PRIORITIZR WORKSHOP MANUAL

Jeffrey O. Hanson

2020-07-21

Contents

Chapter 1

Welcome!

Here you will find the manual for the prioritizr workshop held at Massey University, Palmerston North, New Zealand. **Before you arrive at the workshop, you should make sure that you have correctly set up your computer for the workshop and you have downloaded the data from here. Since we cannot guarantee a stable Internet connection during the workshop, 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 developing prioritizations. In other words, we want to give you the knowledge base and confidence needed to start applying systematic conservation planning to your own work.

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. **Most importantly, the code needed to answer the questions in this workshop are almost always located in the same section as the question. So if you are stuck, try rereading the example code and see if you can modify it to answer the question.** Please note that the first thing an instructor will ask you will be "what have you tried so far?". We can't help you if you haven't tried anything.

2.2 Setting up your computer

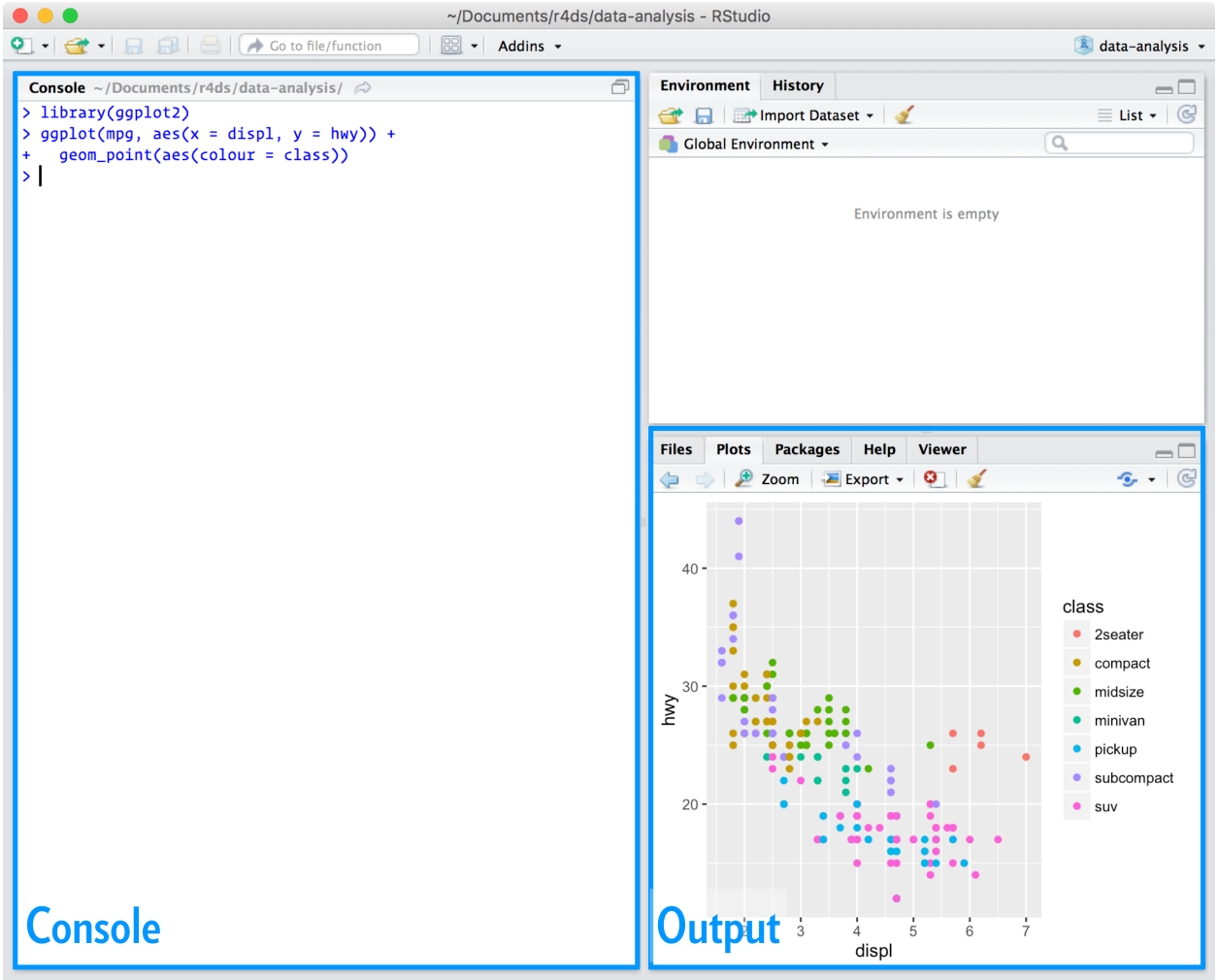
You will need to have both R and RStudio installed on your computer to complete this workshop. Although it is not imperative that you have the latest version of RStudio installed, **you will need the latest version of R installed (i.e. version 4.0.0)**. Please note that you might need administrative permissions to install these programs. After installing them, you will also need to install some R packages too.

2.2.1 R

The R statistical computing environment can be downloaded from the Comprehensive R Archive Network (CRAN). Specifically, you can download the latest version of R (version 4.0.0) 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).

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 enjoyable. During this workshop, you will interact with R through RStudio—meaning that you will open RStudio to code in R. You can download the latest version of RStudio here: http://www.rstudio.com/download. When you start RStudio, you will see two main parts of the interface:



You can type R code into the *Console* and press the enter key to run code.

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 fifteen thousand R packages available on CRAN. Each of these R packages are developed to perform a specific task, 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 almost always not "can I use R to …?" but "what R package can I use to …?". During this workshop, we will use several R packages. To install these R packages, please enter the code below in the *Console* part of the RStudio interface and press enter. Note that you will require an Internet connection and the installation process may take some time to complete.

```
install.packages(c("sf", "dplyr", "sp", "rgeos", "rgdal", "raster",
                   "units", "prioritizr", "prioritizrdata", "Rsymphony",
                   "mapview", "assertthat", "remotes", "gridExtra",
                   "BiocManager"))
BiocManager::install("lpsymphony")
```

2.3 Further reading

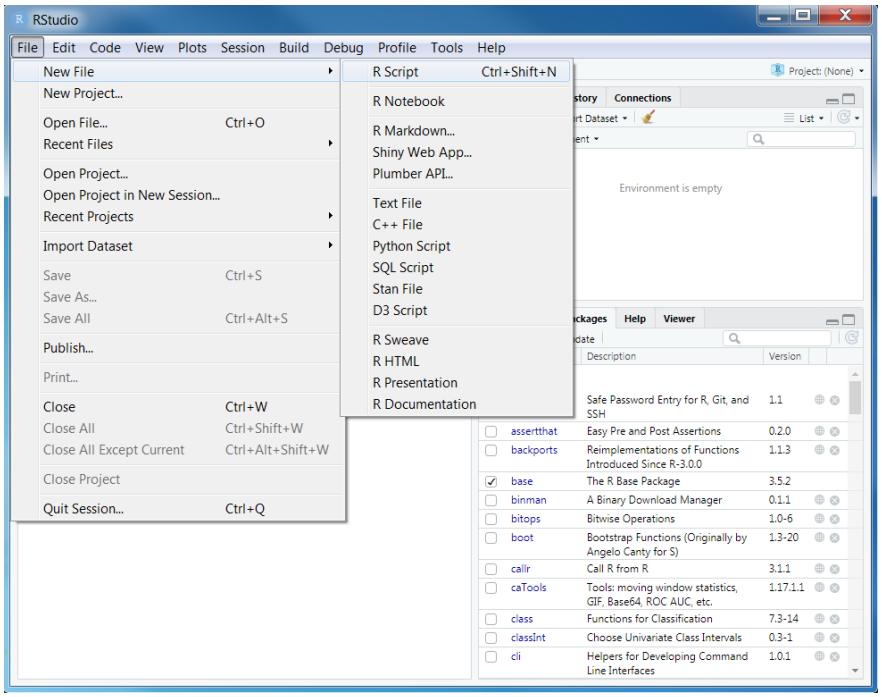
There is a wealth of resources available for learning how to use R. Although not required for this workshop, I would highly recommend that you read *R for Data Science* by Garrett Grolemund and Hadley Wickham. **This veritable trove of R goodness is freely available online.** If you spend a week going 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.

