Introduction to maps with R and ggplot

Robert Cope

This workshop will be a practical introduction to the basics of making maps in R. Maps can be complex and often seem annoying to work with, but with a little familiarity with R you can create useful and attractive maps quickly and easily. We will focus on some common use cases and not assume anything beyond ggplot2 basics covered previously in this series.

Maps

Maps are incredibly useful in any part of science that deals with physical space, the environment, populations of humans or animals, mobility, etc.. However we may not have training in how to create them – this certainly wasn't part of my undergraduate training in maths/stats, or something I learned as a PhD student in ecology. Fortunately when you are a little familiar with R, and have some understanding of how different types of map data works, it can be relatively straightforward to create maps that are both useful and attractive.

One challenge is that there are many different packages available that each do different things, and new packages have been introduced over time. The aim of this workshop is to introduce a few such packages in a practical way. While this may not be exhaustive, it provides a starting point from which you can proceed.

The workshop assumes some level of comfort with ggplot2, and will make use of the tidyverse packages generally for convenience.

Preamble

We need some packages, and the workshop data.

```r
download.file('https://github.com/robert-cope/RmapsWorkshopData/blob/master/data.zip?raw=true',
              destfile='data.zip')
unzip('data.zip')

#Please try to work on your own computer rather than the server
#and if you are asked about installing from source, choose no

install.packages('ggmap')
install.packages('tidyverse')
install.packages('ggrepel')
install.packages('leaflet')

install.packages('sf')

install.packages('raster')
install.packages('rgdal')
```

A base map: ggmap

An easy starting point for producing nice maps is the ggmap packages.

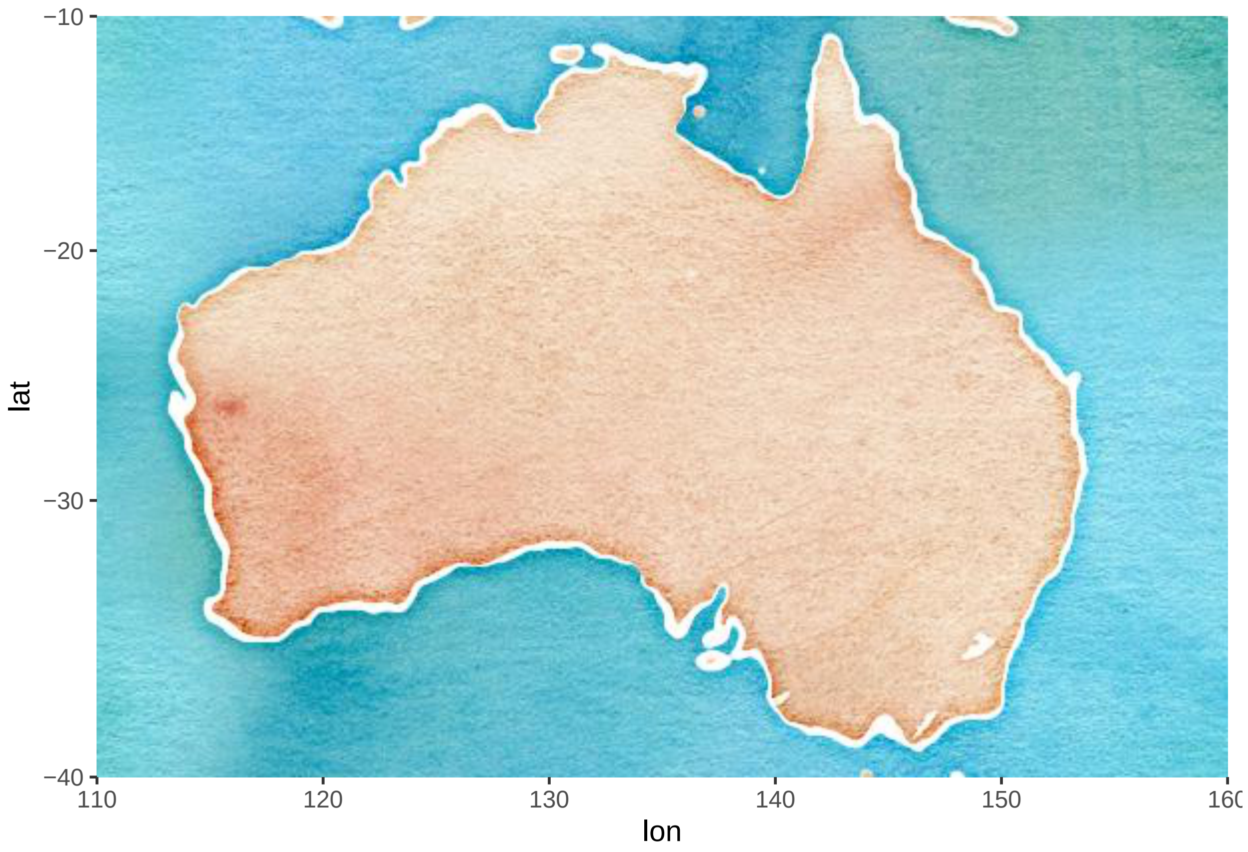
```r
library(ggmap)
```

```
## Loading required package: ggplot2

## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.

## Please cite ggmap if you use it! See citation("ggmap") for details.
```

```r
m1 <- get_stamenmap( bbox = c(left = 110, bottom = -40, right = 160, top = -10),
                     zoom = 4, maptype = "watercolor")
```

```
## Source : http://tile.stamen.com/watercolor/4/12/8.jpg

## Source : http://tile.stamen.com/watercolor/4/13/8.jpg

## Source : http://tile.stamen.com/watercolor/4/14/8.jpg

## Source : http://tile.stamen.com/watercolor/4/15/8.jpg

## Source : http://tile.stamen.com/watercolor/4/12/9.jpg

## Source : http://tile.stamen.com/watercolor/4/13/9.jpg

## Source : http://tile.stamen.com/watercolor/4/14/9.jpg

## Source : http://tile.stamen.com/watercolor/4/15/9.jpg
```

```r
ggmap(m1)
```



