ggplot: spatial

Olivia Burge

12 October 2014

This tutorial is an extension of the introduction to ggplot tutorials (https://github.com/orb16/seminaRs/tree/master/session_6_ggplot), and introduces basic mapping skills. It's actually package-agnostic and if base graphics are your thing, you're catered for too.

First, we cover making a map (in this case of NZ) in base graphics and ggplot and save with/without a transparent background. Transparent backgrounds are very useful for posters which have a coloured background. Then we cover adding points and layers to our base maps.

This is just the beginning of course. You can make inset maps (http://r-nold.blogspot.co.nz/2014/06/creating-inset-map-with-ggplot2.html) and check out the `ggmap` package, particularly the stamen maps.

Maps: base plot and ggplot

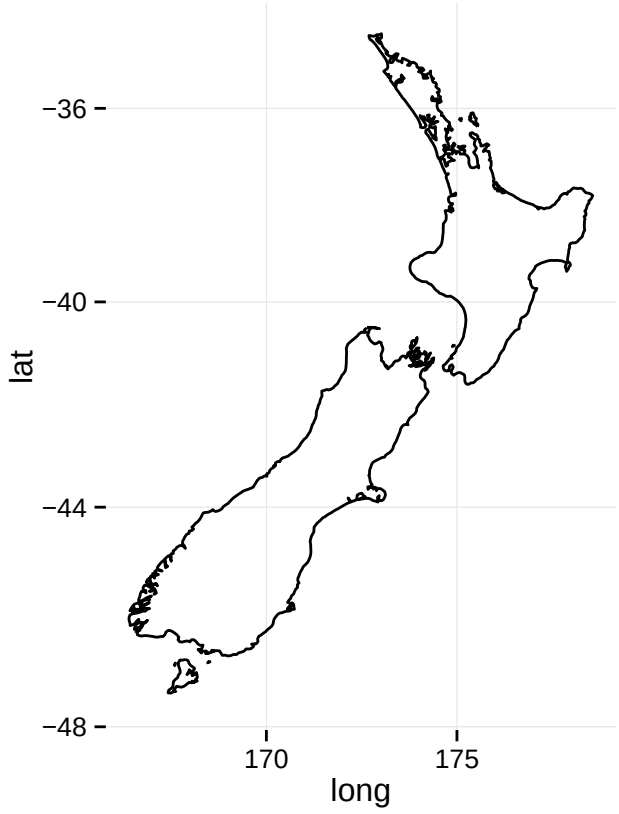
```
map('nz')
```

```
# or:
nzmap <- map_data('nz')
head(nzmap)
```

```
##    long    lat group order        region subregion
## 1 172.7 -34.44     1     1 North.Island       <NA>
## 2 172.8 -34.46     1     2 North.Island       <NA>
## 3 172.9 -34.45     1     3 North.Island       <NA>
## 4 172.9 -34.42     1     4 North.Island       <NA>
## 5 173.0 -34.43     1     5 North.Island       <NA>
## 6 173.0 -34.40     1     6 North.Island       <NA>
```

Now for ggplot - try the first map

```
map1 <- ggplot(nzmap, aes(x = long, y = lat, group = group)) +
  geom_polygon()
map1

# hmmm. change polygon to path, and add coord_map

map2 <- ggplot(nzmap, aes(x = long, y = lat, group = group)) +
  geom_path() +
  coord_map() +
  theme_minimal()
map2
```

