SNPfiltR: an R package for interactive and reproducible SNP filtering

Devon DeRaad[1]

[1]University of Kansas

December 17, 2021

Abstract

Here I describe the novel R package SNPfiltR and demonstrate its functionalities as the backbone of a customizable, reproducible 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 depth, quality, and missing data, then allowing users to set filters based on optimized thresholds, all within a single, cohesive working environment. All SNPfiltR functions require a vcfR object as input, which can be easily generated by reading a SNP dataset stored as a standard vcf file into an R working environment using the function read.vcfR() from the R package vcfR. Performance benchmarking reveals that for moderately sized SNP datasets (up to 50M genotypes with associated quality information), SNPfiltR performs filtering with comparable efficiency to current state of the art command-line-based programs. These benchmarking results indicate that for most reduced-representation genomic datasets, SNPfiltR is an ideal choice for investigating, visualizing, and filtering SNPs as part of a cohesive and easily documentable bioinformatic pipeline. The SNPfiltR package can be downloaded from CRAN with the command [install.packages("SNPfiltR")], and a development version is available from GitHub at: (github.com/DevonDeRaad/SNPfiltR). Additionally, thorough documentation for SNPfiltR, including multiple comprehensive vignettes, is available at the website: (devonderaad.github.io/SNPfiltR/).

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) datasets have quickly become the standard input data for a wide array of phylogenetic and population genetic analyses (Toews et al., 2015). SNP datasets 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 standardized and efficient formatting of the vcf (variant call format) file has become the widely accepted standard (Danecek et al., 2011). By retaining only variant sites (i.e., called SNPs), large population genomic datasets can be stored in vcf files which are manageable both in terms of file size and computational time required for downstream analyses.

As SNP datasets stored and 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, all of which need to be addressed before performing downstream analyses (O'Leary et al., 2018). Furthermore, individual samples may suffer from low sequencing coverage or contamination, preventing confident use in downstream analysis, necessitating their removal from the dataset (Cerca et al., 2021). To address these issues, some SNP calling programs contain built in functionality for filtering output SNP datasets (e.g.,*GATK* and *Stacks*), but these filtering options often leave much to be desired for investigators hoping to perform thorough explorations of parameter space and deeply understand the nuances of their particular SNP dataset. This limited functionality results in a gap in many bioinformatic pipelines, especially for SNP datasets generated via reduced-representation genomic sequencing where de novo assembly and rampant missing data often 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 investigators attempt to confirm that technical issues and missing data are not influencing downstream inferences, before choosing a final set of SNP filtering parameters. As reproducibility is becoming more widely acknowledged as critical for the future of science, effectively documenting this often iterative process of investigating, visualizing, and cleaning large datasets continues to be a major challenge. Current state of the art programs for filtering SNP datasets such as *GATK* and *VCFtools* (Danecek et al., 2011) are highly efficient and parallelizable, making them especially useful for massive datasets, but their command-line interfaces do not lend themselves easily to graphical visualization, leading many investigators to rely on homebrewed scripts for preliminary data investigation. A common homebrewed approach involves generating summary statistic files using a command-line based program or Unix scripting, for investigation and visualization 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. Based on the ubiquity of this approach, there is clearly an outstanding need within the field of evolutionary genomics for user-friendly software that automates and streamlines the process of investigating, visualizing, and filtering SNP datasets.

The R package *SNPfiltR* is designed to fill this bioinformatic gap with a suite of custom functions designed for investigating, visualizing, and filtering reduced-representation SNP datasets within a coherent R-based framework. As input, these functions take SNP datasets stored as standard vcf files, read into an R working environment as 'vcfR' objects, which can be performed in a single step using the function read.vcfR() from the R package *vcfR* (Knaus & Grünwald, 2017). This suite of custom functions from SNPfiltR can then generate automated visualizations of key parameters such as depth, quality, and missing data, and allow users to specify appropriate filtering thresholds based on the nuances of their particular dataset, for rapid filtering directly in R. Functions from *SNPfiltR* can be used in concert with import and export functions from *vcfR* , in order to generate an interactive, customizable SNP filtering pipeline, all within a single R script. The package is publicly available on CRAN via the syntax [install.packages("SNPfiltR")], the development version is available from GitHub at: (github.com/DevonDeRaad/SNPfiltR) and the entire package is thoroughly documented including descriptions and examples for each function and multiple comprehensive vignettes, at the website: (devonderaad.github.io/SNPfiltR/).

