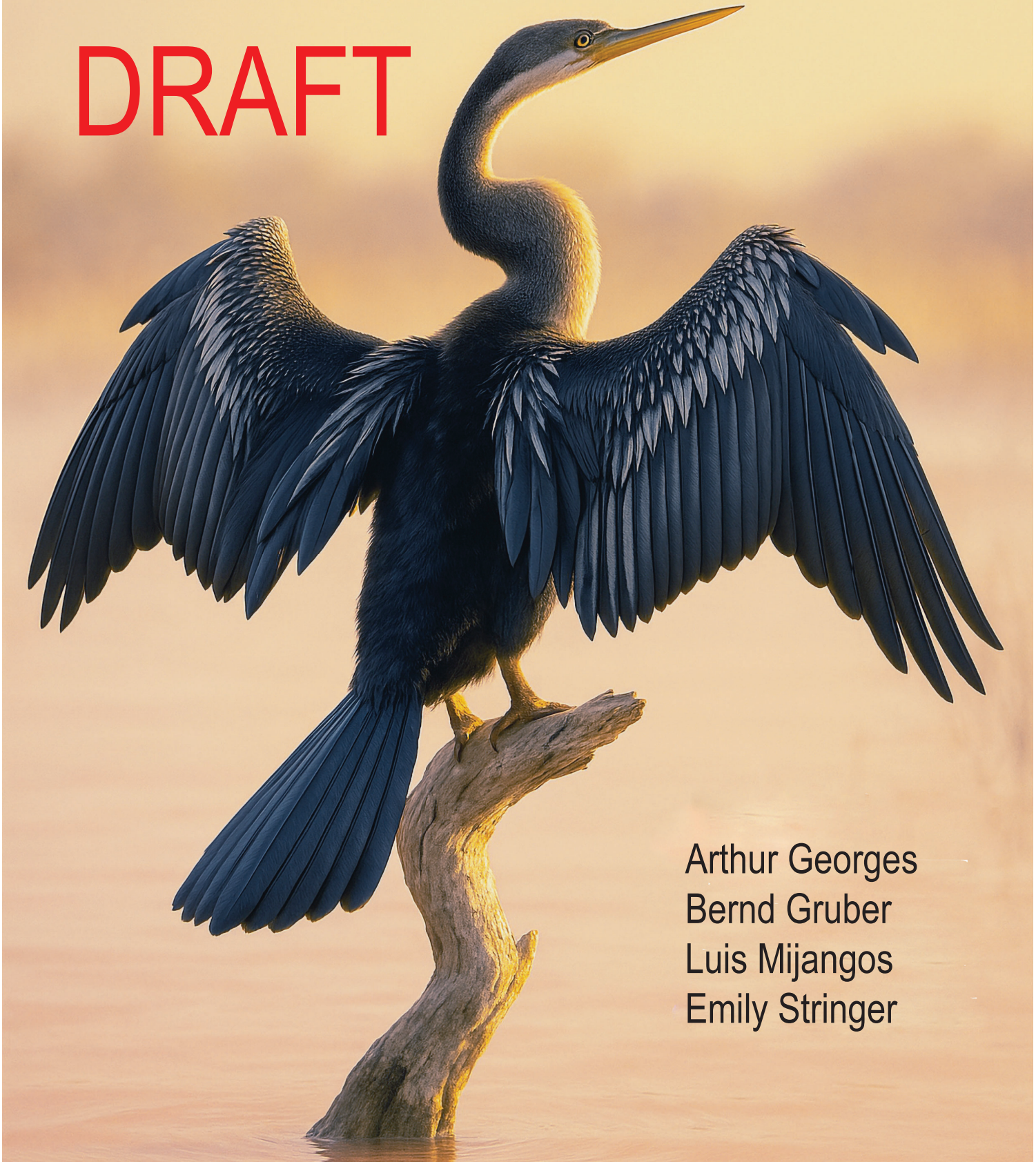
Getting started with dartR

DRAFT



Arthur Georges
Bernd Gruber
Luis Mijangos
Emily Stringer

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

The Institute for Applied Ecology
University of Canberra ACT 2601
Australia

Email: arthur.georges@biomatix.com.au

Publisher: Biomatix Pty Ltd, Sutton NSW 2620, Australia

ISBN: 978-0-646-73235-0

Acknowledgements: We would like to thank the dartR development team for the many discussions that led to the development of dartR and this eBook. We are particularly grateful to Diana Robledo Ruiz, Carlo Pacioni and Peter Unmack. Many people generously provided data to be used in the worked examples and exercises. We thank in particular, Ryan Collie, Steve Donnellan, Michael Mahony, Peter Unmack and Elise Furlan for their generosity. Margarita Medina and Catarina Pujadas provided a detailed review of the eBook before its release. Andrzej Kilian and the staff of Diversity Arrays Technology Pty Ltd assisted greatly in clarifying some of the more difficult aspects of the DArT workflows and pipelines.

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

Foreword

DartR is, at its core, just a software package. Yet it has been a remarkable success, growing beyond its developers' wildest dreams. Today, it is used throughout the world, in countless ways, across agricultural, environmental, and health sectors, and applied to organisms from across the tree of life. In 2025, the global influence of dartR and its community-driven design were recognised with the *ARDC Eureka Prize for Research Software* — a fitting acknowledgment of what can be achieved when open, collaborative science truly works.

What is its secret?

I believe it lies in dartR's roots. It has always been — and will always remain — a grassroots platform, made by and for users with diverse interests. That is how it began, and that is why its focus remains squarely on user needs and ease of use.

This matters now more than ever. Science must deliver solutions to environmental, agricultural and health challenges. Genomics — the original biological *big data* — has emerged as a powerful contributor, yet the sheer volume of information can be overwhelming. Where do you even begin?

DartR provides that gentle start. And this eBook is your start with dartR. It guides you, hands-on, through the breadth of what can be done with DNA datasets in terms of getting started – reading in the data, subsetting and pre-analysis, assessing quality control, filtering to get a set of reliable markers, and exploratory visualisation.

The team behind this eBook have put a great deal of thought and effort into its design, drawing on what users need and how they learn — lessons gained from countless workshops and an active online community.

Whatever your interests or applications, I hope this eBook inspires you to dive into genomic data with confidence and to help solve our most pressing biological challenges.

Good luck!

Dr Oliver Berry

Director
Environomics Future Science Platform
CSIRO, Australia
Perth, November 2024

Table of Contents

Introduction

We are living through a golden age of discovery in population genomics. In just over a decade, we have witnessed an extraordinary growth in both the generation and availability of genetic information across the tree of life. What began with the sequencing of a handful of model species has expanded into a deluge of genomic data encompassing microbes, plants, animals and fungi — from organisms thriving in deep oceans to the rarest reptiles clinging to survival. High-throughput sequencing – once prohibitively expensive and technically daunting – now routinely delivers gigabytes of data at a fraction of former cost, unveiling the immense genetic variation that exists across all levels of biological organisation—from ecosystems and species to populations, individuals, tissues, and even single cells. Every dataset adds another piece to the puzzle of how species adapt, interact, and persist in a changing world.

The revolution is not just in the generation of new data — it is in what *c*an be done with those data. There has been an equally remarkable acceleration in the development of analytical tools capable of extracting biological meaning from these immense datasets. New software packages, programming environments and computational methods appear daily, offering ever more refined ways to visualise, filter, analyse and interpret genomic information.

Together, these two advances – unprecedented rate of data production and rapid tool innovation and development – have transformed how we approach the study of biodiversity, the questions we can now answer, and the pace at which we can advance our understanding.

For researchers and practitioners, this transformation brings both opportunities and challenges. The opportunities lie in addressing questions once thought intractable, including resolving fine-scale population structure, detecting adaptive variation and translating genomic evidence into conservation and management decisions. The challenges lie in mastering the analytical landscape: understanding data formats, applying rigorous pre-analysis filtering and manipulations and choosing appropriate statistical and visualisation methods among an ever-growing suite of options and progressively increasing mathematical complexity.

This eBook is designed to get you started in navigating that landscape in the context of reduced representation sequencing – that is, using a representative sample of single nucleotide polymorphisms (SNPs) to represent genome-wide genetic variation. This resource provides a structured introduction to working with genetic data using the dartR packages (the dartRverse) embedded in the R programming environment. Each chapter builds on the last – from learning the fundamentals of RStudio, through data input and manipulation, to filtering, to exploratory visual analyses such as Principal Components Analysis (PCA). Step by step, you will acquire both the conceptual understanding of the organisation and storage of voluminous SNP data and the practical skills required to confidently and reproducibly interrogate and manipulate real SNP datasets.

This eBook is dynamic. There is the main body of the text which remains stable and does not change unless there is a new edition, and there are the dynamic elements. The dynamic elements include Worked Examples, Exercises, Podcasts and other web components. A significant advantage of having dynamic elements in the eBook

is that they can be refined easily as new insights arise on how to communicate the ideas. Indeed, we strongly encourage feedback on how to improve our materials and mode of delivery.

The pedagogical approach builds upon the Self-paced Learning Modules developed with funding from the Centre for the Advancement of University Teaching (CAUT). This gives you control over your own learning and the pace at which you are comfortable learning. It recognises that deep learning is achieved through recall – recall of the concepts and information presented in overview materials to integrate that knowledge in the context of problem solving. In progressing through the eBook, you should ideally first listen to the podcasts generated by AI as an audio overview of each chapter, read the introductory material in each chapter, then follow through the worked examples. However, understanding what you read and being held by the hand through worked examples are only first steps. Getting the most from this eBook requires particular attention to integrating and applying what you understand to solve the problems presented by the set exercises, and to your own datasets as you progress. Then, both the conceptual understanding and the practical skills you learn will stay with you.

To conclude, it is a great time to be a biologist – a time when curiosity can be matched with an abundance of data, when questions can be scaled to the level of whole genomes, and when the computational impediments to discovery are rapidly dissolving through the efforts of an army of computational scientists and biomathematicians.

We hope you enjoy working through this introductory eBook on dartR and herald the concurrent development of an advanced eBook on population genetics for agriculture, fisheries, forestry, population ecology and conservation biology.

Chapter 1

RStudio Refresher

Contents

Preamble

Learning Outcomes

The objective of this chapter is to introduce you to to the RStudio graphics user interface (GUI) for R. This GUI will be used throughout this eBook as the environment for undertaking analyses.

After completing this chapter you should have a basic understanding of the fundamentals of the RStudio environment, including creating and executing R scripts, montoring progress and examining output.

RStudio is a mature working environment and we do not cover all features. As your skills develop you will be able to expand your knowledge beyond the basics presented here.

Prerequisites

As this is the first chapter, there are no specific prerequisites.

Artificial intelligence tools are invaluable these days for assisting with RStudio, and you should become familiar with at least one such tool. We recommend **Claude AI**.

You may also find it helpful to listen to a **podcast summary** of the introductory material in this chapter.

Workflow

This Chapter introduces R and RStudio, with a focus on the latter. The latest versions of both packages will need to be installed.

Because the subject material is primarily to do with the RStudio graphics user interface and the R programming environment provided by RStudio, you need to follow the instructions in the Chapter and where required, action those instructions in RStudio.

There are also Exercises to stretch your capabilities. You will need to download the Worked Examples and Exercises to the RStudio Tutorial Pane, and associated data files.

Additional Reading

Many tutorials on RStudio can be found online, each with its strengths and weaknesses. To expand your knowledge and gain experience in advanced topics in RStudio you should select the option that best suits your tastes.

For a more comprehensive introduction to R programming, you might like to refer to the **official documentation**. There is also a useful **cheat sheet** put out by the developers of RStudio.

Session 1-1: Introduction to RStudio

What is R?

R is a statistical computing language based on an earlier implementation of a programming language called S available in commercial form. R is in the public domain.

R was created by Ross Ihaka and Robert Gentleman (hence the name R) at the University of Auckland, New Zealand, and is now developed by the R Development Core Team.

Many statistical packages on the market, such as SAS, SPSS and Statistica are regarded as fourth-generation statistical programming languages. The R programming language is a hybrid between a third-generation language such as C or FORTRAN and a fourth-generation language such as SAS. This provides for much greater flexibility for the analyst but demands much more in terms of programming skills.

R supports a wide variety of statistical and numerical techniques, with comparable benchmark results to Octave and its proprietary counterpart MATLAB. R also provides the analyst with a very wide range of packages, which are user-submitted program **libraries**, for specific functions or specific areas of study. As a result, R is one of the most comprehensive statistical analysis systems on the market. R has exceptionally good graphical capacities, and can be used to produce publication-quality graphs.

The library we will be primarily using is **dartRverse**, a collection of scripts to facilitate analysis of Single Nucleotide Polymorphisms (SNPs) generated by commercial providers such as Diversity Arrays Technology Pty Ltd (DArT). Although dartR has some unique analyses, it is primarily for data manipulation, exploratory analysis, and a conduit to other packages used for SNP analysis.

To work through the chapter in this eBook, you will require familiarity with the R GUI, RStudio, but you will not necessarily need to be well versed in R programming.

What is RStudio

RStudio is an integrated development environment for R. It provides a robust set of tools to help you write and execute R code efficiently.

Here are some features of RStudio:

- **Code Editor**: RStudio provides syntax highlighting, code completion, and other powerful editing tools for R.
- **Interactive Console**: You can write and execute R commands directly in the interactive console, allowing for immediate execution and feedback.
- **Plotting and Visualisation**: It integrates with R's plotting capabilities and provides a window for viewing plots and graphs created with R.
- **Package Management**: RStudio provides a user-friendly interface to manage R packages, helping to install, update, and manage the libraries you need.

RStudio is available as a free open-source version that can be installed on Windows, macOS, and Linux.

Whether you are a beginner or an experienced R programmer, RStudio offers a comprehensive set of tools to make your work with R more productive and enjoyable.

The Command Line Interface

When you first start RStudio, a graphical user interface opens with many features to assist you (Figure 1-1). After an introductory text appears in the Console, a **Command Prompt** is presented (>), and the system awaits instructions.

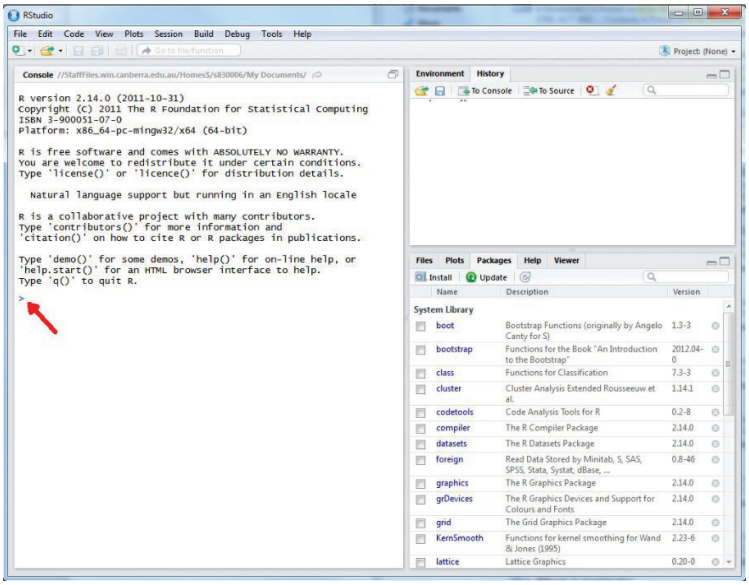


Figure 1-1. R as it appears when it first starts. The R Console window and two other windows are visible. The Program Editor and R Graphics windows do not appear until required.

Before we move on, there are a couple of little tricks worth mentioning. The first is that the Console can be cleared of text using control-L (`^l`). The second tip is that the up-arrow will recall previously submitted commands, which will save you a lot of typing. Try these as you go along.

The simplest way of using R is to supply instructions to the console.

```
sum <- 125 + 172
```

Here we are adding two numbers and putting the answer in the scalar object (single-valued vector) called `sum`.

You can view the contents of an object simply by giving its name in response to the command prompt.

```
sum
[1] 297
```

Try some assignment statements for your self to undertake some arithmetic.

Here is a slightly more complex assignment statement.

```
beetles <- c(15.2,12.1,17.8,13.9,16.4,15.1)
```

There is a lot to this simple command. What we are doing here is creating an ordered set of values, referred to as a **vector** in R terminology. In this case, the data are lengths of beetle elytra. The concatenate function `c()` is used to create the vector which is then assigned to the **object** `beetles` using the **assignment operator** `<-`. The object `beetles` is called an object because it is a self-contained entity with associated attributes that are recognised in subsequent calculations.

Again, you can view the contents of an object simply by giving its name in response to the command prompt.

```
beetles
[1] 15.2 12.1 17.8 13.9 16.4 15.1
```

Instructions to the command line are terminated with a return (↵) or a semi-colon (`;`). R instructions are case-sensitive, so the objects `beetles`, `Beetles` and `BEETLES` are all considered separate objects. It is wise to adopt a consistent practice, such as always using lowercase unless uppercase is demanded by the R syntax.

Spaces matter, sometimes. You will need to watch that.

If you instruct R to undertake some action and do not assign it to an object, then R will direct the results of the instructions to the screen. For example, requesting R to create the vector of beetle elytra without assigning it to `beetles` will result in the vector being listed on the screen.

```
c(15.2,12.1,17.8,13.9,16.4,15.1)
[1] 15.2 12.1 17.8 13.9 16.4 15.1
```

Try some assignment statements again, but this time directing the results to the screen. A bit like a simple calculator.