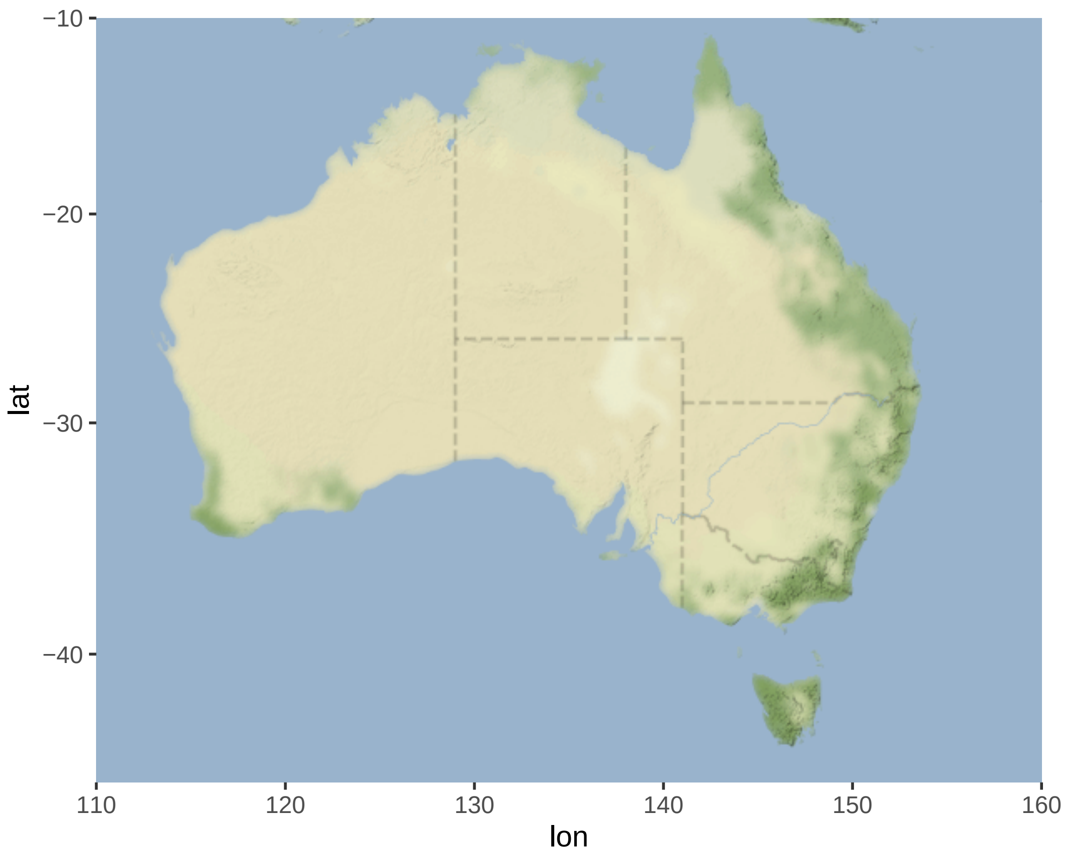
Task 1: Tasmania is missing. Edit your map to include it. (bottom = -45 does this)

