PRIORITIZR WORKSHOP MANUAL

Jeffrey O. Hanson

2019-09-25

Contents

Chapter 1

Welcome!

Here you will find the manual for the prioritizr module of the *Spatial Conservation Prioritization: Concepts, Methods and Application* workshop held at CIBIO-InBIO, Vairão, Portugal. This manual contains instructions for setting up your computer, external resources, data used in the workshop, and teaching materials. **Before you arrive at the workshop, you should make sure that you have correctly set up your computer for the workshop. We cannot guarantee a reliable internet connection during the workshop, and so you may be unable to complete the workshop if you have not set up your computer beforehand.**

Chapter 2

Introduction

2.1 Overview

The aim of this workshop is to help you get started with using the prioritizr R package for systematic conservation planning. It is not designed to give you a comprehensive overview and you will not become an expert after completing this workshop. Instead, we want to help you understand the core principles of conservation planning and guide you through some of the common tasks involved with generating prioritizations. Phrased provocatively, we want to give you the knowledge base and confidence needed to start applying systematic conservation planning to your own work.

One of the best ways, perhaps, to learn something new is to just give it a go. As such, this workshop involves generating prioritizations for protected area establishment and performing analyses to understand them. This manual is divided into two main parts, the **Resources** part provides explanations and example R code for performing common prioritization tasks (e.g. importing data, creating target data, solving conservation problems, calculating shortfalls) and the **Exercises** part contains a systematic conservation planning case-study. During this workshop, your aim is to work through the case-study in the **Exercise** part and complete the tasks and answer the questions. Since the **Exercise** part does not contain any R code, you will need to read the **Resources** part for information on performing the tasks. This means that all of the code needed to complete the **Exercise** part are in the **Resources** part—you just need to find them!

Finally, you are not alone in this workshop. If you are having trouble, please put your hand up and one of the instructors will help you as soon as they can. You can also ask the people sitting next to you for help too. Please note that the first thing an instructor will ask you will (probably) be "what have you tried

so far?". We can't help you if you haven't tried anything.

2.2 Setting up your computer

You will need to have both R and RStudio installed on your computer to complete this workshop. Although it is not imperative that you have the latest version of RStudio installed, **you will need the latest version of R installed**. After installing these programs, you will also need to install various R packages too.

2.2.1 R

The R statistical computing environment can be downloaded from the Comprehensive R Archive Network (CRAN). You can download the latest version of R (version 3.6.1) from here: https://cloud.r-project.org/. Please note that you will need to download the correct file for your operating system (i.e. Linux, Mac OSX, Windows). You may also require administrative permissions to complete the installation process.

2.2.2 RStudio

RStudio is an integrated development environment (IDE). In other words, it is a program that is designed to make your R programming experience more pleasant. During this workshop, you will interact with R through RStudio—meaning that you will open RStudio when you wish interact with R. You can download the latest version of RStudio here: http://www.rstudio.com/download. RStudio is updated several times during the year, and it will tell you when an update is available. When you start RStudio, you will see two key parts of the user interface:

You can type R code into the *Console* part of the user interface and press the enter key to run them.

2.2.3 R packages

An R package is a collection of R code and documentation that can be installed to enhance the standard R environment with additional functionality. Currently, there are over ten thousand R packages available on CRAN. Each of these R packages (mostly) aim to serve a specific need, such as reading Excel spreadsheets, downloading satellite imagery data, downloading and cleaning protected area data, or fitting environmental niche models. In fact, R has such a diverse ecosystem of R packages, that the question is (generally) not "can I