As an object, `beetles` can be used in subsequent calculations. For example,

```
mean(beetles)
[1] 15.08333
```

R programs often comprise a series of nested instructions, and the same result could have been obtained by using

```
mean(c(15.2,12.1,17.8,13.9,16.4,15.1))
[1] 15.08333
```

This is the advantage of an object-oriented approach to programming.

The Graphics Interface

When a command requires more sophisticated output, R will open a purpose-built window. The most useful of these is the graphics window.

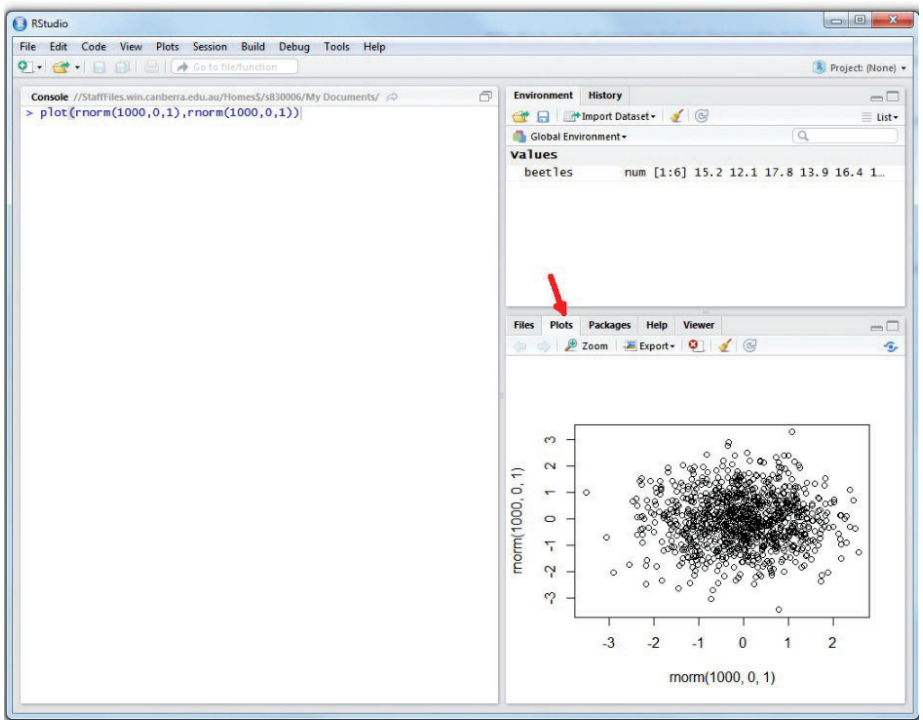
A scatter plot of 1000 pairs of coordinates drawn at random from a bivariate standard normal distribution (mean=0, stdev=1) is made by combining the `plot()` function with the `rnorm()` function as follows:

```
plot(rnorm(1000,0,1),rnorm(1000,0,1))
```

Run this command to see if you can replicate the output below.

The result is shown in Figure 1-2. This scatter plot can be saved to a file or copied to the clipboard by right-clicking on the graphics window and choosing the desired outcome.

Figure 1-2. R as it appears after activating the graphics window.



You can pull the graphics out into its own window with the [Zoom] tab, or export the image in one of the standard formats using the [Export] tab.

The Editor Interface

Using the Command Line Interface is great for a quick analysis, but it is essentially a calculator mode. Once you have done the calculations, you walk away only with the results. In more substantial analyses, we need to better manage the set of programming instructions needed to do the job. We do this using the **R editor**, which can be accessed from the file menu

```
File | New File | R Script
```

or by typing

```
control-N (^N)
```

Open the RStudio Editor.

The idea is to type all instructions in the RStudio editor for progressive submission or for submission as a block. At the end of the process, you have a complete program listing that can be saved to disk for later use.

The R editor is not all that sophisticated. Each instruction is typed in on its own line. A line can be submitted for execution by placing the cursor on it and typing control-enter (^↵). Alternatively, blocks of instructions can be highlighted and submitted in the same way. This allows progressive debugging of the program as it is constructed.

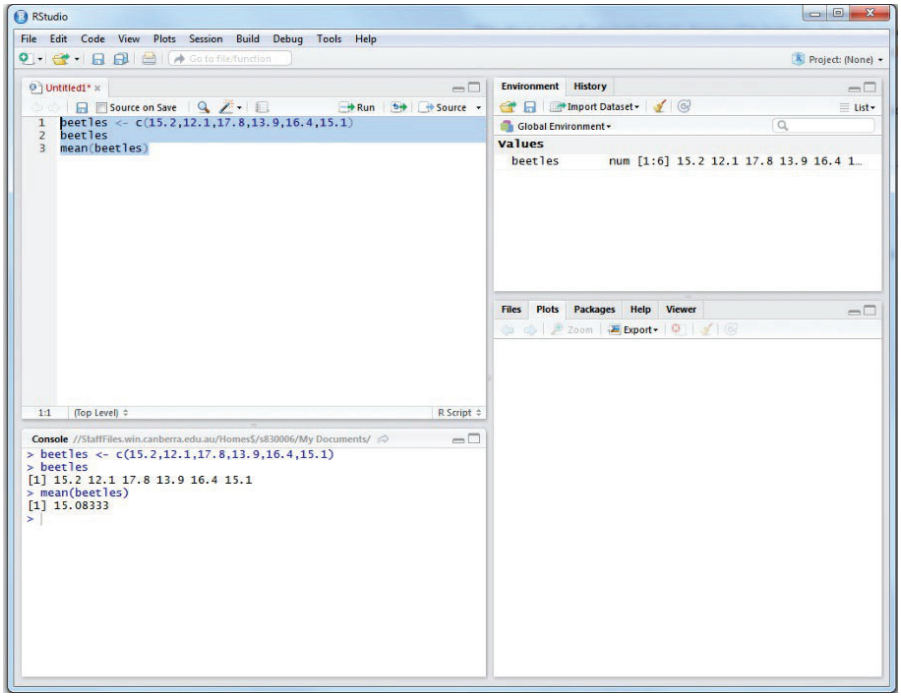
It is wise to include abundant comments as part of your programs, so that you can understand them later or pass them to others in a comprehensible form. Comments are preceded by the # character and terminated by a return (↵).

Our simple R program, with comments added is shown in Figure 1-3.

```
# Creating and displaying a list of beetle measures
beetles <- c(15.2,12.1,17.8,13.9,16.4,15.1)
beetles
mean(beetles)
```

Type the above script into the RStudio Editor. Highlight the text and submit it for execution using control-Enter (^↵)

Figure 1-3. RStudio as it appears after submitting a simple program.



You screen should look like the one shown in Figure 1-3.

Accessing Packages

Not many users of R program all the scripts that they require to undertake a task. This would be like reinventing the wheel. Instead, it is possible to access scripts written by those who have come before you, and who have made those scripts available as a package. A complete list of available packages can be obtained from the ***Comprehensive R Archive Network*** known as CRAN.

You will generally identify the packages you require after some investigative work on the web, by talking to colleagues or taking pointers from the literature. For example, the package reshape2 is useful for rearranging data, and can be accessed using the RStudio menus (`Packages | Install Packages`) or using the statement

```
install.packages("reshape2")
```

Now you have a go. Install reshape2

You may need administrator rights to install packages. Packages are only installed once, not every time you require them.

The directory where packages are stored is called the library. R comes with a standard set of packages. A list of installed packages can be obtained using

```
library()
```

or by examining the list using the Packages menu.

Apart from those included in the standard implementation of R, packages are, once installed, loaded for use in a session with the library function.

```
library(reshape2)
```

Now that you have installed the package rshape2, load the package

A list of loaded packages is obtained with

```
search()
```

Error Handling and Help

When you make an error in the syntax of commands given to R, the program will respond with some form of diagnostic message. Sometimes these are self-explanatory, sometimes they are not.

Command line help

Fortunately, R has very extensive help documentation. If you know the exact name of the function you want help with (e.g. hist for plotting histograms), help can be obtained using

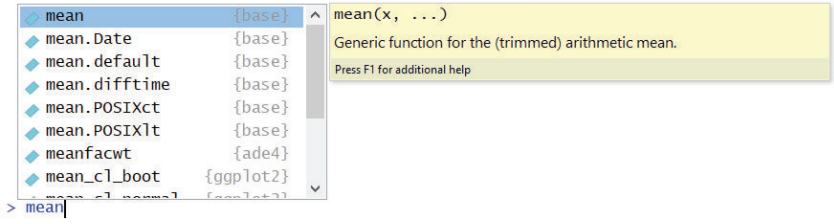
```
?mean
```

A window is displayed with help on the R function to generate the arithmetic mean. Note that the help gives you a list of possible parameters to pass to the function and gives some simple examples of its operation.

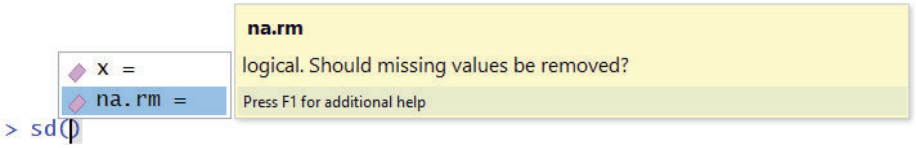
Use the `?` to generate help on the functions `mean`, `sd`, `plot` and `matrix`

Dynamic help

RStudio provides help on the fly. For example, as you are typing a function, the functions that begin with the letters you have typed are displayed as a menu. You can select the one you want, then hit tab to bring it into your statement.



Help is also available by typing a tab after having selected a function. A list of parameters is displayed. Moving among the parameters gives help on each one.



Try this for yourself with some of the functions you have used so far

Help menu

More extensive help can be obtained from the help files using the [Help] tab of RStudio. Here you can use the Search Engine and Keywords link to access a wide range of information on the operations of R.

Since you have already requested help on the mean function, you need to click on the home button to get the menu shown in Figure 1-4.

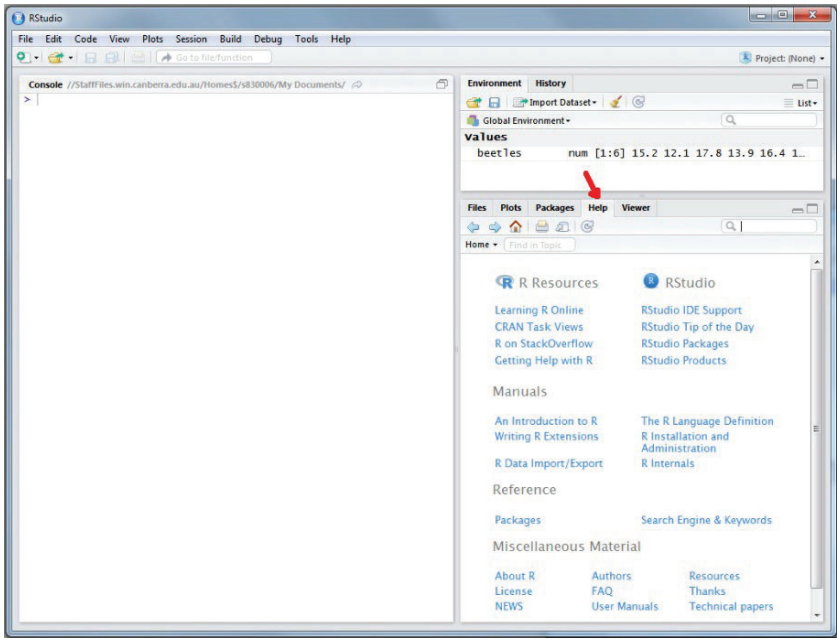


Figure 1-4. Useful information available using the [Help] tab.

Vignettes

Some packages in R have what are called **vignettes**. These are how-to guides for topics and usually offer gentle introductions and examples. Alternatively, you can view vignettes from any loaded package by going to the 'Vignettes' menu and selecting the required package name. This will give a list of all available vignettes for you to open. Sometimes this menu doesn't appear until you load a package that has a vignette.

The Web

The web and Google are good places to turn for assistance. An excellent quick reference to R can be found on **http://www.statmethods.net/**. Of course, you can also use an AI package like ChatGPT or Claude to assist you in navigating RStudio and R. These aids are becoming more and more sophisticated, almost to the point where natural languages, such as English are the new programming language.

Managing Your Workspace

Starting a New Session

RStudio facilitates the management of workflow by defining a **workspace** to hold your objects – vectors, dataframes, user-defined functions and the like. A workspace and associated files can be saved at the end of a session and reloaded at a later time when you want to continue the analysis.

Managing workflows in RStudio and R can be difficult so we need some basic rules to minimise confusion.

- Identify discrete projects or analyses and create a separate **working directory** for each one. This way you will avoid having a jumble of objects from many analyses in your workspace.
- Create an **RStudio project** to facilitate paging in and out of different projects and analyses. Link your RStudio project to your working directory for each defined project.
- Tidy up after each session by removing all unwanted and temporary objects before saving your workspace.
- Use standard file naming conventions, such as `filename.R` for R programs, `filename.csv` for raw comma-deliminted data files and `filename.Rdata` for R binary files.

Once you have started RStudio, you need to create a new project using `File| New Project`. We will do this later.

RStudio may ask you to save existing work before opening a new project. You should do this if you have important work that has been executed in a previously open project.

Terminating a Session

Exit a session by exiting from RStudio, at which time you will be asked whether you wish to save your workspace.

Resuming a Session

If you have saved your session on exit, RStudio will resume where you left off by reopening the session.

If you have defined projects, you can reopen RStudio and load the project to resume analysis.

Managing your Objects

The active objects associated with your workspace are listed when you select the [Global Environment] tab in RStudio.

In addition, a number of useful functions are useful for managing your workspace.

`setwd("c://R_analysis")`	Sets the default directory for files, and an action that can also be done from the R-studio menu `Session | Working Directory | Choose Directory`.
`getwd()`	Gets the default directory.

`ls()`	provides a list of objects in your current workspace, or use the `Environment` tab.
`rm(object)`	deletes an object from your workspace.
`rm(list=ls())`	deletes all objects from your workspace.
`sessionInfo()`	provides information about your session.

Setting a default directory

The best way to manage your work is to keep the files associated with each project in a separate directory on disk. To set the default directory use

```
setwd("C:/Users/username/Documents/eBook1")
```

R will then look in the directory `eBook1` when locating a file to read, and to write a file. Note the direction of the backslashes in the file specification.

Set the working directory to a useful location by adding the above line to your script in the RStudio Editor. You can replace `eBook1` with a directory of your choice. Submit the code for execution.

You can identify the location of the default directory, if you forget where it is, using

```
getwd()
```

You can get a listing of your current files in the working directory using

```
dir()
```

Alternatively, use the `Files` tab to view files in your working directory.

Setting up a Project

A project in RStudio is a convenient way to bundle together all the files related to a particular analysis, including code, data, documentation and output. Let's look at RStudio projects in more detail.

RStudio projects make it straightforward to pick up one of many analyses from where you left off, reproduce your work and collaborate with others. Here's how they help:

- **Isolation**: RStudio projects help to ensure that the files associated with an analysis or project are isolated in their own directory, making it easier to organise files, figure out what files are involved in a given analysis, and to move the analysis or project from one computer to another.
- **Paths**: When you open a project, RStudio sets the working directory to the project's directory.
- **Workspaces**: A project has its own R workspace. This allows you to move from one project to another and easily pick up exactly where you left off in each case.
- **Version Control Integration**: RStudio projects can be integrated with version control systems like **Git**. Each project can have its own links to a specific repository.

You can create a new project in RStudio by going to `File | New Project` and then following the prompts to create a new directory or associate an existing directory with a project. This will create an `.Rproj` file in the directory which stores project-specific settings. By double-clicking on this `.Rproj` file or re-opening the project, you can reinstate all its associated settings.

Accessing Worked Examples and Exercises

First, set up a project and associated working directory as outlined above. Use a sensible name that you will remember, like `eBook1`. You only need to do this once.

Data files for the Worked Examples, Exercises and Integrative Exercises covered by this eBook can be downloaded to your working directory by running the following command in your RStudio Console.

```
source("http://georges.biomatix.org/storage/app/media/eBook%20Introduction%20to%20dartR/eBook_data_download.txt")
```

A list of downloaded files will be shown. They will be held in a subdirectory of your working directory called `data`.

