snp(vcfR, cutoff = .85)
#check how many SNPs and samples are left
vcfR
#> ***** Object of Class vcfR *****
#> 95 samples
#> 36 CHROMs
#> 16,307 variants
#> Object size: 111.2 Mb
#> 5.743 percent missing data
#> *****        *****         *****
```

Write out the filtered vcf, and a linkage filtered version, for downstream analyses

```
#write out vcf with all SNPs
vcfR::write.vcf(vcfR, "~/Downloads/aphelocoma.filtered.vcf.gz")
#linkage filter vcf to thin SNPs to one per 500bp
vcfR.thin<-distance_thin(vcfR, min.distance = 500)
#> 2803 out of 16307 input SNPs were not located within 500 base-pairs of another SNP
#write out thinned vcf
vcfR::write.vcf(vcfR.thin, "~/Downloads/aphelocoma.filtered.thinned.vcf.gz")
```

FIGURE 6 Visualizations comparing sample clustering patterns across potential per SNP completeness thresholds using t-distributed stochastic neighbour embedding (t-SNE). Following thorough visualization, I used the function missing_by_snp() to implement a per SNP completeness threshold, and the function distance_thin() to generate an additional data set of unlinked SNPs. Finally, I used the vcfR package to write both filtered vcf files to disc, where they are ready to use as input for downstream analyses

(Figures 1 and 2). Although reading large data sets into local memory still represents a challenge, the results presented here indicate that the computational power of the R programming language is sufficient for analysing most reduced-representation SNP data sets. Furthermore, the ability to store intermediate filtered vcf files in the local memory of an R work session, rather than having to write each file to disk, may actually result in overall time savings for users of an R-based SNP filtering pipeline (Figure 1). The pipeline shown here (Figures 3–6) gives a fully worked example of using the SNPFILTR and VCFR packages to implement a fully R-based pipeline for streamlining the complicated and iterative process of optimizing filtering parameters for a next-generation sequencing data set, including generalizable suggestions about best practices. By extending the current bioinformatic tools available in R for filtering SNP data sets, SNPFILTR will allow users to spend less time investigating and testing filtering parameters, and more time resolving evolutionary mysteries with genomic data.

ACKNOWLEDGEMENTS

I would like to acknowledge the authors of the R package VCFR, Brian J. Knaus, and Nikolas J. Grünwald, for developing and documenting resources for reading and manipulating SNP data using R, specifically the exceptional GitHub site: knausb.github.io/vcfR_documentation/index.html which provided invaluable guidance in the development of this package. I would also like to acknowledge Jonathan Puritz whose excellent SNP filtering tutorial (ddocent.com/filtering/) was essential for developing an understanding of best practices for filtering reduced-representation genomic SNP data sets.

CONFLICT OF INTEREST

The author declares no conflicts of interest.

AUTHOR CONTRIBUTION

This software was fully conceived, written, and documented by the sole author, Devon A. DeRaad.

OPEN RESEARCH BADGES

This article has earned an Open Data Badge for making publicly available the digitally-shareable data necessary to reproduce the reported results. The data is available at https://datadryad.org/stash/dataset/doi:10.5061/dryad.qh8sh.

DATA AVAILABILITY STATEMENT

The example SNP data set distributed with SNPFILTR can be directly accessed by downloading the SNPFILTR package from CRAN and loading the data set into the working environment, using the following code install.packages("SNPFILTR"); data(vcfR.example), in an R session. The UCE SNP data set used for the SNPFILTR UCE SNP filtering vignette is publicly available for download at: (datadryad.org/stash/dataset/doi:10.5061/dryad.qh8sh). The RADseq SNP data set used to generate the example SNP filtering pipeline shown in this study is available here: (https://doi.org/10.5281/zenodo.6382078), along with the input vcf files used for accuracy and performance benchmarking. The SNPFILTR package is stably documented on GitHub at: github.com/DevonDeRaad/SNPfiltR and at the dedicated site: devonderaad.github.io/SNPfiltR/ DeRaad (2022).

