Programming with R — A Beginners' Guide for Geoscientists

1 - Introduction

Tobias Stephan

09/02/2022

Contents

Obtaining R

R is available for Linux, MacOS, and Windows. Software can be downloaded from The Comprehensive R Archive Network (CRAN): (https://cran.r-project.org/)

For windows: - click **Download R for Windows** - click **install R for the first time** - Download R-4.X.X

For Mac: - Download the R-4.X.X package for your OS X

For Linux Debian/Ubuntu: - `sudo apt-get update` - `sudo apt-get install r-base`

Editor

Editors are software programs that enable the user to create and edit text files. They also offer compilation, syntax highlighting, and many more useful features. The undisputed best editor for R is **RStudio**. Please download the free editor at https://www.rstudio.com/products/rstudio/download/ for your system and follow the instructions of the installer.

PLEASE INSTALL THESE BEFORE CONTINUING!

RStudio is what we will focus on using in this course. You will not only love it from the very beginning but as you advance in your skills you will not run out of reasons to love RStudio.

You have currently installed RStudio Desktop, however there is a server version available which allows for shared coding projects and a central R installation so that users do not need to manage things by themselves.

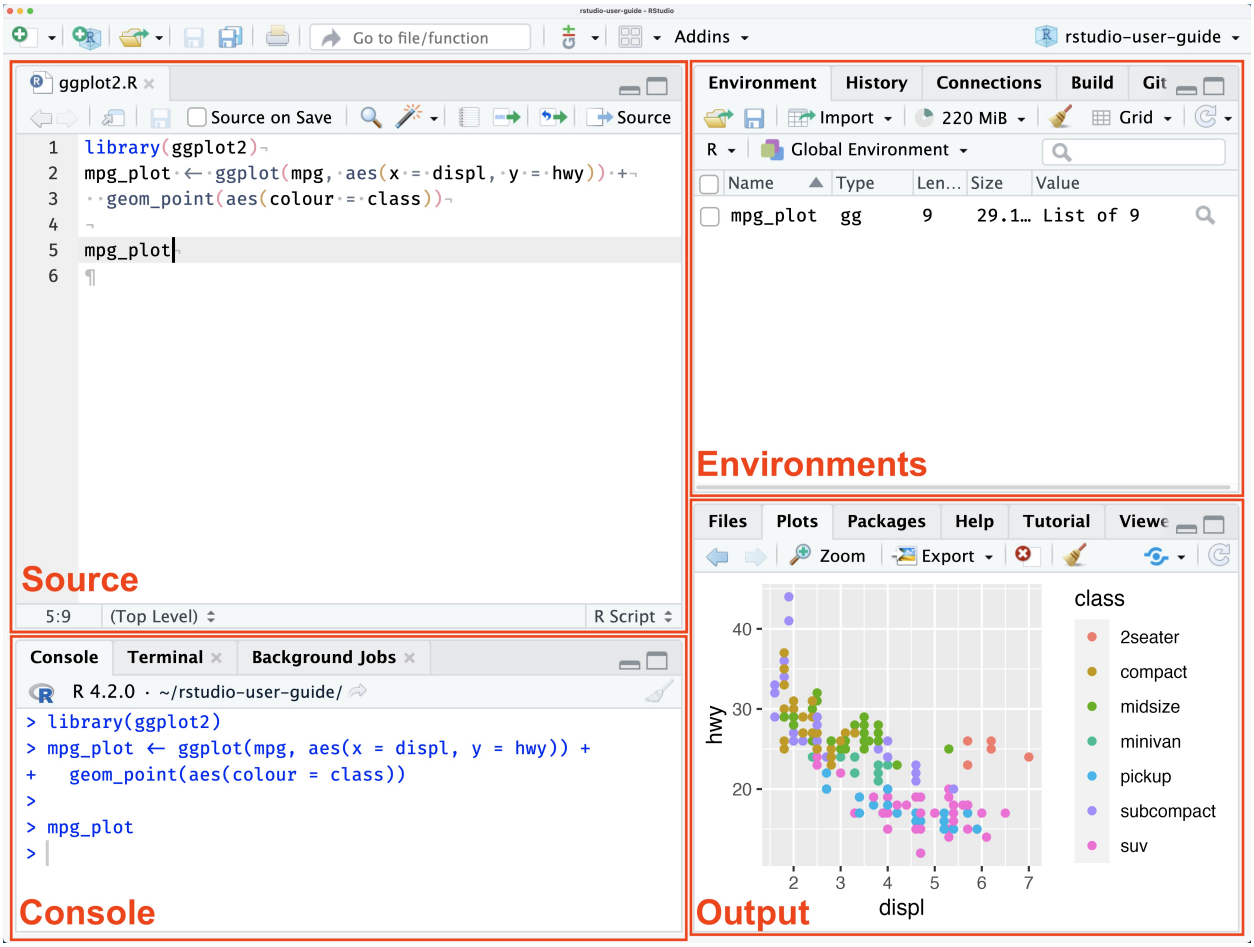

Figure 1: RStudio panes

Coding Basics

Operators

You can use R as a calculator with the following operators:

Operator	Description
+	addition
-	subtraction
*	multiplication
/	division
^ or **	exponentiation
%%	modulus (remainder from division)
%/%	integer Division

```
1 + 2
```

```
## [1] 3
```

```
1 - 2 # everything that comes after the '#' character will be ignored.
```

```
## [1] -1
```

```
# People therefor use it to comment the lines
1 / 200 * 30 # Comment
```

```
## [1] 0.15
```

```
(59 + 73 + 2) / 3 # Distributive property
```

```
## [1] 44.66667
```

```
15 / 2
```

```
## [1] 7.5
```

```
15 %/% 2 # the integer divison (%/%) rounds the result down to the nearest whole number
```

```
## [1] 7
```

```
364 %% 360 # Modulo operation
```

```
## [1] 4
```

Logical Operators

In programming, you often need to know if an expression is true or false. You can evaluate any expression in R, and get one of two answers, `TRUE` or `FALSE`.

Logical Operators ("Booleans") include:

Operator	Description
>	greater than
>=	greater than or equal to
==	exactly equal to
!=	not equal to
&	AND
|	OR
!	NOT

When you compare two values, the expression is evaluated and R returns the logical answer, i.e. the binary values `TRUE` or `FALSE`:

```
1 < 2
```

```
## [1] TRUE
```

```
1 == 2
```

```
## [1] FALSE
```

```
1 > 2
```

```
## [1] FALSE
```

```
1 != 2
```

```
## [1] TRUE
```

```
1 & 2 < 3
```

```
## [1] TRUE
```

```
1 | 2 < 2
```

```
## [1] TRUE
```

```
TRUE
```

```
## [1] TRUE
```

```
FALSE
```

```
## [1] FALSE
```

```
!TRUE
```

```
## [1] FALSE
```

Creating new variables (assignment)

You can create new objects with `<-`:

```
x <- 3 * 4
```

All R statements where you create objects, assignment statements, have the same form:

```
object_name <- value
```

When reading that code say "object name gets value" in your head.

> You will make lots of assignments and `<-` is a pain to type. Don't be lazy and use `=`: it will work, but it will cause confusion later. Instead, use RStudio's keyboard shortcut: `Alt` + `-` (the minus sign). Notice that RStudio automagically surrounds `<-` with spaces, which is a good code formatting practice. Code is miserable to read on a good day, so giveyoureyesabreak and use spaces.

Object names must start with a letter, and can only contain letters, numbers, `_` and `.`. You want your object names to be descriptive, so you'll need a convention for multiple words. It is recommended to use **snake_case** where you separate lowercase words with `_`.

```
i_use_snake_case
otherPeopleUseCamelCase
some.people.use.periods
And_aFew.People_RENOUNCEconvention
```

You can inspect an object by typing its name:

```
x
```

```
## [1] 12
```

Make another assignment:

```
this_is_a_really_long_name <- 2.5
```

To inspect this object, try out RStudio's completion facility: type "this", press `TAB`, add characters until you have a unique prefix, then press return.

Calling functions

R has a large collection of built-in functions that are called like this:

```
function_name(arg1 = val1, arg2 = val2, ...)
```

Let's try using `seq()` which makes regular **seq**uences of numbers and, while we're at it, learn more helpful features of RStudio. Type se and hit TAB. A popup shows you possible completions. Specify `seq()` by typing more (a "q") to disambiguate, or by using ↑/↓ arrows to select. Notice the floating tooltip that pops up, reminding you of the function's arguments and purpose. If you want more help, press F1 to get all the details in the help tab in the lower right pane.

Press TAB once more when you've selected the function you want. RStudio will add matching opening (`(`) and closing (`)`) parentheses for you. Type the arguments `1`, `10` and hit return.

```
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

If you make an assignment, you don't get to see the value. You're then tempted to immediately double-check the result:

```
y <- seq(1, 10)
y
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Conditional statements (if. . . else)

If

An **if statement** is written with the `if` keyword, and it is used to specify a block of code to be executed if a condition is `TRUE`:

```
a <- 1
b <- 200

if (b > a) {
  print("b is greater than a")
}
```

```
## [1] "b is greater than a"
```

R uses curly brackets { } to define the scope in the code.