There are different map background images available. The help `?get_stamenmap` gives you the different map type options.

```r
m2 <- get_stamenmap( bbox = c(left = 110, bottom = -45, right = 160, top = -10),
                     zoom = 4, maptype = "terrain-background")
```

```
## Source : http://tile.stamen.com/terrain-background/4/12/8.png

## Source : http://tile.stamen.com/terrain-background/4/13/8.png

## Source : http://tile.stamen.com/terrain-background/4/14/8.png

## Source : http://tile.stamen.com/terrain-background/4/15/8.png

## Source : http://tile.stamen.com/terrain-background/4/12/9.png

## Source : http://tile.stamen.com/terrain-background/4/13/9.png

## Source : http://tile.stamen.com/terrain-background/4/14/9.png

## Source : http://tile.stamen.com/terrain-background/4/15/9.png

## Source : http://tile.stamen.com/terrain-background/4/12/10.png

## Source : http://tile.stamen.com/terrain-background/4/13/10.png

## Source : http://tile.stamen.com/terrain-background/4/14/10.png

## Source : http://tile.stamen.com/terrain-background/4/15/10.png
```

```r
ggmap(m2)
```



Historically, you could use different map sources, e.g., openstreetmap. You can also use google maps, but this requires an API key which you have to set up yourself. There are instructions how to do this if you search for ggmap in google. The google API also gives you access to geocoding, i.e., when you ask for “Australia” or “Canberra” it will give you coordinates.

The simplest case: adding points.

Looking for a nice example dataset I came accross this paper

Westgate, M. J., Scheele, B. C., Ikin, K., Hoefer, A. M., Beaty, R. M., Evans, M., … & Driscoll, D. A. (2015). Citizen science program shows urban areas have lower occurrence of frog species, but not accelerated declines. *PloS one*, 10(11), e0140973.

https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0140973

https://datadryad.org/stash/dataset/doi:10.5061/dryad.75s51

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------- tidyverse 1.3.0 --
```

```
## v tibble  3.0.2     v dplyr   1.0.0
## v tidyr   1.1.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.5.0
## v purrr   0.3.4
```

```
## -- Conflicts ------------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
sites <- read_csv('data/Frogwatch/Frogwatch_sites.csv')
```

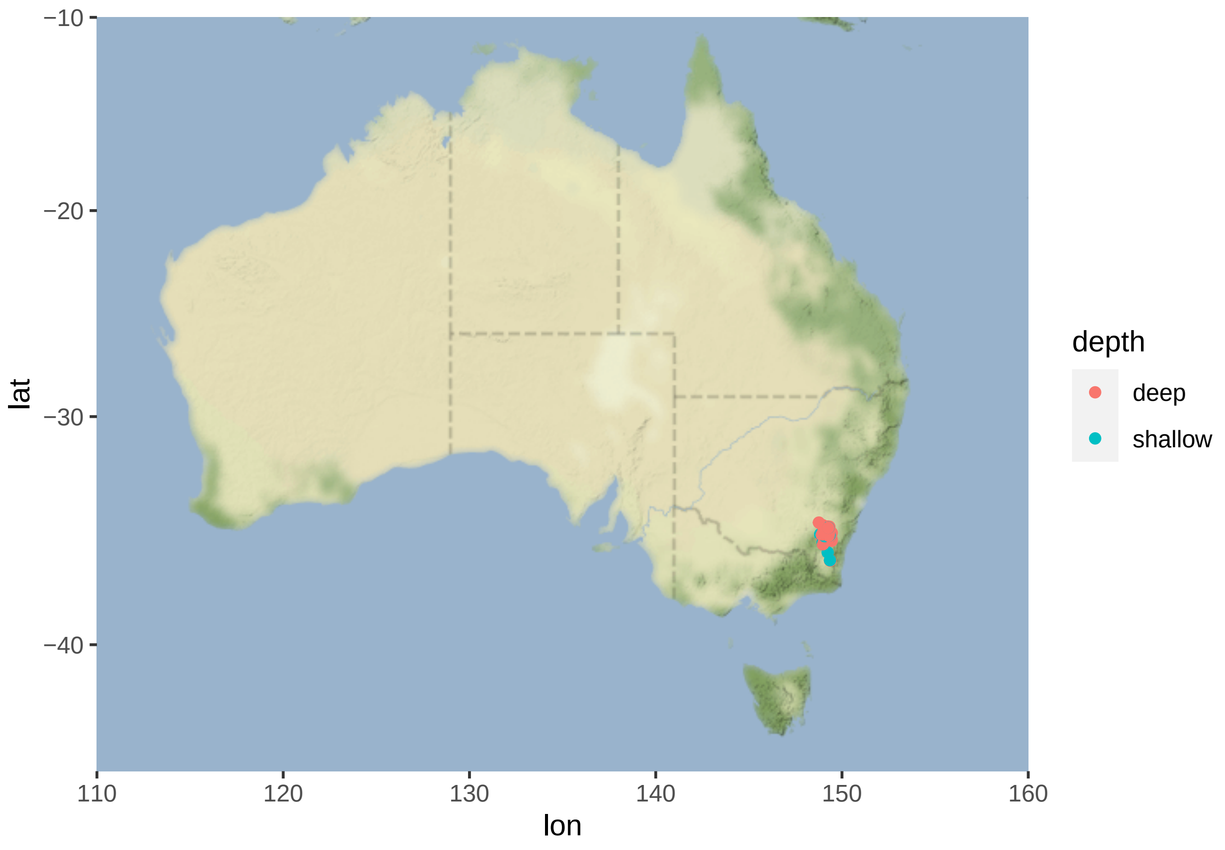
```
## Parsed with column specification:
## cols(
##   site.code = col_character(),
##   log.size = col_double(),
##   depth = col_character(),
##   water.type = col_character(),
##   latitude = col_double(),
##   longitude = col_double(),
##   cpar = col_double(),
##   csig = col_double(),
##   limdum = col_double(),
##   limper = col_double(),
##   limtas = col_double(),
##   lper = col_double(),
##   lver = col_double(),
##   ulae = col_double(),
##   richness = col_double()
## )
```