Materials and methods

Example datasets

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

I used additional example datasets to provide fully worked vignettes integrating functions from *SNPfiltR* and *vcfR* into fully R-based, customizable SNP filtering pipelines for genomic datasets 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 (UCE's) (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 *Stacks* v.2.41 (Rochette et al., 2019). This empirical dataset from throughout the entire distribution of Scrub-Jays (genus *Aphelocoma*) across North America, will be publicly released via Dryad, upon publication. 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 dataset was the focus of McCormack et al. (McCormack et al., 2016), and is publicly available for download via the Dryad repository associated with this paper at: (datadryad.org/stash/dataset/doi:10.5061/dryad.qh8sh).

Novel functions for visualizing and filtering SNP datasets in R

The *SNPfiltR* package relies on the efficient import and export functions of the *vcfR* package to efficiently 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. Once a vcf file has been read into the local R working environment as a vcfR object, it is immediately available in proper input format for all *SNPfiltR*functions. Each *SNPfiltR* function can be run without specified thresholds or cutoffs, (e.g., *hard_filter(vcfR=vcfR.object)*) to visualize the parameter space that will be filtered, without performing filtering, allowing users to quickly make informed decisions based on patterns specific to their datasets, and implement their chosen filtering thresholds (e.g., *hard_-filter(vcfR=vcfR.object, depth=5, gq=30)*). *SNPfiltR* contains a suite of commonly implemented filters for genomic datasets, 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 physical linkage. While most of these filters can be implemented in other programs (e.g., *VCFtools* and *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 or skipped at the discretion of the user, to build an interactive SNP filtering pipeline customized to the specific needs of a given genomic dataset.

Beyond simply filtering, I also developed functions to automate the process of investigating the effects of missing data on a SNP dataset. The *SNPfiltR* functions *assess_missing_data_pca()* and*assess_missing_data_tsne()* are designed to perform dimensionality reduction on highly multi-dimensional SNP datasets, using principal components analysis (PCA) via the R package adegenet (Jombart, 2008) and t-distributed stochastic neighbor embedding implemented via the R package *Rtsne* (Krijthe & van der Maaten, 2015), respectively. Each of these functions then visualizes the similarity between input samples in two-dimensional space, across user specified missing data per SNP 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), by setting clustering = TRUE, if they wish to assess the effect of missing data on objective sample clustering assignments. Finally each of these functions will generate an additional visualization of sample similarity in two-dimensional space with samples color-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()* , in order to ensure that user specified missing data thresholds both per sample and per SNP are sufficient for mitigating the effects of missing data in driving patterns of sample clustering for your specific dataset before performing downstream population genetic or phylogenetic analyses.

Performance benchmarking

To evaluate the performance of *SNPfiltR* , I compared filtering runtimes with the widely used program *VCFtools* (Maechler et al., 2018). *VCFtools* is a highly efficient command-line based program written in Perl and C++, which is frequently used for filtering vcf files according to various quality metrics. 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 a minimum depth per called genotype = 5 and a minimum genotype quality per genotype = 30. I then compared runtimes across three different approaches; 1) using the R function SNPfiltR::hard_filter() on a vcf file that has already been read into the local memory as a vcfR object, 2) wrapping the R function vcfR::read.vcf() inside of a call to SNPfiltR::hard_filter(), to first read the given vcf file into the local R working environment as a vcfR object, and then to perform filtering on the vcfR object, and 3) directly specifying the full path to the given vcf file to *VCFtools* to filter the dataset and output a new, filtered vcf file. For each of these approaches, I recorded the runtime for filtering each of eight vcf files, subset from a real empirical vcf file, each containing 100 samples, and varying from 10K to 500K SNPs. 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 typical computing power), and exact runtimes were recorded with a precision of $1/1000^{\text{th}}$ 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 example of this benchmarking process is available at: (devonderaad.github.io/SNPfiltR/articles/performance-benchmarking.html#benchmark-10k-1).