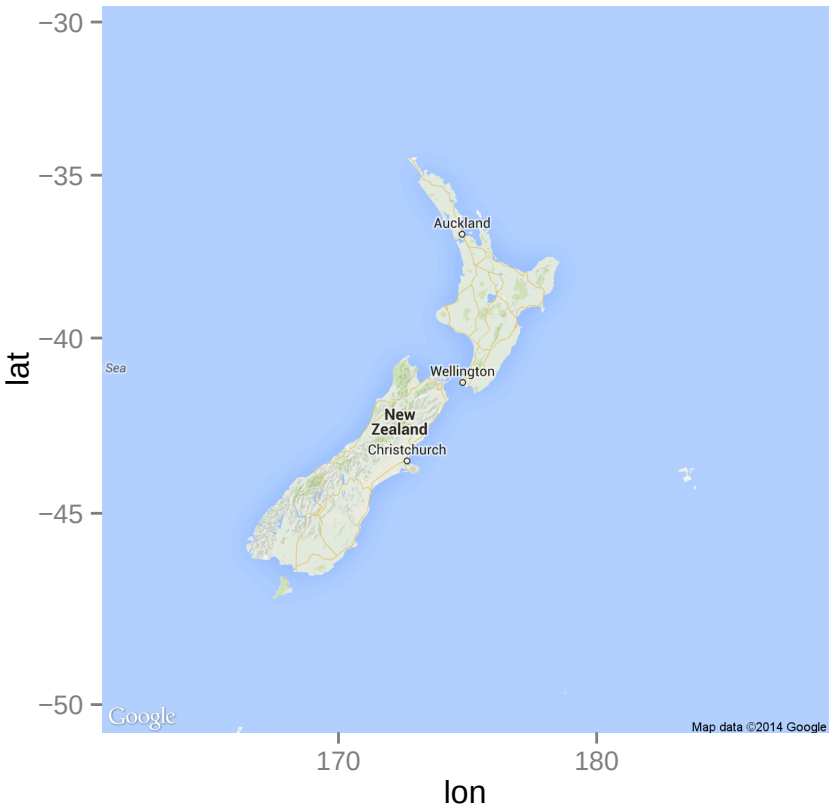


Next up is getting the google map:

```
ggnz<-get_map('nz', source='google', maptype='road',
              zoom = 5)
```

```
## Map from URL : http://maps.googleapis.com/maps/api/staticmap?center=nz&zoom=5&size=%20640x640&scale=
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
## Information from URL : http://maps.googleapis.com/maps/api/geocode/json?address=nz&sensor=false
## Google Maps API Terms of Service : http://developers.google.com/maps/terms
```

```
map3 <-ggmap(ggnz)
map3
```



Adding location data to maps:

Firstly, we get some data. We are first trying the RCurl package to get the data off a github site (mine, in fact). It's freely available on the web, but I've added it here for ease of access.

If this doesn't work, uncomment the lines below where the data is entered manually (and correctly, hopefully!). Note that if you highlight the commented lines and go up to the "Code" menu option you can 'uncomment' them all at once.

data

```
# get data

url_kakariki <- getURL("https://raw.githubusercontent.com/orb16/seminaRs/master/session_6_ggplot/raw_da
    ssl.verifypeer = FALSE)

awaraikakariki <- read.csv(text = url_kakariki, header = TRUE, stringsAsFactors = FALSE,
    fileEncoding = "latin1")
awaraikakariki <- awaraikakariki[c("site", "lat", "lon")]

url_ramsar <- getURL("https://raw.githubusercontent.com/orb16/seminaRs/master/session_6_ggplot/raw_data_
    ssl.verifypeer = FALSE)

ramsar <- read.table(text = url_ramsar, sep = ",", stringsAsFactors = FALSE,
    fileEncoding = "latin1", header = T)
```

```
ramsar <- ramsar[c("site", "lat", "lon")]

# awaraikakariki <- data.frame(site = c('Ashburton lakes', 'Awarua-Waituna
# Wetland', 'Whangamarino'), lat = c(-43.56417, -46.56667, -37.30000), lon =
# c(171.1892, 168.5167, 175.1167))

# ramsar <- data.frame( site = c('Awarua Wetland (formerly Waituna Lagoon)',
# 'Farewell Spit', 'Whangamarino', 'Kopuatai Peat Dome', 'Firth of Thames',
# 'Manawatu river mouth and estuary'), lat = c(-46.56666667, -40.53333333,
# -37.3, -37.43333333, -37.21666667, -40.48333333), lon = c(168.5166667,
# 172.8333333, 175.1166667, 175.55, 175.3833333, 175.2333333) )
```

base graphics method

This is going to save your map as "foo.png" to whichever working directory you have (NB, it will likely be where you've saved this .Rmd file, unless you set it explicitly.) If you are knitting this file, you'll need to delete the `eval=FALSE` in the curly brackets to make it actually save.

```
png("foo.png", height= 640, width= 480)
map('nz')
points(x = ramsar$lon, y = ramsar$lat, pch = 16, col="slateblue4")
dev.off()

# prints with a white background.
# try making a transparent one adding "bg = transparent" to png()
```

ggplot method

And with ggplot. Here we `rbind` the data. This is like `merge`, but since we have all the same column names and number of columns, we can `rbind`. `rbind` is like gluing one dataset below the other, and just dropping one of the header columns (hence having to have same number and names of cols).

```
ramsar$data <- "Ramsar wetlands"
awaraikakariki$data <- "Arawai kakariki wetlands"
```