Next, install dartRverse.

```
install.packages("dartRverse")
```

Load the package and its libraries into your R Session.

```
library(dartRverse)
```

Set the global verbosity to 3.

```
gl.set.verbosity(3)
```

If using windows, install the worked examples and exercises using

```
install.packages("eBook1.zip", repo=NULL, binary=TRUE)
```

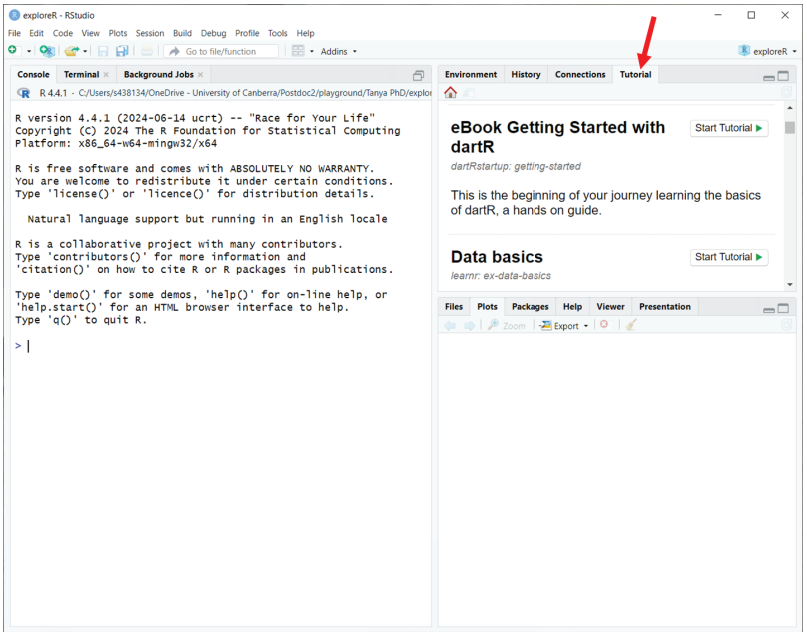
If using Mac/Linux, use

```
install.packages("eBook1.tar.gz", repo=NULL, binary=TRUE)
```

You only have to install these packages once.

To access the worked examples, navigate to the RStudio Tutorial Pane and open *eBook Getting Started with dartR* (Figure 1-5).

Figure 1-5. Worked examples and exercises available via the [Tutorial] tab.



Alternatively, use

```
learnr::run_tutorial("getting-started",package =
         "dartRstartup")
```

which will open the tutorial in a separate pop-up window.

Similarly, to access the exercises, navigate to the RStudio Tutorial Pane then open *eBook Getting Started with dartR* and navigate to the exercises for Chapter 1.

Winding Up

Where have we come?

The above sections were designed to give you an overview of the operation of R through the RStudio graphical user interface. Having completed this Session, you should now be familiar with the following concepts.

- RStudio is an integrated development environment (IDE) with a graphical user interface (GUI) that provides the R console, script editor, graphics, and help tools in one place.
- R packages are first installed, then loaded to become available for use.
- RStudio establishes a workspace. Managing the objects in that workspace is challenging for the new user of R, but proficiency will come with practice.
- Projects can be established to keep all the data, code and environment in place on save, which allows you to pick up later where you left off.
- R has abundant sources of help, including placing a `?` before a command, using tab to list options for a function, referring to the vignette if one has been provided in the loaded packages, and of course, Dr Google.

Ende

Chapter 2

A Primer on R Programming

Contents

Preamble

Learning Outcomes

This Chapter is optional. The objective is to introduce you to R programming. The package dartR does not require an in-depth knowledge of R programming, so those not wishing to dive deeper into the R programming aspects of SNP data analysis can skip this chapter.

After completing this chapter, you should have a basic understanding of the fundamentals of the R language and programming in R. This includes how to import data into the R workflow, an understanding of the data structures R uses to hold data, and how to manipulate those data with assignment statements, conditional statements and iteration statements.

Most importantly, you should be empowered to expand your knowledge beyond the basics presented here by knowing where to find information on R operations and capabilities is located and how to access it.

Prerequisites

RStudio is rquired to work through this chapter so you will need to have already completed *Chapter 1. A primer on RStudio.*

You will have been introduced to the concept of assignment using the `<-` operator in Chapter 1. This knowledge is assumed.

Artificial intelligence tools are invaluable these days for assisting with R code, and you should become familiar with at least one such tool. We recommend **Claude AI**.

You may also find it helpful to listen to a **podcast summary** of the introductory material in this chapter.

Workflow

This Chapter follows a formula. First some **Theory**, unencombered by competing options and technical asides, then some **Worked Examples** where you are led through analyses using RCode Boxes, then some **Practical Nuances**, finishing with some **Exercises** where you are left on your own to integrate the knowledge you have gained in a problem-solving context.

Because this is an optional Chapter, the Worked Examples may be treated as revisionary, and yhe code run within the RCode Boxes without referring to RStudio. Or you can progressively copy the code to your RStudio Editor to execute it and explore more options on your own.

Additional Reading

For more advanced treatment of R programming refer to the **official documentation**. There are also innumerable tutorials on R programming available online, each with its strengths and weaknesses. To expand your knowledge and gain experience in advanced topics in R programming, you should select one of these resources that best suits your tastes. We recommend the **R Programming** course from the r package **swirl**. You might also like to refer to the online short course ***Introduction to R*** by Jonathan Cornelissen.

Cheat Sheets are also good to have at hand. There is one on **R programming** and one on **functions** that are particularly useful.

Session 2-1: Key Programming Concepts

What is R?

We introduced R briefly in Chapter 1. R is a specialised programming language that has become the go-to tool for statisticians, researchers and data scientists worldwide.

What sets R apart is its vast ecosystem of specialised packages that extend its capabilities into virtually every field of research, from genomics and climate science to finance and marketing analytics.

The language's strength lies in its ability to seamlessly combine data manipulation, statistical analysis, and visualisation in a single environment, allowing users to import raw data, clean and transform it, perform sophisticated analyses, and create publication-quality graphs all within the same workflow.

While R has a steeper learning curve than point-and-click software, its power and flexibility make it invaluable for anyone working with substantial datasets or requiring advanced statistical techniques. The fact that R is completely free and open-source has contributed to its widespread adoption in academia and increasingly in industry, where organisations appreciate both its cost-effectiveness and the reproducible, transparent nature of R-based analyses.

The versatility of R has led to many different styles in the way the program is used. A programmer will use R in a very different way from someone using R to undertake statistical analyses. This module presents only one style. As your skills develop and you learn more about the capabilities and options of the programming language, you will develop your own style.

R as a Programming Language

R is a programming language combined with the features of a statistical package. As with most programming languages, R comprises:

- A set of **keywords** and **operators** that the programmer combines to form statements or instructions to be executed. R is unusual in that many of its commands are function calls, and so its keyword set is largely made up of the function names.
- A clearly defined **sequence** in which these instructions are executed.
- **Data Structures** for holding data to be used in analyses.
- Mechanisms for input and output of data to those structures, often referred to as **I/O**.
- A mechanism for **branching**, subject to some condition being met.
- A mechanism for **iteration**, that is, repeated execution of a sequence of statements while a condition is met until a condition is met, or some fixed number of times.

If you think in these terms, once you learn one programming language you can move quickly to another programming language. We will learn more about these

concepts as we move through the practical components of this Module, not necessarily in this order.

Figure 2-1. A simple R Program. The program is executed in sequence one statement at a time from the top. The "for" statement results in the enclosed code being repeated for each line of data. Branching is provided with an "if" statement.

```
# Read the data from the clipboard, tab delimited
# Note: Cut from Excel
  forest <- read.delim("clipboard")
# View the contents of the dataframe forest to confirm
  forest
# Make dataframe forest the default
  attach(forest)
# calculate timber yield separately for each species
  yield <- vector(1:length(species))
  for (i in 1:length(species)){
    if (species[i] == "radiata") {
      yield[i] <-
       density[i]*3.1416*diameter[i]*height[i]*0.85
    } else {
      yield[i] <-
       density[i]*3.1416*diameter[i]*height[i]*0.62
    } # Terminate the if statement
  } # Terminate the for loop
# Add yield to the dataframe turtle
  forest <- cbind(forest,yield)
# Examine forest to confirm
  forest
```

The script in Figure 1-1 illustrates many of these elements. The R script has a syntax (keywords and operators) unique to the R programming language. It is read and executed from top to bottom (sequence). It contains code for branching (the `if` statement) and for iteration or looping (the `for` statement). The I/O is evident in reading an Excel spreadsheet and in outputting the results locally to the screen.

Copy the script and pass it to your AI Client with a request to explain the code to you. Ask your AI Client to critique the code.

Unique attributes of R

R differs from other programming languages conceptually, structurally, and philosophically. Many of these differences arise because R was explicitly built for statistics and data analysis, not for general-purpose computing.

- R works with objects, that is, with self-contained entities with defined attributes. These entities may be a scalar value, a vector of values, a matrix or even a function. The advantage of objects is that when you use one in calculations, R knows what the object is from its attributes and how to handle it in the calculations.
- R uses vectorised operations, meaning you can perform calculations or comparisons on entire datasets at once, without writing loops.
- Many statistical and graphical functions are built-in, such as `lm()` for linear modelling, `t.test()` for testing means and `plot()` for displaying data.
- Because functions are objects, functions can act upon functions, as do members of the `apply()` family of functions for example.

- Variables in R don't need type declarations — they adapt dynamically. This is both a feature and a fault depending upon context.
- The R scripts are interpreted rather than compiled so the programming environment is interactive and exploratory, which encourages incremental development of scripts and visualisation of results as the analysis progresses.
- R employs functions that have a single object returned rather than the more flexible subroutines available in other languages.
- R is not a structured language like C++, though more structured than Perl. This means that some discipline needs to be applied if your R programs are going to be useful to the future you or others who may come to take over your programming responsibilities.
- R is slow to execute in comparison with general purpose compiled programs like FORTRAN or Julia.

Good Programming Practice

Logic

- Work out the logic of your programming task before beginning to code.
- Structure your R scripts according to best-practice principles. A good, structured script uses a hierarchy of logical control structures — each with a single-entry point and a single exit point. Then the flow of execution can be readily understood top-to-bottom.

  Avoid jumping around the script or using esoteric code such as changing global variables inside a function. It might be clever, but it is counterproductive when working in a team.

Style

- Use consistent naming conventions such as `item_case` or `item.case` for variables and functions, and a different convention for scripts.
- Comment profusely using the `#` operator (see Figure 1-1) and explain why, not just what.
- Indent statements within `if` and `for` constructs to clearly show the body of commands that apply within these constructs.
- Group related code using blank lines and section headers, e.g. `##### DO THE JOB`

There are some good style guides to follow such as the **tidyverse style guide**, which will help with consistency.

Functions

- Encapsulate repeated tasks, even small ones, into functions.
- Include defensive checks and defaults in functions. Artificial Intelligence aids, such as Claude or ChatGPT are good at identifying and trapping potential sources of error.
- Use **Roxygen** comments (`#'`) for documentation at the head of your functions.

Computations

- Avoid explicit loops when vectorisation is possible, for example

```
# Bad
  for (i in 1:length(x)) y[i] <- x[i]^2
# Good
  y <- x^2
```

- Remove from loops evaluations that can be done before the loop is executed.
- Load only what you need. Do not import an entire package if you are to use only one of its functions. Use `importFrom` instead.
- Remove objects that are not to be used again, with `rm()`.

Clarity over efficiency

Remember, R programming is as much about clarity in communication as it is computation. Readable, reproducible and thoroughly tested code is more valuable than clever code that is comprehensible only to you, and likely not comprehensible even to you weeks or months later.

Session 2-2: Worked Examples

To access the Worked Examples, navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR* and navigate to the worked examples listed below.

Worked Example 2-1: Data Structures

This worked example will run through the R programming using various data structures call objects.

Worked Example 2-2 Input/Output

This worked example will run through the R programming code for importing and export of data into the datastructures covered in the previous session.

Worked Example 2-3: Controlling workflow

This worked example will run through the R programming code for controlling workflow using IF and FOR constructs and other elements specific to the R programming language.

Worked Example 2-4: Functions

This worked example will run through the R programming code on user-defined functions.

Session 2-3: Practical Nuances

When things go wrong

There is a dreaded shadow that hangs over all who engage in computing — SYNTAX. If you don't get it right, the program will not work. Finding out why it will not work is not always easy, though many simple mistakes will be immediately evident.

AI Tools like Claude and ChatGP have revolutionised our approaches to debugging our scripts. Both are fully familiar with R and dartR. They are not infallible but are of immense value in identifying bugs and potential logical flaws in your programming.

Nevertheless, we provide some advice on how to proceed with manual troubleshooting.

Help tools – first port of call

```
?summary              # Get help for a specific function
??regression          # Search for functions related to "regression"
help.search("mean")   # Search help files for a term
```

Failure to Balance Quotes

String values are usually identified as such by enclosing them in quotes. If you fail to supply the terminal quote, a very common error, then R passes all that follows the initial quote into the string variable. Usually this is evident because the R Console responds with a + rather than the usual command prompt >. You need to break out of this using the Esc key, locate and rectify the error, and resumit the code.

Failure to Balance Brackets

Many R commands comprise nested function calls, and so you have brackets within brackets. These two need to be balanced. Blocks of code also are contained within brackets ({ and }), and these need to be balanced. RStudio will identify balanced brackets to assist with debugging. Merely mouse-over a bracket to highlight its counterpart, or press Ctrl + P to jump between the opening and closing bracket.

Package not loaded

```
glPCA()
Error in glPCA() : could not find function "glPCA"
```

When drawing functions from third-party providers, you need to first install the package with `install.packages("new.package")` and then load the package with `library(new.package)`.

Incorrect syntax for function calls

R functions are very particular in what they expect as arguments, and often the order of the arguments is quite important. When a function fails, the first step is to call up the help files using the `?` prefix. An example is

```
?summary
```

This command opens up the help files in a new window. The help information contains full details of the syntax for the `summary()` function, which should enable you to identify and rectify the problem with your program.

Mixing Up Characters with Numbers

Mis-spelling variable names and function names is a very common mistake, and one that requires attention to detail. A common mistake is to use a lowercase L (l) in place of the number one (1) or the letter O in place of the number 0. Look for this when your program does not work for reasons that are not immediately obvious.

Mixing Up Variable Names with Function Names.

It is best to avoid using the pre-existing names of R functions as variable names. You will find that we have done this in this Module – using length as the name of a variable when there exists a function `length()` that returns the number of values in a vector. This can lead to considerable confusion – length(length) for example – and is best avoided

Object Does Not Exist

This is a common error and can arise for reasons as simple as mis-spelling its name, or for more complex reasons such as the object not falling on the search path.

The first step is to check that the object exists by typing its name and submitting it to the R Console. This will verify its existence and display its contents.

If the object is part of a larger object, such as a dataframe, R might not be able to find it. You can identify this problem by typing its full specification, such as `my.data$Wt`.

Debugging

Debugging a program is what programmers do, but for those not used to programming, it can be a real block. The idea is to be systematic about your approach to identifying and rectifying bugs in your programs.

The best way is to run each line of your program one at a time, checking the progress of the analysis. Bugs often beget bugs, so it is important to start at the top of the program.

Bugs derived from incorrect syntax are the easiest to resolve. Those involving logical errors take a bit more mental energy.

Exercises

Now is a good time to try some more of what you have learned so far on some exercises and your own data.

The Exercises are also found by navigating to the RStudio Tutorial Pane and then open the session *eBook Getting Started with dartR*. Navigate from there to the exercises under Chapter 2.

Winding Up

Where have we come?

We have learned that there are multiple elements to consider when using any programming language, including

- I/O – how to read data into R and output results to the console or to disk.
- Data structures – how R stores data in dataframes, vectors, matrices, arrays and lists.
- Computational elements fundamental to all programming – keywords, sequence of progression, branching and iteration.
- Functions – the special role of functions in R, built-in functions, third-party functions and functions you construct yourself.

You also should now have

- Gained detailed knowledge of the tools for undertaking computations on data held in the various R data structures.
- Come to understand how data structures that contain the results of these computations can be output to disk as csv files, text files or compact binary files.

Through the worked examples, you should appreciate how this all hangs together in the form of R scripts.

Most importantly, you should be empowered to expand your knowledge beyond the basics presented here by knowing where information on R operations and capabilities is located and how it can be accessed. Modern tools such as AI Clients are invaluable in assisting you to rapidly build capability for R programming.

Where have we not gone?

Graphics

R has outstanding capabilities in graphics. We have not touched on these capabilities. That is left to the many guides and books available on this topic, and for AI to assist you in writing code to produce publication-ready graphics.

Tidyverse

Tidyverse and other related programming tools in this family are used by many programmers to enhance productivity. We have focused largely on base R as a foundation from which to build your learning. It is a matter of taste as to whether you want to take advantage of the power of tools like Tidyverse.

Ende

Chapter 3

Data Input and Structures

Contents

Preamble

Learning Outcomes

The objective of this chapter is to introduce you to the manner in which dartR stores genotype data and associated metadata, and how in input and export data from dartR (I/O).

After completing this chapter you should have a basic understanding of the fundamentals of dartR storage including the genotypes for each individual and associated metadata for each locus and each individual. You will also learn how to read your data into dartR and how to output the results of your analyses and manipulations. An important method for outputting data is to save your dataset in a compact binary form for later use. In this Chapter, we also cover the output of data in a form for use by third-party software packages.

This is an introductory guide. You should be empowered to expand your knowledge beyond the basics presented here by exploring the help on these topics and associated scripts, and by applying the knowledge in other tutorials to come.

Prerequisites

RStudio is an integrated development environment (IDE) with a graphical user interface (GUI) used to manage R workflow and so to work through this chapter, you will need to have already completed *Chapter 1. A primer on RStudio.* For more advanced treatment refer to the **official documentation**.

Artificial intelligence tools are invaluable these days for assisting with R code and providing summary information on topics relevant to this Chapter, and you should become familiar with at least one such tool. We recommend **Claude AI**.

You may also find it helpful to listen to the **podcast summary** of the overview material in this chapter.