```
sites
```

```
## Warning: `...` is not empty.
##
## We detected these problematic arguments:
## * `needs_dots`
##
## These dots only exist to allow future extensions and should be empty.
## Did you misspecify an argument?
```

```
## # A tibble: 320 x 15
##    site.code log.size depth water.type latitude longitude  cpar  csig limdum
##    <chr>        <dbl> <chr> <chr>         <dbl>     <dbl> <dbl> <dbl>  <dbl>
##  1 AMA100       2.91  deep  moving        -35.2      149.     1     1      1
##  2 AMH100       0.554 shal~ moving        -35.2      149.     0     0      0
##  3 ANB100       2.01  deep  moving        -35.3      149.     0     1      1
##  4 ANU004       1.75  deep  moving        -35.3      149.     0     1      1
##  5 ANU018       2.38  deep  still         -35.3      149.     0     1      0
##  6 ANU019       1.75  deep  still         -35.3      149.     0     1      1
##  7 ANU020       2.16  deep  still         -35.3      149.     1     1      1
##  8 ANU021       0.892 shal~ still         -35.3      149.     1     1      0
##  9 ANU022       0.957 shal~ moving        -35.3      149.     1     1      0
## 10 ANU022B      1.13  deep  moving        -35.3      149.     0     1      0
## # ... with 310 more rows, and 6 more variables: limper <dbl>, limtas <dbl>,
## #   lper <dbl>, lver <dbl>, ulae <dbl>, richness <dbl>
```

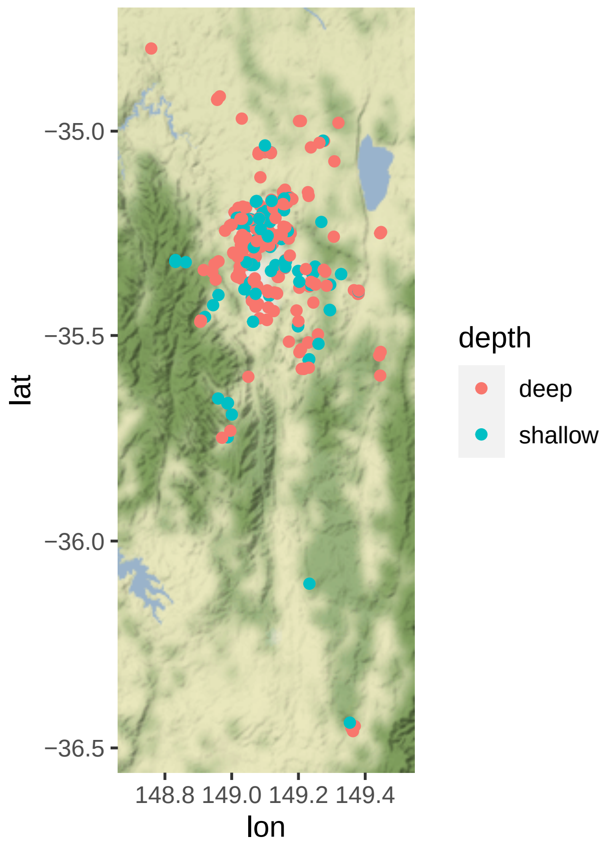
```r
ggmap(m2) +
  geom_point(data = sites,
             aes(x=longitude,y=latitude,col=depth))
```



