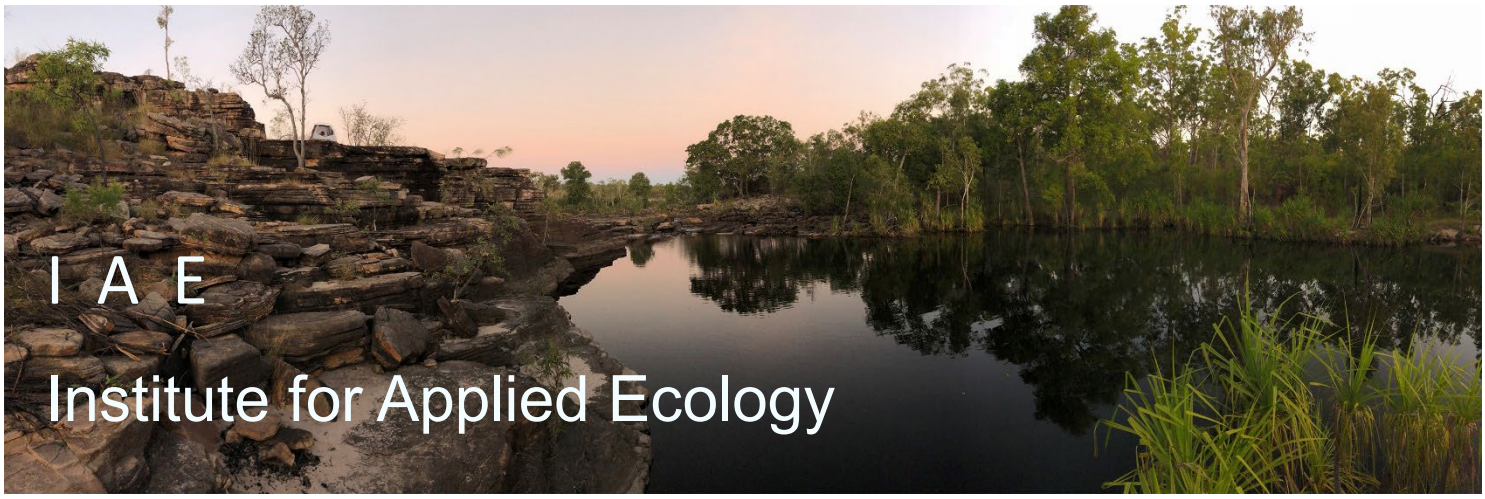
Wildlife Genomics@UC

UNIVERSITY OF CANBERRA

SNP Analysis using dartR

RStudio Refresher

Version 3



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

The Institute for Applied Ecology
University of Canberra ACT 2601
Australia

Email: arthur.georges@biomatix.com.au

Citation: Georges, A. (2025). RStudio Refresher. Version 3. Institute for Applied Ecology, University of Canberra, Canberra ACT 2617 Australia.

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

Contents

RStudio Refresher

What is R?

R is a statistical computing language based on an earlier implementation of a programming language called S. S is still available in the commercial form of S-plus, whereas 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 is 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 capacity, and can be used to produce publication-quality graphs.

The library we will be primarily using in this workshop is **dartR**, a collection of scripts to facilitate analysis of data provided by Diversity Arrays Technology Pty Ltd. Although dartR has some unique analyses, it is primarily for data manipulation, exploratory analysis, and a conduit to other packages used for SNP analysis.

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. In this workshop, you will require familiarity with the R GUI, RSTudio, but 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 Visualization**: 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 various platforms like 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 R-studio, a graphical user interface opens with many features to assist you (Figure 1-1). After some introductory text appears in the Console, a **Command Prompt** is presented (>), and the system awaits instructions.