Workflow

You will have already established a project and associated working directory and downloaded the required data files to the `./data` subdirectory of your working directory. The Tutorial Pane in RStudio should contain the Worked Examples and Exercises.

The workflow from this point follows a formula.

- **Theory**, unencombered by competing options and technical asides;
- **Worked Examples** where you are led through analyses with sample code;
- **Exercises** where you are left on your own to integrate the knowledge you have gained in a problem-solving context.
- **Review** the learning outcomes and the section on Where have we come? to confirm that you are moving to the next chapter with all that is required under your belt.

Session 1: Introduction to DArTSeq

Sequencing

Diversity Arrays Technology Pty Ltd (DArT) is a private company that specialises in genotyping by sequencing. Their approach is one of genome complexity reduction. But what does this mean?

Basically, DArTSeq is a method that extracts reproducible genomic variation across many individuals at an affordable cost. The technique digests genomic DNA using pairs of restriction enzymes (cutters) (Figure 1). When the DNA is cut at two locations within a reasonable distance of each other, the fragment is available for sequencing using the Illumina short-read platforms. Hence, the data are representational in the sense that they are generated for a random but reproducible selection of small fragments of sequence only, fragments that exhibit variation at the level of single base pairs (SNPs).

The first step in the process involves the selection of restriction enzymes that provide the best balance between getting an adequate fraction of the genome represented, an adequate read depth for each fragment, and adequate levels of polymorphism. This is species-specific and so requires some initial optimisation.

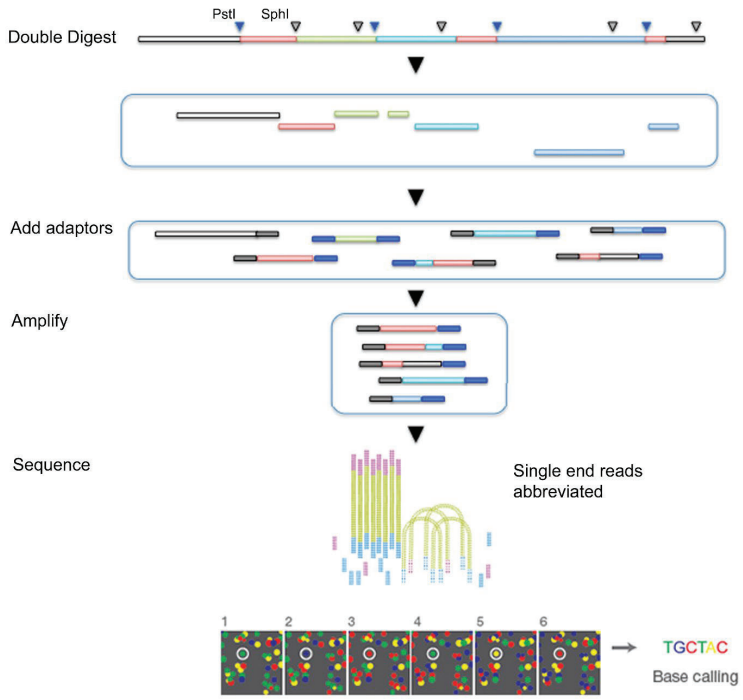


Figure 3-1. *A diagram showing the workflow for representational sequencing using the services of DArT.*

Once the best restriction enzymes are selected, say PstI (recognition sequence 5'-CTGCA|G-3') and SphI (5'-GCATG|C-3'), then the DNA is digested, and various adaptors are added to the sequence fragments to allow Illumina short-read sequencing to proceed. These additional terminal sequences include a barcode to allow disaggregation of the sequences for each sample during later analysis.

The fragments of DNA selected by this process are sequenced in an abbreviated process to yield a set of raw "sequence tags" each of around 75 bp. They are filtered on sequence quality, particularly in the barcode region, truncated to 69 bp and stacked by sequence similarity. A series of proprietary filters are then applied to select those sequence tags that include a reliable SNP marker.

In particular, one third of samples are processed twice as technical replicates, from DNA and using independent adaptors, through to allelic calls. Scoring consistency (repeatability) is used as the main selection criterion for high-quality and low error-rate markers.

These DArT analysis pipelines have been tested against hundreds of controlled crosses to verify Mendelian behaviour of the resultant SNPs as part of their commercial operations.

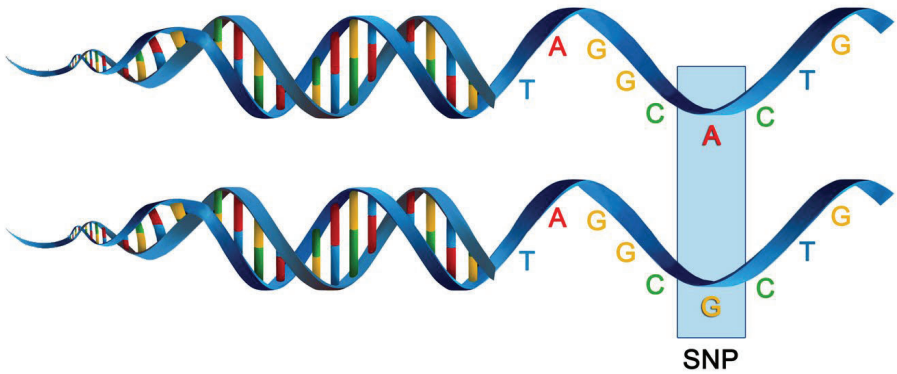
When you come to publish, you may receive requests to be more elaborative than you are able to, because of the proprietary nature of the pipelines. DArT Pty Ltd is a private company and needs to hold some of its proprietary analyses in-house. Note that other companies with whom you interact, including Illumina, do the same. The work is reproducible in that using the same service/equipment on the same samples will yield the same result. Most journals accept this.

The SNP dataset

SNPs, or single nucleotide polymorphisms, are single base pair mutations at a nuclear locus (Figure 2). That nuclear locus is represented in the dataset by two sequence tags which, at a heterozygous locus, take on two allelic states, one referred to as the reference state, the other as the alternate or SNP state.

Because it is extremely rare for a mutation to occur twice at the same site in the genome (perhaps with the exception of Eucalypts), the SNP data are considered to be effectively biallelic. Sites with more than two states that occur rarely are typically eliminated in the quality control steps as they are bundled with multiallelic sites arising from multiple copy sequences (e.g. as would arise from gene duplications) removed during preliminary filtering.

Figure 3-2. *A diagram illustrating what is meant by a SNP (single nucleotide polymorphism)*



The data can be represented in a table of SNP bases (A, T, C or G), with two states for each individual at each locus in diploid organisms (Table 1). The data shown in Table 1 are the base pairs on each haplotype of the sequence tag associated with the locus. The haplotypes are not phased with the reference and alternate alleles chosen arbitrarily.

***Table 3-1**. A table of genotypes for 10 individuals scored at 11 loci.*

	Ind 01	Ind 02	Ind 03	Ind 04	Ind 05	Ind 06	Ind 07	Ind 08	Ind 09	Ind 10
Locus 1	A/A	A/A	A/A	A/A	A/G	A/A	A/A	A/A	A/A	-/-
Locus 2	C/C	C/C	C/C	C/C	C/C	C/C	C/T	C/C	C/C	C/C
Locus 3	C/G	G/G	G/G	G/G	G/G	C/C	C/C	C/C	C/C	C/C
Locus 4	A/A	A/T	A/A	A/T	T/T	A/A	A/A	A/A	A/A	A/A
Locus 5	A/A	A/A	A/A	A/A	-/-	A/G	A/A	A/A	A/A	A/A
Locus 6	C/C	C/C	C/C	C/C	C/C	C/C	C/T	C/C	C/C	C/C
Locus 7	C/G	G/G	G/G	G/G	G/G	C/C	C/C	C/C	C/C	C/C
Locus 8	A/A	A/T	A/A	A/T	T/T	A/A	A/A	A/A	A/A	A/A
Locus 9	A/A	A/A	A/A	A/A	A/A	A/A	A/A	A/A	A/A	A/A
Locus 10	C/C	C/C	C/C	C/C	C/C	C/C	C/T	C/C	C/C	C/C
Locus 11	C/G	G/G	G/G	G/G	G/G	C/C	C/C	C/C	C/C	C/C

Alternatively, because the data are biallelic, it is computationally convenient to code the data as 0 for homozygotes for one allele, 1 for heterozygotes, and 2 for homozygotes of the other allele.

The reference allele is arbitrarily taken to be the most common allele, so 0 is the score for homozygous reference, and 2 is the score for homozygous alternate or SNP state. NA indicates that the SNP could not be scored (Table 2).

***Table 3-2.** A table of genotypes for 10 individuals with numerical scores at 11 loci.*

	Ind01	Ind02	Ind03	Ind04	Ind05	Ind06	Ind07	Ind08	Ind09	Ind10
Locus 1	0	0	0	0	1	0	0	0	0	NA
Locus 2	0	0	0	0	0	0	1	0	0	0
Locus 3	1	2	2	2	2	0	0	0	0	0
Locus 4	0	1	0	1	2	0	0	0	0	0
Locus 5	0	0	0	0	NA	1	0	0	0	0
Locus 6	0	0	0	0	0	0	1	0	0	0
Locus 7	1	2	2	2	2	0	0	0	0	0
Locus 8	0	1	0	1	2	0	0	0	0	0
Locus 9	0	0	0	0	0	0	0	0	0	0
Locus 10	0	0	0	0	0	0	1	0	0	0
Locus 11	1	2	2	2	2	0	0	0	0	0

Table 2 shows the form the data are stored in dartR, **though note that it departs from the coding arrangement used by DArT**.

Some sequence tags might contain more than one SNP, in which case they are likely to be closely linked when passed from parent to offspring. These may need consideration when preparing your data for analysis. Note that multiple SNPs occurring in the same sequence tag are each represented as a separate data record in the dataset.

The SNP data are provided in two forms by DArT, which will be described later.

SilicoDArT

As well as individuals varying in allelic composition at SNP sites, they can also vary at the restriction sites used to pull the representation from the genome. A mutation at one or both of the restriction sites will result in allelic dropout or null alleles. The presence or absence of particular sequence tags across individuals provides a source of information additional to the SNP data. DArT refers to these data as SilicoDArT markers.

Broadly, SilicoDArT markers can be considered analogous to AFLPs (Amplified Fragment Length Polymorphisms).

DArT provide this second dataset of presence (1) or absence (0) scores for sequence tags across individuals (Table 3). The filtering pipeline applied to

generate these data has been highly optimised for reliability, so do not be tempted to use the null alleles (missing data) present in the SNP dataset.

Table 3-3. *A table of SilicoDArT genotypes (presence/absence) for 10 individuals scored at 11 loci.*

	Ind01	Ind02	Ind03	Ind04	Ind05	Ind06	Ind07	Ind08	Ind09	Ind10
Locus 01	0	1	0	0	0	0	1	1	0	1
Locus 02	1	0	1	1	1	1	1	0	0	1
Locus 03	1	1	0	1	0	0	0	0	1	0
Locus 04	1	1	0	1	1	0	1	0	0	0
Locus 05	0	0	0	0	1	NA	1	0	1	0
Locus 06	1	1	1	0	1	1	1	0	1	1
Locus 07	1	1	NA	0	0	1	1	0	1	0
Locus 08	1	0	1	0	0	1	1	0	1	1
Locus 09	1	1	0	0	0	1	1	1	0	1
Locus 10	0	0	1	1	0	1	1	1	NA	1
Locus 11	0	1	1	0	1	1	0	0	1	1

Note that unlike the SNP data, NA represents a truly missing value, in that the state, presence or absence of the sequence tag could not be determined. Note also that the SilicoDArT dataset will include the presence or absence of sequence tags that do not contain a SNP.

Where have we come?

The above Session was designed to give you a very brief overview of the DArT pipelines for producing SNP and associated data. Having completed this Session, you should now be familiar with the following concepts.

- The concept of a SNP marker and how they are generated.
- The distinction between DArTSeq and SilicoDArT datasets.
- The coding used for SNP genotypes – 0 for homozygous reference, 2 for homozygous alternate, 1 for heterozygous, and NA for 'missing'.
- The coding used for SilicoDArT genotypes – 0 for absent, 1 for present, and NA for missing.

Session 2: Getting data into dartR

A sensible workflow

Let us begin by jumping the gun and defining a sensible pipeline for entering your data, as a context for the material in this and subsequent Sessions.

1. Examine the data provided by DArT in Excel to confirm that it conforms to expectations of the dartR package.

   For the SNP data, there needs to be a `AlleleID` column, and the start and end columns of the locus metadata need to be defined by the columns headed by asterisks. The row with the locus metadata labels needs to be the same row that holds the individual (= specimen or sample labels). This is usually the case, but some older datasets may need a little modification.

   For the SilicoDArt data, there needs to be a `CloneID` column and the locus metadata needs to be defined by the columns headed by asterisks.

2. Prepare the metadata associated with each individual. This dataset, stored in csv format, contains at a minimum the individual/specimen labels in a column headed `id`, and a population column that assigns individuals to groups or populations in a column headed `pop`. Other columns are optional but might include latitude, longitude of capture, sex, or other possible groupings or other information related to the individuals.

3. Read the data into dartR

We elaborate on this workflow in the sections that follow.

How dartR stores SNP data

The package dartR relies on the SNP data being stored in a compact form using a bit-level coding scheme. SNP data coded in this way are held in a genlight object defined in R package adegenet, described in ***Analysing genome-wide SNP data using adegenet 2.0.0*** by Thibaut Jombart and Caitlin Collins.

The complex storage arrangement of genlight objects is hidden from the user because it is accompanied by a number of "accessors". These allow the data to be accessed in a way similar to manipulating standard objects in R, such as lists, vectors and matrices.

A genlight object can be considered a matrix containing the SNP data encoded in a particular way (Figure 3). The matrix entities (rows) are the individuals, and the attributes (columns) are the SNP loci. In the body of this individual by locus matrix are the SNP data, coded as 0 for homozygous reference state, 1 for heterozygous, and 2 for homozygous alternate (or SNP) state.

Note: This coding is quite different from that used by DArT in their 1-Row and 2-Row csv files provided as part of their report.

Note also that a genlight object used by dartR differs in some important respects from the default genlight object of adgenet (a dartR genlight object is a superset of an adegenet genlight object). By this we mean that all functions in the adegenet package work on dartR genlight objects, but dartR genlight objects have

other essential components. So creating a genlight object to hold your data manually from a vcf or csv format requires a few steps in addition to importing the data to an adegenet genlight object.

Genlight objects not only contain SNP data, but also allow attachment of locus metadata to loci and of individual metadata to individuals/samples (Figure 3). The data and metadata are dataframes and handled sensibly by dartR following {adegenet} accessors [e.g. nLoc(), nInd(), pop()].

Locus metadata

The locus metadata included in the genlight object are those provided as part of your DArT report. These metadata are obtained from the DArT csv file when it is read into the genlight object. The locus metadata are held in an R data.frame that is associated with the SNP data as part of the genlight object.

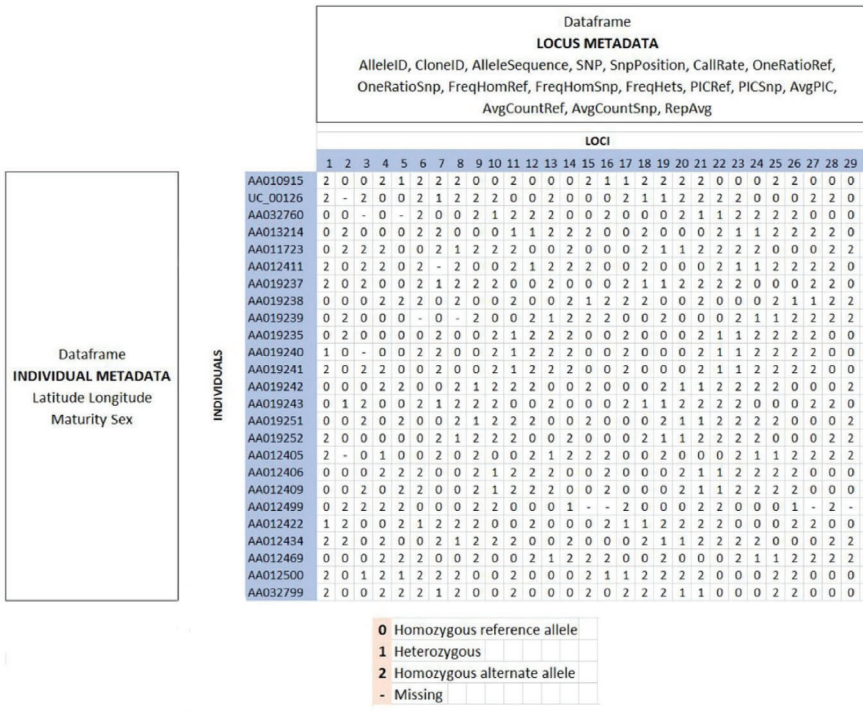


Figure 3-3. *A diagrammatic representation to illustrate the arrangment for storing data in a genlight object. The data are genotypes (0, 1, 2) in locus by individual matrix.*

The locus metadata would typically include:

SNP	the mutational change and its position in the sequence tag, referenced from zero
SnpPosition	position (zero is position 1) in the sequence tag of the defined SNP variant base
TrimmedSequence	(optional)The sequence containing the SNP or SNPs (the sequence tag), trimmed of adaptors.
CallRate	proportion of samples for which the genotype call is non-missing (that is, not "-")
OneRatioRef	proportion of samples for which the genotype score is 0
OneRatioSnp	proportion of samples for which the genotype score is 2

FreqHomRef	proportion of samples homozygous for the Reference allele
FreqHomSnp	proportion of samples homozygous for the Alternate (SNP) allele
FreqHets	proportion of samples which score as heterozygous, that is, scored as 1
PICRef	polymorphism information content (PIC) for the Reference allele
PICSnp	polymorphism information content (PIC) for the SNP
AvgPIC	average of the polymorphism information content (PIC) of the Reference and SNP alleles
AvgCountRef	sum of the tag read counts for all samples, divided by the number of samples with non-zero tag read counts, for the Reference allele row
AvgCountSnp	sum of the tag read counts for all samples, divided by the number of samples with non-zero tag read counts, for the Alternate (SNP) allele row
RepAvg	proportion of technical replicate assay pairs for which the marker score is consistent

In addition, dartR calculates the minor allele frequency and an estimate of read depth and stores them in the locus metadata.