Chapter 3

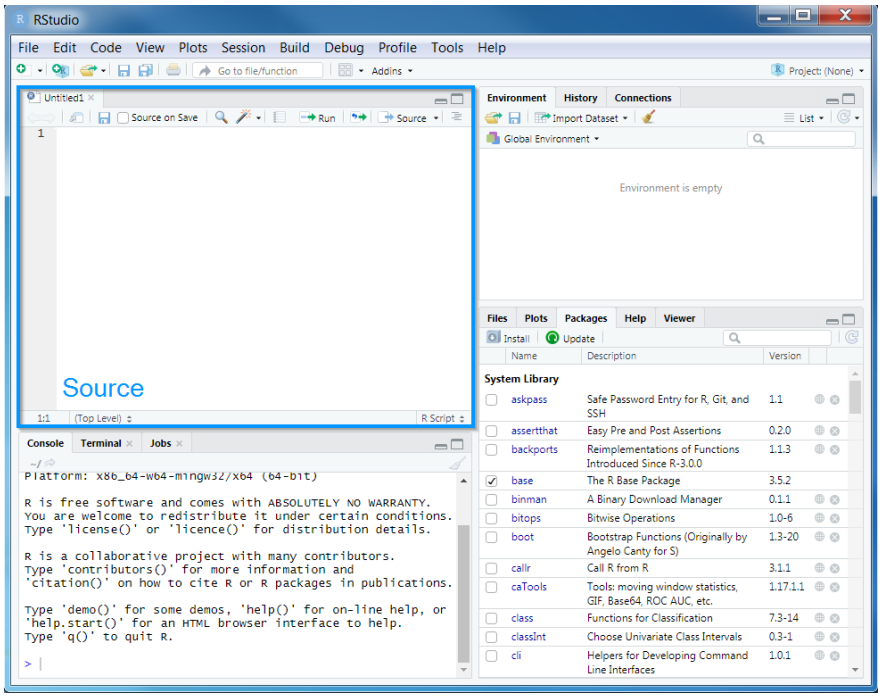
Data

3.1 Starting out

We will start by opening RStudio. Ideally, you will have already installed both R and Rstudio before the workshop. If you have not done this already, then please see the Setting up your computer section. **During this workshop, please do not copy and paste code from the workshop manual into RStudio. Instead, please write it out yourself in an R script.** When programming, you will spend a lot of time fixing coding mistakes—that is, debugging your code—so it is best to get used to making mistakes now when you have people here to help you. You can create a new R script by clicking on *File* in the RStudio menu bar, then *New File*, and then *R Script*.



After creating a new script, you will notice that a new *Source* panel has appeared. In the *Source* panel, you can type and edit code before you run it. You can run code in the *Source* panel by placing the cursor (i.e. the blinking line) on the desired line of code and pressing `Control + Enter` on your keyboard (or `CMD + Enter` if you are using an Apple computer). You can save the code in the *Source* panel by pressing `Control + s` on your keyboard (or `CMD + s` if you are using an Apple computer).



You can also make notes and write your answers to the workshop questions inside the R script. When writing notes and answers, add a `#` symbol so that the text following the `#` symbol is treated as a comment and not code. This means that you don't have to worry about highlighting specific parts of the script to avoid errors.

```
# this is a comment and R will ignore this text if you run it
# R will run the code below because it does not start with a # symbol
print("this is not a comment")
```

```
## [1] "this is not a comment"
```

```
# you can also add comments to the same line of R code too
print("this is also not a comment") # but this is a comment
```

```
## [1] "this is also not a comment"
```

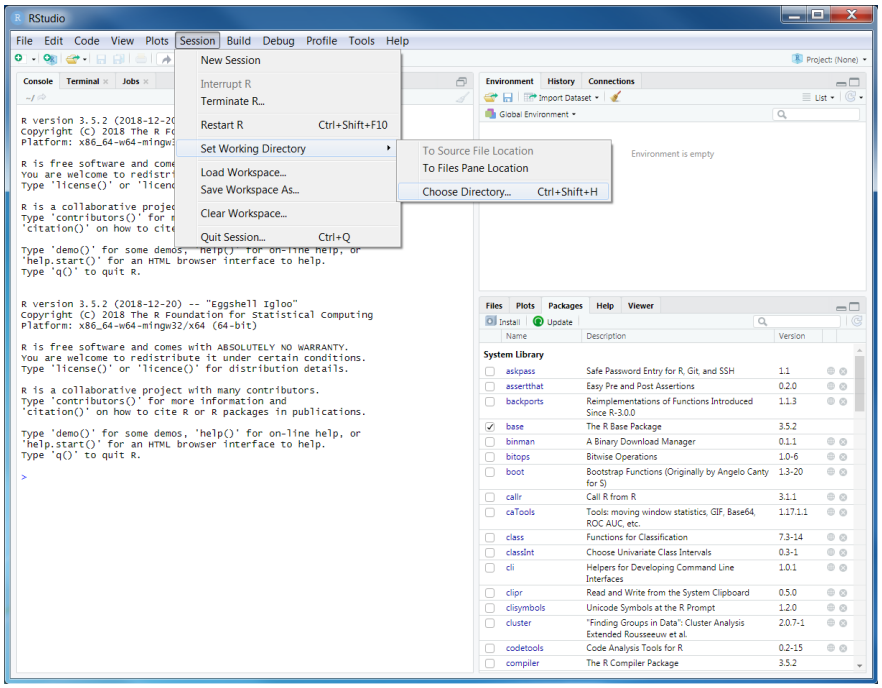
Remember to save your script regularly to ensure that you don't lose anything in the event that RStudio crashes (e.g. using `Control + s` or `CMD + s`)!

3.2 Attaching packages

Now we will set up our R session for the workshop. Specifically, enter the following R code to attach the R packages used in this workshop.

```
# load packages
library(prioritizr)
library(sf)
library(rgdal)
library(raster)
library(rgeos)
require(mapview)
library(units)
library(scales)
library(assertthat)
library(gridExtra)
library(dplyr)
```

You should have already downloaded the data. If you have not already done so, you can download it from here: https://github.com/prioritizr/cibio-workshop/raw/master/data.zip. After downloading the data, you can unzip the data into a new folder. Next, you will need to set the working directory to this new folder. To achieve this, click on the *Session* button on the RStudio menu bar, then click *Set Working Directory*, and then *Choose Directory*.



Now navigate to the folder where you unzipped the data and select *Open*. You can verify that you have correctly set the working directory using the following R code. You should see the output `TRUE` in the *Console* panel.

```
file.exists("data/pu.shp")
```

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

3.3 Data import

Now that we have downloaded the dataset, we will need to import it into our R session. Specifically, this data was obtained from the "Introduction to Marxan" course and was originally a subset of a larger spatial prioritization project performed under contract to Australia's Department of Environment and Water Resources. It contains vector-based planning unit data (`pu.shp`) and the raster-based data describing the spatial distributions of 33 vegetation classes (`vegetation.tif`) in southern Tasmania, Australia. Please note this dataset is only provided for teaching purposes and should not be used for any real-world conservation planning. We can import the data into our R session using the following code.

```
# import planning unit data
pu_data <- as(read_sf("data/pu.shp"), "Spatial")

# format columns in planning unit data
pu_data$locked_in <- as.logical(pu_data$locked_in)
pu_data$locked_out <- as.logical(pu_data$locked_out)

# import vegetation data
veg_data <- stack("data/vegetation.tif")
```

3.4 Planning unit data

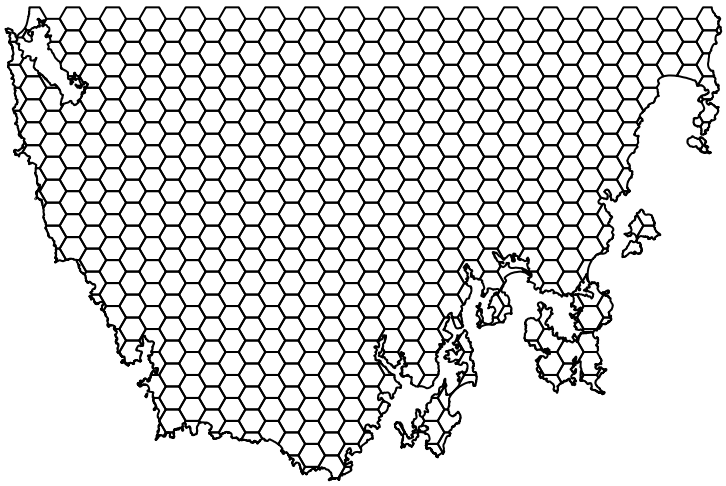
The planning unit data contains spatial data describing the geometry for each planning unit and attribute data with information about each planning unit (e.g. cost values). Let's investigate the `pu_data` object. The attribute data contains 5 columns with contain the following information:

- `id`: unique identifiers for each planning unit
- `cost`: acquisition cost values for each planning unit (millions of Australian dollars).
- `status`: status information for each planning unit (only relevant with Marxan)
- `locked_in`: logical values (i.e. `TRUE`/`FALSE`) indicating if planning units are covered by protected areas or not.
- `locked_out`: logical values (i.e. `TRUE`/`FALSE`) indicating if planning units cannot be managed as a protected area because they contain are too degraded.

```
# print a short summary of the data
print(pu_data)
```

```
## class       : SpatialPolygonsDataFrame
## features    : 557
## extent      : 1115191, 1385011, -4840595, -4662354  (xmin, xmax, ymin, ymax)
## crs         : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellp
## variables   : 5
## names       :   id,            cost, status, locked_in, locked_out
## min values  :  574, 2.5534941249227,      0,         0,          0
## max values  : 1130, 47.238336402701,      2,         1,          1
```

```
# plot the planning unit data
plot(pu_data)
```



```
# plot an interactive map of the planning unit data
mapview(pu_data)
```

```
# print the structure of object
str(pu_data, max.level = 2)
```

```
## Formal class 'SpatialPolygonsDataFrame' [package "sp"] with 5 slots
##   ..@ data       :'data.frame':  557 obs. of  5 variables:
##   ..@ polygons   :List of 557
##   ..@ plotOrder  : int [1:557] 103 219 21 314 206 131 316 327 231 218 ...
##   ..@ bbox       : num [1:2, 1:2] 1115191 -4840595 1385011 -4662354
##   .. ..- attr(*, "dimnames")=List of 2
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
```

```
# print the class of the object
class(pu_data)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
```

```
## [1] "sp"
```

```
# print the slots of the object
slotNames(pu_data)
```

```
## [1] "data"        "polygons"    "plotOrder"   "bbox"        "proj4string"
```

```
# print the coordinate reference system
print(pu_data@proj4string)
```

```
## CRS arguments:
##  +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0
## +ellps=GRS80 +units=m +no_defs
```

```
# print number of planning units (geometries) in the data
nrow(pu_data)
```

```
## [1] 557
```

```
# print the first six rows in the data
head(pu_data@data)
```

```
##    id     cost status locked_in locked_out
## 1 574 28.60687      0     FALSE      FALSE
## 2 575 30.83416      0     FALSE      FALSE
## 3 576 38.75511      0     FALSE      FALSE
## 4 577 38.11618      2      TRUE      FALSE
## 5 578 33.33462      2      TRUE      FALSE
## 6 579 42.98948      2      TRUE      FALSE
```

```
# print the first six values in the cost column of the attribute data
head(pu_data$cost)
```

```
## [1] 28.60687 30.83416 38.75511 38.11618 33.33462 42.98948
```

```
# print the highest cost value
max(pu_data$cost)
```

```
## [1] 47.23834
```

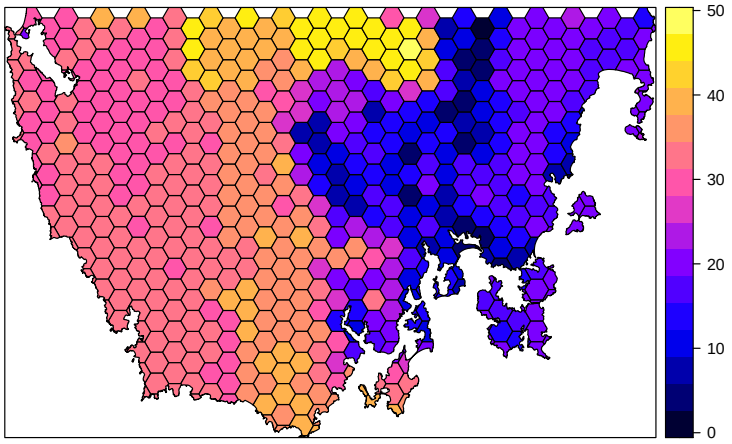
```
# print the smallest cost value
min(pu_data$cost)
```

```
## [1] 2.553494
```

```
# print average cost value
mean(pu_data$cost)
```

```
## [1] 26.65837
```

```
# plot a map of the planning unit cost data
spplot(pu_data, "cost")
```



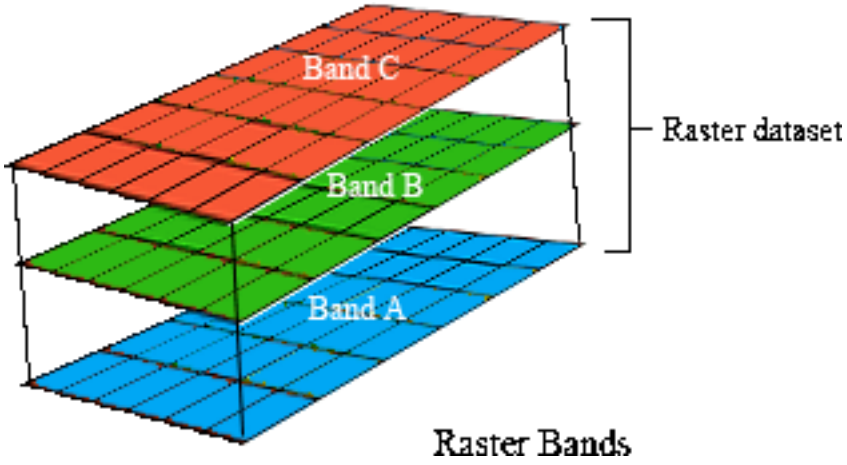

```
# plot an interactive map of the planning unit cost data
mapview(pu_data, zcol = "cost")
```

Now, you can try and answer some questions about the planning unit data.

1. How many planning units are in the planning unit data? 2. What is the highest cost value? 3. Is there a spatial pattern in the planning unit cost values (hint: use `plot` to make a map)?

3.5 Vegetation data

The vegetation data describe the spatial distribution of 33 vegetation classes in the study area. This data is in a raster format and so the data are organized using a grid comprising square grid cells that are each the same size. In our case, the raster data contains multiple layers (also called "bands") and each layer has corresponds to a spatial grid with exactly the same area and has exactly the same dimensionality (i.e. number of rows, columns, and cells). In this dataset, there are 33 different regular spatial grids layered on top of each other – with each layer corresponding to a different vegetation class – and each of these layers contains a grid with 171 rows, 319 columns, and 54549 cells. Within each layer, each cell corresponds to a 1 by 1 km square. The values associated with each grid cell indicate the (one) presence or (zero) absence of a given vegetation class in the cell.