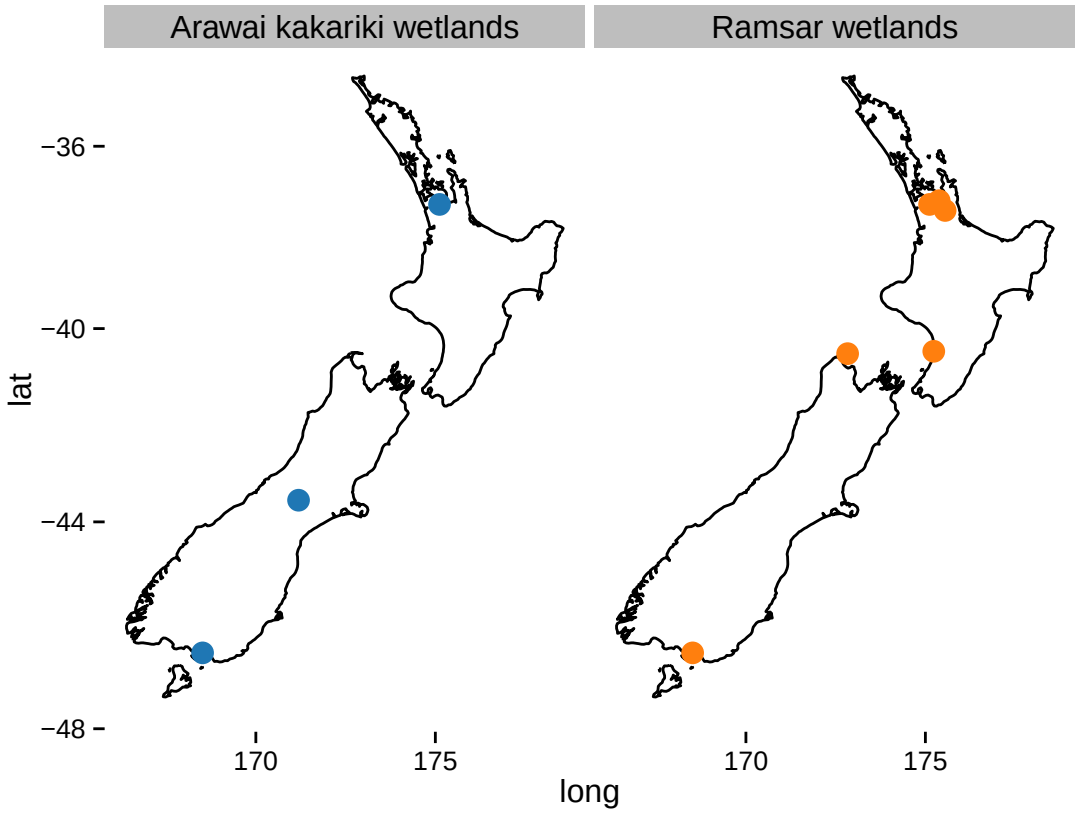
This part above makes a new factor in each dataset, before we `rbind` them. Have a look to see the before and after.

```
nzwetlands <- rbind(ramsar, awaraikakariki)
```

The map: a lot of code. But if you go through line by line, nothing too scary. The `facet_wrap()` is what makes two maps, splitting on the variable we created earlier. All the `theme` stuff at the end is mainly to make it transparent, and to get rid of the lines behind the plot. `strip.text` changes the size of the text in the headers at the top.

Note: we can combine different dataframes into one plot by specifying the datasource manually for some layers. We do that here with `geom_point()`, which uses the `nzwetlands` dataframe. Whereas the default for other layers is the `nzmap` data (first line).

```
map_bg <- ggplot(nzmap, aes(x = long, y = lat)) +
   geom_path(aes(group = group)) +
   coord_map() +
   geom_point(data = nzwetlands,
              size = 4,
              aes(x = lon, y = lat, colour = data)) +
   scale_colour_tableau(guide = FALSE) +
   facet_wrap(~ data) +
   theme_minimal() +
   theme(strip.text = element_text(size = 12),
         strip.background = element_rect(fill = "grey", colour = NA),
         panel.grid.major = element_blank(),
         panel.grid.minor = element_blank(),
         panel.background = element_rect(fill = "transparent",
                                         colour = NA),
         plot.background = element_rect(fill = "transparent",
                                        colour = NA))

#ggsave(plot = map_bg, "map_bg.png", bg = "transparent")
map_bg
```




```
#this is for a transparent map.
#Delete all the references to transparent to get a white bg.
```

Plotting shapefile data

To run this part of the tutotorial you'll need the packages listed at the start installed, and the raw data from koordinates.com/