Results

Performance benchmarking

I compared runtimes between using *VCFtools* to filter a series of vcf files according to simple genotype quality thresholds (minimum depth = 5, minimum genotype quality = 30), and using the *SNPfiltR*function hard_filter() to perform the same filtering protocol on the same input files. Each vcf file contained between 10K and 500K SNPs for 100 samples, and we benchmarked *SNPfiltR* separately under a scenario where the vcf file had already been read into the local memory of the R working environment, and a scenario where the vcf file was required to be read from disk before filtering. Across three replicates of each iteration, we found that when the vcf file had already been stored as a vcfR object in the R working environment, the*SNPfiltR* function hard_filter() performed filtering and returned a filtered object, on average, more rapidly than *VCFtools* used to perform the identical filtering (Fig. 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* , then this approach takes consistently 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 (Fig. 1), indicating that this step scales poorly as the number of SNPs in the input vcf file increases, compared to the filtering process itself whether executed using *SNPfiltR* or *VCFtools* .

Quality filtering for a scrub-jay RADseq SNP dataset

To provide an example of using SNPfiltR and vcfR to build a comprehensive, R-based SNP filtering pipeline, I began by reading an unfiltered vcf file into my R working environment using the function*read.vcfR()* from the *vcfR* package, resulting in a vcfR object containing 210,336 SNPs for 115 samples,derived from RAD sequencing of samples from throughout the range of the scrub-jays. Using the *SNPfiltR* function *hard_filter()* to filter genotypes to minimum thresholds of a depth of 5 reads, and a genotype quality of 30 resulted in the removal of 32.92% and 2.01% of called genotypes, respectively (Fig. 2). Filtering to retain only bi-allelic loci using the function *filter_biallelic()* , removed no SNPs, as there were no SNPs with more than two alleles. An additional quality filter offered by *SNPfiltR* is the function *filter_allele_balance()* , which implements a quality filter based on allele balance, or the ratio of reads for each allele in called heterozygous genotypes. This function follows the recommendations of *Ddocent* (Puritz et al., 2014), by removing called heterozygous genotypes where the allele balance falls outside of the range .25-.75, and resulted in the removal of 7.56% of heterozygous genotypes, or .39% of all called genotypes, in our scrub-jay SNP dataset. Next, I implemented a maximum depth filter, which is crucial for RAD datasets, where paralogously assembled loci (which will display outlier sequencing depth) can result in problematic and misleading downstream inferences (O'Leary et al., 2018). Because depth of coverage can vary widely based on project design, understanding the distribution of read depths across all called SNPs is essential for making an informed decision about a maximum depth cutoff. I used the*SNPfiltR* function *max_depth()* to visualize the distribution of mean read depth per sample for all called SNPs, and then set a maximum mean depth cutoff of 100 reads, resulting in 12.85% of all SNPs being removed from the dataset. As a final quality filtering step, I implemented a minor allele count filter using the*SNPfiltR* function *min_mac()* in order to remove invariant SNPs resulting from genotype-based quality filtering steps, which may have resulted in all minor allele genotype calls being converted to 'NA' for individual SNPs. This filter requiring one minor allele in each SNP removed of 55.87% of SNPs, resulting in 80,885 quality filtered SNPs remaining in the filtered vcfR object for further investigation (Fig. 2).

Missing data filtering for a scrub-jay RADseq SNP dataset

I then utilized the dedicated visualization tools offered by*SNPfiltR* to investigate patterns of missing data by individual sample and by SNP for this quality filtered scrub-jay SNP dataset (Fig. 3). The function *missing_-by_sample()* reveals that missing data is distributed relatively equally across a priori identified species groups, and that with all 115 samples included, there are hardly any SNPs that reach a 90% completeness threshold. A visualization of the proportion of missing genotype calls in each sample shows that samples vary along a relatively continuous distribution from missing less than 20% of genotype calls to missing nearly 100% of genotype calls. Using the *missing_by_sample()* function, I filtered with a proportion missing genotypes per sample threshold of 81%, resulting in 20 samples being dropped from the dataset (Fig. 3). Because SNPs may have become invariant if all minor allele genotypes were removed when these samples were dropped, I again implemented a minor allele count filter, with a minimum of one minor allele genotype per SNP, to remove invariant sites, resulting in .61% of remaining SNPs being dropped.