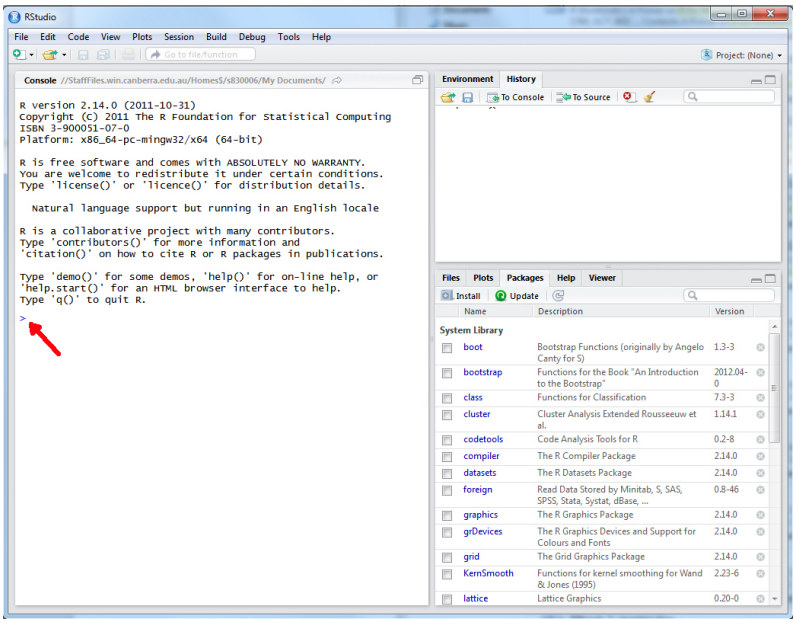


Figure 1-1. R as it appears when it first starts. The R Console window and two other windows are visible. The Program Editor and R Graphics windows do not appear until required.

Before we move on, there are a couple of little tricks here that are 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 can be used 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 as separate objects. It is wise to adopt a consistent practice, such as always using lower case unless upper case 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
```

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

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

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

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.

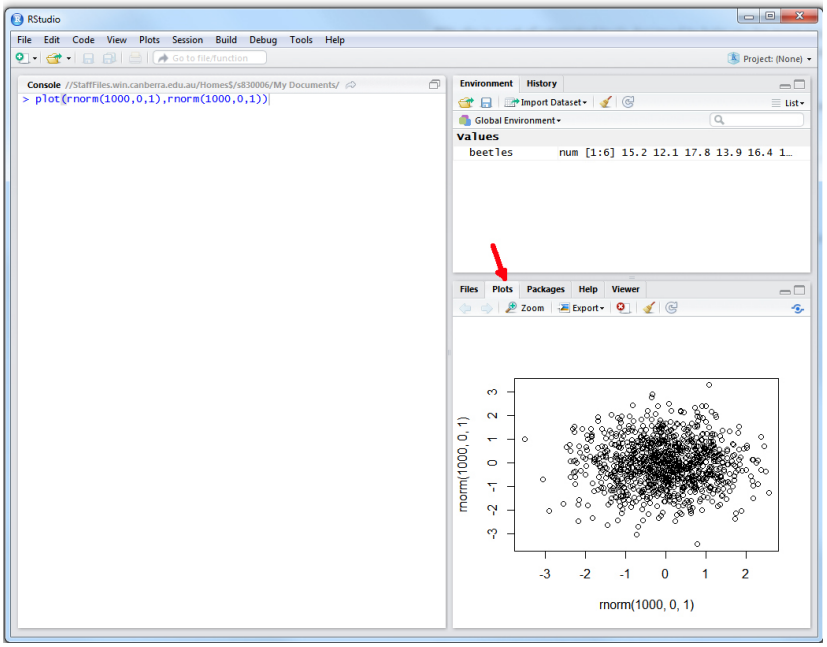
A scatter plot of 1000 pairs of coordinates drawn at random from a bivariate standard normal distribution (mean=0, stdev=1) is made by combining the `plot()` function with the `rnorm()` function as follows:

```
> 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 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 R 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 R 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 preceeded by the # character and terminated by a return (↵).