ORCID

Devon A. DeRaad https://orcid.org/0000-0003-3105-985X

REFERENCES

Cerca, J., Maurstad, M. F., Rochette, N. C., Rivera-Colón, A. G., Rayamajhi, N., Catchen, J. M., & Struck, T. H. (2021). Removing the bad apples: A simple bioinformatic method to improve loci-recovery in de novo RADseq data for non-model organisms. *Methods in Ecology and Evolution*, *12*(5), 805–817. https://doi.org/10.1111/2041-210X.13562

Danecek, P., Auton, A., Abecasis, G., Albers, C. A., Banks, E., DePristo, M. A., Handsaker, R. E., Lunter, G., Marth, G. T., Sherry, S. T., McVean, G., & Durbin, R. (2011). The variant call format and VCFtools. *Bioinformatics*, *27*(15), 2156–2158. https://doi.org/10.1093/bioinformatics/btr330

Danecek, P., Bonfield, J. K., Liddle, J., Marshall, J., Ohan, V., Pollard, M. O., Whitwham, A., Keane, T., McCarthy, S. A., Davies, R. M., & Li, H. (2021). Twelve years of SAMtools and BCFtools. *GigaScience*, *10*(2), giab008. https://doi.org/10.1093/gigascience/giab008

Davey, J. W., & Blaxter, M. L. (2010). RADSeq: Next-generation population genetics. *Briefings in Functional Genomics*, *9*(5–6), 416–423. https://doi.org/10.1093/bfgp/elq031

deJong, M. J., deJong, J. F., Hoelzel, A. R., & Janke, A. (2021). SambaR: An R package for fast, easy and reproducible population-genetic analyses of biallelic SNP data sets. *Molecular Ecology Resources*, *21*(4), 1369–1379. https://doi.org/10.1111/1755-0998.13339

DeRaad, D. A. (2022). SNPfiltR: An R package for interactive and reproducible SNP filtering. *GitHub Repository*. github.com/DevonDeRaad/SNPfiltR. https://doi.org/10.5281/zenodo.6284749

Eaton, D. A. R., & Overcast, I. (2020). ipyrad: Interactive assembly and analysis of RADseq datasets. *Bioinformatics*, *36*(8), 2592–2594. https://doi.org/10.1093/bioinformatics/btz966

Faircloth, B. C. (2016). PHYLUCE is a software package for the analysis of conserved genomic loci. *Bioinformatics*, *32*(5), 786–788. https://doi.org/10.1093/bioinformatics/btv646

Faircloth, B. C., McCormack, J. E., Crawford, N. G., Harvey, M. G., Brumfield, R. T., & Glenn, T. C. (2012). Ultraconserved elements anchor thousands of genetic markers spanning multiple evolutionary timescales. *Systematic Biology*, *61*(5), 717–726. https://doi.org/10.1093/sysbio/sys004

Gandrud, C. (2018). *Reproducible research with R and RStudio*. Chapman and Hall/CRC.

Gompert, Z., & Buerkle, C. A. (2010). introgress: A software package for mapping components of isolation in hybrids. *Molecular Ecology Resources*, *10*(2), 378–384. https://doi.org/10.1111/j.1755-0998.2009.02733.x

Gruber, B., Unmack, P. J., Berry, O. F., & Georges, A. (2018). dartr: An r package to facilitate analysis of SNP data generated from reduced representation genome sequencing. *Molecular Ecology Resources*, *18*(3), 691–699. https://doi.org/10.1111/1755-0998.12745

Jombart, T. (2008). adegenet: A R package for the multivariate analysis of genetic markers. *Bioinformatics*, *24*(11), 1403–1405. https://doi.org/10.1093/bioinformatics/btn129

Knaus, B. J., & Grünwald, N. J. (2017). vcfr: A package to manipulate and visualize variant call format data in R. *Molecular Ecology Resources*, *17*(1), 44–53. https://doi.org/10.1111/1755-0998.12549

Krijthe, J. H., & van derMaaten, L. (2015). *Rtsne: T-distributed stochastic neighbor embedding using Barnes-Hut implementation*. R

package version 0.13, https://github.com/jkrijthe/Rtsne [Computer software]

Maechler, M., Rousseeuw, P., Struyf, A., Hubert, M., & Hornik, K. (2018). *cluster: Cluster analysis basics and extensions*. R package version 2.0.7-1 [Computer software].