I then used the *SNPfiltR* function *missing_by_snp()* to visualize the proportion of missing data in each sample across a reasonable set of potential per-SNP completeness thresholds (Fig. 3). This visualization shows a continuous distribution of missing data within retained samples and no visible outlier samples, indicating that we have successfully dropped problematic samples from the dataset. Dotplots show a strong negative correlation between total proportion missing data and the total number of SNPs retained in the dataset, across potential per-SNP filtering thresholds. I chose to implement a per-SNP completeness cutoff of 85% using the function*missing_by_snp()* , resulting in a final, quality and missing data filtered SNP dataset containing 95 samples, 16,307 SNPs, and 5.7% total missing genotypes (Fig. 3).

To ensure that the implemented 85% missing data threshold effectively prevents patterns of missing data within individuals from driving overall clustering patterns, I then used the function*assess_missing_data_-pca()* to visualize sample clustering across 75% and 85% completeness per SNP completeness thresholds (Fig. 4). At both thresholds, all samples visually cluster according to a priori assignment to species groups. When samples are colored according to proportion missing data, it becomes evident that within species groups, samples with the most missing data are clustered the least tightly, indicating increased uncertainty in assignment. Between the 75% and 85% per SNP completeness thresholds, the more restrictive threshold slightly reduces the effect of missing data in these most loosely assigned samples (Fig. 4). Sample clustering using t-SNE reveals additional population substructure within species groups and shows no indication that missing data is driving patterns of clustering either between or within groups (Fig. 4). A final filter for physical linkage, using the *SNPfiltR* function *distance_thin()* to remove all SNPs separated by less than 500 base-pairs, resulted in a quality and missing data filtered, unlinked SNP dataset of 2,803 SNPs ready for input in downstream analyses.

Discussion

Historically, programs designed for performing computationally intensive bioinformatic processes have rarely been implemented in the R language because the requirement that datasets be read into local memory can cause computational bottlenecks with large input file sizes. Here I showed that the R package *SNPfiltR* can be used to filter moderate sized reduced-representation SNP datasets with runtimes comparable to state-of-the-art programs implemented in highly efficient languages such as Perl and C++. While benchmarking confirmed that reading large files into the local memory of an R working environment scales poorly with increasing input file size, the *vcfR* and *SNPfiltR* packages can be used in tandem to read and quality filter a SNP dataset containing 50M genotypes and associated quality information in less than two minutes on a personal laptop. This size SNP dataset (50M genotypes, or 500K genotypes for 100 samples) is realistic for a set of unfiltered SNP calls resulting from a moderate to large sized reduced-representation genomic sequencing project, indicating that the computational power of the R language has been generally overlooked for the purposes of processing and filtering reduced-representation genomic SNP datasets. *SNPfiltR* takes advantage of this previously overlooked computational power, and unlike existing programs designed for SNP filtering, harnesses the widely commended data visualization capabilities of R, allowing users to design an interactive and customizable SNP filtering pipelines within a single R script.

While many existing R packages are capable of working with SNP data, no existing R package contains functions for automated visualization and filtering of SNP data comparable to those offered by *SNPfiltR* . A few packages focus on directly reading and manipulating SNP data (e.g.,*vcfR* (Knaus & Grünwald, 2017) and *dartR* (Gruber et al., 2018)), but largely require custom scripting using R syntax if users wish to filter and visualize their SNP datasets, leaving a need for automated SNP visualization and filtering functions. *SNPfiltR* is complementary to these packages, extending their functionalities with modular functions that automate key visualization and filtering steps, allowing the rapid generation of full SNP filtering pipelines in R. Notably, functions from the *SNPfiltR* package rely on vcfR objects as input, which can be directly read in from vcf files using the function *read.vcfR()* from the *vcfR* package. For this reason, we strongly recommend that users of the *SNPfiltR* package also cite the *vcfR* package as part of their integrative SNP filtering pipelines. A suite of additional R packages exist for performing downstream phylogenetic and population genetic analyses on high-quality SNP datasets (e.g., *APE* (Paradis & Schliep, 2019),*stAMPP* (Pembleton et al., 2013), *SNPrelate* (Zheng et al., 2012), *adegenet* (Jombart, 2008), *sambaR* (de Jong et al., 2021), and *introgress* (Gompert & Buerkle, 2010)).*SNPfiltR* is complementary to these packages as well, as each*SNPfiltR* function returns a filtered vcfR object which can be easily converted into a myriad of object classes within R for further analysis using any of these dedicated population genetic programs.