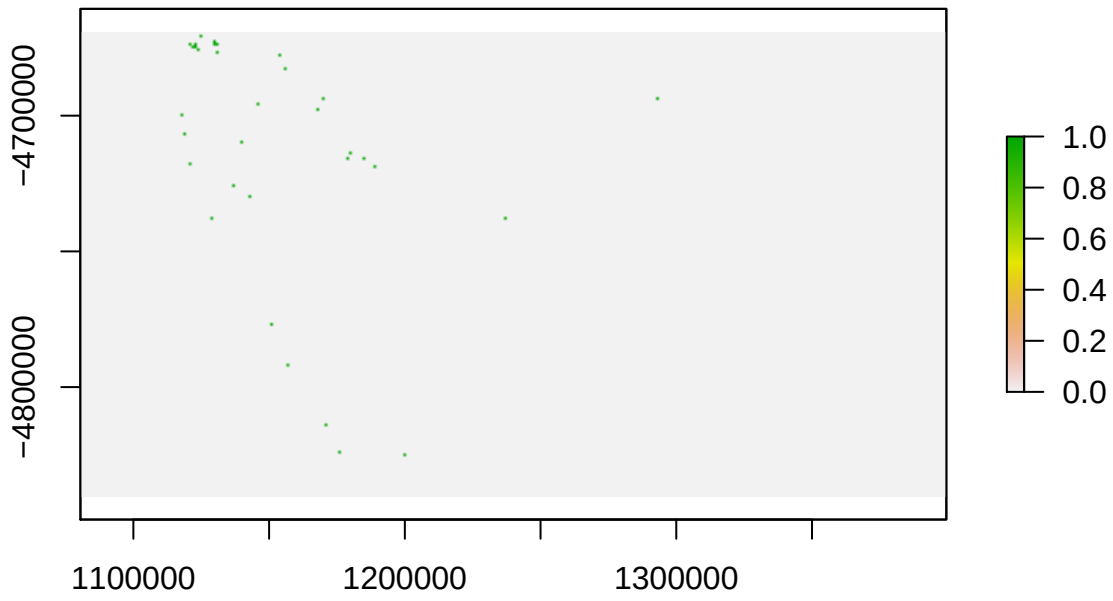


Let's explore the vegetation data.

```
# print a short summary of the data
print(veg_data)
```

```
## class      : RasterStack
## dimensions : 171, 319, 54549, 33  (nrow, ncol, ncell, nlayers)
## resolution : 1000, 1000  (x, y)
## extent     : 1080496, 1399496, -4840217, -4669217  (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps
## names      : vegetation.1, vegetation.2, vegetation.3, vegetation.4, vegetation.5, ve
## min values :            0,            0,            0,            0,            0,
## max values :            1,            1,            1,            1,            1,
```

```
# plot a map of the 20th vegetation class
plot(veg_data[[20]])
```




```
# plot an interactive map of the 20th vegetation class
mapview(veg_data[[20]])
```

```
# print number of rows in the data
nrow(veg_data)
```

```
## [1] 171
```

```
# print number of columns  in the data
ncol(veg_data)
```

```
## [1] 319
```

```
# print number of cells in the data
ncell(veg_data)
```

```
## [1] 54549
```

```
# print number of layers in the data
nlayers(veg_data)
```

```
## [1] 33
```

```
# print  resolution on the x-axis
xres(veg_data)
```

```
## [1] 1000
```

```
# print resolution on the y-axis
yres(veg_data)
```

```
## [1] 1000
```

```
# print spatial extent of the grid, i.e. coordinates for corners
extent(veg_data)
```

```
## class      : Extent
## xmin       : 1080496
## xmax       : 1399496
## ymin       : -4840217
## ymax       : -4669217
```

```
# print the coordinate reference system
print(veg_data@crs)
```

```
## CRS arguments:
##  +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0
## +ellps=GRS80 +units=m +no_defs
```

```
# print a summary of the first layer in the stack
print(veg_data[[1]])
```

```
## class      : RasterLayer
## band       : 1  (of  33  bands)
```

```
## dimensions : 171, 319, 54549  (nrow, ncol, ncell)
## resolution : 1000, 1000  (x, y)
## extent     : 1080496, 1399496, -4840217, -4669217  (xmin, xmax, ymin, ymax)
## crs        : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellps
## source     : /home/travis/build/prioritizr/massey-workshop/data/vegetation.tif
## names      : vegetation.1
## values     : 0, 1  (min, max)
```

```
# print the value in the 800th cell in the first layer of the stack
print(veg_data[[1]][800])
```

```
##
## 0
```

```
# print the value of the cell located in the 30th row and the 60th column of
# the first layer
print(veg_data[[1]][30, 60])
```

```
##
## 0
```

```
# calculate the sum of all the cell values in the first layer
cellStats(veg_data[[1]], "sum")
```

```
## [1] 17
```

```
# calculate the maximum value of all the cell values in the first layer
cellStats(veg_data[[1]], "max")
```

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

```
# calculate the minimum value of all the cell values in the first layer
cellStats(veg_data[[1]], "min")
```

```
## [1] 0
```

```
# calculate the mean value of all the cell values in the first layer
cellStats(veg_data[[1]], "mean")
```

```
## [1] 0.0003116464
```

Now, you can try and answer some questions about the vegetation data.

1. What part of the study area is the 13th vegetation class found in (hint: make a map)? For instance, is it in the south-eastern part of the study area? 2. What proportion of cells contain the 12th vegetation class? 3. Which vegetation class is the most abundant (i.e. present in the greatest number of cells)?

Chapter 4

Gap analysis

4.1 Introduction

Before we begin to prioritize areas for protected area establishment, we should first understand how well existing protected areas are conserving our biodiversity features (i.e. native vegetation classes in Tasmania, Australia). This step is critical: we cannot develop plans to improve conservation of biodiversity if we don't understand how well existing policies are currently conserving biodiversity! To achieve this, we can perform a "gap analysis". A gap analysis involves calculating how well each of our biodiversity features (i.e. vegetation classes in this exercise) are represented (covered) by protected areas. Next, we compare current representation by protected areas of each feature (e.g. 5% of their spatial distribution covered by protected areas) to a target threshold (e.g. 20% of their spatial distribution covered by protected areas). This target threshold denotes the minimum amount (e.g. minimum proportion of spatial distribution) that we need of each feature to be represented in the protected area system. Ideally, targets should be based on an estimate of how much area or habitat is needed for ecosystem function or species persistence. In practice, targets are generally set using simple rules of thumb (e.g. 10% or 20%), policy (17%; https://www.cbd.int/sp/targets/rationale/target-11) or standard practices (e.g. setting targets for species based on geographic range size) [Butchart et al., 2015, Rodrigues et al., 2004].

4.2 Feature abundance

Now we will perform some preliminary calculations to explore the data. First, we will calculate how much of each vegetation feature occurs inside each planning unit (i.e. the abundance of the features). To achieve this, we will use the `problem` function to create an empty conservation planning problem that only contains the planning unit and biodiversity data. We will then use the `feature_abundances` function to calculate the total amount of each feature in each planning unit.

```r
# create prioritizr problem with only the data
p0 <- problem(pu_data, veg_data, cost_column = "cost")

# print empty problem,
# we can see that only the cost and feature data are defined
print(p0)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (557 units)
##   cost:           min: 2.55349, max: 47.23834
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (33 features)
##   objective:      none
##   targets:        none
##   decisions:      default
##   constraints:    <none>
##   penalties:      <none>
##   portfolio:      default
##   solver:         default
```

```r
# calculate amount of each feature in each planning unit
abundance_data <- feature_abundances(p0)

# print abundance data
print(abundance_data)
```

```
## # A tibble: 33 x 3
##    feature       absolute_abundance relative_abundance
##    <chr>                      <dbl>              <dbl>
##  1 vegetation.1                16.                   1
##  2 vegetation.2                12.2                  1
##  3 vegetation.3                11.4                  1
##  4 vegetation.4                17.0                  1
##  5 vegetation.5                11.1                  1
##  6 vegetation.6                15.2                  1
##  7 vegetation.7                27.1                  1
##  8 vegetation.8                16.1                  1
##  9 vegetation.9                18.0                  1
## 10 vegetation.10               19.5                  1
## # ... with 23 more rows
```

```
# note that only the first ten rows are printed,
# this is because the abundance_data object is a tibble (i.e. tbl_df) object
# and not a standard data.frame object
print(class(abundance_data))
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
# we can print all of the rows in abundance_data like this
print(abundance_data, n = Inf)
```

```
## # A tibble: 33 x 3
##    feature       absolute_abundance relative_abundance
##    <chr>                      <dbl>              <dbl>
##  1 vegetation.1                16.                   1
##  2 vegetation.2                12.2                  1
##  3 vegetation.3                11.4                  1
##  4 vegetation.4                17.0                  1
##  5 vegetation.5                11.1                  1
##  6 vegetation.6                15.2                  1
##  7 vegetation.7                27.1                  1
##  8 vegetation.8                16.1                  1
##  9 vegetation.9                18.0                  1
## 10 vegetation.10               19.5                  1
## 11 vegetation.11               22.5                  1
## 12 vegetation.12              852.                   1
## 13 vegetation.13              178.                   1
## 14 vegetation.14               11.2                  1
## 15 vegetation.15               19.2                  1
## 16 vegetation.16               14.1                  1
## 17 vegetation.17              208.                   1
## 18 vegetation.18               14.4                  1
## 19 vegetation.19               16.0                  1
## 20 vegetation.20               17.9                  1
## 21 vegetation.21               17.5                  1
## 22 vegetation.22              292.                   1
## 23 vegetation.23               21.4                  1
## 24 vegetation.24              164.                   1
## 25 vegetation.25              715.                   1
## 26 vegetation.26               25.0                  1
## 27 vegetation.27               17.7                  1
## 28 vegetation.28               17.7                  1
## 29 vegetation.29               24.2                  1
## 30 vegetation.30               59.6                  1
```

```
## 31 vegetation.31              84.0                  1
## 32 vegetation.32              10.0                  1
## 33 vegetation.33              65.0                  1
```

