EDS 223: week 1 lab

Ruth Oliver

2022-10-03

introduction

In this lab, we'll explore the basics of map-making in R using the **tmap** package. The following materials are modified from Chapter 9 of Geocomputation with R by Rovin Lovelace

prerequisites

Let's install all the necessary packages to reproduce code from Geocomp with R.

```r
#install.packages("spDataLarge", repos = "https://geocompr.r-universe.dev")
#install.packages("remotes")
#remotes::install_github("geocompx/geocompkg")
```

Let's load all necessary packages.

```r
library(sf)
library(terra)
library(dplyr)
library(spData)
library(spDataLarge)
remotes::install_github("r-tmap/tmap@v4")
library(tmap)    # for static and interactive maps
library(leaflet) # for interactive maps
library(ggplot2) # tidyverse data visualization package
```

Let's also read in some data from the **spDataLarge** package to work with later.

```r
nz_elev = rast(system.file("raster/nz_elev.tif", package = "spDataLarge"))
```

map-making basics

Let's start with a pre-loaded spatial object representing the states of New Zealand

```r
nz
```

- we're going to start by using the **tmap** package to make some basic maps
- **tmap** can work with spatial objects of a variety of clasess, meaning it's highly versatile
- this approach relies on a series of functions that typically start with "tm_"

the first element is always “tm_shape”

```
tm_shape(nz) +
  tm_fill()
```

now let’s plot just the boundaries

```
tm_shape(nz) +
  tm_borders()
```

and the shapes and boundaries together

```
tm_shape(nz) +
  tm_fill() +
  tm_borders()
```

map objects

- **tmap** can store maps as *objects*
- this means that we store a base map and add additional layers later

```
map_nz <- tm_shape(nz) +
  tm_polygons()
class(map_nz)
```

```
## [1] "tmap"
```

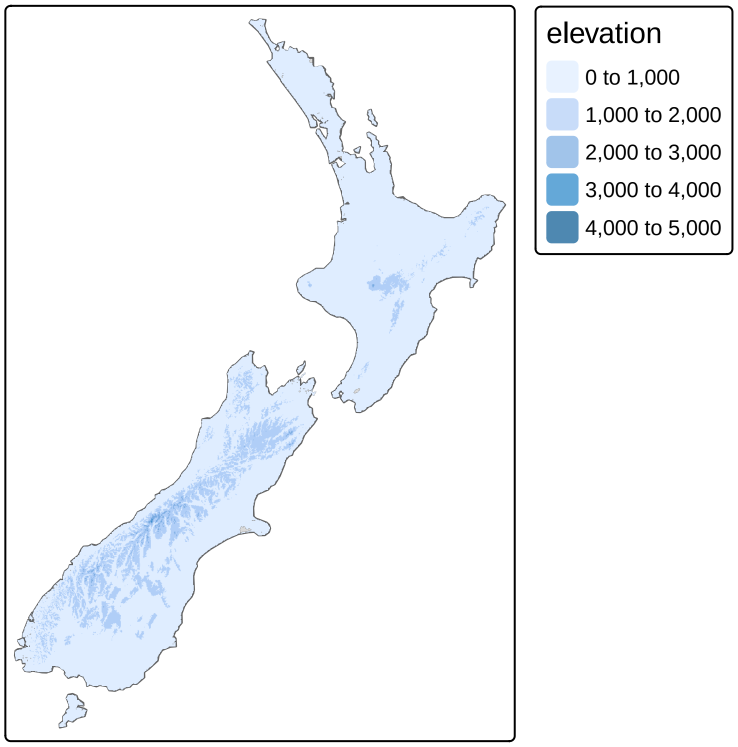
- we can add new shapes on top of the base map
- when we add a new shape, all subsequent aesthetic functions refer to it, until we add a new shape

In this case, we’re adding a layer with information on elevation and this layer to have 70% transparency.

```
map_nz1 <- map_nz +
  tm_shape(nz_elev) +
  tm_raster(col_alpha = 0.7)

map_nz1
```

```
## SpatRaster object downsampled to 1141 by 877 cells.
```