It is widely accepted that the universe of elegant, open-source R based tools such as Rstudio and Rmarkdown allow for exceptional interactivity and reproducibility (Gandrud, 2018). Additionally, the performance benchmarking results presented here indicate that the computational power of the R programming language is sufficient for analyzing most reduced-representation SNP datasets, despite that this practice seems relatively rare. The *SNPfiltR* package takes advantage of this previously unrecognized opportunity and provides custom functions designed to fully integrate the investigation, visualization, and filtering of a SNP dataset into a single coherent R framework. The filtering functions offered by *SNPfiltR* perform competitively with current state of the art SNP filtering programs on moderately sized datasets, indicating that bioinformaticians ought to consider implementing fully R-based pipelines for streamlining the often complicated and iterative process of optimizing filtering parameters for next-generation sequencing datasets. By extending the current bioinformatic tools available in R for filtering SNP datasets, the*SNPfiltR* package 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.

Data availability

The example SNP dataset distributed with *SNPfiltR* can be directly accessed by downloading the *SNPfiltR* package from CRAN and loading the dataset into the working environment, using the following code *install.packages("SNPfiltR"); data(vcfR.example)* , in an R session. The UCE SNP dataset 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 dataset used to generate the example SNP filtering pipeline shown in this paper will be released in a Dryad repository upon manuscript acceptance. The SNPfiltR package is stably documented on github at: github.com/DevonDeRaad/RADstackshelpR and at the dedicated site: devonderaad.github.io/SNPfiltR/

Author contributions

This software was fully conceived, written, and documented by the sole author, DAD.

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). 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

de Jong, M. J., de Jong, 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

[dataset]DeRaad, D. A. (2021). SNPfiltR: an R package for interactive and reproducible SNP filtering; GitHub repository. github.com/DevonDeRaad/SNPfiltR; Dryad DOI: to come.

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 der Maaten, L. (2015). *Rtsne: T-distributed stochastic neighbor embedding using Barnes-Hut implementation* (R package version 0.13, URL 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).

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).

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

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. Vienna, Austria: R Foundation for Statistical Computing.*

Rochette, N. C., Rivera-Colon, 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