The `abundance_data` object contains three columns. The `feature` column contains the name of each feature (derived from `names(veg_data)`), the `absolute_abundance` column contains the total amount of each feature in all the planning units, and the `relative_abundance` column contains the total amount of each feature in the planning units expressed as a proportion of the total amount in the underlying raster data. Since all the raster cells containing vegetation overlap with the planning units, all of the values in the `relative_abundance` column are equal to one (meaning 100%). Now let's add a new column with the feature abundances expressed in area units (i.e. km^2).

```
# add new column with feature abundances in km^2
abundance_data$absolute_abundance_km2 <-
  (abundance_data$absolute_abundance * prod(res(veg_data))) %>%
  set_units(m^2) %>%
  set_units(km^2)

# print abundance data
print(abundance_data)
```

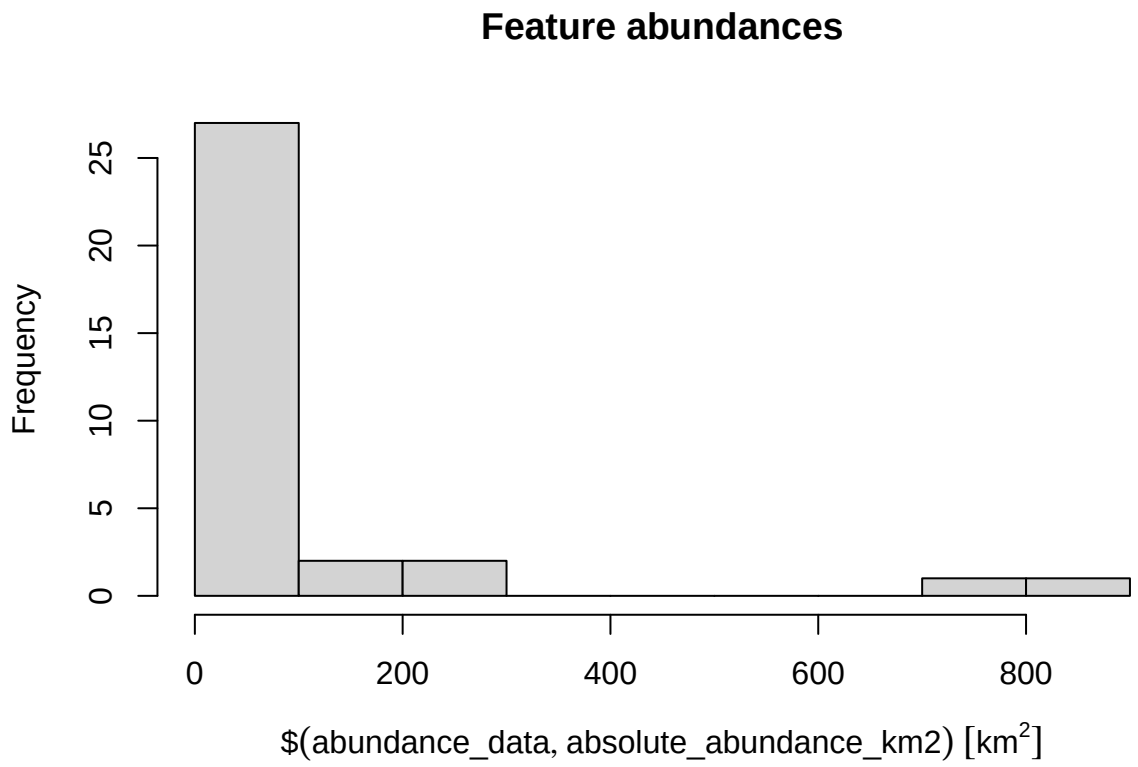
```
## # A tibble: 33 x 4
##    feature       absolute_abundance relative_abundance absolute_abundance_km2
##    <chr>                      <dbl>              <dbl>                  [km^2]
##  1 vegetation.1                16.                 1                16.00000
##  2 vegetation.2                12.2                1                12.23304
##  3 vegetation.3                11.4                1                11.41238
##  4 vegetation.4                17.0                1                16.97727
##  5 vegetation.5                11.1                1                11.11602
##  6 vegetation.6                15.2                1                15.23127
##  7 vegetation.7                27.1                1                27.11938
##  8 vegetation.8                16.1                1                16.13828
##  9 vegetation.9                18.0                1                17.99032
## 10 vegetation.10               19.5                1                19.54535
## # ... with 23 more rows
```

Now let's explore the abundance data.

```
# calculate the average abundance of the features
mean(abundance_data$absolute_abundance_km2)
```

```
## 91.81647 [km^2]
```

```r
# plot histogram of the features' abundances
hist(abundance_data$absolute_abundance_km2, main = "Feature abundances")
```




```r
# find the abundance of the feature with the largest abundance
max(abundance_data$absolute_abundance_km2)
```

```
## 851.9835 [km^2]
```

```r
# find the name of the feature with the largest abundance
abundance_data$feature[which.max(abundance_data$absolute_abundance_km2)]
```

```
## [1] "vegetation.12"
```

Now, try to answer the following questions.

1. What is the median abundance of the features (hint: `median`)? 2. What is the name of the feature with smallest abundance? 3. How many features have a total abundance greater than 100 km^2 (hint: use `sum(abundance_data$absolute_abundance_km2 > set_units(threshold, km^2)` with the correct `threshold` value)?

4.3 Feature representation

After calculating the total amount of each feature in the planning units (i.e. the features' abundances), we will now calculate the amount of each feature in the planning units that are covered by protected areas (i.e. feature representation by protected areas). We can complete this task using the `feature_representation` function. This function requires (i) a conservation problem object with the planning unit and biodiversity data and also (ii) an object representing a solution to the problem (i.e an object in the same format as the planning unit data with values indicating if the planning units are selected or not).

```
# create column in planning unit data with binary values (zeros and ones)
# indicating if a planning unit is covered by protected areas or not
pu_data$pa_status <- as.numeric(pu_data$locked_in)

# calculate feature representation by protected areas
repr_data <- feature_representation(p0, pu_data[, "pa_status"])

# print feature representation data
print(repr_data)
```

```
## # A tibble: 33 x 3
##    feature       absolute_held relative_held
##    <chr>                 <dbl>         <dbl>
##  1 vegetation.1          0             0
##  2 vegetation.2          0             0
##  3 vegetation.3          0             0
##  4 vegetation.4          0             0
##  5 vegetation.5          0             0
##  6 vegetation.6          0             0
##  7 vegetation.7          0             0
##  8 vegetation.8          0             0
##  9 vegetation.9          0.600         0.0334
## 10 vegetation.10         0             0
## # ... with 23 more rows
```

Similar to the abundance data before, the `repr_data` object contains three columns. The `feature` column contains the name of each feature, the `absolute_held` column shows the total amount of each feature held in the solution (i.e. the planning units covered by protected areas), and the `relative_held` column shows the proportion of each feature held in the solution (i.e. the proportion of each feature's spatial distribution held in protected areas). Since the `absolute_held` values correspond to the number of grid cells in the `veg_data` object with overlap with protected areas, let's convert them to area units (i.e. km^2) so we can report them.

```
# add new column with the areas represented in km^2
repr_data$absolute_held_km2 <-
  (repr_data$absolute_held * prod(res(veg_data))) %>%
  set_units(m^2) %>%
  set_units(km^2)

# print representation data
print(repr_data)
```

```
## # A tibble: 33 x 4
##    feature       absolute_held relative_held absolute_held_km2
##    <chr>                 <dbl>         <dbl>            [km^2]
##  1 vegetation.1          0             0              0.0000000
##  2 vegetation.2          0             0              0.0000000
##  3 vegetation.3          0             0              0.0000000
##  4 vegetation.4          0             0              0.0000000
##  5 vegetation.5          0             0              0.0000000
##  6 vegetation.6          0             0              0.0000000
##  7 vegetation.7          0             0              0.0000000
##  8 vegetation.8          0             0              0.0000000
##  9 vegetation.9          0.600         0.0334         0.6001696
## 10 vegetation.10         0             0              0.0000000
## # ... with 23 more rows
```

Now let's investigate how well the species are represented.