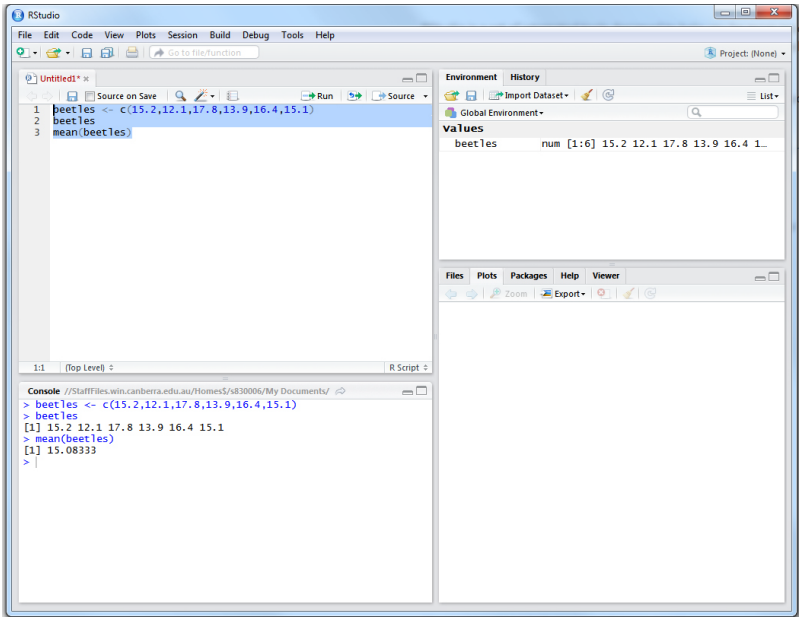
Our simple R program, with comments added is shown in Figure 1-3.

```
# 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. R as it appears after submitting a simple program.



You screen should look like that displayed 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 (https://cran.r-project.org/).

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 R-studio 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 on (e.g. hist for plotting histograms), help can be obtained using

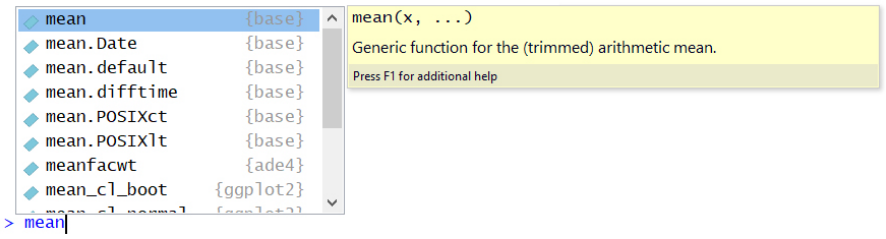
```
> ?mean
```

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.

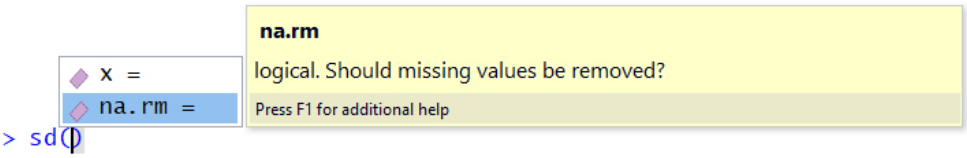
Use the `?` to generate help on the functions `mean`, `sd`, `plot` and `matrix`

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 in to 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.

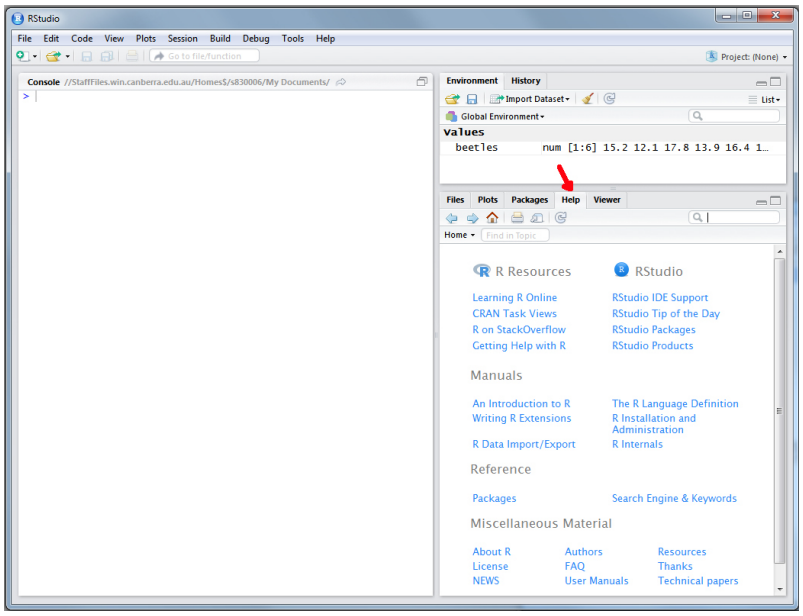


Try this for yourself with some of the functions you have used so far

Help menu

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

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 which 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 English is the new programming language.

Managing Your Workspace

Starting a New Session

R 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 workflow can be difficult in R, and we need some basic rules to minimize confusion.

- Identify discrete projects or analyses and create a separate Windows directory for each one. This way you will avoid having a jumble of objects from many analyses in your workspace.
- 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 data files and `filename.Rdata` for R binary files.

Once you have started R, you need to start a new project using `File>New Project`. R will prompt you for a directory in which to save all temporary and working files, and the project image if you choose to save it later.

R 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 R, at which time you will be asked whether or not 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.

Managing your Objects

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

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

`> setwd("c://R_analysis")`	Sets the default directory for files, and action that can also be done from the R-studio menus.
`> ls()`	provides a list of objects in your current workspace.
`> 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 ensure that the files associated with each project is in a separate directory on disk. To set the default directory use