These metadata variables are held in the genlight object as part of a data.frame called loc.metrics.

Depending on the report from DArT you may have additional, fewer or different `loc.metrics` (e.g. `TrimmedSequence` may be available only on request).

These metadata are used by dartR for various purposes, so if any are missing from your dataset, some analyses will not be possible. For example, `TrimmedSequence` is used to generate output for subsequent phylogenetic analyses that require estimates of base frequencies and transition and transversion ratios.

`AlleleID` is essential (with its very special format), and dartR scripts for loading your data sets will terminate with an error message if this is not present.

Individual metadata

Individual (=specimen/sample) metadata are user-specified, and do not come from DArT. Individual metadata are held in a second dataframe associated with the SNP data in the genlight object (Figure 3).

Two special individual metrics are:

- `id` Unique identifier for the individual or specimen that links back to the physical sample
- `pop` A label for the biological population from which the individual was drawn

Individual metrics are supplied by the user by way of a metadata file, provided at the time of reading the SNP data into the dartR genlight object. The metadata file

is a comma-delimited file, usually named `ind_metrics.csv` or similar, that contains labelled columns. The file must have a column named `id` that contains the individual (=specimen or sample labels) and a column named `pop`, which contains the populations to which individuals are assigned.

These special metrics can be accessed using:

```
pop(gl)
popNames(gl)
indNames(gl)
```

A number of other user-defined metrics can be included in the metadata file. Examples of user-defined metadata for individuals include:

- sex Sex of the individual (Male, Female)
- maturity Maturity of the individual (Adult, Subadult, juvenile)
- lat Latitude of the location of collection
- long Longitude of the location of collection

These optional data are provided by the user in the same metadata file used to assign id labels and assign individuals to populations at the time of reading in the data.

The individual metadata are held in the genlight object as a dataframe named `ind.metrics`.

Flags

The genlight object used by dartR has some additional information not normally accessed by the user. If these data are not in the genlight object, various functions may throw an error.

To ensure your manually generated genlight object (say, converted from a vcf file) is compliant, be sure to use

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

How dartR stores SilicoDArT data

dartR also stores SilicoDArT presence/absence data in a genlight object but distinguishes the data from SNP data by setting ploidy=1.

The locus metadata would typically include:

AlleleSequence	Sequence of the tag which is present in samples with genotype score "1"
TrimmedSequence	Same as the full sequence, but with removed adapters in short marker tags
AvgReadDepth	Sum of the tag read counts for all samples, divided by the number of samples with non-zero tag read counts.
StDevReadDepth	Standard deviation of the number of tag reads for all samples with non-zero tag read counts

CallRate	Proportion of samples for which the genotype call is either "1" or "0", rather than "-"
CloneID	Unique identifier of the sequence tag
OneRatio	Proportion of samples for which the genotype score is "1"
PIC	Polymorphism Information Content
Qpmr	Average of the normalised non-zero tag read counts divided by the standard deviation of the normalised non-zero tag read counts (If standard deviation is zero or near zero, the Qpmr value will be 100).
Reproducibility	Proportion of technical replicate assay pairs for which the marker score is consistent

Note that the locus metadata supplied by DArT may vary from service to service. The SilicoDarT data and associated metadata can be accessed in the same way as for SNP data, as described above.

Reading DArT files into a genlight object

SNP genotypes

SNP data can be read into a genlight object using `gl.read.dart()`. This function intelligently interrogates the input csv file to determine

- if the file is a 1-row or 2-row format, as supplied by DArT Pty Ltd.
- the number of locus metadata columns to be input before reading the SNP data themselves.
- the number of lines to skip at the top of the csv file before reading the specimen IDs and then the SNP data themselves.
- if there are any errors in the data.

An example of the function used to input data is as follows:

```
gl <- gl.read.dart(filename="sample_data_2Row.csv",
      ind.metafile="sample_metadata.csv")
```

The filename specifies the csv file provided by DArT, and the `ind.metafile` parameter specifies the csv file that contains metrics associated with each individual (id, pop, sex, environmental data, etc).

The `gl.read.dart` function

- transfers the SNP states to the body of the genlight object (as 0, 1, 2, NA);
- transfers the locus metadata (call rate, AvgPIC, etc) to the locus metadata dataframe (`gl@other$loc.metadata`) and calculates a few more;
- transfers individual metadata (plate, Specimen id, etc) to the individual metadata dataframe (`gl@other$ind.metadata`); and
- transfers additional individual metadata provided via the `ind.metafile` parameter to the individual metadata dataframe.

The resultant dartR genlight object contains the SNP genotypes, the individual metadata and the locus metadata.

SilicoDArT genotypes

SNP data can be read into a genlight object using `gl.read.silicodart()`. This function intelligently interrogates the input csv file to determine

- the number of locus metadata columns to be input (the first typically being `CloneID` and the last `Reproducibility`).
- the number of lines to skip at the top of the csv file before reading the specimen IDs and then the SNP data themselves.
- if there are any errors in the data.

An example of the function used to input data is as follows:

```
gl <- gl.read.silicodart(
     filename="sample_data_silicodart.csv",
     ind.metafile="sample_metadata.csv")
```

The filename specifies the csv file provided by DArT, and the `ind.metafile` parameter specifies the csv file that contains metrics associated with each individual (e.g. id, pop, sex, environmental data, etc).

The resultant genlight object contains the SilicoDArT presence/absence genotypes, the individual metadata and the locus metadata.

Reading non-DArT files into a dartR genlight object

If you are working with data that have not been prepared by DArT, you can still input the data to dartR provided you can get it into the appropriate format.

Unfortunately the vcf format is not a strict standard, so it has been difficult to capture all of the variations in our `gl.read.vcf()` script. Nevertheless, `gl.read.vcf()` should be your first port of call.

As with `gl.read.dart()`, the script requires the name of the data file (.vcf) and the name of an `ind.metafile` that contains the metadata for individuals.

```
gl <- gl.read.vcf(vcffile="Arabian_Oryx.vcf",
      ind.metafile="Oryx.ind.metadata.csv")
```

You can try this for yourself by navigating to and undertaking the Exercise on the Arabian Oryx, reading a vcf file.

If you do have difficulties, another method for inputting data from a vcf file to dartR is to extract the genotypes in A/T format (- for missing) as a comma-delimited `.csv` file, then prepare two additional `.csv` files, one with the locus metrics and one with the individual metrics.

The locus metrics should include only those locus attributes that cannot be calculated from the data. The locus names should correspond to those in the genotype matrix. Similarly, the individual names should correspond to those in the genotype matrix.

The three files can then be read into dartR with

```
gl.read.csv(filename=<genotype csv name>,
     ind.metafile=<individuals csv name>,
     loc.metafile=<locus csv name>, verbose=3)
```

Session 3: Getting data out of dartR

Saving a genlight object

Reading the data in from an Excel spreadsheet and converting to a genlight object takes a lot of computation, and so time. You will also have done some tidying up of the data. It is sensible to save your genlight object in binary form using

```
gl.set.wd(getwd())
gl.save(gl,file="tmp.Rdata")
```

and then read it in again with

```
gl.new <- gl.load("tmp.Rdata")
```

Saving genotypes as a csv file

The function

```
write.csv(gl,"genotypes.csv"))
```

will write the genotypes to a `csv` file with the loci as column headings and the individuals as row headings.

Exporting data for use by third party software

Package dartR has options for exporting a genlight object to a format for input to third-party software packages. Examples include:

`gl2plink()`	Exports a genlight object into PLINK format. PLINK is a widely used, open-source software toolset for genome-wide association studies (GWAS) and population-based genetic analyses.
`gl2genepop()`	Exports a genlight object into GENEPOP format. GENEPOP is a software package designed for population genetics analyses. It performs tests for Hardy–Weinberg equilibrium, linkage disequilibrium, genetic differentiation, and calculates F-statistics from allele frequency data, typically for microsatellite or SNP markers.
`gl2faststructure()`	Exports a genlight object into fastSTRUCTURE format. fastSTRUCTURE is software designed to infer population structure from large-scale genotype (SNP) data.
`gl2fasta()`	Exports the trimmed sequence tags or SNPs into standard fastA format.

`gl2phylip()` Calculates a Euclidean distance matrix from the genotypes and exports the matrix in a form suitable for the Phylip distance phylogeny analyses.

A full list of conversion functions can be obtained by typing

`??gl2` (without a return)

and examining the pulldown menu.

Session 4: Worked Examples

Background

Here we introduce a dataset that will be used in the worked examples in this and following chapters. It is drawn from studies of the Canberra grassland earless dragon (*Tympanocryptis lineata*), a critically endangered species that is now found in a few remaining patches of lowland natural temperate grassland in the Australian Capital Territory. They are relatively sedentary, moving up to 100 metres in six weeks and have home ranges between 0.1 and 0.4 hectares. One would expect this low rate of movement and presumably dispersal to be reflected in the genetic structure of the species across isolated patches.

The ACT Government provides a **link to the information on the Canberra earless dragon**, including conservation planning at Territory and National level.

Montoring of the species showed a rapid decline in numbers between 2002 and 2006, a decline that has continued to the point of bringing the species to the brink of extinction. A study of genetic diversity is among the many initiatives to build the knowledge necessary to stem and reverse this decline. Are the sub-populations of the species in their isolated remnant patches losing genetic diversity to a point where they will find it difficult to bounce back even with intervention, and what strategies can be adopted to maximise genetic diversity in each of the declining sub-populations?

These are big questions. Answering them is beyond the scope of this introductory eBook. Our objective here is to use the data collected between 2002 and the present to illustrate, by way of worked examples, the techniques within the scope of this eBook. We start in Chapter 3 by pulling the data into dartR, examining the data, and outputting the data in various forms.

The data used in this Worked Example were collected as research by PhD student Ryan Collie and are unpublished at the time of writing. We have modified the data heavily for the sake of illustration, so they should not be used for any other purpose. If you wish to access the original data, contact the originators directly (Bernd Gruber).

Setting up

Open RStudio. To keep things organised, a first step is to set up a working directory and an RStudio project as as outlined in the Workflow. Use a sensible

name that you will remember, like `eBook1`. You only need to do this once as described in Chapter 1.

Set the global verbosity to 3 so you get to see the full progress log when you submit commands.

```
gl.set.verbosity(3)
```

To access the Worked Examples, use

```
install.packages('eBook_Startup')
```

You only have to do this once. Navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*.

Worked Example 3-1: SNP Data

This worked example on the earless dragon illustrates how to input data into dartR, some basic functions for assessing whether the input was successful, and how to save the data in compact binary form for later use.

Navigate to the Worked Examples for Chapter 3.

Once you have input the data and are happy with it, you will be asked to save it in binary form for use in the next chaper.

Exercises

We have held you by the hand through the Worked Examples. It is time to integrate what you have learned with some exercises on input, storage in defined data structures and output of SNP and SilicoDArT data.

Navigating to the RStudio Tutorial Pane and then opening the *eBook Getting Started with dartR*. Navigate to the Exercises for Chapter 3.

Winding up

Where have we come?

In this Session, we covered a range of topics on data entry, storage of data and some preliminary approaches to examining those data. Having completed the Session, you should understand

- What is a sensible pipeline for preliminary handling of your SNP data.
- How a genlight object is organised in terms of the SNP scores (which are different from the scores used by DArT) and how locus and sample metadata are associated with the genotypes.
- The different types of locus metadata generated by DArT, and how to look up what each metric means.
- How to read data from DArT into a genlight object.
- How to read data generated independently from DArT into a genlight object.
- How to interrogate the locus and individual (specimen/sample) metadata.

- How to save a genlight object (genotypes, individual metadata, locus metadata) into compact binary form for later use.
- How to output data for use by third party software.

Chapter 4

Data Manipulation

Contents

Preamble

Learning Outcomes

The objective of this chapter is to introduce you to the options available for manipulating your data after you have loaded them in into dartR.

After completing this chapter, you should have a basic understanding of the fundamentals of the darR functions for deleting populations, individuals and loci, recoding populations and individuals, assigning individuals to populations, merging populations, renaming populations, and other routine tasks.

Most importantly, you should be empowered to expand your knowledge beyond the basics presented here by exploring the help on these topics and associated scripts, and by applying the knowledge in other tutorials to come.

Prerequisites

RStudio is an integrated development environment (IDE) with a graphical user interface (GUI) used to manage R workflow and so to work through this chapter, you will need to have already completed *Chapter 1. A primer on RStudio.* For more advanced treatment refer to the **official documentation**.

You should have completed Chapter 3 on data entry and data structures before attempting this Chapter.

Artificial intelligence tools are invaluable these days for assisting with R code and providing summary information on topics relevant to this Chapter, and you should become familiar with at least one such tool. We recommend **Claude AI**.

You may also find it helpful to listen to the **podcast summary** of the overview material in this chapter.

Workflow

We assume that you have established the RStudio project `eBook1` and downloaded the datasets required for all Chapters. Package `eBook_Startup` has been installed. All ready to go.

The workflow from this point again follows a formula.

- **Theory**, unencombered by competing options and technical asides;
- **Worked Examples** where you are led through analyses with sample code;
- **Exercises** where you are left on your own to integrate the knowledge you have gained in a problem-solving context.
- **Review** the learning outcomes and the section on Where have we come? to confirm that you are moving to the next chapter with all that is required under your belt.

The only difference with Chapter 4 is that it cycles through these elements twice, once in a sesson on dartR reports and one in a session on manipulating data.

Session 1: Interrogating a genlight object

Revision

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

`nLoc()`	number of loci
`locNames()`	list of loci
`nInd()`	number of individuals (specimens or samples)
`indNames()`	list of individuals
`nPop()`	number of populations
`popNames()`	list of populations
`pop()`	list of population assignments for each individual
`table(pop())`	generate a table of sample size by population
`m <- as.matrix()`	generate a matrix of the SNP scores, with 0 as homozygous reference, 2 as homozygous alternate, and 1 as heterozygous.

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

Report Functions

We covered some of the above options for interrogating a SNP dataset in dartR in Chapter 3. In this Chapter 4, we will carry this beyond the basic interrogation with genlight accessors to introduce some of the report functions in dartR.

These report functions are particularly important because they inform analyses down the track, particularly the filtering functions described in Chapter 5. Indeed, they come in pairs, each report function having a corresponding function that applies a filter. The report function provides the information necessary for a judgement on the threshold to use for filtering.

The report functions typically apply to both SNP and SilicoDArT datasets. The common report functions that are matched with filtering functions are listed below.

`gl.report.callrate()`	summarises CallRate values
`gl.report.reproducibility()`	summarises repAvg (SNP) or reproducibility (SilicoDArT) values
`gl.report.monomorphs()`	provides a count of polymorphic and monomorphic loci
`gl.report.secondaries()`	provides a count of loci that are secondaries, that is, loci that reside on the one sequence tag.
`gl.report.rdepth()`	reports the estimate of average read depth for each locus

`gl.report.hamming()`	reports on Hamming distances between sequence tags
`gl.report.overshoot()`	reports loci for which the SNP has been trimmed along with the adaptor sequence
`gl.report.taglength()`	reports a frequency tabulation of sequence tag lengths
`gl.report.sexlinked()`	reports the number and type of loci likely to be in sex chromosomes

A full list of reporting functions can be obtained by typing

```
??gl.report
```

and examining the pulldown menu.

