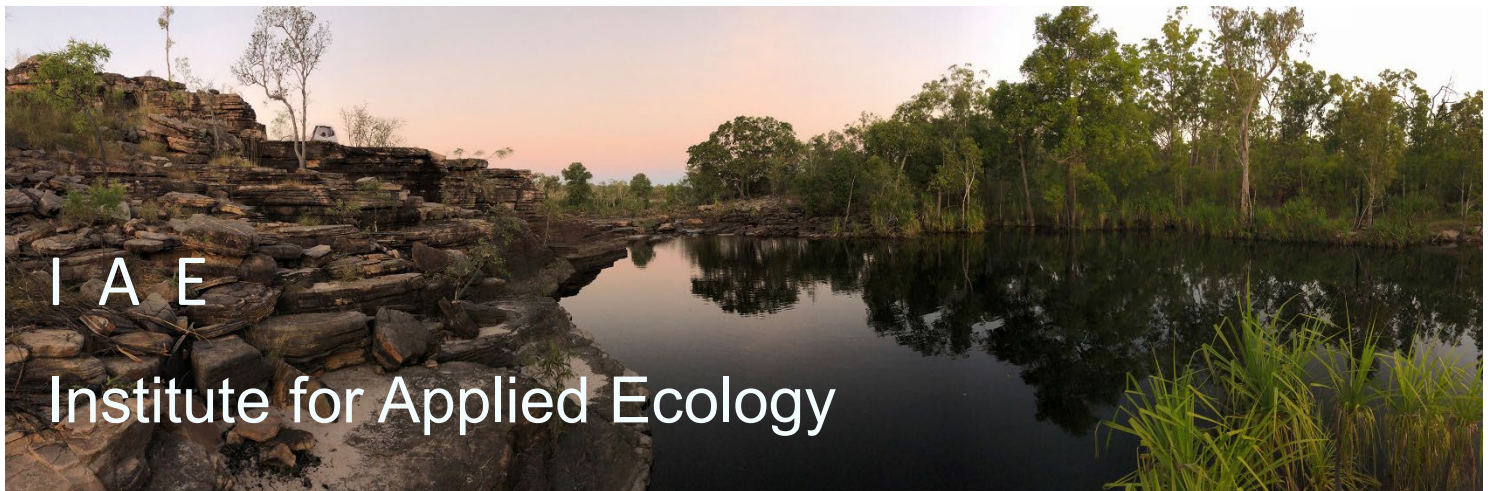


SNP Analysis using dartR



Interrogating and Manipulating Data

Version 4



Copies of the latest version of this tutorial are available from:

The Institute for Applied Ecology
University of Canberra ACT 2601
Australia

Email: arthur.georges@biomatix.com.au

Copyright @ 2025 Arthur Georges

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, including electronic, mechanical, photographic, or magnetic, without the prior written permission.

Such permission would normally be granted for educational purposes, to be used with or without modification, provided that due acknowledgement is given.

Citation: Georges, A., Gruber, B., Mijangos, J.L., and Robledo-Ruiz, D. (2025). Interrogating and manipulating data using dartR. Version 4. Institute for Applied Ecology, University of Canberra, Canberra, ACT, 2617.

dartR is a collaboration between the University of Canberra, CSIRO and Diversity Arrays Technology, and is supported with funding from the ACT Priority Investment Program, CSIRO and the University of Canberra.



Contents

Session 1: Interrogating a genlight object	4
Examine attributes and basic attributes	4
Attributes	4
Basic Summary	5
Examine metrics	5
Use adegenet accessors	6
Exercises	6
Use dartR report functions	7
Example: Reporting on Call Rate	7
Where have we come?	11
Session 2. Data Manipulation	12
Overview	12
An early cautionary note	12
Removing populations and individuals	13
Dropping populations, individuals and loci	13
Selection on an individual metric	13
Replacing the pop variable	13
Subsampling loci and individuals	14
Recalculating locus metrics	14
Recode tables	15
Tidy up the workspace	16
Where have we come?	16
Session 3: Additional Exercises	17
Exercise 1: Platypus attributes	17
Exercise 2: Platypus dataset	17
Exercise 3: Platypus reports	17
Exercise 4: Platypus subsetting	17
Exercise 5: Possums	18

Session 1: Interrogating a genlight object



If you are coming back to us, load your project, confirm the working directory, and do not forget to set the default directory for files to `getwd()` using `gl.set.wd()`.