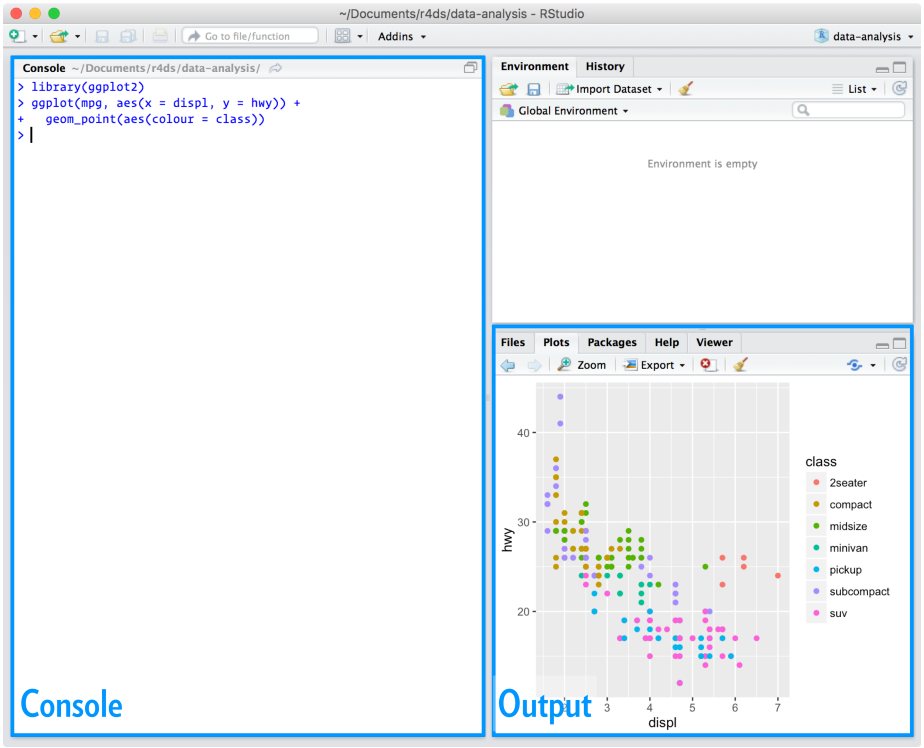
~/Documents/r4ds/data-analysis - RStudio
Go to file/function
Addins
data-analysis
Console ~/Documents/r4ds/data-analysis/
> library(ggplot2)
> ggplot(mpg, aes(x = displ, y = hwy)) +
+ geom_point(aes(colour = class))
>
Environment
History
Import Dataset
List
Global Environment
Environment is empty
Files
Plots
Packages
Help
Viewer
Zoom
Export
hwy
displ
class
2seater
compact
midsize
minivan
pickup
subcompact
suv
Console
Output

Figure 2.1:

use R to do …?" but "what R package can I use to …?". During this workshop, we will use various R packages. To install these R packages, please run enter the code below in the *Console* part of the RStudio interface and press enter. Please note that you will require an internet connection to install the packages and the installation process may take a while to complete.

```
install.packages(c("sf", "tidyverse", "sp", "rgeos", "rgdal", "raster",
                   "prioritizr", "prioritizrdata" "Rsymphony", "mapview"))
```

2.3 Further reading

There is a wealth of resources available for learning how to use R. Although not required for this workshop, I would highly recommend that you read *R for Data Science* by Garrett Grolemund and Hadley Wickham. **This veritable trove of R goodness is freely available online.** If you spend a week going through this book then you will save months debugging and rerunning incorrect code. I would urge any and all ecologists – especially those working on Masters or PhD degrees – to read this book. I even bought this book as a Christmas present for my sister—and, yes, she was happy to receive it! For intermediate users looking to skill-up, I would recommend the *The Art of R Programming: A Tour of Statistical Software Design* by Norman Matloff and *Advanced R* by Hadley Wickham. Finally, if you wish to learn more about using R as a geospatial information system (GIS), I would recommend *Geocomputation with R* by Robin Lovelace, Jakub Nowosad, and Jannes Muenchow which is also freely available online. I also recommend *Applied Spatial Data Analysis* by Roger S. Bivand, Edzer Pebesma, and Virgilio Gómez-Rubio too.

Part I

Resources

Chapter 3

Data import and export

- How can I verify that I have entered the correct file path?
  - You can check that a file exists on your computer using the `file.exists` function. For instance, to check that the file `data.txt` exists, you can use `file.exists("data.txt")`.
- I am having trouble importing data, even though I am specifying the folder?
  - When we specify file paths on Windows systems, we use the `\` character to delimit different folders. For instance, the complete file path for a file might be `C:\Users\kelly\Downloads\cost-data.tif`. However, R treats the `\` character as a special character. This means that if you want to use the `\` character, you need to specify it as `/`. Thus if you want to refer to the file `C:\Users\kelly\Downloads\cost-data.tif` in an R environment, you need to specify it as `C:/Users/kelly/Downloads/cost-data.tif`.
- How can I import spatial raster data?
  - Spatial raster data, such as ASCII (`.asc`) or GeoTIFF (`.tif`) files, can be imported using the `stack` function from the raster R packages. For example, if the file name for your raster data was called `cost-data.tif` then you could import it using `"cost_data <- stack("cost-data.tif")`. Note that you will probably need to specify the folder path for the file too. For example, if you downloaded the file to the Downloads folder on your computer, then you might need to use something like this `"cost_data <- stack("C:/Users/kelly/Downloads/cost-data.tif")"` depending on your system user name.
- How can I import spatial vector data?