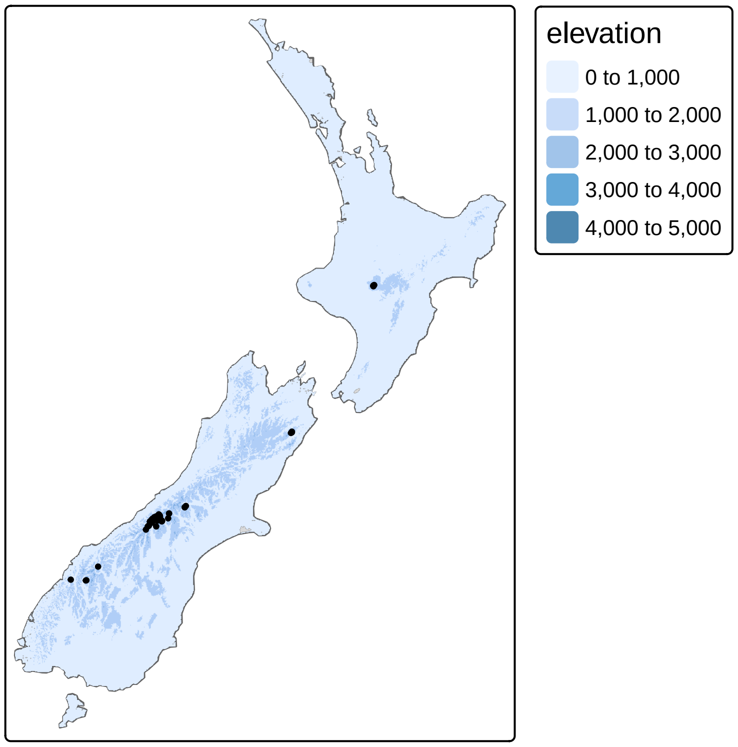


we can add points designating high points in the country

```
map_nz2 <- map_nz1 +
  tm_shape(nz_height) +
  tm_dots()

map_nz2
```

```
## SpatRaster object downsampled to 1141 by 877 cells.
```



aesthetic basics

- up until now, we've been working with the default aesthetics
- there are 2 types of aesthetics: fixed and those that change with the value of a variable
- **tmap** works differently than **ggplot2** and doesn't use the "aes()" function

Let's start by changing some fixed aesthetics... First, let's change the color used to fill the NZ shapes.

```
tm_shape(nz) +
  tm_polygons(fill = "red")
```

now change the color of the boundaries

```
tm_shape(nz) +
  tm_polygons(col = "blue")
```

and the width of the boundary lines

```
tm_shape(nz) +
  tm_polygons(lwd = 3)
```

and the line type of the boundary lines

```r
tm_shape(nz) +
  tm_polygons(lty = 2)
```

all together now!

```r
tm_shape(nz) +
  tm_polygons(fill = "red",
              fill_alpha = 0.3,
              col = "blue",
              lwd = 3,
              lty = 2)
```

Now let's change the colors based on a value. We noticed that the New Zealand dataset has a column with each state's land area