Examine attributes and basic attributes

Attributes

The first step in examining your genlight object is to simply type its name. This will give you the attributes associated with the object, which is of Class "genlight".

```
gl <- testset.gl
gl

/// GENLIGHT OBJECT //////////

// 250 genotypes, 255 binary SNPs, size: 751.3 Kb
7868 (12.34 %) missing data

// Basic content
@gen: list of 250 SNPbin
@ploidy: ploidy of each individual (range: 2-2)

// Optional content
@ind.names: 250 individual labels
@loc.names: 255 locus labels
@loc.all: 255 alleles
@position: integer storing positions of the SNPs
@pop: population of each individual (group size range: 1-11)
@other: a list containing: loc.metrics latlong ind.metrics
```

Note: If it is a SNP dataset, the ploidy of each individual will be reported as (range 2-2). If it is a SilicoDART dataset, the ploidy will be reported as (range: 1-1)



Try this for yourself using `gl` or the genlight object containing your own data

Displayed are the number of genotypes (individuals/specimens/samples), the size of the genlight object, and the number of missing values. The ploidy value should be 2-2 for SNP data for a diploid organism (dartR does not have support for polyploid organisms), so if it is something else, you have a problem with your data. SilicoDART presence absence data has the ploidy set to 1-1.

Slots containing important information are listed, such as `@position`, which lists the position of the SNPs in the sequence tags (referenced from 0 as position 1). The `@other` slot is particularly important, because it holds the `loc.metrics` from Diversity Arrays Technology Pty Ltd (DART) and your `ind.metrics`.

If any of the optional content slots indicated above are missing, consider running

```
gl <- gl.compliance.check(gl)
```

which will render the genlight object compliant with dartR.

Basic Summary

To obtain a basic summary of statistics for a `genlight` object, use

```
gl.report.basics(platypus.gl)
```

This function will provide the following information, which is self explanatory.

```
SUMMARY STATISTICS
Datatype: SNP
Loci: 1000
Individuals: 81
Populations: 3

Values: 81000
      0    1    2    NA
percent 70.6 11.6 10.5  7.3

Monomorphic Loci: 359
Loci all NA: 6
Individuals all NA: 0

Sample Sizes
SEVERN_ABOVE SEVERN_BELOW TENTERFIELD
      23           17           41

Loci all NA across individuals by Population
  SEVERN_ABOVE SEVERN_BELOW TENTERFIELD
           9           9           6

Individuals all NA across loci by Population
  SEVERN_ABOVE SEVERN_BELOW TENTERFIELD
           0           0           0

Locus Metrics
AlleleID, CloneID, AlleleSequence, TrimmedSequence,
Chrom_Platypus_Chrom_NCBIV1, ChromPos_Platypus_Chrom_NCBIV1,
AlnCnt_Platypus_Chrom_NCBIV1, AlnEval_Platypus_Chrom_NCBIV1, SNP,
SnpPosition, CallRate, OneRatioRef, OneRatioSnp, FreqHomRef, FreqHomSnp,
FreqHets, PICRef, PIC_Snp, AvgPIC, AvgCountRef, AvgCountSnp, RepAvg, clone,
uid, rdepth, monomorphs, maf, OneRatio, PIC

Individual Metrics
id, pop, lat, lon, Microchip, Weight, Sex, AgeClass, concentration, PlateID,
Row, Column, order_dart, id_b, service, plate_location, sex

History
gl.read.dart(filename=infile1,ind.metafile=infile2,nas="-",probar = TRUE)
gl.recode.pop(x=platy_temp,pop.recode="new_pop_assignments.csv")
gl.keep.pop(x=platy,pop.list=c("TENTERFIELD","SEVERN_ABOVE","SEVERN_BELOW"))
gl.subsample.loci(x = platy, n = 1000, method = "random")
```

Examine metrics

To access metadata directly you can use commands of the form

```
cr <- gl@other$loc.metrics$CallRate
hist(cr)
```



Try this for yourself to examine `CallRate`, `RepAvg`, `rdepth` and other locus metrics in `testset.gl`. See if you can access individual metadata like `sex`.