```
> setwd("C:/Users/username/Documents/R_demo")
```

R will then look in the directory R_demo 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 somewhere useful, by adding the above line to your script in the RStudio Editor. Submit it 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 the files in the working directory using

```
> dir()
```

Setting up a Project

A project in RStudio is a convenient way to bundle together all the files related to a particular analysis, including any code, data, documentation and output.

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

- **Isolation**: RStudio projects help to ensure that the files associated with an analysis are isolated in their own directory, which makes it easier to organize files, figure out what files are involved in a given analysis, and to move the analysis 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 particularl links to 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 into a project. This will create a .Rproj file in the directory which stores project-specific settings. By clicking on this .Rproj file or re-opening the project, you can reinstate all its associated settings.

Where have we come?

The above Session was 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 the following concepts.

- R has available a Graphical User Interface (GUI) called R-studio, and within it, the R Console, R Editor Window, Graphics Output Window, and various Help Windows.
- R packages are first installed, then loaded to become available for use.
- R 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 ? in front of 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.

Exercises

Exercise 1: Scalars and Basic Arithmetic

- Open RStudio and create a new script with the comment line

```
# Exercise 1: Scalars and Basic Arithmetic
```

- Create scalar objects x and y by assigning values to them using assignment statements (e.g. <−).
- Perform basic arithmetic operations such as addition, subtraction, multiplication, division and exponentiation.
- Add and run a statement to create a new object z with the value equal to an algebraic combination of x and y.
- Display the contents of the scalar object z.

Exercise 2: Vectors

- Add the comment line to your script.

```
# Exercise 2: Vectors
```

- Create and run statement to define a vector v containing the first 10 positive integers.
- Add and run a statement to display the contents of the vector object v.
- Add and run statements to calculate the mean, median, and standard deviation of v.

Exercise 3: Larger Vectors

- Add the comment line to your script.

```
# Exercise 3: Larger Vectors
```

- Obtain and read the help on the `rnorm()` function.
- Add and run a statement to generate 1000 random values drawn from a normal distribution with a mean of zero and a standard deviation of one. Note that these values appear in the console window.
- Adjust and run that statement so that the 1000 values are assigned to a vector v.
- Obtain and read the help on the `hist()` function.
- Add and run a statement to plot a histogram of vector v in the plot window. No need for bells and whistles.

Exercise 4: Install a library

- Add the comment line to your script.

```
# Exercise 4: Install a library
```

- Install the library ggplot2 from CRAN.
- Load the library ggplot2.

- Cut and paste the following code into your script in the Editor Window.

```
# Create a data frame
data <- data.frame(value = v)
# Create the ggplot object
p <- ggplot(data, aes(x = value))
# Add the histogram layer
p + geom_histogram(binwidth=0.5,fill="blue",color="black")
+ labs(title="Histogram of Values",x="Value",y="Frequency")
```

- Run this section of code to display an improved histogram in the Plot Window.

Exercise 5: Save your Script

- Check that your script is nice and tidy, and add any additional comment lines (program for your future you).
- Check what directory is your working directory.
- Save your script to your working directory.

Exercise 6: Create a Project

- Create a new project and associate it with your working directory.
- Save the project using a memorable name relevant to the purpose of your script.
- Close the project.
- Exit RStudio.
- Start RStudio again and open your project.