Else If

The `else if` keyword is R's way of saying "if the previous conditions were not true, then try this condition":

```r
a <- 33
b <- 33

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print("a and b are equal")
}
```

```
## [1] "a and b are equal"
```

Else

The `else` keyword catches anything which isn't caught by the preceding conditions:

```r
a <- 200
b <- 33

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print("a and b are equal")
} else {
  print("a is greater than b")
}
```

```
## [1] "a is greater than b"
```

AND

The `&` symbol (and) is a logical operator, and is used to combine conditional statements:

```r
a <- 200
b <- 33
c <- 500

if (a > b & c > a) {
  print("Both conditions are true")
}
```

```
## [1] "Both conditions are true"
```

OR

The `|` symbol (or) is a logical operator, and is used to combine conditional statements:

```r
a <- 200
b <- 33
c <- 500

if (a > b | a > c) {
  print("At least one of the conditions is true")
}
```

```
## [1] "At least one of the conditions is true"
```

Loops

A for loop is used to iterate over a vector in R programming.

```
# Below is an example to count the number of even numbers in a vector.
x <- c(2, 5, 3, 9, 8, 11, 6)
count <- 0
for (i in x) {
  if (i %% 2 == 0) {
    count <- count + 1
  }
}
print(count)
```

```
## [1] 3
```

Packages

Packages are collections of R functions, data, and compiled code in a well-defined format. The directory where packages are stored is called the library. R comes with a standard set of packages. Others are available for download and installation. Once installed, they have to be loaded into the session to be used.

Adding packages

Befor using any function of a package, you have to install the package.

Packages can be installed from the R's own official package library **CRAN** (Comprehensive R Archive Network) or from Github (the largest platform for any developer of open-source software).

> There are other sources, incl. private repositories. However, I do not recommend to download packages from such resources as the packages are not tested, documented, nor is there a guarantee that they will even cause any harm to your system.

```
install.packages("remotes") # install package 'remotes' from CRAN
```

```
remotes::install_github("tobiste/tectonicr") # install package 'tectonicr' from my repository on Github
```

You only have to install the package once. After that you have to load the the package. `library()` allows to call the entire package, i.e. all the functions of the package:

```
library("tectonicr")
```

`p_load()` from the package `pacman` combines the install (if necessary) and the loading of a package:

```
install.packages("pacman")
pacman::p_load("dplyr")
```

If you only want to use a single function from a package, you just prompt the function by adding the package's name and `::` before the function call:

```
dplyr::glimpse(iris) # "iris" is a pre-installed example dataset
```

```
## Rows: 150
## Columns: 5
## $ Sepal.Length <dbl> 5.1, 4.9, 4.7, 4.6, 5.0, 5.4, 4.6, 5.0, 4.4, 4.9, 5.4, 4.~
## $ Sepal.Width  <dbl> 3.5, 3.0, 3.2, 3.1, 3.6, 3.9, 3.4, 3.4, 2.9, 3.1, 3.7, 3.~
## $ Petal.Length <dbl> 1.4, 1.4, 1.3, 1.5, 1.4, 1.7, 1.4, 1.5, 1.4, 1.5, 1.5, 1.~
## $ Petal.Width  <dbl> 0.2, 0.2, 0.2, 0.2, 0.2, 0.4, 0.3, 0.2, 0.2, 0.1, 0.2, 0.~
## $ Species      <fct> setosa, setosa, setosa, setosa, setosa, setosa, setosa, s~
```

Help

Packges from CRAN or Github resources have a thorough documentation. Every function, therefor, has a description and details of the in- and output arguments, the mathematical background, references, and many more. You can simply call this documentation by using `?` and the name of the function:

```
?max
help(c)
```

Functions

In addition to R's built-in functions, you can also define your own:

```
cube <- function(n) {
  return(n^3)
}

# Using this function to take the cube of 3:
c3 <- cube(3)
```

Useful resources

- **R for Data Science** written by Hadley Wickham – the chief scientist of RStudio and many packages. This **free** book will teach you how to do data science with R: You'll learn how to get your data into R, get it into the most useful structure, transform it, visualize it and model it. https://r4ds.had.co.nz/
- **Cheatsheets** Eclectic, graphical, easy to understand, and handy summaries for some of the most important packages: https://www.rstudio.com/resources/cheatsheets/
- **Stackoverflow** Stack Overflow is a question and answer website for professional and enthusiast programmers. For most cases, the site is the first stop to find an answer. **There is an answer for almost every question!** The answers are rated, and the most useful answers are on top! https://stackoverflow.com/
- **R-bloggers** content contributed by bloggers who write about R (in English). This includes useful tutorials. https://www.r-bloggers.com/