Remind yourself of the variables in the metadata using

```
names(gl@other$loc.metrics)
and
names(gl@other$ind.metrics)
```

or by running
`gl.report.basics(gl)`

Use adegenet accessors



You can also interrogate the genlight object using commands built into the {adegenet} package.

<code>nLoc(gl)</code>	number of loci
<code>locNames(gl)</code>	list of loci
<code>nInd(gl)</code>	number of individuals (specimens or samples)
<code>indNames(gl)</code>	list of individuals
<code>nPop(gl)</code>	number of populations
<code>popNames(gl)</code>	list of populations
<code>pop(gl)</code>	list of population assignments for each individual
<code>m <- as.matrix(gl)</code>	generate a matrix of the SNP scores, with 0 as homozygous reference, 2 as homozygous alternate, and 1 as heterozygous.
<code>gl.smearplot(gl)</code>	a smear plot of individual against locus, useful for gross pattern identification and assessment of allelic dropout. This version adds individual labels and is only useful if there are not too many individuals.

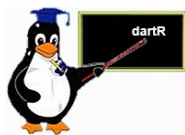
These are all useful for interrogating your genlight object, and of course can be used in your R scripts to subset and manipulate your data.



Exercises

1. Copy the genlight object `testset.gl` to a working genlight object `gl`.
2. How many loci are represented in this dataset?
3. How many individuals have been scored?
4. Are the individuals assigned to populations and if so, how many populations? What are the names of the populations?
5. Examine the genotypes for the first 5 individuals for the first 10 loci.
6. How are missing values represented?
7. Examine the structure of the dataset in a smear plot. What can you say about allelic dropout?
8. Redo activities 2-7 with your own data.

Use dartR report functions



Various report scripts provide summaries for SNP and SilicoDArT datasets, often as a prelude to filtering. Note: Do not assign the output of the report to your genlight object or you will overwrite it. The common report functions that are used as a prelude to filtering are listed below.

<code>gl.report.callrate</code>	summarises CallRate values
<code>gl.report.reproducibility</code>	summarises repAvg (SNP) or reproducibility (SilicoDArT) values
<code>gl.report.monomorphs</code>	provides a count of polymorphic and monomorphic loci
<code>gl.report.secondaries</code>	provides a count of loci that are secondaries, that is, loci that reside on the one sequence tag.
<code>gl.report.rdepth</code>	reports the estimate of average read depth for each locus
<code>gl.report.hamming</code>	reports on Hamming distances between sequence tags
<code>gl.report.overshoot</code>	reports loci for which the SNP has been trimmed along with the adaptor sequence
<code>gl.report.taglength</code>	reports a frequency tabulation of sequence tag lengths

Example: Reporting on Call Rate

As an example, let us consider reporting on call rate across loci.

```
gl.report.callrate(testset.gl)
```

```
Starting gl.report.callrate
Processing SNP data
Reporting Call Rate by Locus
No. of loci = 255
No. of individuals = 250
Minimum      : 0.401932
1st quantile : 0.7879225
Median       : 0.971014
Mean         : 0.8634915
3r quantile  : 0.993237
Maximum      : 1
Missing Rate Overall: 0.12
```