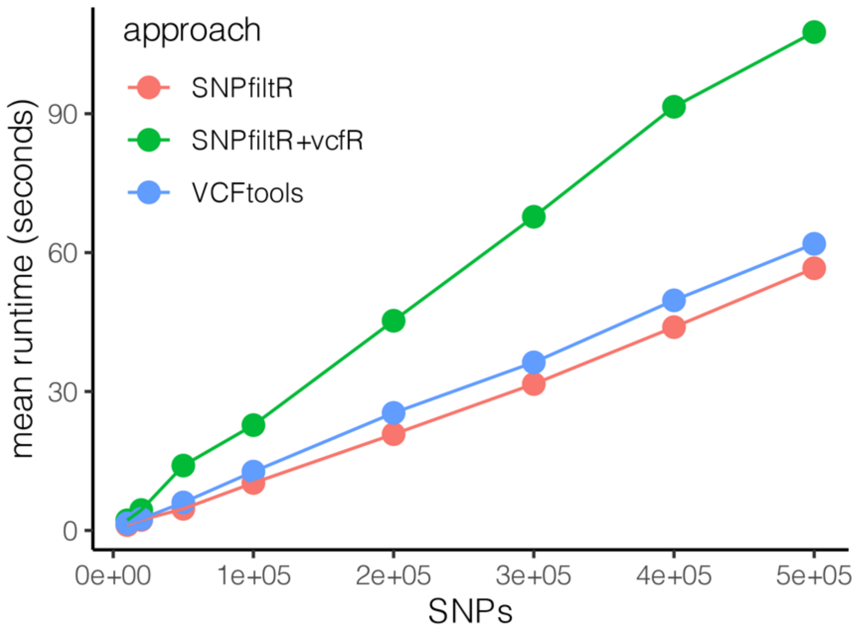
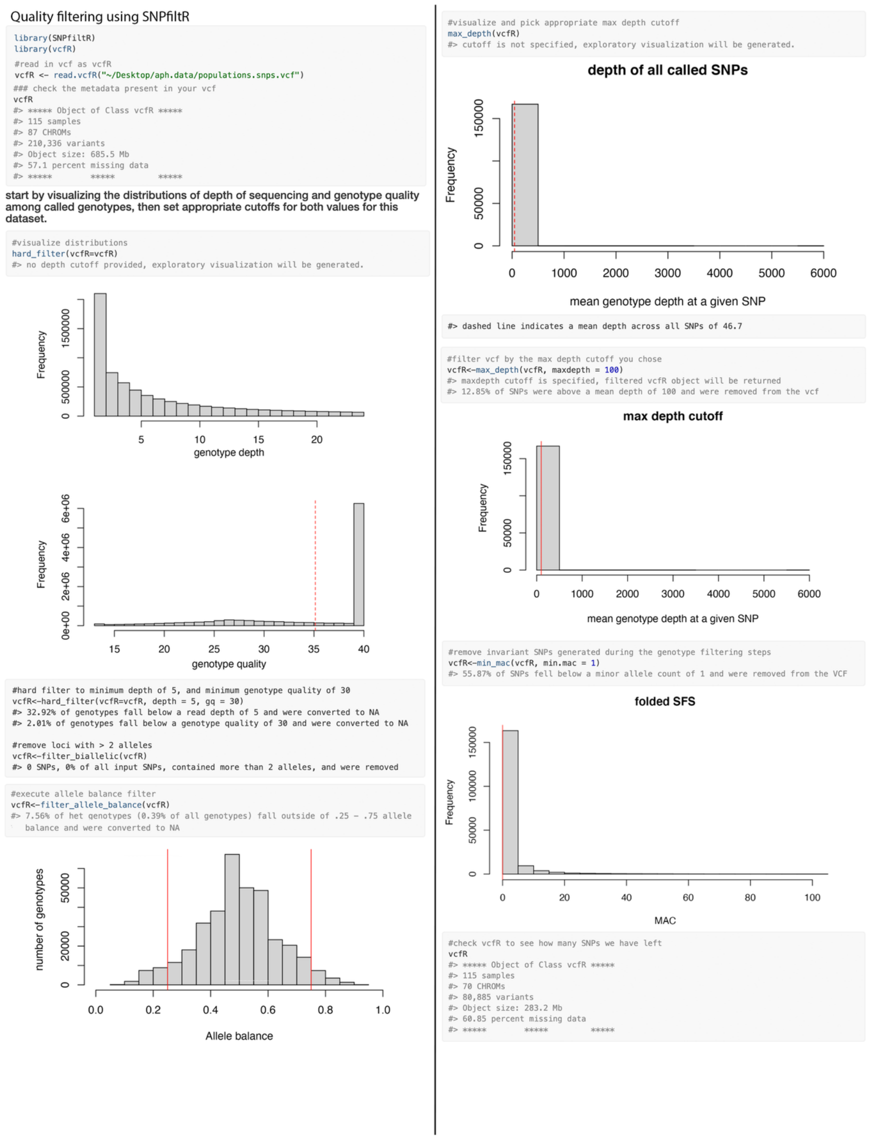


Figure 1 . Dotplot showing mean runtimes for filtering vcf files according to the following thresholds: minimum depth per genotype = 5, and minimum genotype quality per genotype = 30. Each input vcf file contained between 10K and 500K SNPs for 100 individual samples. The three approaches to performing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