McCormack, J. E., Tsai, W. L. E., & Faircloth, B. C. (2016). Sequence capture of ultraconserved elements from bird museum specimens. *Molecular Ecology Resources*, *16*(5), 1189–1203. https://doi.org/10.1111/1755-0998.12466

McKenna, A., Hanna, M., Banks, E., Sivachenko, A., Cibulskis, K., Kernytsky, A., Garimella, K., Altshuler, D., Gabriel, S., Daly, M., & DePristo, M. A. (2010). The genome analysis toolkit: A mapreduce framework for analyzing next-generation DNA sequencing data. *Genome Research*, *20*(9), 1297–1303. https://doi.org/10.1101/gr.107524.110

Mersmann, O., Beleites, C., Hurling, R., & Friedman, A. (2015). *Microbenchmark: Accurate timing functions*. R package version 1.4.2 [Computer software].

O'Leary, S. J., Puritz, J. B., Willis, S. C., Hollenbeck, C. M., & Portnoy, D. S. (2018). These aren't the loci you'e looking for: Principles of effective SNP filtering for molecular ecologists. *Molecular Ecology*, *27*(16), 3193–3206. https://doi.org/10.1111/mec.14792

Papin, J. A., Gabhann, F. M., Sauro, H. M., Nickerson, D., & Rampadarath, A. (2020). Improving reproducibility in computational biology research. *PLoS Computational Biology*, *16*(5), e1007881. https://doi.org/10.1371/journal.pcbi.1007881

Paradis, E., & Schliep, K. (2019). ape 5.0: An environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, *35*(3), 526–528. https://doi.org/10.1093/bioinformatics/bty633

Pembleton, L. W., Cogan, N. O. I., & Forster, J. W. (2013). StAMPP: An R package for calculation of genetic differentiation and structure of mixed-ploidy level populations. *Molecular Ecology Resources*, *13*(5), 946–952. https://doi.org/10.1111/1755-0998.12129

Puritz, J. B., Hollenbeck, C. M., & Gold, J. R. (2014). dDocent: A RADseq, variant-calling pipeline designed for population genomics of non-model organisms. *PeerJ*, *2*, e431. https://doi.org/10.7717/peerj.431

R Core Team (2019). *R: A language and environment for statistical computing*. R Foundation for Statistical Computing.

Rochette, N. C., Rivera-Colón, A. G., & Catchen, J. M. (2019). Stacks 2: Analytical methods for paired-end sequencing improve RADseq-based population genomics. *Molecular Ecology*, *28*(21), 4737–4754. https://doi.org/10.1111/mec.15253

Toews, D. P. L., Campagna, L., Taylor, S. A., Balakrishnan, C. N., Baldassarre, D. T., Deane-Coe, P. E., Harvey, M. G., Hooper, D. M., Irwin, D. E., Judy, C. D., Mason, N. A., McCormack, J. E., McCracken, K. G., Oliveros, C. H., Safran, R. J., Scordato, E. S. C., Stryjewski, K. F., Tigano, A., Uy, J. A. C., & Winger, B. M. (2015). Genomic approaches to understanding population divergence and speciation in birds. *The Auk*, *133*(1), 13–30. https://doi.org/10.1642/AUK-15-51.1

Zheng, X., Levine, D., Shen, J., Gogarten, S. M., Laurie, C., & Weir, B. S. (2012). A high-performance computing toolset for relatedness and principal component analysis of SNP data. *Bioinformatics*, *28*(24), 3326–3328. https://doi.org/10.1093/bioinformatics/bts606

How to cite this article: DeRaad, D. A. (2022). SNPFILTR: An R package for interactive and reproducible SNP filtering. *Molecular Ecology Resources*, *00*, 1–11. https://doi.org/10.1111/1755-0998.13618