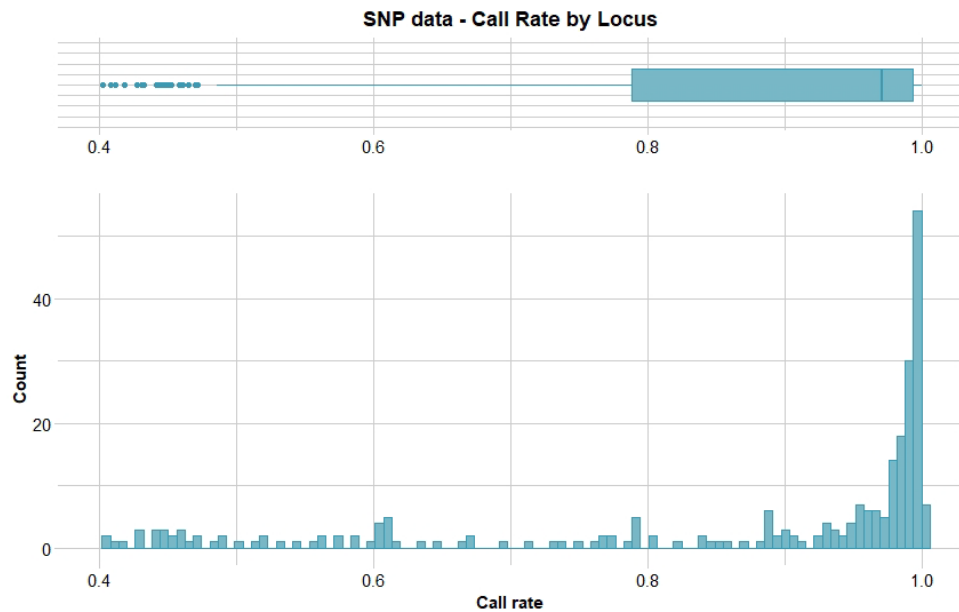
	Quantile	Threshold	Retained	Percent	Filtered	Percent
1	100%	1.0000000	7	2.7	248	97.3
2	95%	0.9990340	19	7.5	236	92.5
3	90%	0.9971010	34	13.3	221	86.7
4	85%	0.9961350	44	17.3	211	82.7
5	80%	0.9951690	53	20.8	202	79.2
6	75%	0.9932370	67	26.3	188	73.7
7	70%	0.9911108	77	30.2	178	69.8
8	65%	0.9884060	91	35.7	164	64.3

```

9      60% 0.9855070      103    40.4      152    59.6
10     55% 0.9803862      115    45.1      140    54.9
11     50% 0.9710140      128    50.2      127    49.8
12     45% 0.9577778      140    54.9      115    45.1
13     40% 0.9389374      153    60.0      102    40.0
14     35% 0.9021252      166    65.1      89     34.9
15     30% 0.8606764      178    69.8      77     30.2
16     25% 0.7879225      191    74.9      64     25.1
17     20% 0.6672464      204    80.0      51     20.0
18     15% 0.6039616      216    84.7      39     15.3
19     10% 0.5151690      229    89.8      26     10.2
20      5% 0.4506281      242    94.9      13      5.1
21      0% 0.4019320      255   100.0       0      0.0
Saving the ggplot to session tempfile
Saving tabulation to session tempfile
Completed: gl.report.callrate

```

The report is fairly self-evident. The table provides you with a basis for deciding an appropriate Call Rate to use as a threshold for filtering loci. For example, if you reject all loci that fail to call in any individual, you will lose all but 30.9% of your loci. That is too stringent. However, a threshold of 0.95 might be appropriate, with 67.8% of loci retained. The plot might make the decision simpler.



You can consider call rate across individuals using

```
gl.report.callrate(testset.gl, method='ind')
```

```

Starting gl.report.callrate
Processing SNP data
Reporting Call Rate by Individual
No. of loci = 255
No. of individuals = 250
Minimum      : 0.7490196
1st quantile : 0.8666667
Median       : 0.8823529
Mean         : 0.8765804
3r quantile  : 0.8941176
Maximum      : 0.9333333
Missing Rate Overall: 0.12