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.
```

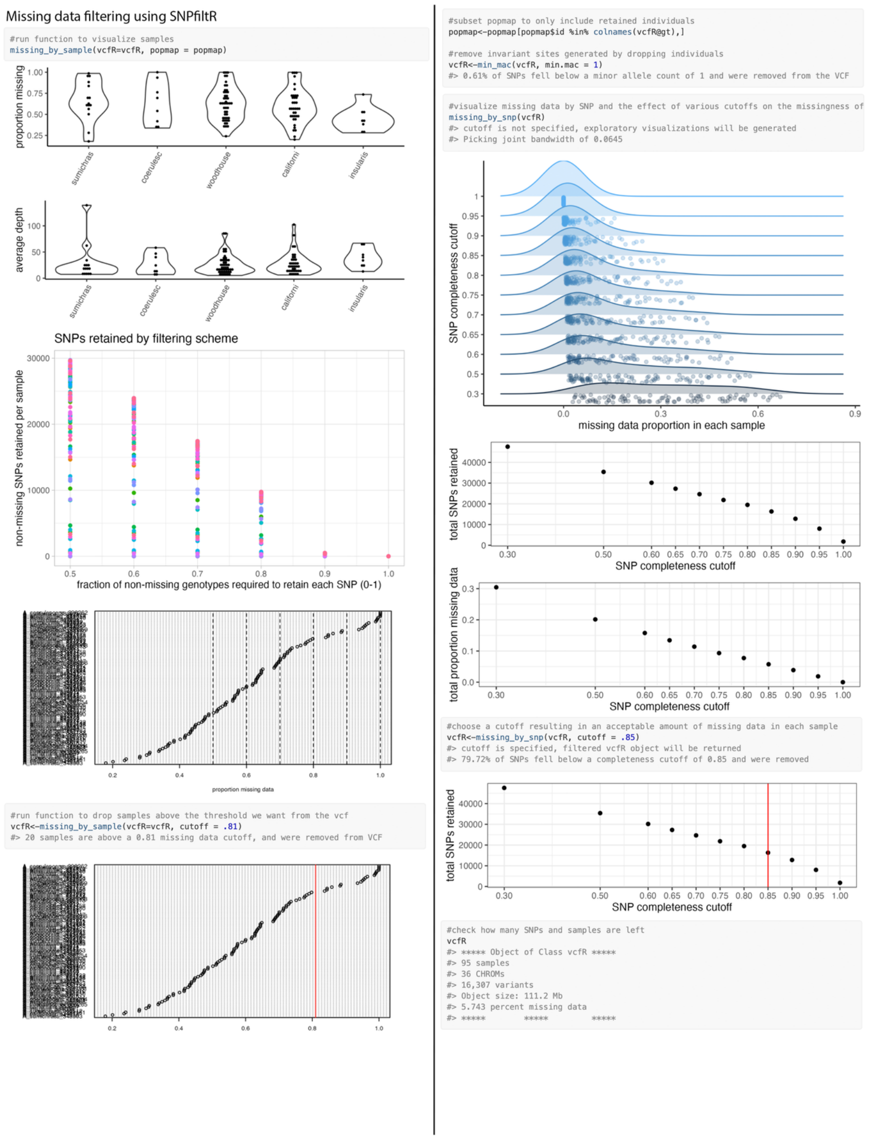
```
#> dashed line indicates a mean depth across all SNPs of 46.7
```

```
#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 2. 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.



```
#run function to visualize samples
missing_by_sample(vcfR=vcfR, popmap = popmap)
```

```
#run function to drop samples above the threshold we want from the vcf
vcfR<-missing_by_sample(vcfR=vcfR, cutoff = .81)
#> 20 samples are above a 0.81 missing data cutoff, and were removed from VCF
```