Example: Reporting on Call Rate

As an example, let us consider reporting on call rate across loci. Recall that call rate is the rate at which individuals have been successfully called for a locus, or that loci have been called for an individual depending on the context.

```
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 insist on retaining only those loci that have successfully been called across all individuals

(setting threshold=1.0), you will retain only 7 loci with the loss of 97.3% of scored loci. That is way too stringent. However, a threshold of 0.95 might be appropriate, with 54.9% of loci retained.

The plot can make the decision simpler.

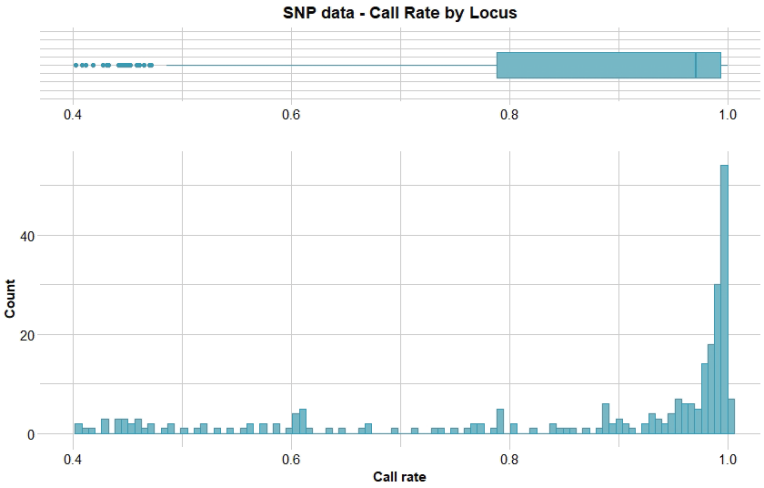


Figure 4-1. Standard output of the report callrate function showing the distribution of successful calls across loci. This enables an assessment of the quality of the run.

Each report function prints out a table of thresholds with the accompanying losses should that threshold be used to filter and a frequency distribution plot.

The report functions provide quality estimates for your dataset and allow you to determine whether remedial action is required.

Smear plot

An alternative way of visualizing the call rate across individuals and loc is to examine a smear plot. In the plot generated by `gl.smearplot(testset.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 exceptionally low call rates, and some individuals have a poor call rates for some loci (horizontal white bands). One would typically examine a smear plot before and after any manipulations to evaluate the impact of remedial action.

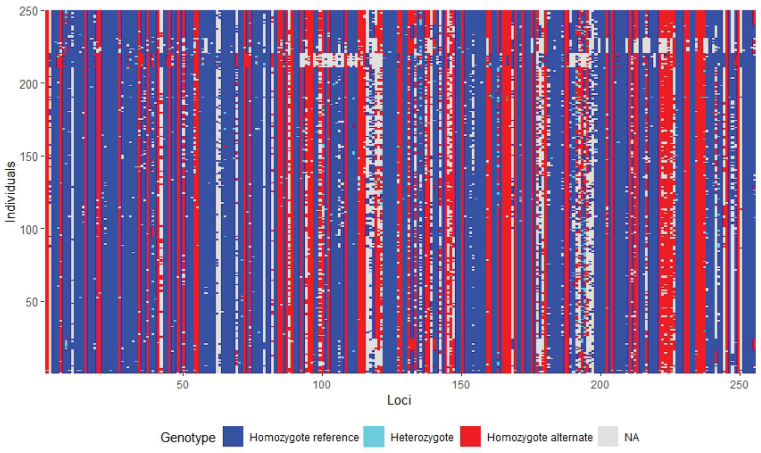


Figure 4-2. A smear plot showing the SNP state for loci (x axis) and individuals (y axis). A key consideration is the pattern of missing values (NA) which provides an indication of where remedial action might be required.

Worked Example 4-1: Basic Attributes and QC

This worked example uses the earless dragon dataset we input and then saved in Chapter 3. In this example, we illustrate the application of functions to provide a basic summary of the contents of the dataset, application of the standard adegenet accessors, and application of the dartR report functions.

Open the RStudio project you set up in Chapter 3, namely `eBook_Project`.

Set the global verbosity to 3 so you get to see the full progress log when you submit commands.

You will be asked to load the earless dragon dataset that was created in Chapter 3, namely `tympo_SNP_raw.RData`.

To access the Worked Examples, navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*.

Undertake Worked Example 4-1.

Exercises

Now is a good time to try some exercises and to try out what you have learned so far on your own data. These exercises ask you analyse a small SNP dataset and a small SilicoDArT dataset to see what you have learned so far.

Navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*. Navigate to the Exercises for Chapter 4. Undertake the first two exercises.

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 familiarising yourself with your dataset at the most fundamental level, for assessing the quality of the SNP and SilicoDArT calls and for detecting any errors or outliers likely to cause problems in later analyses.

Session 2. Data Manipulation

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 that are perhaps duplicated, and recalculating the locus metrics to include 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 lead to serious miscalculations. 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 feature not included 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, Renaming and Creating Populations

Dropping populations

The following set of functions applies to populations and individuals.

`gl <- gl.drop.pop()` remove listed populations

`gl <- gl.keep.pop()` keep only the listed populations

Merging and renaming a population

You can merge two populations or rename a population, using

`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.

Reassigning a population using an individual metric

It may also be convenient to **permanently** 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. If all individuals in an existing population are reassigned, the original population will be dropped.

Bulk population reassignment or deletion [Recode Tables]

An alternative way to manipulate population assignments is to use a recode table in the form of a comma-delimited csv file. This is efficient if you wish to make lot of changes.

With a recode table you can rename, merge and delete populations. The recode table is a csv file that can be set aside as part of your preliminary workflow to ensure reproducibility of your initial manipulations.

Note that populations or individuals assigned the label 'Delete' will be removed from the genlight object when the recode table is applied.

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

An interactive method of effecting population changes can be achieved with

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

Subsampling populations

There may be occasions when you want to subsample populations in a genlight object containing SNP or SilicoDArT data. This can be done, with or without replacement, using

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

This option is particularly useful when used in combination with population assignment, for example:

```
lat <- as.character(latitude)
long <- as.character(longitude)
latlong <- paste0(lat,long)
gl2 <- gl.subsample.pop(gl, n=100 as.pop=latlon,
      replacement=FALSE)
```

Object `gl2` will contain a subsample of 100 randomly-selected locations in the dataset.

Removing and Renaming Individuals

Keeping and dropping individuals

The following set of functions applies to populations and individuals.

`gl <- gl.drop.ind()` remove listed individuals

`gl <- gl.keep.ind()` keep only the listed individuals

Bulk removal of individuals using an individual metric

You can select individuals to be kept or dropped based on one of the 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.

Reassigning individuals to a new population

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

`gl <- gl.define.pop()` create a new population for listed individuals; their old population assignments will be discarded.

Bulk individual reassignment or deletion [Recode Tables]

An alternative way to manipulate 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, as might occur when the genotype names submitted to the sequence provider are different or abbreviated labels for the individuals.

The recode table is a csv file that can be set aside as part of your preliminary workflow to ensure reproducibility of your initial manipulations.

Note that individuals assigned the label 'Delete' will be removed from the genlight object when the recode table is applied.

`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.ind()`	apply the specified ind.recode table to the individuals
`gl <- gl.edit.recode.ind()`	edit individual assignments, and apply the changes on closure

Subsampling individuals

There may be occasions when you want to subsample individuals in a genlight object containing SNP or SilicoDArT data. This can be done, with or without replacement, using:

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

Removing and Renaming Loci

Keeping and dropping loci

The following set of functions applies to populations and individuals.

`gl <- gl.drop.loc()` remove listed loci

`gl <- gl.keep.loc()` keep only the listed loci

Bulk locus reassignment or deletion [Recode Tables]

An alternative way to manipulate 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, as might occur when the genotype names submitted to the sequence provider are different or abbreviated labels for the individuals.

The recode table is a csv file that can be set aside as part of your preliminary workflow to ensure reproducibility of your initial manipulations.

Note that individuals assigned the label 'Delete' will be removed from the genlight object when the recode table is applied.

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

Subsampling loci

There may be occasions when you want to subsample loci in a genlight object containing SNP or SilicoDArT data. This can be done, with or without replacement, using

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

Recalculating locus metrics

Several scripts delete individuals, and so many of the metadata variables provided by Diversity Arrays Technology or your service provider no longer apply. For example, deleting individuals with a low call rate across loci will alter the call rate for each locus and so the `CallRate` for loci provided initially by DArT or your service provider will be incorrect in the new genlight object.

There are parameters to set in each of the relevant dartR scripts to recalculate the locus metadata. Flags are set as to whether the locus data are no longer correct, and action taken to update them when needed. So dartR looks after this behind the scenes.

However, if you are going to access the locus metadata directly in your workflows, be sure to do your manipulations and then run

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

Some functions will generate monomorphic loci. These are not deleted automatically because the need to do so depends on subsequent analyses. Instead a warning is issued. You can then choose to delete these monomorphic loci using

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

Refer to Chapter 5 for further details.

Worked Example 4-2: Manipulating data

This worked example will run through the application of functions used to manipulate data prior to more serious analyses. The genlight object contains data from a real-world example, the earless grassland dragon.

To access the Worked Examples, navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*.

You will be asked again to load the earless dragon dataset that was created in Chapter 3, namely `tympo_SNP_raw.RData`.

Undertake Worked Example 4-2.

Exercises

Now is a good time to try some more of what you have learned so far on some exercises and your own data. These exercises ask you analyse a small SNP dataset and a small SilicoDArT dataset to see what you have learned so far.

The Exercises are also found by navigating to the RStudio Tutorial Pane and then opening the *eBook Getting Started with dartR*. Navigate to the Exercises for Chapter 4.

Undertake the remaining exercises under Chapter 4.

Winding up

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:

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

Ende

Chapter 5

Filtering

Contents

Preamble

Learning Outcomes

The objective of this chapter is to introduce you to the options available for filtering your data based on an assessment of quality using the report functions introduced in Chapter 3.

After completing this chapter you should have a basic understanding of the fundamentals of the darR functions for filtering SNPs and SilicoDArT calls based on such criteria as call rate, reproducibility, read depth, monomorphic loci and many other attributes. In some cases, individuals may be filtered.

Most importantly, you should be empowered to expand your knowledge beyond the basics presented here by exploring the help on these topics and associated scripts, and by applying the knowledge in your work.

Prerequisites

RStudio is an integrated development environment (IDE) with a graphical user interface (GUI) used to manage R workflow and so to work through this chapter, you will need to have already completed *Chapter 1. A primer on RStudio.* For more advanced treatment refer to the **official documentation**.

You should have completed Chapter 3 on data entry and data structures and Chapter 4 on data manipulations before attempting this Chapter.

Artificial intelligence tools are invaluable these days for assisting with R code and providing summary information on topics relevant to this Chapter, and you should become familiar with at least one such tool. We recommend **Claude AI**.

You may also find it helpful to listen to the **podcast summary** of the overview material in this chapter.

Workflow

We assume that you have established the RStudio project `eBook1` and downloaded the datasets required for all Chapters. Package `eBook_Startup` has been installed. All ready to go.

The workflow from this point again follows a formula.

- **Theory**, unencumbered by competing options and technical asides;
- **Worked Examples** where you are led through analyses with sample code;
- **More Theory**, this time covering caveats and nuances.
- **Exercises** where you are left on your own to integrate the knowledge you have gained in a problem-solving context.
- **Review** the learning outcomes and the section on Where have we come? to confirm that you are moving to the next chapter with all that is required under your belt.

Session 1: Basic Filtering

Overview

Calling SNPs in genotyping by sequencing is not an exact science. Judgements need to be made at various points in the workflow to increase the likelihood of a correct call at a particular locus. Diversity Arrays Pty Ltd (DArT) has already done much of the filtering of the sequences used to generate your SNPs that would normally be undertaken by researchers who generate their own ddRAD data. Here we present some other filters that you might wish to apply to increase the reliability of the data retained in your SNP or SilicoDArT dataset.

For each filter parameter offered by dartR, we provide both a report function and a filtering function (e.g. `gl.report.reproducibility` is associated with `gl.filter.reproducability`). The reporting function provides descriptive statistics for the parameter of choice. Inspection of these outputs will assist you to identify appropriate filter thresholds. Hence, as a standard workflow, it is a good idea to run the `gl.report` functions in advance of applying each `gl.filter` function to provide a foundation for selecting thresholds.

Several filters are available to improve the quality of the data represented in your genlight object. Some of these are specific to Diversity Arrays data (e.g. `gl.filter.reproducibility`), but most will work on any data provided they are in the appropriate genlight format.

The basic ones, in no particular order, are:

`gl <- gl.filter.reproducibility()`	filter out loci for which the reproducibility (strictly repeatability) is less than a specified threshold, say threshold = 0.99
`gl <- gl.filter.callrate()`	filter out loci or individuals for which the call rate (rate of non-missing values) is less than a specified threshold, say threshold = 0.95
`gl <- gl.filter.monomorphs()`	filter out monomorphic loci and loci that are scored all NA
`gl <- gl.filter.allna`	filter out loci that are all missing values (NA)
`gl <-gl.filter.secondaries()`	filter out SNPs that share a sequence tag, except one retained at random [or the best based on reproducibility (RepAvg) and information content (AvgPIC)].
`gl <- gl.filter.hamming()`	filter out loci where the corresponding sequence tags differ from each other by less than a specified number of base pairs

`gl <- gl.filter.rdepth()`	filter out loci with exceptionally low or high read depth (coverage)
`gl <- gl.filter.taglength()`	filter out loci for which the tag length is less that a threshold
`gl <- gl.filter.overshoot()`	filter out loci where the SNP location lies outside the trimmed sequence tag
`gl <- gl.drop.sexlinked()`	filter out loci that are likely to be in sex chromosomes

The order of filtering can be important and requires some thought. As an example, filtering on call rate by individual before filtering on call rate by locus or choosing the alternative order will depend on the weight placed on losing individuals versus losing loci. The filtering steps may also change depending on the analysis you intend to run. There is no one-fits-all approach to SNP filtering. Refer to Session 2.

An Example: Filtering on reproducibility

As an additional quality control measure, a selection of samples is processed twice by DArT using independent adaptors from DNA through to allelic calls. The consistency of scoring across these technical replicates is referred to as reproducibility. Let's examine the distribution of reproducibility measures (RepAvg) in our dataset.

```
gl.report.reproducibility(testset.gl,plot=TRUE)
```

```
Starting gl.report.reproducibility
Processing SNP data
Reporting Repeatability by Locus
No. of loci = 255
No. of individuals = 250
Minimum      :  0.959459
1st quartile :  1
Median       :  1
Mean         :  0.9981525
3r quartile  :  1
Maximum      :  1
Missing Rate Overall:  0.12

Quantile Threshold Retained Percent Filtered Percent
1      100% 1.0000000      214    83.9       41    16.1
2       95% 1.0000000      214    83.9       41    16.1
3       90% 1.0000000      214    83.9       41    16.1
4       85% 1.0000000      214    83.9       41    16.1
5       80% 1.0000000      214    83.9       41    16.1
6       75% 1.0000000      214    83.9       41    16.1
7       70% 1.0000000      214    83.9       41    16.1
8       65% 1.0000000      214    83.9       41    16.1
9       60% 1.0000000      214    83.9       41    16.1
10      55% 1.0000000      214    83.9       41    16.1
11      50% 1.0000000      214    83.9       41    16.1
12      45% 1.0000000      214    83.9       41    16.1
13      40% 1.0000000      214    83.9       41    16.1
14      35% 1.0000000      214    83.9       41    16.1
15      30% 1.0000000      214    83.9       41    16.1
16      25% 1.0000000      214    83.9       41    16.1
17      20% 1.0000000      214    83.9       41    16.1
18      15% 0.9976873      216    84.7       39    15.3
19      10% 0.9945940      229    89.8       26    10.2
20       5% 0.9883732      242    94.9       13     5.1
21       0% 0.9594590      255   100.0        0     0.0

Completed: gl.report.reproducibility
```

This output is useful in that it provides an indication of how much data will be lost, when filtering, for each choice of a threshold value. Clearly, with a minimum reproducibility of 0.96 and a maximum of 1 across loci, we can be fairly stringent in our choice of a threshold without shedding many loci. A value of 0.99 will not result in the loss of much data (16.1%). The judgement can also be made on the basis of the graphical output.

You may be tempted on the basis of this graph to filter very stringently on a threshold of 1 (perfect reproducibility). The DArT team recommend against this as it could potentially introduce an ascertainment bias.

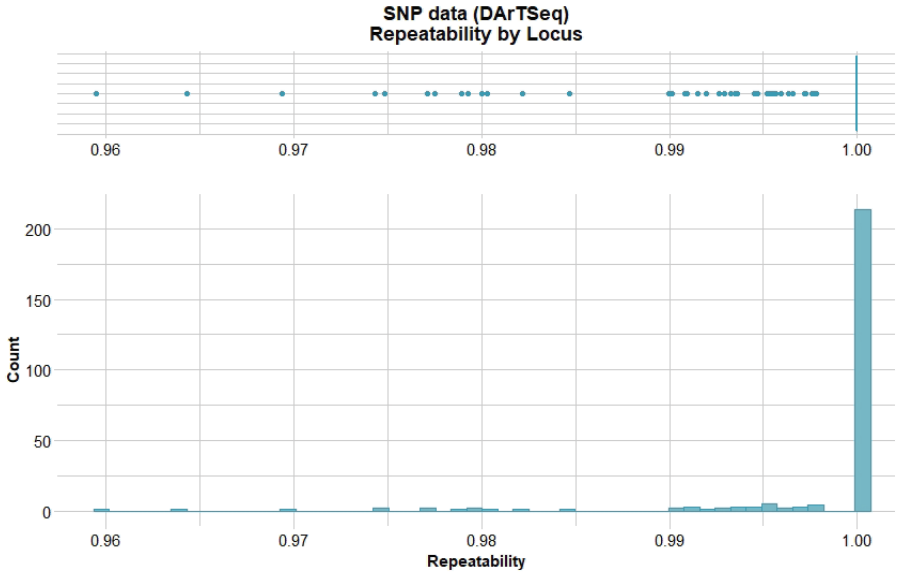


Figure 5-1. Standard graphic produced by the report reproducibility function showing the quality of the SNP loci based on technical replicates.

We now filter by selecting a threshold value of 0.99.