1. What is the average proportion of the features held in protected areas (hint: use `mean(table$relative_held)` with the correct `table` name)? 2. If we set a target of 10% coverage by protected areas, how many features fail to meet this target (hint: use `sum(table$relative_held >= target_value)` with the correct `table` name)? 3. If we set a target of 20% coverage by protected areas, how many features fail to meet this target? 4. Is there a relationship between the total abundance of a feature and how well it is represented by protected areas (hint: `plot(abundance_data$absolute_abundance ~ repr_data$relative_held)`)?

Chapter 5

Spatial prioritizations

5.1 Introduction

Here we will develop prioritizations to identify priority areas for protected area establishment. Its worth noting that prioritizr is a decision support tool (similar to Marxan and Zonation). This means that it is designed to help you make decisions—it can't make decisions for you.

5.2 Starting out simple

To start things off, let's keep things simple. Let's create a prioritization using the minimum set formulation of the reserve selection problem. This formulation means that we want a solution that will meet the targets for our biodiversity features for minimum cost. Here, we will set 5% targets for each vegetation class and use the data in the `cost` column to specify acquisition costs. Although we strongly recommend using Gurobi to solve problems (with `add_gurobi_solver`), we will use the lpsymphony solver in this workshop since it is easier to install. The Gurobi solver is much faster than the lpsymphony solver (see here for installation instructions).

```
# print planning unit data
print(pu_data)
```

```
## class       : SpatialPolygonsDataFrame
## features    : 557
## extent      : 1115191, 1385011, -4840595, -4662354  (xmin, xmax, ymin, ymax)
## crs         : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellp
## variables   : 6
## names       :   id,            cost, status, locked_in, locked_out, pa_status
## min values  :  574, 2.5534941249227,      0,         0,          0,         0
## max values  : 1130, 47.238336402701,      2,         1,          1,         1
```

```r
# make prioritization problem
p1 <- problem(pu_data, veg_data, cost_column = "cost") %>%
      add_min_set_objective() %>%
      add_relative_targets(0.05) %>% # 5% representation targets
      add_binary_decisions() %>%
      add_lpsymphony_solver()

# print problem
print(p1)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (557 units)
##   cost:           min: 2.55349, max: 47.23834
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (33 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.05, max: 0.05)]
##   decisions:      Binary decision
##   constraints:    <none>
##   penalties:      <none>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```r
# solve problem
s1 <- solve(p1)

# print solution, the solution_1 column contains the solution values
# indicating if a planning unit is (1) selected or (0) not
print(s1)
```

```
## class       : SpatialPolygonsDataFrame
## features    : 557
## extent      : 1115191, 1385011, -4840595, -4662354  (xmin, xmax, ymin, ymax)
## crs         : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellp
## variables   : 7
## names       :   id,            cost, status, locked_in, locked_out, pa_status, soluti
## min values  :  574, 2.5534941249227,      0,         0,          0,         0,
## max values  : 1130, 47.238336402701,      2,         1,          1,         1,
```

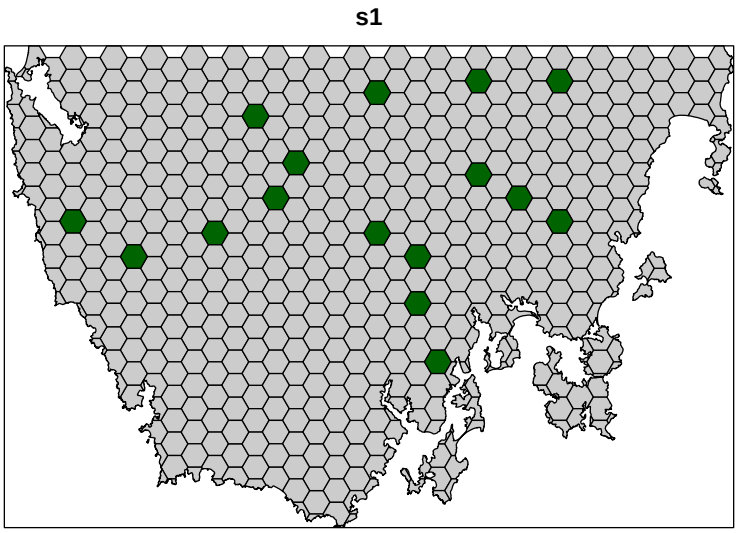
```r
# calculate number of planning units selected in the prioritization
sum(s1$solution_1)
```

```
## [1] 16
```

```
# calculate total cost of the prioritization
sum(s1$solution_1 * s1$cost)
```

```
## [1] 408.3382
```

```
# plot solution
# selected = green, not selected = grey
spplot(s1, "solution_1", col.regions = c("grey80", "darkgreen"), main = "s1",
       colorkey = FALSE)
```



Now let's examine the solution.

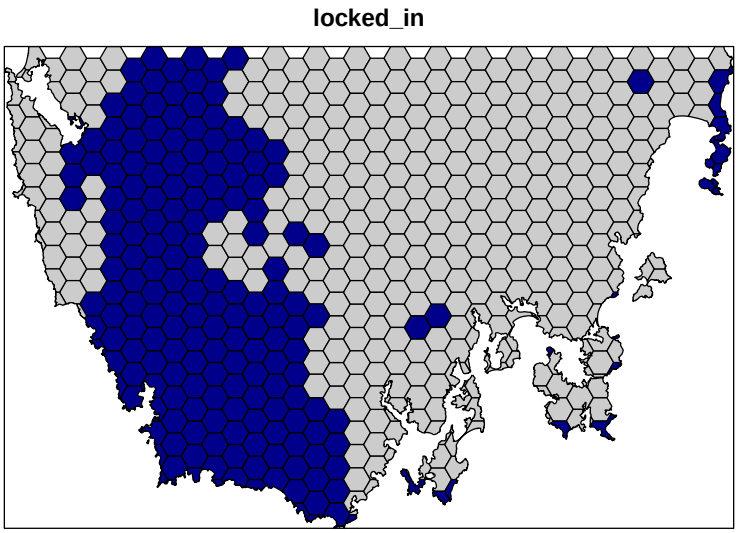
1. How many planing units were selected in the prioritization? What proportion of planning units were selected in the prioritization? 2. Is there a pattern in the spatial distribution of the priority areas? 3. Can you verify that all of the targets were met in the prioritization (hint: `feature_representation(p1, s1[, "solution_1"])`)?

5.3 Adding complexity

Our first prioritization suffers many limitations, so let's add additional constraints to the problem to make it more useful. First, let's lock in planing units that are already by covered protected areas. If some vegetation communities are already secured inside existing protected areas, then we might not need to add as many new protected areas to the existing protected area system to meet their targets. Since our planning unit data (`pu_da`) already contains

this information in the `locked_in` column, we can use this column name to specify which planning units should be locked in.

```
# plot locked_in data
# TRUE = blue, FALSE = grey
spplot(pu_data, "locked_in", col.regions = c("grey80", "darkblue"),
       main = "locked_in", colorkey = FALSE)
```

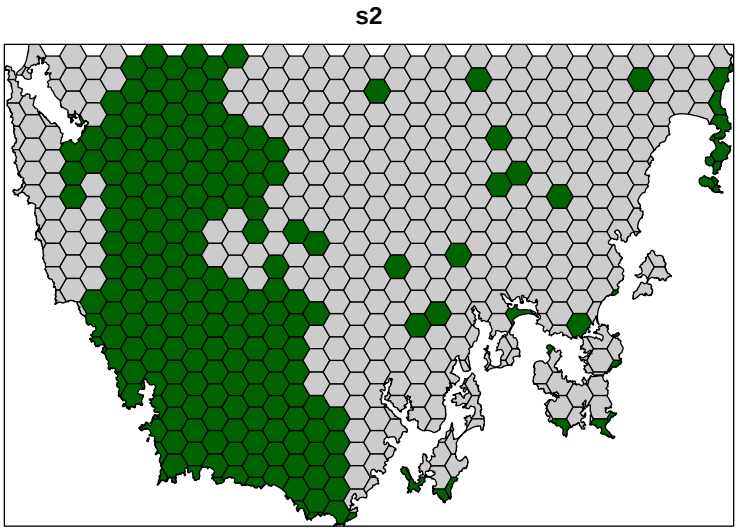



```
# make prioritization problem
p2 <- problem(pu_data, veg_data, cost_column = "cost") %>%
      add_min_set_objective() %>%
      add_relative_targets(0.05) %>%
      add_locked_in_constraints("locked_in") %>%
      add_binary_decisions() %>%
      add_lpsymphony_solver()

# print problem
print(p2)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (557 units)
##   cost:           min: 2.55349, max: 47.23834
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (33 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.05, max: 0.05)]
##   decisions:      Binary decision
##   constraints:    <Locked in planning units [203 locked units]>
##   penalties:      <none>
```

```
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```
# solve problem
s2 <- solve(p2)

# plot solution
# selected = green, not selected = grey
spplot(s2, "solution_1", col.regions = c("grey80", "darkgreen"), main = "s2",
       colorkey = FALSE)
```