  - Spatial vector data, such as shapefiles (`.shp`), can be imported using the `readOGR` function in the rgdal R package. For example, if the file name for your shapefile was `animals.shp`, then you could import it using `animal_data <- readOGR("animals.shp")`. Note that you will probably need to specify the folder path for the file too. For example, if you downloaded the file to the Downloads folder on your computer, then you might need to use something like this `"cost_data <- stack("C:/Users/frank/Downloads/animals.tif")"` depending on your system user name.

- How can I import tabular data?

  - Tabular data, such as comma-separated (`.csv`) files, can be imported using the readr R package. The `read_csv` function can be used to import files where comma characters have been used to delimit different files (i.e. `.csv` files). Additionally, the `read_tsv` function can be used to import files where tab characters have been used to delimit different files (i.e. `.tsv` files). For example, you can import a comma-separated file called `animal-names.csv` using `animal_name_data <- read_csv("animal-names.csv")`.

- How can I export spatial raster data?

- How can I export spatial vector data?

- How can I export spreadsheet data?

Chapter 4

Data manipulation

TODO.

Chapter 5

Data visualization

TODO.

Chapter 6

Spatial prioritizations

TODO.

Part II

Exercises

Chapter 7

Data exploration

TODO.

Chapter 8

Gap analysis

TODO.

Chapter 9

Spatial prioritization

TODO.

Chapter 10

Irreplaceability

TODO.

Part III

Appendix

Chapter 11

Error glossary

If you are having trouble running a specific line of code or a function, then one of the best ways to find help is to read (i) the error message and (ii) the documentation. Believe it or not, a human actually wrote the function that is throwing that (probably) annoying and (probably) indecipherable error message and they (almost always) genuinely want to help you. Instead of (or in addition to) getting frustrated at an error, you might choose to appreciate the fact that someone went to the trouble of writing additional code so that a description of what went wrong would be displayed. Indeed, since there are an unlimited numbers of ways that code can be incorrect, throwing a short and precise error message is actually helpful because it tells you *what went wrong* rather than *potentially incorrectly guessing what should be fixed*. After reading the error message, take the time to think about what it means. Debugging is the process of iterating through each and every assumption that you made until you identify which of your assumptions is incorrect. This is liberating because it means it is within your power to fix the code. When reading an error message, try to interpret each word individually – you might have to google technical terms to understand them, that's OK – and then put them together to try to understand what they mean together. For instance, consider the code below:

```
read.table("fred.txt")
```

```
## Warning in file(file, "rt"): cannot open file 'fred.txt': No such file or
## directory
```

```
## Error in file(file, "rt"): cannot open the connection
```

This tells us that R `cannot open the connection` to the file `"fred.txt"` because `No such file or directory` was found. Now we have to think of all the reasons why this could be true. Is it possible that we have misspelled the filename? Perhaps we meant `"fed.txt"`, `"fran.txt"`, or `"fred.csv"`? Or perhaps we need to specify a folder name, such as `"data/fred.txt"`? Or maybe

we forgot to save the data after entering it into Excel?

Now let us consider another example:

```
a <- 1
b <- "c"
a + b
```

```
## Error in a + b: non-numeric argument to binary operator
```

This error message is a bit more cryptic—but it mentions something about a `non-numeric argument`. This would indicate that something went wrong because something was not a number. If we look at the code, can we see something that is not a number? Yes, `c` is not a number. It is a character-type object (i.e. `"k"`). Does it make sense to add the letter "k" to the number 1? Well, not in R. If we wanted to create the character `"c1"` we would have to use a function designed for this purpose (e.g. `paste`). Alternatively, if we wanted to employ the hexadecimal numeral system, where letters count as numbers, then we would need to specify this.

Let's look at another example:

```
d <- c("a", "b, "c")
```

```
## Error: <text>:1:18: unexpected symbol
## 1: d <- c("a", "b, "c
##                      ^
```

This error message mentions an `unexpected symbol`. If a symbol is unexpected, then this would suggest that the our error is occurring because we have typed an additional symbol or are missing a symbol. If we carefully look at the code for missing symbols, we can see that we are missing a quote (`"`) symbol after after the `b`.