```
gl <- gl.filter.reproducibility( testset.gl,
         threshold=0.99, verbose = 3)
```

```
Starting gl.filter.reproducibility
  Processing a SNP dataset
  Identifying loci with repeatability below : 0.99
  Removing loci with repeatability less than 0.99

  Summary of filtered dataset
  Retaining loci with repeatability >= 0.99
  Original no. of loci: 255
  No. of loci discarded: 14
  No. of loci retained: 241
  No. of individuals: 250
  No. of populations:  30

Completed: gl.filter.reproducibility
```

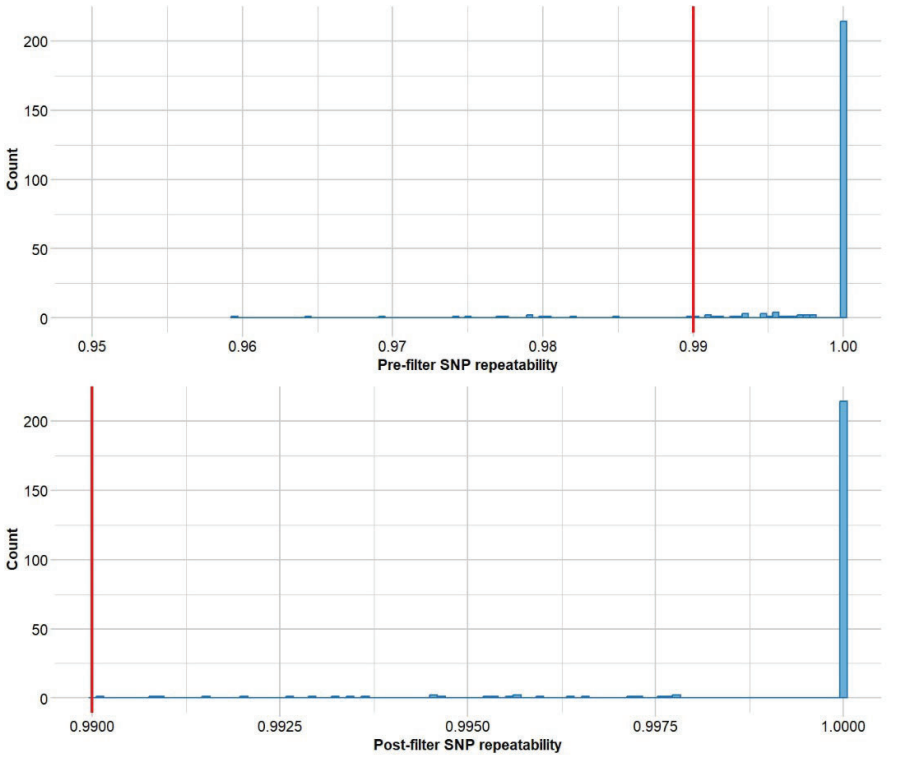


Figure 5-2. Standard graphic produced by the filter on reproducibility function showing the quality of the SNP loci based on technical replicates before and after filtering.

Only 14 loci out of 255 were removed.

Worked Examples 5-1 – Basic Filtering

This worked example uses the earless dragon dataset we input and then saved in Chapter 3. In this example, we illustrate the application of functions filtering SNP and SilicoDArT data.

Open the RStudio project you set up in Chapter 3, namely `eBook_Project`.

Set the global verbosity to 3 so you get to see the full progress log when you submit commands.

You will be asked to load the earless dragon dataset that was created in Chapter 3, namely `tympo_SNP_raw.RData`.

To access the Worked Examples, navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*.

Undertake Worked Example 5-1.

Where have we come?

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

- Filter on call rate, reproducibility, secondaries, read depth, sex linkage and hamming distance.

- Recalculate locus metrics after deleting individuals or populations as part of the filtering process.
- Filter out resultant monomorphic loci.

Having played with the various filters, you should also have a good appreciation of how to use the `gl.report` functions to select appropriate thresholds for filtering and how to introduce a sequence of filtering steps into your workflows.

Session 2: Nuances of Filtering

dartR is designed to steamline workflows and analyses. This does not relieve the user from applying judgement based on sound biological and scientific understanding. Analyses rest on assumptions, models and limitations. This applies to filtering. There is no one recipe for applying filters to SNP and SilicoDArT data. Context is important, both in terms of the scenario in which the data were collected and plans for future analysis. This section attempts to alert you to the considerations that are required in pre-filtering using dartR.

Pre-dartR Filtering

Sequences generated by DArT are processed using proprietary analytical pipelines before the report (containing the SNP and SilicoDArT markers) is provided to the client and the data are passed to dartR.

In the DArT workflow, poor quality sequences are first filtered, applying more stringent selection criteria to the barcode region than to the rest of the sequence (minimum barcode Phred score 30, pass percentage 75; minimum whole-read Phred score 10, pass percentage 50). In that way, assignment of the sequences to specific samples in the sample disaggregation step is very reliable. In a report based on full sequence volume (high), approximately 2,500,000 (±7%) sequences per sample are identified and used in marker calling.

These sequences are truncated to 69 bp (including some adaptor sequence where the fragments were shorter than 69 bp) and aggregated into clusters by the DART fast clustering algorithm. A Hamming distance threshold of 3 bp is used as a threshold for identifying distinct sequence, taking advantage of the fixed fragment length.

The sequences are then error-corrected using an algorithm that corrects a low-quality base (Phred score <20) to a corresponding high-quality singleton tag (Phred score >25); where there is more than one distinct high-quality tag, the sequence with the low-quality base is discarded.

Identical sequences are then collapsed. These error-corrected sequences are analysed using DART software (DArT GS14 pipeline) to output candidate SNP markers. SNP markers are identified within each cluster by examining parameters calculated for each sequence across all samples – primarily average and variance of sequencing depth, the average counts for each SNP allele and the call rate (proportion of samples for which the marker is scored).

Where three sequences survive filtering to this point, the two variants with the highest read depth are selected. The final average read depth per locus depends on the genome size, the degree of representation delivered by the chosen restriction enzymes and the sequencing intensity (sequence volume).

As an additional quality control, a selection of samples (30 to 100% of samples depending on the number of plates submitted) is processed twice using independent adaptors from DNA through to allelic calls. These are technical replicates. Scoring consistency (reproducibility or more strictly, repeatability) is used as the main selection criterion for high quality/low error rate markers.

The DArT analysis pipelines (DS14) have been tested against hundreds of controlled crosses to verify the Mendelian behaviour of the resultant SNPs, and the parameter thresholds have been fine tuned as part of their commercial operations.

The bottom line is that, although analytical pipelines are proprietary and commercial in confidence, a substantial amount of filtering of sequence tags is undertaken before you receive the SNP and SilicoDArT markers and report from DArT, filtering that ensures quality outcomes in terms of the veracity of the resultant SNPs.

Package dartR picks up the analysis at this point. The additional filtering that you may choose to undertake using dartR depends upon the research questions and other considerations as outlined below.

Stacks Pre-Filtering

A full protocol for calling SNPs from reduced representation data has been published -- Rochette, N. & Catchen, J. (2017). Deriving genotypes from RAD-seq short-read data using Stacks. Nature Protocols, 12(12), 2640–2659. Please refer to that article and the **Stacks V2 Manual**.

Package dartR picks up the analysis after the SNPs have been called in Stacks. The additional filtering that you may choose to undertake using dartR depends upon the research questions and other considerations as outlined below.

Filtering Order

It is often unclear as to what order the filtering steps should be performed in your dartR workflow. Does one filter on call rate by individual first, then call rate by locus, or the other way around? There is no correct answer to this, as it depends on whether you value retaining loci over retaining individuals. Usually, you would not want to lose individuals from your dataset, so filtering out those loci with low call rates would come first, filtering on individuals second, though this is not always the case. A starting point for considering a workflow might be:

1. Filter on sex linkage.
2. Optionally filter on read depth if the Diversity Arrays threshold is considered too lenient.
3. Filter out loci with a reproducibility below a particular threshold (say 0.98).
4. Filter out loci with a call rate below a particular threshold (say 0.95).

5. Filter out individuals with a call rate below a particular threshold (say 0.80).
6. Filter out secondaries (all but one SNP retained per sequence tag).
7. Optionally filter for monomorphs created by the removal of individuals.

An example of a filtering sequence might be

```
gl <- gl.filter.sexlinked(gl, system="xy")
gl <- gl.filter.secondaries(gl)
gl <- gl.filter.rdepth(gl)
gl <- gl.filter.reproducibility(gl)
gl <- gl.filter.callrate(gl, method="loc")

gl <- gl.filter.callrate(gl, method="ind")
gl <- gl.filter.monomorphs(gl)
```

Filtering Thresholds

Strategy

The take-home message from this section is to be careful not to over-filter. The objective is to achieve an appropriate balance between signal-to-noise ratio and not eliminating noise altogether at the expense of losing some signal. This balance will depend on sequence intensity (volume in the context of genome size), sample quality and most importantly, downstream applications.

There is no right answer. **A good strategy is to undertake an exploratory analysis first, whereby you experiment with filtering**. Perhaps start with very stringent filtering, examine the impact on the final analysis, then progressively relax the stringency to select the minimal filtering regime that still delivers a stable outcome. Play with your data to get a feel for the balance between signal to noise ratio, in the context of the analyses you propose to subsequently undertake.

What is sequence volume?

Three services are routinely offered by DArT, differing in the volume of sequence generated. For genomes larger than 1 Gbp, a sequence volume of 2.5 million is recommended (high). For genomes ranging from 300 Mbp to 1 Gbp, a sequence volume of 1.2 million is recommended (medium). For genomes less than 300 Mbp, a sequence volume of 800,000 is recommended (low).

The average read depth of the sequence tags depends upon the genome size, the sequencing volume (low, medium, high), the proportion of the genome that is represented after the restriction enzymes are applied and after size selection, and the proportion of mis-targeted sequence in the sample (e.g. arising from bacterial contamination). Average read depth per sequence tag is calculated by dartR using `gl.report.rdepth()`.

Average read depth can be obtained using:

```
mean(gl@other$loc.metrics$rdepth)
```

This is reported also by `gl.report.basicstats()`.

Filtering on Reproducibility

As mentioned above, a proportion of samples is processed twice, using independent adaptors, from DNA through to allelic calls. These are technical replicates and their scoring consistency (repeatability) is used as the main selection criterion for high-quality/low error-rate markers. Reproducibility values may depend on the number of plates run in a particular service, because the proportion of samples selected for technical replication will vary (usually 60-70%, but as high as 100% (partial plate) or as low as 30%).

While it is tempting to filter on 100% reproducibility, this is typically over-kill. One needs to appreciate that the reproducibility of technical replicates can depend on the quality of the input DNA. If it is contaminated, say with bacterial DNA or a blood parasite, then the target sequence volume can be dramatically reduced, the error-rate inflated and the reproducibility of technical replicates across samples for a locus substantially affected. Furthermore, the presence of inhibitors can affect the efficiency of restriction enzymes in a context specific way, again leading to inflation of the error rate and systematic differences between technical replicates.

These factors, contamination and presence of PCR inhibitors, have a greater influence on heterozygous than homozygous sites, and so differentially affect loci that are more polymorphic than others. Furthermore, experience shows that 90% of the resultant error rates occur in 10% of samples; filtering too stringently will lead to loss of loci that are otherwise reproducible for 90% of samples.

DArT has already used technical replicates in their earlier pipelines to select reliable markers to yield a basic level of reproducibility in the data provided to you.

So by all means filter on reproducibility, but consider the consequential loss of loci and do not filter so stringently as to decimate your locus count, as this can lead to systematic biases and unnecessary loss of informative loci. You should carefully consider the plot of the distribution of reproducibility values in the exploratory analysis (`gl.report.reproducibility`).

Filtering on Call Rate

When filtering loci on Call Rate, one needs to be aware that failure to call a SNP at a given locus is typically a result of a mutation at one or both of the restriction sites. That is, a missing value typically represents a null allele, captured in the companion SilicoDArT dataset. In the SNP dataset, highly polymorphic regions of the genome, those with an abundance of informative SNPs, are also most prone to null alleles. Hence, SNPs in highly polymorphic regions will be differentially eliminated by overzealous filtering on Call Rate.

Again, by all means filter loci on Call Rate, but consider the consequential loss of loci and do not filter so stringently as to decimate your locus count, as this can lead to systematic biases (highly polymorphic loci differentially eliminated) and unnecessary loss of informative loci. Carefully consider the plot of the distribution of Call Rate values in the exploratory analyses (`gl.report.callrate`). Depending upon the subsequent analyses (see below), a Call Rate threshold of 0.90 might be considered appropriate for filtering loci.

There is no strict rule when filtering individuals on Call Rate. One should examine which individuals are likely to be lost for a given threshold and consider the impact of their loss on the subsequent analysis and interpretation. Of course, if individuals have exceptionally low Call Rates, their deletion would be highly recommended. If such individuals are of particular importance to the study, re-extraction and re-running the samples should be considered.

Imputation

Some analyses are intolerant of any missing values (e.g. classical Principal Components Analysis). Removing all missing values is often too wasteful of data – one must either remove all loci that contain one or more missing values, or remove all individuals that contain one or more missing values. An alternative is to impute the missing values. There are a number of approaches to imputation.

- **Global Imputation:** Missing values are replaced by an estimate of the allelic value generated from the average allele frequencies for all the data taken collectively.
- **Local Imputation:** Missing values are replaced by an estimate of the allelic value generated from the average allele frequencies or from a Hardy-Weinberg model at that locus in the population from which the individual was drawn.
- **Nearest Neighbour:** Missing values are replaced by the value at the corresponding locus for the nearest neighbouring individual (nearest using Euclidean genetic distance)
- **Random:** Missing data are substituted by random values (0, 1 or 2).

The appropriate statement is

```
gl <- gl.impute(gl)
```

Filtering on Read Depth

Read depth has a considerable influence on the veracity of SNP calls. Read depth estimates for sequence tags generated from high-volume sequencing typically range from 3 to 1000 for single-copy sequence. The low values arise because of chance variation in the coverage of particular bases. The high values arise because of the differential efficiency of the PCR steps. Under some circumstances, it might be sensible to push the lower threshold for read depth higher than has been used by DArT in their report. Analyses that rely heavily on the accuracy of calls of heterozygotes may require a higher threshold for read depth, say 10*x*, for example. The call is