```

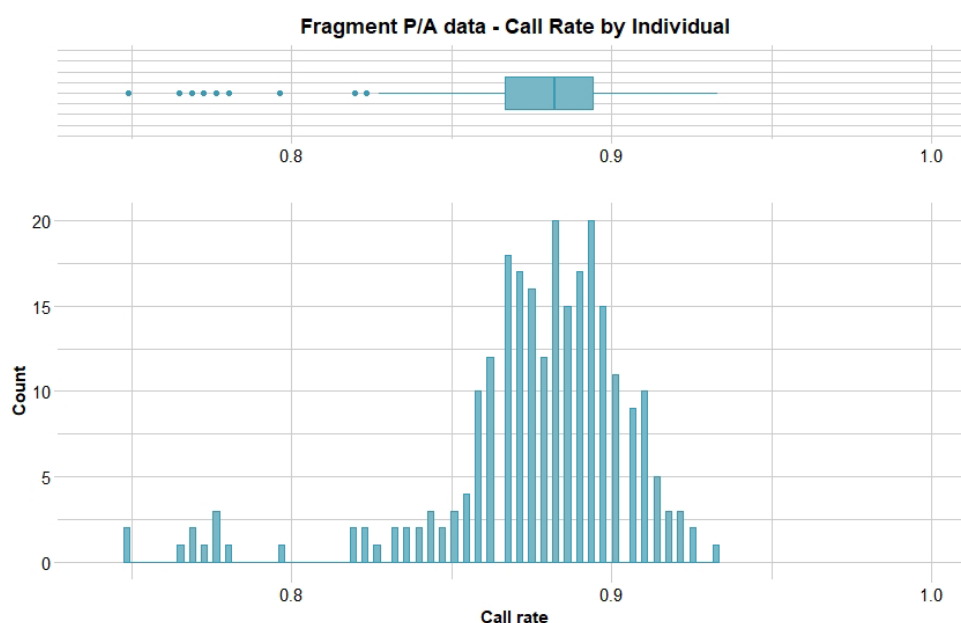
Quantile	Threshold	Retained	Percent Filtered	Percent		
1	100%	0.9333333	1	0.4	249	99.6
2	95%	0.9137255	14	5.6	236	94.4
3	90%	0.9058824	33	13.2	217	86.8
4	85%	0.9019608	44	17.6	206	82.4

5	80%	0.8980392	59	23.6	191	76.4
6	75%	0.8941176	79	31.6	171	68.4
7	70%	0.8941176	79	31.6	171	68.4
8	65%	0.8901961	96	38.4	154	61.6
9	60%	0.8862745	111	44.4	139	55.6
10	55%	0.8823529	131	52.4	119	47.6
11	50%	0.8823529	131	52.4	119	47.6
12	45%	0.8784314	143	57.2	107	42.8
13	40%	0.8745098	159	63.6	91	36.4
14	35%	0.8705882	176	70.4	74	29.6
15	30%	0.8705882	176	70.4	74	29.6
16	25%	0.8666667	194	77.6	56	22.4
17	20%	0.8627451	206	82.4	44	17.6
18	15%	0.8588235	216	86.4	34	13.6
19	10%	0.8466667	225	90.0	25	10.0
20	5%	0.8213725	237	94.8	13	5.2
21	0%	0.7490196	250	100.0	0	0.0