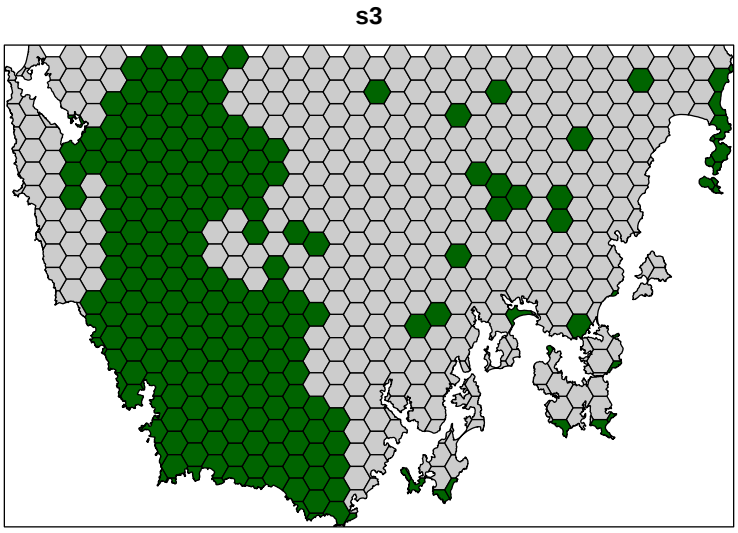
Let's pretend that we talked to an expert on the vegetation communities in our study system and they recommended that a 10% target was needed for each vegetation class. So, equipped with this information, let's set the targets to 10%.

```
# make prioritization problem
p3 <- problem(pu_data, veg_data, cost_column = "cost") %>%
      add_min_set_objective() %>%
      add_relative_targets(0.1) %>%
      add_locked_in_constraints("locked_in") %>%
      add_binary_decisions() %>%
      add_lpsymphony_solver()

# print problem
print(p3)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (557 units)
##   cost:           min: 2.55349, max: 47.23834
```

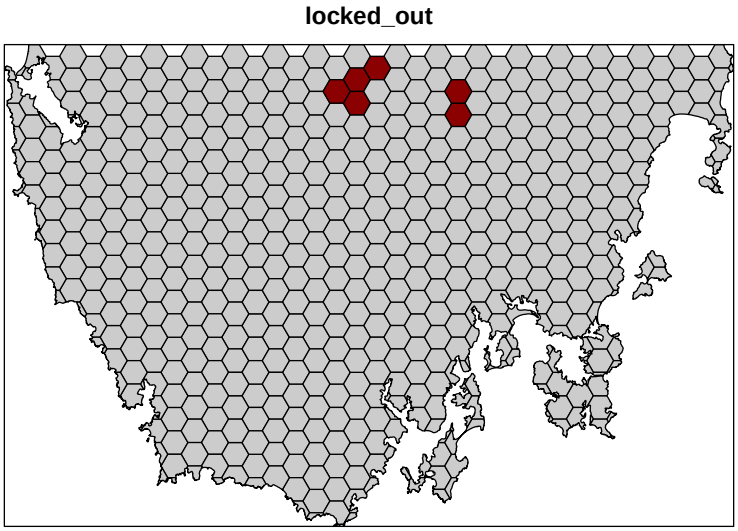
```
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (33 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.1, max: 0.1)]
##   decisions:      Binary decision
##   constraints:    <Locked in planning units [203 locked units]>
##   penalties:      <none>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```
# solve problem
s3 <- solve(p3)

# plot solution
# selected = green, not selected = grey
spplot(s3, "solution_1", col.regions = c("grey80", "darkgreen"), main = "s3",
       colorkey = FALSE)
```



Next, let's lock out highly degraded areas. Similar to before, this information is present in our planning unit data so we can use the `locked_out` column name to achieve this.

```
# plot locked_out data
# TRUE = red, FALSE = grey
spplot(pu_data, "locked_out", col.regions = c("grey80", "darkred"),
       main = "locked_out", colorkey = FALSE)
```



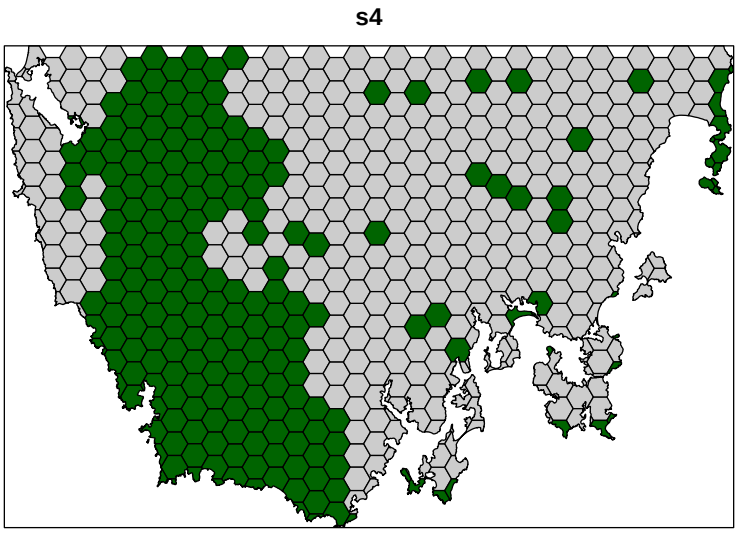

```
# make prioritization problem
p4 <- problem(pu_data, veg_data, cost_column = "cost") %>%
      add_min_set_objective() %>%
      add_relative_targets(0.1) %>%
      add_locked_in_constraints("locked_in") %>%
      add_locked_out_constraints("locked_out") %>%
      add_binary_decisions() %>%
      add_lpsymphony_solver()
```

```
# print problem
print(p4)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (557 units)
##   cost:           min: 2.55349, max: 47.23834
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (33 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.1, max: 0.1)]
##   decisions:      Binary decision
##   constraints:    <Locked out planning units [6 locked units]
##                    Locked in planning units [203 locked units]>
##   penalties:      <none>
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```
# solve problem
s4 <- solve(p4)
```

```
# plot solution
# selected = green, not selected = grey
spplot(s4, "solution_1", col.regions = c("grey80", "darkgreen"), main = "s4",
       colorkey = FALSE)
```



Now, let's compare the solutions.

1. What is the cost of the planning units selected in `s2`, `s3`, and `s4`? 2. How many planning units are in `s2`, `s3`, and `s4`? 3. Do the solutions with more planning units have a greater cost? Why (or why not)? 4. Why does the first solution (`s1`) cost less than the second solution with protected areas locked into the solution (`s2`)? 5. Why does the third solution (`s3`) cost less than the fourth solution solution with highly degraded areas locked out (`s4`)?

5.4 Penalizing fragmentation

Plans for protected area systems should promote connectivity. However, the prioritizations we have made so far have been highly fragmented. To address this issue, we can add penalties to our conservation planning problem to penalize fragmentation. These penalties work by specifying a trade-off between the primary objective (here, solution cost) and fragmentation (i.e. total exposed boundary length) using a penalty value. If we set the penalty value too low, then we will end up with a solution that is nearly identical to the previous solution. If we set the penalty value too high, then prioritizr will (1) take a long time to solve the problem and (2) we will end up with a solution that contains lots of extra planning units that are not needed. This is because the minimizing fragmentation is considered so much more important than solution cost that the optimal solution is simply to select as many planning units as possible.

As a rule of thumb, we generally want penalty values between 0.00001 and 0.01. However, finding a useful penalty value requires calibration. The "correct" penalty value depends on the size of the planning units, the main objective values (e.g. cost values), and the effect of fragmentation on biodiversity persistence. Let's create a new problem that is similar to our previous problem (`p4`)—except that it contains boundary length penalties and a slightly higher optimality gap to reduce runtime (default is 0.1)—and solve it. Since our planning unit data is in a spatial format (i.e. vector or raster data), prioritizr can automatically calculate the boundary data for us.