```
gl.filter.rdepth(gl,threshold=10)
```

Of course, how far you can push the read depth threshold depends very much on the sequencing volume of the service taken in the context of the genome size. You should examine the average read depth for your data in making this decision.

Influence of Planned Analyses

Possibly the most influential consideration on filtering is the purpose for which the filtered dataset is to be used.

If you are planning to generate ***high-resolution linkage maps***, then highly stringent filtering is warranted, and DArT would adjust their pipelines accordingly upon request. Similarly, if you are contemplating an analysis of ***relatedness*** or ***inbreeding***, then stringent filtering might be warranted. In any case, it would be wise to start out with high stringency and then progressively relax the stringency and examine the impact on outcomes.

If your focus is on identifying ***sex-linked markers***, which rely heavily on identifying markers that are heterozygous in XY individuals and homozygous in XX individuals, then a filtering threshold for read depth of 10x would be sensible. Sequencing volume in the context of genome size will provide an upper limit to how far you can push the read depth threshold, so in some cases, a high sequence volume service will need to be requested. Because the pipelines of DArT depend in part on Mendelian behaviour in the selection of SNP markers, and sex-linked loci do not behave in Mendelian fashion, you might also ask them to relax the stringency of some aspects of their filtering in generating the report.

Obviously, if your focus is on identifying ***sex-linked markers***, then do no filtering before using our functions to identify sex linked loci.

The proof is in the pudding when searching for sex linked markers, so if you get the balance of considerations wrong, the cost lies in the number of false positives that will be generated, and additional work at the validation stage.

If your focus is on ***Genome-wide Association Studies*** (GWAS) or identifying ***loci under selection***, then noise in the data will not associate with phenotype, and so filtering can be less stringent in order to maximise the chances of identifying promising associations.

Principal Coordinates Analysis (PCA) and, to a lesser extent, ***Principal Components Analysis*** (PCoA), rely on fully populated data matrices (no missing values). Classical PCA for example, cannot easily accommodate missing values. To overcome this, a balance must be struck between filtering on Call Rate and imputing the values of those missing values that remain. More stringent filtering on Call Rate, and less imputation; but more stringent filtering on Call Rate, more loss of potentially useful information or valuable samples. Refer to Georges et al. (2023, BioRxiv https://doi.org/10.1101/2023.03.22.533737) for further discussion of this point.

Conclusion

There is no hard and fast rule to guide decisions on filtering SNP datasets prior to a substantive analysis. The decisions are based on comparing the distribution of the parameters to be filtered (using one of the `gl.report` functions) and the purpose to which the filtered dataset is to be put. Sequencing volume can constrain options, and the likelihood of some level of contamination of samples or presence of inhibitors of the restriction enzymes can have a bearing on the decisions. It is important not to over-filter because of the risk of introducing a level of systematic bias and because of the unnecessary loss of informative loci or individuals. An experimental approach is recommended, whereby different filtering regimes are tried (from stringent to less stringent) and checked for influence on analysis outcomes.

Exercises

Now is a good time to try some exercises and to try out what you have learned so far on your own data. These exercises ask you analyse some SNP and SilicoDArT datasets to see what you have learned so far.

Navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*. Navigate to the Exercises for Chapter 5. Undertake the exercises.

Winding up

Where have we come?

The above Session was designed to give you an overview of the considerations that come into play when filtering your data using dartR. Having completed this Session, you should:

- Appreciate to some degree the pre-processing that is undertaken by DArT before you receive your report.
- Understand some of the considerations that come into play in deciding the order in which to apply dartR filters.
- Be able to apply the filtering in a nuanced manner to avoid over-filtering, which depends of course on the particular research question you are addressing as much as attributes of the data.

Further reading

O'Leary, S.J., Puritz, J.B., Willis, S.C., Hollenbeck, C.M. and Portnoy, D.S. (2018). These aren't the loci you're looking for: Principles of effective SNP filtering for molecular ecologists. Molecular Ecology 27: 3193-3206.

Ende

Chapter 6

Exploratory Visualisation with PCA

Contents

Preamble

Learning Outcomes

The objective of this chapter is to introduce you to the options available for visualizing structure in SNP and SilicoDArT datasets using a dimension-reduction technique called Principal Components Analysis (PCA).

After completing this Chapter, you should have a basic intuitive understanding of PCA and how it can be used to condense and examine genetic structure visually. This chapter is a basic introduction to PCA, a topic that will be developed further in a future eBook on Advanced Topics in dartR.

You also will have skills in adjusting your graphics, saving your graphics, and familiarity with options for refining your graphics outside dartR using ggplot.

Prerequisites

RStudio is an integrated development environment (IDE) with a graphical user interface (GUI) used to manage R workflow including for PCA and so to work through this chapter, you will need to have already completed *Chapter 1. A primer on RStudio.* For more advanced treatment, refer to the **official documentation**.

You should have completed Chapters 3-5 on data entry and data structures, manipulating data and filtering data before attempting this Chapter.

Artificial intelligence tools are invaluable these days for assisting with R code and providing summary information on topics relevant to this Chapter, and you should become familiar with at least one such tool. We recommend **Claude AI**.

You may also find it helpful to listen to the **podcast summary** of the overview material in this chapter.

Workflow

We assume that you have established the RStudio project `eBook1` and downloaded the datasets required for all Chapters. Package `eBook_Startup` has been installed. All ready to go.

The workflow from this point again follows a formula.

- **Theory**, unencombered by competing options and technical asides;
- **Worked Examples** where you are led through analyses with sample code;
- **Exercises** where you are left on your own to integrate the knowledge you have gained in a problem-solving context.
- **Review** the learning outcomes and the section on Where have we come? to confirm that you are moving forward with all that is required under your belt.

Session 6-1: Introduction to PCA

Overview of Principal Components Analysis

Once the data are entered to dartR, manipulated and filtered, we are still confronted with a very substantial body of data. There could be hundreds of individuals and thousands of loci scored for those individuals. Typically, we are looking for evidence of genetic structure in the data, but visually perusing the dataset will provide little insight. We need a technique to separate out signal from noise.

By far the most common approach to visualising structure in a multivariable dataset, such as a SNP or SilicoDArT dataset, is Principal Components Analysis (Jolliffe, 2002) or PCA. There are at least four computational approaches to calculating a PCA, differing in computational efficiency and tractability (Wu et al., 1997). Essentially PCA takes a data matrix (SNP genotypes or SilicoDArT) and considers the individuals to be entities and the SNP loci to be attributes. Each individual is defined by the SNP genotype scores across the loci. PCA is a very intuitive technique that can be understood in its essentials without reference to the underlying mathematics.

Geometric Representation

First, the position of each individual is placed in a space defined by the SNP loci (each locus representing a dimension in a physical space). If we consider the three individuals A, B and C in the space defined by SNP locus 1 and locus 2, we have individual A located at point (1,1) indicating that it is heterozygous at both loci (Figure 6-1). Individual B is heterozygous at locus 1 and homozygous for the alternate allele at locus 2, and individual C is homozygous for the alternate allele at both locus 1 and 2. An individual homozygous for the reference allele at both loci would lie at the origin,

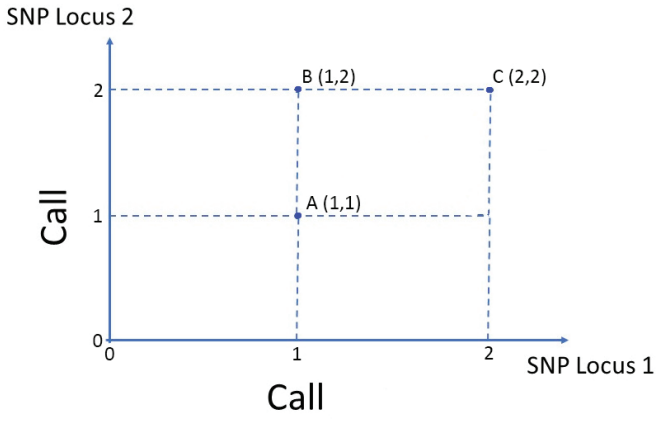


Figure 6-1. *A diagram showing the representation of three individuals (A, B and C) in a space defined by SNP Locus 1 and 2.*

It takes a little stretch of the imagination to extend this to three dimensions with the addition of a third SNP locus ,with the positions of individuals A, B and C defined by their coordinates (0, 1 or 2) along each of the three dimensions. We can extend this thinking further to define the positions of individuals A, B and C in a space defined by their scores on all available SNP loci, that is, in a multivariable space. So,

all the individuals are represented by a cloud of points in a multivariable space with the axes defined by the SNP loci.

Orthogonal Basis

The challenge now addressed by PCA is that the dimensions defined by the SNP loci do not represent a particularly good basis for representing the individuals in a space. This is because the information contained in one SNP locus is not necessarily independent of the information contained in a second SNP locus. They might be linked in the genome or there might be genetic structure in the dataset. If two loci are tightly linked, or both drawn from a highly divergent population, the information on the position of individual A along the dimension corresponding to the first locus will carry with it almost all the information on the position of individual A along the dimension corresponding to the second locus. The two dimensions defining the space are not orthogonal but rather are interrelated. Movement along one mandates movement along the other. They are not authogonal axes.

PCA analysis centres the data on the multivariable mean, then takes linear combinations of the SNP axes to generate orthogonal composite axes (Principal Components) that form a typical set of Cartesian axes. Each individual is represented in the new space by their new scores on the orthogonal Principal Components.

Ordination

The next step in the analysis is to rotate the Principal Component axes to form an ordinated space. The SNP locus space is realigned by rotation (linear transformation) to yield new orthogonal axes ordered on the contribution of variance (represented by their eigenvalues) in the direction of the axes (defined by their eigenvector). So, the first Principal Component is aligned in the direction of maximum variance in the data, the second Principal Component is aligned in the direction of maximum variance orthogonal to the first, and so on.

This process maintains the relative positions of the individuals.

Dimensional Reduction

Because the resultant axes are ordered on the amount of information they contain (the variance), the first few axes, preferably the first two or three, tend to contain information on any structure in the data (signal), and later axes tend to contain only noise (Gauch, 1982) and can be ignored. Classical PCA is thus a powerful visual technique for examining structure in the SNP dataset.

Example

An example of a PCA is provided in Figure 6-2. This is from a study of genetic variation among individuals in the genus *Mixophyes*, the barred frogs. After some preliminary filtering for call rate (< 0.90), secondaries, and other parameters, the dataset contained 19,680 loci scored for each of 77 individuals. Clearly, perusing a data matrix with 77 entities scored for 19,680 attributes is not going to yield any insight to whatever genetic structure may be present. By ordinating a space defined by new orthogonal axes calculated from the 19,680 loci scores across individuals,

we are able to disentangle signal from noise. The first two Principal Components explain 45% of the variation in the data, so we set aside remaining variation (mainly noise) to interpret the genetic structure within the dataset.

Clearly there are three groupings. Two represent existing well-defined species: *Mixophyes balbus* and *Mixophyes fleayi.* A third grouping is unexpected and represents a putative new species, previously considered to be *Mixophyes balbus.* Further analysis following the exploratory PCA confirms that this third grouping is indeed a new species, and it is named as *Mixophyes australis*. Refer to the published paper for a full analysis (Mahony et al., 2023).

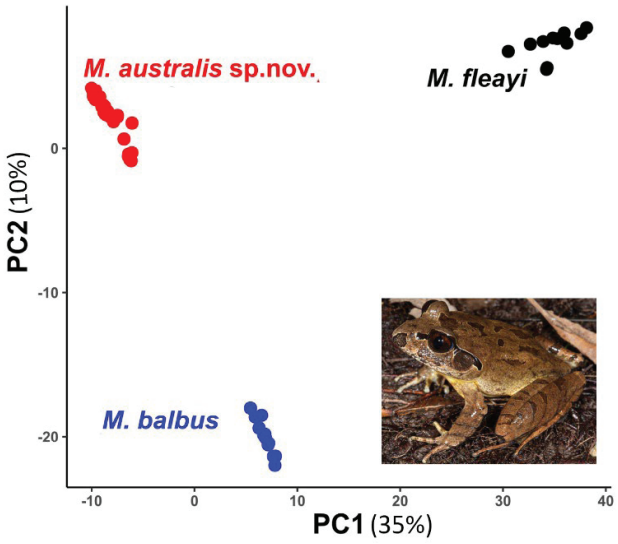


Figure 6-2. *A principal components analysis (PCA) of SNP variation for 77 individuals of the frogs in the genus Mixophyes (barred frogs) each scored for 19,680 loci.*

Worked Example

Now let us consider a worked example. This worked example will run through the application of functions used for exploratory visualisation using PCA prior to more serious analyses. The genlight object contains data from a real-world example, the earless grassland dragon.

To access the Worked Examples, navigate to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*.

You will be asked again to load the earless dragon dataset that was created in Chapter 3, namely `tympo_SNP_raw.RData`.

Undertake Worked Example 6-1.

Exercises

Try some more of what you have learned so far on some exercises and your own data. These exercises ask you to analyse read datasets to apply what you have learned on exploratory visualisation.

The Exercises are also found by navigating to the RStudio Tutorial Pane and then open the *eBook Getting Started with dartR*. Navigate to the Exercises for Chapter 6.

Winding up

Where have we come?

In this Session, you have learnt how to reduce the dimensionality with a technique called classical Principal Components Analysis (PCA) and how to visualise the results. You should now have:

- a basic appreciation of the rationale of PCA
- practical experience in conducting a PCA on SNP and SilicoDArT data
- an appreciation that PCA is an exploratory tool for examining genetic structure within the dataset as a prelude to further analysis to address interesting hypotheses arising.

PCA is a complex topic, and this Chapter gives only a brief introduction. Further reading will be required to appreciate and apply the full power of the analysis on real datasets.

Ende

Integrative Exercises

One of the fundamental steps in achieving deep learning is to take what you have read and heard in a comprehensive coverage of the topic and then recall, integrate and use that information in a problem-solving context.

This can be done by applying what you have learned from this eBook to solve the challenges you face as you progress your own projects and craft solutions to achieve the desired outcomes.

However, our experience is that the world is a busy place with many distractions and by the time you come to apply what you have learned, much of that knowledge will have faded.

We strongly encourage you to complete some of the exercises we have crafted for you, which cover the full range of skills and knowledge that is covered in this eBook. We call them integrative exercises. It is in undertaking these exercises that the real, deep learning will occur. It will stay with you.

To access these exercises, navigate to the RStudio Tutorial Pane and then open the eBook Getting Started with dartR. Navigate to the Integrative Exercises. Choose those that are best aligned with your work or study interests.

Concluding Remarks

In completing this eBook, you have made your first dip in the waters of genetic analysis using representational sequencing – SNP profiles. The journey through this eBook has taken you from the fundamentals of R and RStudio, through the logic of data structures and workflows, to the first steps of exploring and visualising genetic variation using dartR. Along the way, you have learned how to import, manage, filter and analyse SNP and SilicoDArT datasets, and to present your results in basic ways that reveal the biological patterns hidden within the data.

These are not just technical skills. They are tools to provide you with access to more sophisticated analyses that are the foundation of modern population genetics in the genomic era. The capacity to move seamlessly from raw genotypes to interpretable patterns of structure, diversity and differentiation is now essential for biologists across disciplines – from conservation and evolutionary ecology to systematics and genomics. As genetic technologies advance and data volumes continue to expand, the ability to handle these data efficiently and reproducibly will be central to effective research in a data-rich world.

This first eBook and its subject material, dartRverse, have aimed to make that journey approachable, allowing you to explore real genomic data with confidence and curiosity. The eBook has focused on the basic mechanics of dartR and RStudio – establishing a practical and conceptual grounding that empowers you to work confidently with your own data. But it is only the beginning.

A companion volume – *Advanced Topics in Population Genetics using dartR* – is in preparation. It will build on the foundation laid here, introducing more sophisticated analyses and interpretations. You will explore methods for estimating population differentiation (e.g., F_{ST}, G_{ST}, and related metrics), inferring population structure, estimating gene flow and effective population size, identifying loci under selection, visualising spatial genetic patterns, new approaches to species delimitation, and integrating genomic data with environmental, demographic, and geographic information. These advanced techniques will open a window into the evolutionary and ecological processes that shape genetic diversity across landscapes and through time.

As analytical tools and computational power continue to evolve, the integration of genomic data into ecology and conservation biology will only deepen. The ability to think critically about data, to explore it visually and to interpret data within an evolutionary framework will remain at the heart of this transformation.

As we said in the introduction, it is a remarkable time to be a biologist – perhaps the most exciting time in the history of our discipline. Never before have we had the means to examine life at such resolution, nor the tools to connect genetic variation to ecological and evolutionary processes at scale. Whether your interests lie in conservation, systematics, physiology, evolutionary theory, or applications in agriculture, fisheries and forestry, the skills you are developing now will open new doors. They will serve as a bridge to discoveries, new collaborations, and new ways of studying the amazing diversity of life.

Take what you have learned here, keep exploring, and look forward to the next step, where we delve deeper into the rich analytical landscape of *Population Genetics using dartR.*

Bibliography

Georges, A., Gruber, B., Pauly, G.B., Adams. M., White, D., Young, M.J., Kilian, A., Zhang, X., Shaffer, H.B. and Unmack, P.J. (2018). Genome-wide SNP markers breathe new life into phylogeography and species delimitation for the problematic short-necked turtles (Chelidae: *Emydura*) of eastern Australia. *Molecular Ecology* 27:5195-5213. **https://doi.org/10.1111/mec.14925** .

Gruber, B., Unmack, P.J., Berry, O. and Georges, A. (2018). dartR: an R package to facilitate analysis of SNP data generated from reduced representation genome sequencing. *Molecular Ecology Resources* 18:691–699. **https://doi.org/10.1111/1755-0998** .

Jolliffe, I.T. (2022). *Principal Component Analysis*. Second Edition. Springer-Verlag, New York.

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

Jombart, T. and Ahmed, I. (2011). Adegenet 1.3-1: New tools for the analysis of genome-wide SNP data. *Bioinformatics* 27:3070–3071. https://doi.org/ **10.1093/bioinformatics/btr521** .

Kilian, A., Wenzl, P., Huttner, E., Carling, J., Xia, L., Blois, H., Caig, V., Heller-Uszynska, K., Jaccoud, D., Hopper, C., Aschenbrenner-Kilian, M., Evers, M., Peng, K., Cayla, C., Hok, P. and Uszynski, G. (2012). Diversity Arrays Technology: a generic genome profiling technology on open platforms. In F. Pompanon & A. Bonin (Eds.), *Data Production and Analysis in Population Genomics: Methods and Protocols* (pp. 67–89). Humana Press. **https://doi.org/10.1007/978-1-61779-870-2_5** .

Mahony, M., Bertozzi, T., Guzinski, T. and Donnellan, S.C. (2023). A new species of barred frog, Mixophyes (Anura: Myobatrachidae) from south-eastern Australia identified by molecular genetic analyses. *Zootaxa* 5297: 301–336. https://doi.org/**10.11646/zootaxa.5297.3.1** .

Mijangos, J., Gruber, B., Berry, O., Pacioni, C. and Georges, A. 2022. dartR v2: an accessible genetic analysis platform for conservation, ecology, and agriculture. *Methods in Ecology and Evolution* 13:2150–2158. **https://doi.org/10.1111/2041-210X.13918** .

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

Rochette, N. and Catchen, J. (2017). Deriving genotypes from RAD-seq short-read data using Stacks. *Nature Protocols* 12:2640–2659. **https://doi.org/10.1038/nprot.2017.123** .

Sansaloni, C., Petroli, C., Jaccoud, D., Carling, J., Detering, F., Grattapaglia, D. and Kilian, A. (2011). Diversity Arrays Technology (DArT) and next-generation

sequencing combined: genome-wide, high throughput, highly informative genotyping for molecular breeding of *Eucalyptus*. *BMC Proceedings* 5 :P54. **https://doi.org/10.1186/1753-6561-5-s7-p54** .