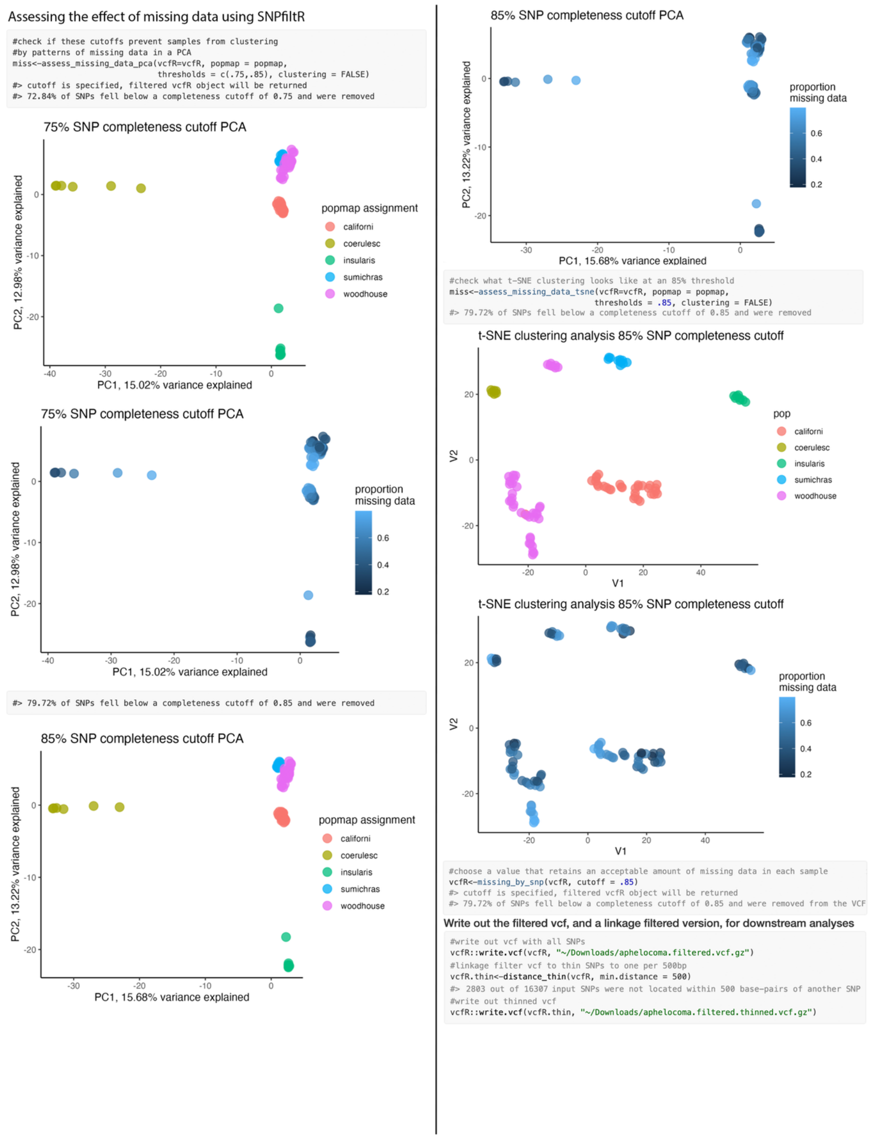
```
#subset popmap to only include retained individuals
popmap<-popmap[popmap$id %in% colnames(vcfR@gt),]

#remove invariant sites generated by dropping individuals
vcfR<-min_mac(vcfR, min.mac = 1)
#> 0.61% of SNPs fell below a minor allele count of 1 and were removed from the VCF
```

```
#visualize missing data by SNP and the effect of various cutoffs on the missingness of
missing_by_snp(vcfR)
#> cutoff is not specified, exploratory visualizations will be generated
#> Picking joint bandwidth of 0.0645
```

```
#choose a cutoff resulting in an acceptable amount of missing data in each sample
vcfR<-missing_by_snp(vcfR, cutoff = .85)
#> cutoff is specified, filtered vcfR object will be returned
#> 79.72% of SNPs fell below a completeness cutoff of 0.85 and were removed
```

```
#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
#> *****        *****         *****
```

Figure 3. An example using *SNPfiltR* to visualize missing data by sample and by SNP, and then filter a previously quality filtered vcf file (Fig. 2) based on user specified missing data thresholds both per sample and per SNP.


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
```

```
#check what t-SNE clustering looks like at an 85% threshold
miss<-assess_missing_data_tsne(vcfR=vcfR, popmap = popmap,
                               thresholds = .85, clustering = FALSE)
#> 79.72% of SNPs fell below a completeness cutoff of 0.85 and were removed
```

```
#choose a value that retains an acceptable amount of missing data in each sample
vcfR<-missing_by_snp(vcfR, cutoff = .85)
#> cutoff is specified, filtered vcfR object will be returned
#> 79.72% of SNPs fell below a completeness cutoff of 0.85 and were removed from the VCF
```

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")
```

75% SNP completeness cutoff PCA
PC2, 12.98% variance explained
PC1, 15.02% variance explained
popmap assignment
californi
coerulesc
insularis
sumichras
woodhouse
proportion missing data
0.6
0.4
0.2
85% SNP completeness cutoff PCA
PC2, 13.22% variance explained
PC1, 15.68% variance explained
t-SNE clustering analysis 85% SNP completeness cutoff
V1
V2
pop

Figure 4. An example using *SNPfiltR* to validate that the user specified missing data thresholds (Fig. 3) remove enough missing data to avoid missing data driving sample clustering patterns. The example finishes with writing a fully filtered vcf file to disc, for input into future downstream analyses.