Saving the ggplot to session tempfile

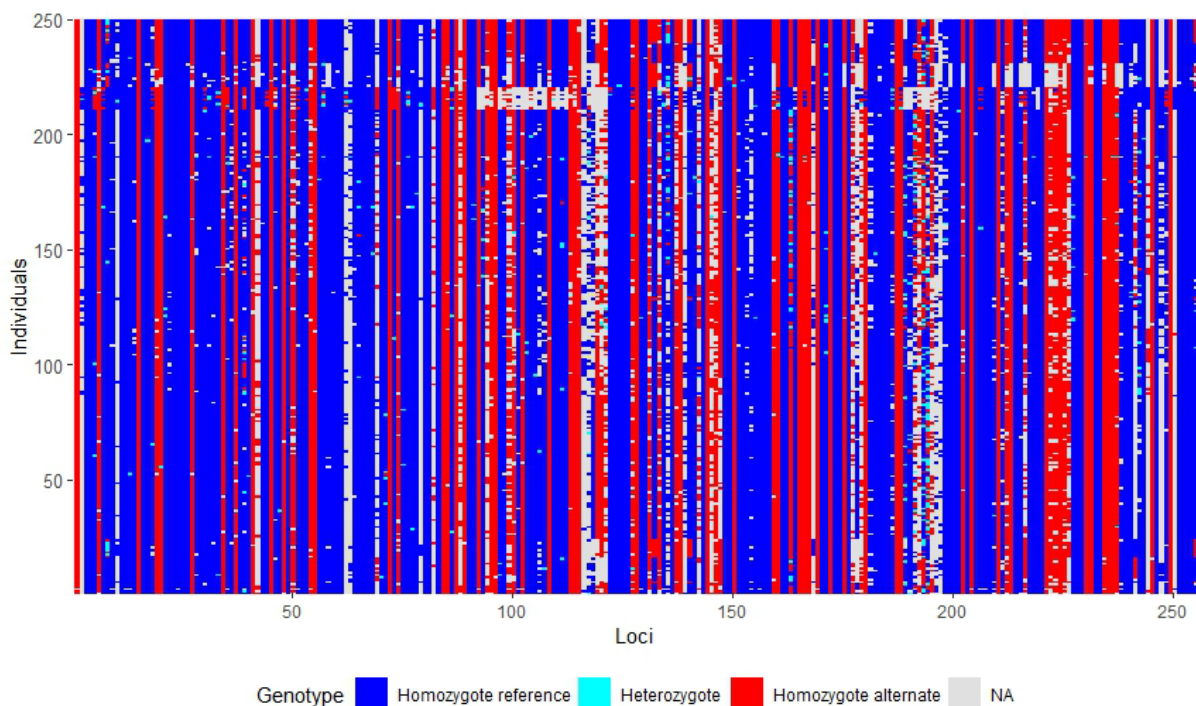
Saving tabulation to session tempfile

Completed: gl.report.callrate



Again, the printout and plot gives you a basis for selecting a threshold for filtering individuals with a poor call rate.

An alternative way of visualizing the call rate across individuals and loc is to examine a smear plot. `gl.smeaplot(gl)` SNP loci are shown along the x-axis and individuals/specimens along the y-axis. Missing data are shown as white, so it can be seen that some loci (vertical white stripes) have an exceptionally low call rate, and some individuals have a poor call rate for some loci (horizontal white bands).



Try this for yourself using the sample SilicoDart dataset

```
gl.report.callrate(gs)
gl.report.callrate(gs, method='ind')
gl.smeaplot(gs)
```

All of the report functions operate in a similar, though not identical, manner.



Generate reports for `reprocubility`, `rdepth`, `secondaries`, `overshoot`, `taglength`, `monomorphs` and other locus metadata in `gl`. Try it on your own data.

Where have we come?



In this Session, you have learnt the various ways to interrogate a `genlight` object, namely by

- simply typing its name
- examining the locus and individual metrics
- using `adegenet` accessors
- using the `dartR` report functions

These techniques are important for familiarizing yourself with your dataset at the most fundamental level, and for detecting any errors or outliers likely to cause problems in later analyses.

Session 2. Data Manipulation



If you are coming back to us, load your project, confirm the working directory, and do not forget to set the default directory for files to `getwd()` using `gl.set.wd()`.

Overview



Almost certainly you will need to undertake various manipulations of your data prior to analysis. This might include altering or correcting some individual or population labels, removing some individuals included in error or perhaps duplicated, and recalculating the locus metrics to include some new metrics not provided by Diversity Arrays Technology.

During the course of analysis as well, you will need to amalgamate individuals from sampling sites, delete populations (say the outgroup taxa for an analysis directed at the ingroup taxa alone), or examine statistics from one population on its own.

An early cautionary note

While it is possible to manipulate the content of `genlight` objects directly, this can result in serious miscalculation. For example, deleting loci directly with

```
index <- glNA(gl, alleleAsUnit=FALSE)<= 0.05*nInd(gl)
gl <- gl[, index]
```

is fine, provided you remember to apply the restriction to the locus metrics as well. Otherwise, the locus metrics will become out of sync with the loci, which is potentially disastrous.

```
gl@other$loc.metrics <- gl@other$loc.metrics[index,]
```

Even subsetting is potentially problematic. For example

```
gl <- gl[,1:100]
```

will generate a `genlight` object containing only the first 100 loci, but the locus metrics will retain data for all the original loci. This is a bug in `adegenet` that we have attempted to redress in the definition of a `dartR` object. A `dartR` object is a superset of the `adegenet` `genlight` object. Nevertheless, it is prudent not to bypass the `dartR` scripts and try to manipulate the `genlight` objects directly in R.