Some usful geoscience packages

(list is not complete)

Geochemistry

- `rgr`: Applied Geochemistry EDA (by Geological Survey of Canada (GSC)): Geological Survey of Canada (GSC) functions for exploratory data analysis with applied geochemical data, with special application to the estimation of background ranges and identification of outliers, 'anomalies', to support mineral exploration and environmental studies. Additional functions are provided to support analytical data QA/QC, ANOVA for investigations of field sampling and analytical variability, and utility tasks. `install.packages(rgr)`
- `gmGeostats`: Geostatistics for Compositional Analysis: Support for geostatistical analysis of multivariate data, in particular data with restrictions, e.g. positive amounts data, compositional data, distributional data, microstructural data, etc. It includes descriptive analysis and modelling for such data, both from a two-point Gaussian perspective and multipoint perspective. The methods mainly follow Tolosana-Delgado, Mueller and van den Boogaart (2018) doi:10.1007/s11004-018-9769-3. `install.packages("gmGeostats")`

Geochronology

- `isoplotR`: IsoplotR is a free and open-source substitute for Kenneth Ludwig's popular Isoplot add-in to Microsoft Excel. Provenance analysis. `install.packages(isoplotR)`
- `provenance`: Statistical Toolbox for Sedimentary Provenance Analysis (by P. Vermeesch): Bundles a number of established statistical methods to facilitate the visual interpretation of large datasets in sedimentary geology. Includes functionality for adaptive kernel density estimation, principal component analysis, correspondence analysis, multidimensional scaling, generalised procrustes analysis and individual differences scaling using a variety of dissimilarity measures. Univariate provenance proxies, such as single-grain ages or (isotopic) compositions are compared with the Kolmogorov-Smirnov, Kuiper or Sircombe-Hazelton L2 distances. Categorical provenance proxies such as chemical compositions are compared with the Aitchison and Bray-Curtis distances, and point-counting data with the chi-square distance. Also included are tools to plot compositional and point-counting data on ternary diagrams and point-counting data on radial plots, to calculate the sample size required for specified levels of statistical precision, and to assess the effects of hydraulic sorting on detrital compositions. Includes an intuitive query-based user interface for users who are not proficient in R. `install.packages(provenance)`

Structural geology, tectonics, seismicity - `structR`: Structural geology package for R: My package that provides functions to analyse and plot 3-dimensional orientation data for structural geology (still under development) `remotes::install_github("tobiste/structR")`

- `RockFab`: Rock fabric and strain analysis tools: Provides functions to complete three-dimensional rock fabric and strain analyses following the Rf Phi, Fry, and normalized Fry methods. Also allows for plotting of results (Stereoplot) and interactive 3D visualization functionality. `install.packages(RockFab)`
- `RFOC`: Spherical Distributions and Earthquake Focal Mechanisms: Graphics for statistics on a sphere, as applied to geological fault data, crystallography, earthquake focal mechanisms, radiation patterns, ternary plots and geographical/geological maps. Non-double couple plotting of focal spheres and source type maps are included for statistical analysis of moment tensors. `install.packages(RFOC)`

Multivariable Geostatistics

- `gstat`: Spatial and Spatio-Temporal Geostatistical Modelling, Prediction and Simulation Spatial and spatio-temporal geostatistical modelling, prediction and simulation (e.g. interpolation, krige, variograms)
- `gear` Geostatistical analysis in R
- `georob`: Robust Geostatistical Analysis of Spatial Data: Provides functions for efficiently fitting linear models with spatially correlated errors by robust and Gaussian (Restricted) Maximum Likelihood and for computing robust and customary point and block external-drift Kriging predictions, along with utility functions for variogram modelling in ad hoc geo-statistical analyses, model building, model evaluation by cross-validation, (conditional) simulation of Gaussian processes, unbiased back-transformation of Kriging predictions of log-transformed data.

Remote Sensing

- `raster`
- `RStoolbox`

Generally helpful packages

- `ggplot2` ggplot2 is a system for declarative creating graphics, based on The Grammar of Graphics. You provide the data, tell ggplot2 how to map variables to aesthetics, what graphical primitives to use, and it takes care of the details.
- `patchwork` Easy assembling of plots into one figure
- `dplyr` is a data wrangling package, offering a series of functions (all called as verbs) for data manipulations (select columns, filter rows, mutate variables into new ones, etc.). This has the advantage to nicely integrate with the pipe, jointly with certain convenience functionality in variable selection.
- `sf` similar to `sp` but it is faster and easier to handle with other R packages (e.g. `dplyr`). Might replace sp in future.

- `scico` Scientific colormaps that can be used with `ggplot()`

...

Content | next course: Data