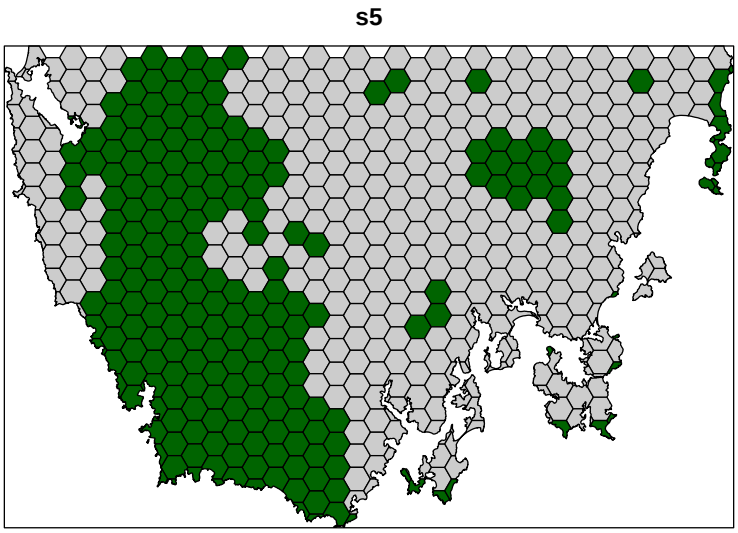
```
# make prioritization problem
p5 <- problem(pu_data, veg_data, cost_column = "cost") %>%
      add_min_set_objective() %>%
      add_boundary_penalties(penalty = 0.001) %>%
      add_relative_targets(0.1) %>%
      add_locked_in_constraints("locked_in") %>%
      add_locked_out_constraints("locked_out") %>%
      add_binary_decisions() %>%
      add_lpsymphony_solver()

# print problem
print(p5)
```

```
## Conservation Problem
##   planning units: SpatialPolygonsDataFrame (557 units)
##   cost:           min: 2.55349, max: 47.23834
##   features:       vegetation.1, vegetation.2, vegetation.3, ... (33 features)
##   objective:      Minimum set objective
##   targets:        Relative targets [targets (min: 0.1, max: 0.1)]
##   decisions:      Binary decision
##   constraints:    <Locked in planning units [203 locked units]
##                    Locked out planning units [6 locked units]>
##   penalties:      <Boundary penalties [edge factor (min: 0.5, max: 0.5), penalty (0.0
##   portfolio:      default
##   solver:         Lpsymphony [first_feasible (0), gap (0.1), time_limit (-1), verbose
```

```
# solve problem,
# note this will take a bit longer than the previous runs
s5 <- solve(p5)

# print solution
print(s5)
```

```
## class       : SpatialPolygonsDataFrame
## features    : 557
## extent      : 1115191, 1385011, -4840595, -4662354  (xmin, xmax, ymin, ymax)
## crs         : +proj=aea +lat_1=-18 +lat_2=-36 +lat_0=0 +lon_0=132 +x_0=0 +y_0=0 +ellp
## variables   : 7
## names       :   id,            cost, status, locked_in, locked_out, pa_status, soluti
## min values  :  574, 2.5534941249227,      0,         0,          0,         0,
## max values  : 1130, 47.238336402701,      2,         1,          1,         1,
```

```
# plot solution
# selected = green, not selected = grey
spplot(s5, "solution_1", col.regions = c("grey80", "darkgreen"), main = "s5",
       colorkey = FALSE)
```



Now let's compare the solutions to the problems with (`s5`) and without (`s4`) the boundary length penalties.

1. What is the cost the fourth (`s4`) and fifth (`s5`) solutions? Why does the fifth solution (`s5`) cost more than the fourth (`s4`) solution? 2. Try setting the penalty value to 0.000000001 (i.e. `1e-9`) instead of 0.0005. What is the cost of the solution now? Is it different from the fourth solution (`s4`) (hint: try plotting the solutions to visualize them)? Is this is a useful penalty value? Why (or why not)? 3. Try setting the penalty value to 0.5. What is the cost of the solution now? Is it different from the fourth solution (`s4`) (hint: try plotting the solutions to visualize them)? Is this a useful penalty value? Why (or why not)?

Chapter 6

Acknowledgements

Many thanks to Icons8 for providing the icons used in this manual and to Yihui Xie for developing the bookdown R package that underpins this manual. We also thank Garrett Grolemund and Hadley Wickham for creating one of the Rstudio screenshots used in this manual that was originally a part of their *R for Data Science* book.

Chapter 7

Session information

```
# print session information
sessionInfo()
```

```
## R version 4.0.0 (2020-04-24)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 16.04.6 LTS
##
## Matrix products: default
## BLAS:   /home/travis/R-bin/lib/R/lib/libRblas.so
## LAPACK: /home/travis/R-bin/lib/R/lib/libRlapack.so
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] dplyr_1.0.0      gridExtra_2.3    assertthat_0.2.1 scales_1.1.1
##  [5] units_0.6-7      rgeos_0.5-3      rgdal_1.5-12     prioritizr_5.0.1
##  [9] proto_1.0.0      sf_0.9-5         raster_3.3-13    sp_1.4-2
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_1.0.5          compiler_4.0.0      pillar_1.4.6
```

```
##  [4] class_7.3-16        tools_4.0.0         uuid_0.1-4
##  [7] digest_0.6.25       gtable_0.3.0        evaluate_0.14
## [10] lifecycle_0.2.0     tibble_3.0.3        lattice_0.20-41
## [13] pkgconfig_2.0.3     rlang_0.4.7         Matrix_1.2-18
## [16] lpsymphony_1.16.0   cli_2.0.2           DBI_1.1.0
## [19] parallel_4.0.0      yaml_2.2.1          xfun_0.15
## [22] e1071_1.7-3         exactextractr_0.4.0 stringr_1.4.0
## [25] knitr_1.29          generics_0.0.2      vctrs_0.3.2
## [28] classInt_0.4-3      grid_4.0.0          tidyselect_1.1.0
## [31] glue_1.4.1          R6_2.4.1            fansi_0.4.1
## [34] rmarkdown_2.3       bookdown_0.20.1     purrr_0.3.4
## [37] magrittr_1.5        codetools_0.2-16    htmltools_0.5.0
## [40] ellipsis_0.3.1      colorspace_1.4-1    utf8_1.1.4
## [43] KernSmooth_2.23-16  stringi_1.4.6       munsell_0.5.0
## [46] crayon_1.3.4
```

Chapter 8

References

Bibliography

Stuart H.M. Butchart, Martin Clarke, Robert J. Smith, Rachel E. Sykes, Jörn P.W. Scharlemann, Mike Harfoot, Graeme M. Buchanan, Ariadne Angulo, Andrew Balmford, Bastian Bertzky, Thomas M. Brooks, Kent E. Carpenter, Mia T. Comeros-Raynal, John Cornell, G. Francesco Ficetola, Lincoln D.C. Fishpool, Richard A. Fuller, Jonas Geldmann, Heather Harwell, Craig Hilton-Taylor, Michael Hoffmann, Ackbar Joolia, Lucas Joppa, Naomi Kingston, Ian May, Amy Milam, Beth Polidoro, Gina Ralph, Nadia Richman, Carlo Rondinini, Daniel B. Segan, Benjamin Skolnik, Mark D. Spalding, Simon N. Stuart, Andy Symes, Joseph Taylor, Piero Visconti, James E.M. Watson, Louisa Wood, and Neil D. Burgess. Shortfalls and solutions for meeting national and global conservation area targets. *Conservation Letters*, 8(5):329–337, 2015.

Ana S. L. Rodrigues, H. Resit Akçakaya, Sandy J. Andelman, Mohamed I. Bakarr, Luigi Boitani, Thomas M. Brooks, Janice S. Chanson, Lincoln D. C. Fishpool, Gustavo A. B. Da Fonseca, Kevin J. Gaston, Michael Hoffmann, Pablo A. Marquet, John D. Pilgrim, Robert L. Pressey, Jan Schipper, Wes Sechrest, Simon N. Stuart, Les G. Underhill, Robert W. Waller, Matthew E. J. Watts, and Xie Yan. Global gap analysis: priority regions for expanding the global protected-area network. *BioScience*, 54(12):1092–1100, 2004.