So even if you are well versed in R programming, and tempted to take shortcuts, they are best avoided.

Instead, there are functions in `dartR` to assist you in manipulating a `genlight` object. It is safest to use them.

Removing populations and individuals

Dropping populations, individuals and loci

The following set of functions apply to populations and individuals.

```
gl <- gl.drop.pop()    remove listed populations
gl <- gl.keep.pop()    keep only the listed populations
gl <- gl.drop.ind()    remove listed individuals
gl <- gl.keep.ind()    keep only the listed individuals
gl <- gl.drop.loc()    remove listed loci
gl <- gl.keep.loc()    keep only the listed loci
```



Copy `testset.gl` to a new genlight object, and list the populations to which individuals are assigned. Delete one or more populations. Delete one or more individuals. Confirm that the population(s) or individual(s) has been dropped. Try this with your own data.

Selection on an individual metric

You can select individuals based on one of your individual metrics using

```
gl <- gl.keep.pop(gl, "Male", as.pop="sex")
```

where `sex` is one of the variables among the individual metrics in `gl@other$ind.metrics`, and where you want to retain only the males. This parameter works with `gl.drop.pop` also.

Replacing the pop variable

It may also be convenient to assign one of your individual metrics as the population variable. Some analysts have a series of different population breakdowns in the individual metrics and move from one to the other during analysis. For example, the `popNames(gl)` might be based on sample sites, but another metric might be `region`. The region can be assigned as the `pop` variable using

```
gl <- gl.reassign.pop(gl, as.pop="region")
```

This will overwrite the existing population assignments with values of the individual metric `region`.

Some functions allow the temporary assignment of an individual metric as the population attribute. For example,

```
gl <- gl.drop.pop(gl, pop.list="NSW", as.pop="region")
```

will use `region` as the population for the purposes of deleting populations (e.g. `NSW`), then revert to the initial population assignments.

Subsampling loci and individuals

There may be occasions when you want to subsample a genlight object containing SNP or SilicoDArT data. This can be done with

```
gl2 <- gl.subsample.loc(gl, n=100, replacement=TRUE)
```

or

```
gl2 <- gl.subsample.ind(gl, n=200, replacement=FALSE)
```

Recalculating locus metrics

Several scripts delete individuals, and so many of the metadata variables provided by Diversity Arrays Technology no longer apply. For example, deleting individuals with a low call rate across loci will alter the call rate and so the [CallRate](#) provided by initially by Diversity Arrays Technology will be incorrect in the new genlight object.

There are parameters to set in each of these scripts to recalculate the locus metadata, but it is best to do your manipulations and then run

```
gl <- gl.recalc.metrics(gl)
```



Recalculate the locus metrics for your new genlight object, using a verbose = 3 to track what the function does.

These scripts will also generate monomorphic loci, where the deleted individuals have all the variation at a locus, and these monomorphic loci can be deleted using

```
gl <- gl.filter.monomorphs(gl)
```



Examine your new genlight object with a smear plot. Remove monomorphic loci (and those that are all NA) from your new genlight object. Again, set verbosity to 3. Re-examine your new genlight object with a smear plot.

You can define a new population for a set of listed individuals, merge two populations, rename a population, using

```
gl <- gl.define.pop()
```

create a new population for listed individuals; their old population assignments will be discarded.

```
gl <- gl.merge.pop()
```

merge two populations under a new name, or if applied to one population, to rename it.

```
gl <- gl.rename.pop()
```

rename a population.



Create a new genlight object from `gl`, at the same time merging two selected populations into one. Confirm that the population reassignment has been achieved.

List the individuals in the genlight object `gl`. Select two individuals and assign them to a new population called `newguys`. Confirm that the population reassignment has been achieved.