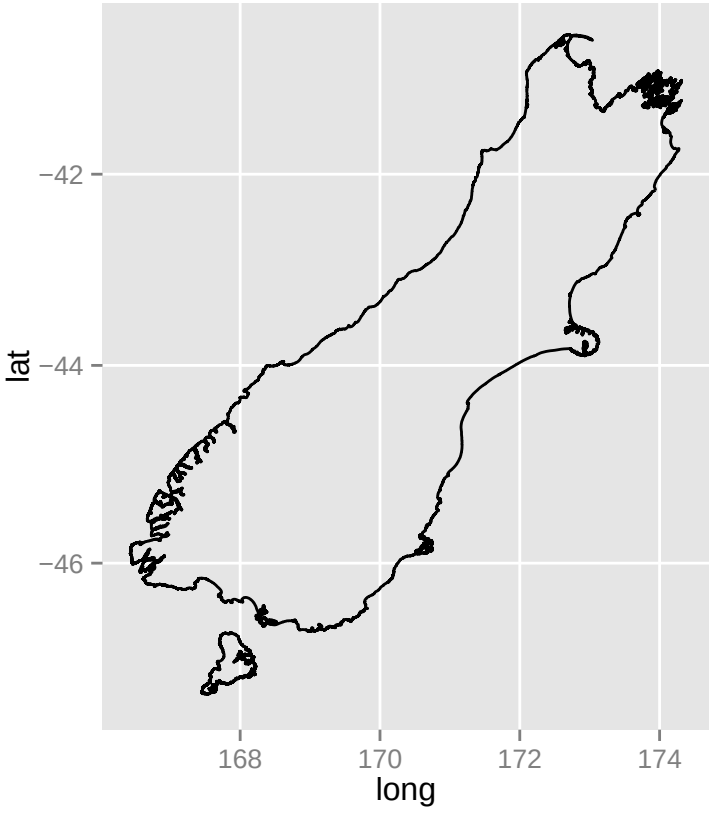
the forests

From: https://koordinates.com/layer/301-nz-fsms6-south-island/

Polygon of the South Island

Next up we plot just the South Island, and Stewart Island (this is what the `coast[]` part is doing)

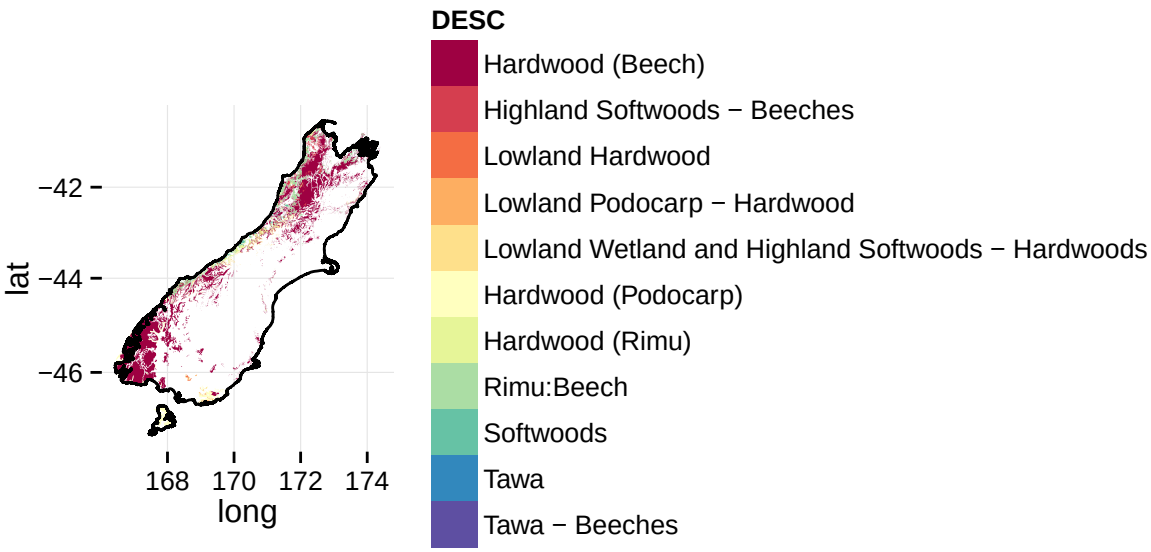
```
#base layer and forest data are from
# koordinates.com - worth a look, quite a bit of data

base_nz<-ggplot(data = coast[coast$id %in% c(2, 0),], aes(x=long, y = lat))+
  geom_path(aes(group = group)) +
  coord_map()
base_nz
```



Forest polygons

Now that that works, we can overlay the polygons. This takes ages, hence getting the map together first, and also, save other things before you run this.. You have been warned!

```
(base_forest <-  ggplot(data = coast[coast$id %in% c(2, 0),],
                        aes(x=long, y = lat))+
  theme_minimal() +
  geom_polygon(data = forests_df,
               aes(y = lat, x = long, group = group, fill = DESC)) +
  geom_path(aes(group = group)) +
  coord_map()+
  scale_fill_brewer(palette = "Spectral"))
```




```
  #ggsave(base_forest, file = "base_forest.pdf",
  #       width = 210, height = 297, units = "mm")

# can do other things - source some contours, for example.
```