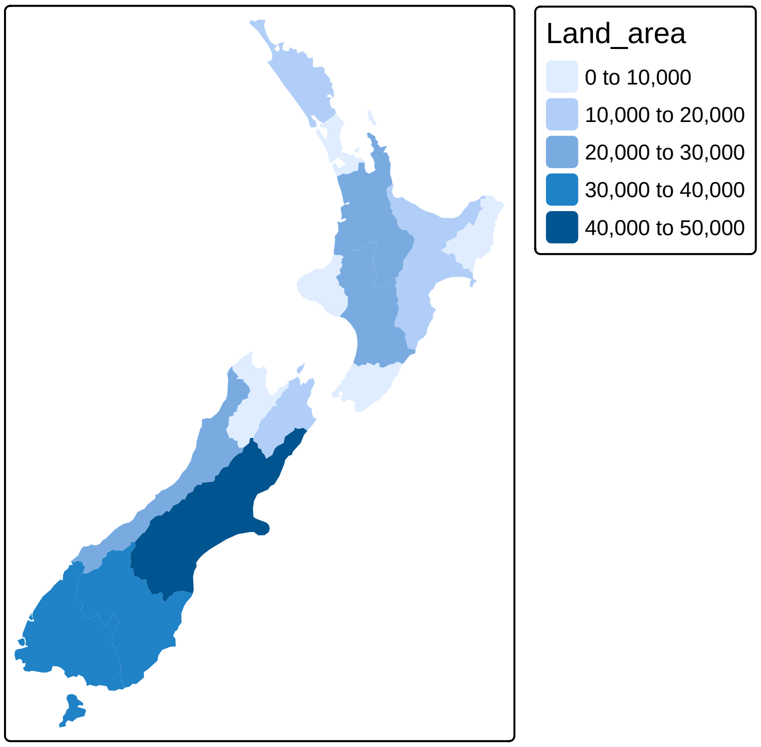
```r
nz
```

Let's try to plot the Land_area column. We might think that the following works, but it doesn't!

```r
#tm_shape(nz) +
#  tm_fill(col = nz$Land_area)
```

Instead, **tmap** is expecting a character string naming the attribute associated with the geometry

```r
tm_shape(nz) +
  tm_fill(fill = "Land_area")
```



We can also add an argument that updates the title of the legend

```r
tm_shape(nz) +
  tm_fill(fill = "Land_area", title = "Area")
```

We can even make it more precise using the “expression” function

```r
tm_shape(nz) +
  tm_fill(fill = "Land_area", title = expression("Area (km"^2*")")) +
  tm_borders()
```

color settings

- how we set and define colors can radically change the interpretation of our maps
- the style argument has several options for breaking data into bins
  - **style = “pretty”** (default) rounds breaks into evenly spaced whole numbers, where possible
  - **style = “equal”** divides input values into bins with equal range (best for uniform distributions)
  - **style = “quantile”** puts the same number of observations into each bin
  - **style = “jenks”** identifies groups of similar values and maximizes differences between bins

Notice how the following maps display the same data, but look quite different

```r
tm_shape(nz) +
  tm_polygons(fill = "Median_income", style = "pretty")
```

```r
tm_shape(nz) +
  tm_polygons(fill = "Median_income", style = "equal")
```

```r
tm_shape(nz) +
  tm_polygons(fill = "Median_income", style = "quantile")
```

```r
tm_shape(nz) +
  tm_polygons(fill = "Median_income", style = "jenks")
```

We can also define custom bins

```r
breaks = c(0, 3, 4, 5) * 10000
tm_shape(nz) +
  tm_polygons(fill = "Median_income", breaks = breaks)
```

- in some cases we might not want to use bins
  - **style = “cont”** displays colors as a continuous spectrum
  - **style = “cat”** uses a unique vale for each categorical value

```r
map_nz +
  tm_shape(nz_elev) +
  tm_raster(col_alpha = 0.7) +
  tm_scale_continuous()
```

```r
map_nz +
  tm_shape(nz) +
  tm_polygons(fill = "Island")
```

map layout

- now that we have the basics, we can turn to all the other elements that make a cohesive map
- **tmap** has lots of options, but we will explore just a few

To clearly give readers the context of our map, we can include a compass and scale bar

```r
map_nz +
  tm_compass(type = "4star", position = c("left", "top")) +
  tm_scale_bar(breaks = c(0, 100, 200), text.size = 1)
```

Instead of using a compass and scale bar, we could add latitude/longitudes graticules

```r
map_nz +
  tm_graticules()
```

We can also update the background color

```r
map_nz +
  tm_graticules() +
  tm_layout(bg.color = "lightblue")
```

faceted and animated maps

- we might have data that varies over time and we want to look at the how it changes
- one approach is by using faceted plots, a series of plots

```r
urb_1970_2030 <- urban_agglomerations %>%
  filter(year %in% c(1970, 1990, 2010, 2030))

tm_shape(world) +
  tm_polygons() +
  tm_shape(urb_1970_2030) +
  tm_symbols(col = "black", border.col = "white", size = "population_millions") +
  tm_facets(by = "year", nrow = 2, free.coords = FALSE)
```

interactive maps

- **tmap** is especially powerful because it allows us to make interactive maps using the same syntax
- all we need to do is enter the interactive mode

```r
tmap_mode("view")
map_nz
```

To go back to regular plotting, we just need enter plotting mode

```r
tmap_mode("plot")
```