Recode tables

An alternative way to manipulate population assignments and individual labels is to use a recode table in the form of a comma-delimited csv file. This is efficient if you have a lot of changes or if you want to keep a good record of individual recoding and population assignments.

`gl <- gl.make.recode.pop()` make a recode table based on existing population labels. You will need to edit the second column of the recode table to specify the new labels to apply.

`gl <- gl.make.recode.ind()` make a recode table based on existing individual labels. You will need to edit the second column of the recode table to specify the new labels to apply. Individuals assigned the new label 'Delete' will be removed from the genlight object.

`gl <- gl.recode.pop()` apply the specified pop.recode table to the populations

`gl <- gl.recode.ind()` apply the specified ind.recode table to the individuals

`gl <- gl.edit.recode.pop()` edit population assignments, and apply the changes on closure

`gl <- gl.edit.recode.ind()` edit population assignments, and apply the changes on closure

Note that populations or individuals assigned the label 'Delete' will be removed from the genlight object when the recode table is applied. If this occurs, you will need to run

`gl <- gl.recalc.metrics()`

to recompute the locus metrics.



1. Just in case you have accidentally modified the genlight object `gl`, recreate it by copying it from `testset.gl`. List the populations and the number of populations.
2. Use `gl.make.recode.pop` to create a draft recode table with a specified name (make sure it is a `.csv` file). Edit this in excel to make changes to the population assignments. Make one population assignment `Delete`.
3. Apply the recode table to genlight object `gl`.
4. List the populations and the number of populations. Have the anticipated changes been made?
5. Now try editing the population assignments with `gl.edit.recode.pop`. List the populations and the number of populations. Have the anticipated changes been made?
6. If you deleted a population, be sure to filter out monomorphic loci and to recalculate the locus metadata.

Tidy up the workspace

We have created files that we will not use again, so they should be removed from the workspace. Check the list under the tab Environment, and use `rm()`.

```
rm(newguys)
```

Where have we come?



The above Session was designed to give you an overview of the scripts in dartR for undertaking basic manipulations of your data. Having completed this Session, you should now be able to:

- Visualize your data in a smear plot to assess structure and frequency of null alleles.
- Delete individuals and populations, rename individuals and populations, merge populations, assign individuals to new populations.
- Recalculate locus metrics after deleting individuals or populations.

Session 3: Additional Exercises



Exercise 1: Platypus attributes

Use a combination of {adegenet} accessors and `gl.report.basics` to answer the following questions.

- How many loci are represented in genlight object `platypus.gl`?
- How many individuals have been scored?
- Are the individuals assigned to populations and if so, how many populations? What are the names of the populations? How many individuals are in each population?
- Redo these activities with your own data.



Exercise 2: Platypus dataset

- Examine the genotypes for the first 5 individuals for the first 10 loci.
- How are missing values are represented? What percentage of scores are missing?
- Plot a histogram of the SNP scores. Why is the count of homozygous reference greater than the count of homozygous alternate?
- Examine the structure of the dataset in a smear plot. What can you say about allelic dropout?
- Redo these activities with your own data.



Exercise 3: Platypus reports

- Generate a report on callrate for the platypus dataset. What do you conclude? Are the data 'clean' and ready for analysis?
- Generate reports on reproducibility, monomorphs, secondaries and read depth. Consider these reports along with that for callrate and devise a strategy for pre-analysis filtering to yield a reliable dataset.
- Redo these activities with your own data.



Exercise 4: Platypus subsetting

- List the population names for the platypus dataset?
- Create a new genlight object `tenterfield.gl` which has data only for the population labelled `tenterfield`. Note that there are two ways of doing this.
- Combine the populations `SEVERN_ABOVE` and `SEVERN_BELOW` into a single population in a new genlight object `platy2.gl`.
- Create a new genlight object `sexes.gl` where the populations are defined by sex `M` or `F`.
- Delete individuals `U_TENT_10` and `U_TENT_14`.
- Try some of these activities with your own data.

Exercise 5: Possums



- Repeat Exercises 1-4 with the platypus dataset.



Ende