```r
xmin <- min(sites$longitude)-0.1
xmax <- max(sites$longitude)+0.1
ymin <- min(sites$latitude)-0.1
ymax <- max(sites$latitude)+0.1


m3 <- get_stamenmap(bbox = c(left = xmin,
                             bottom = ymin,
                             right = xmax,
                             top = ymax),
                    zoom = 8,
                   maptype = "terrain-background")
```

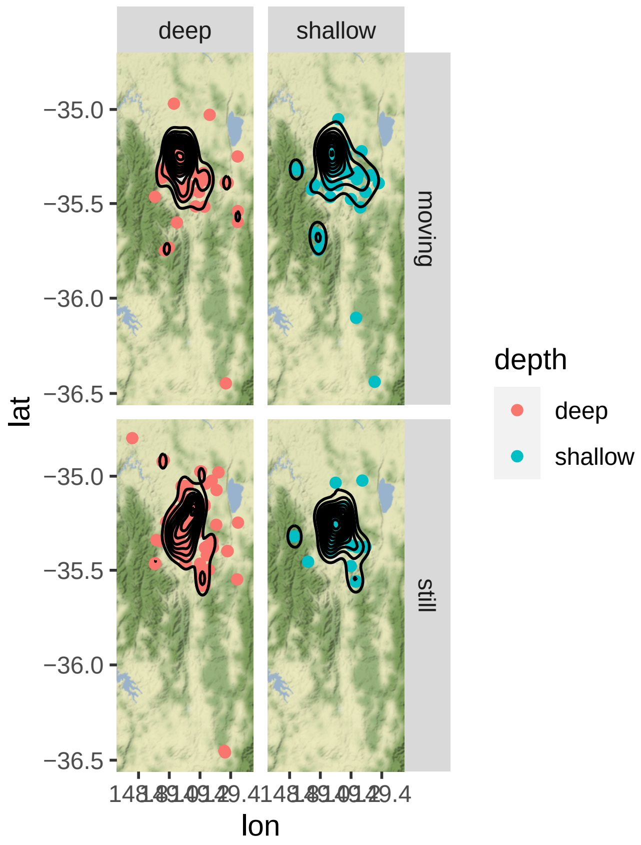
```
## Source : http://tile.stamen.com/terrain-background/8/233/154.png
## Source : http://tile.stamen.com/terrain-background/8/234/154.png
## Source : http://tile.stamen.com/terrain-background/8/233/155.png
## Source : http://tile.stamen.com/terrain-background/8/234/155.png
```

```
ggmap(m3) +
  geom_point(data = sites,
             aes(x=longitude,y=latitude,col=depth))
```



Note that you can do things with these that you would do with ggplot. For example:

```
ggmap(m3) +
  geom_point(data = sites,
             aes(x=longitude,y=latitude,col=depth)) +
  geom_density2d(data = sites,
                 aes(x=longitude,y=latitude),col='black')+
  facet_grid(water.type~depth)
```



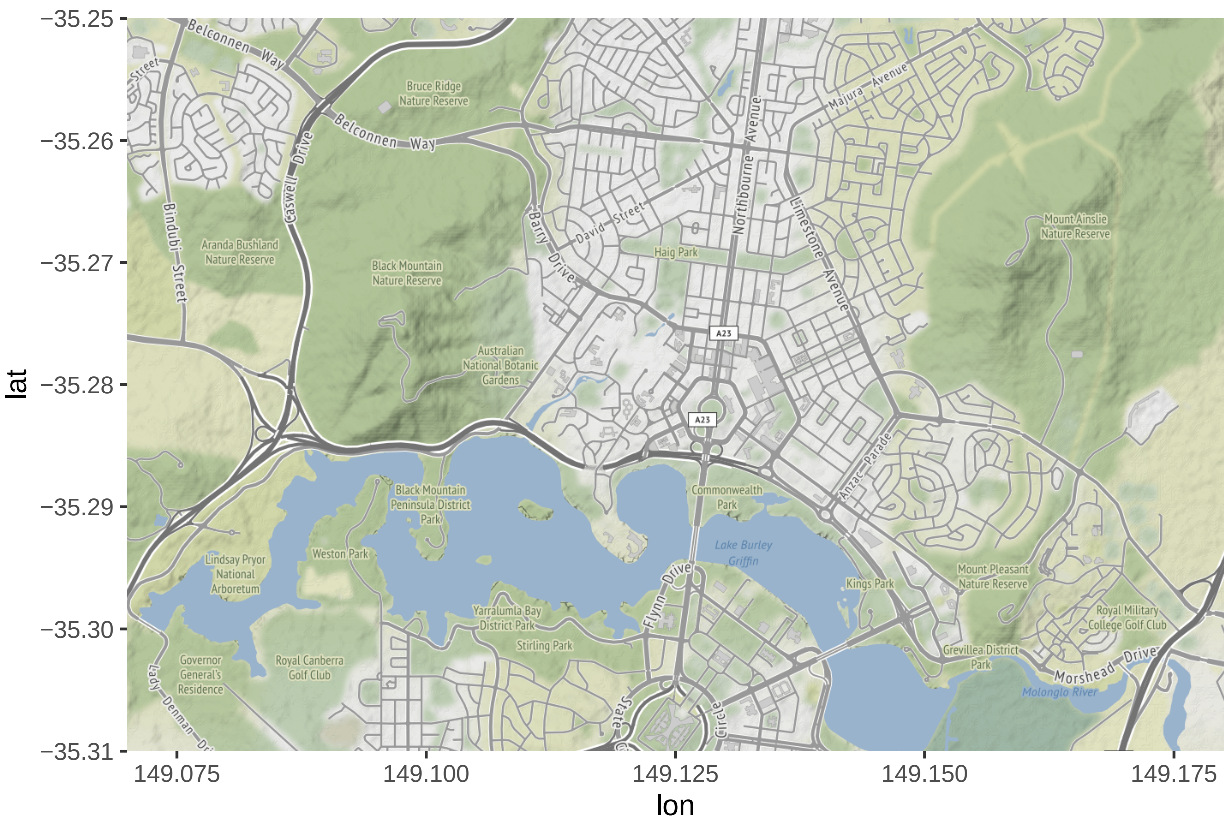
Imagine that I'm most interested in the points around the city / ANU.

Task 2: select a map size that fits campus and the surrounding areas.

```
xmin2 <- 149.07
xmax2 <- 149.18
ymin2 <- -35.31
ymax2 <- -35.25

m4 <- get_stamenmap( bbox = c(left = xmin2,
                              bottom = ymin2,
                              right = xmax2,
                              top = ymax2),
                     zoom = 14,
                     maptype = "terrain")
```

```
## Source : http://tile.stamen.com/terrain/14/14976/9908.png

## Source : http://tile.stamen.com/terrain/14/14977/9908.png

## Source : http://tile.stamen.com/terrain/14/14978/9908.png

## Source : http://tile.stamen.com/terrain/14/14979/9908.png

## Source : http://tile.stamen.com/terrain/14/14980/9908.png

## Source : http://tile.stamen.com/terrain/14/14981/9908.png

## Source : http://tile.stamen.com/terrain/14/14976/9909.png

## Source : http://tile.stamen.com/terrain/14/14977/9909.png
## Source : http://tile.stamen.com/terrain/14/14978/9909.png
## Source : http://tile.stamen.com/terrain/14/14979/9909.png
## Source : http://tile.stamen.com/terrain/14/14980/9909.png
## Source : http://tile.stamen.com/terrain/14/14981/9909.png
## Source : http://tile.stamen.com/terrain/14/14976/9910.png
## Source : http://tile.stamen.com/terrain/14/14977/9910.png
## Source : http://tile.stamen.com/terrain/14/14978/9910.png
## Source : http://tile.stamen.com/terrain/14/14979/9910.png
## Source : http://tile.stamen.com/terrain/14/14980/9910.png
## Source : http://tile.stamen.com/terrain/14/14981/9910.png
## Source : http://tile.stamen.com/terrain/14/14976/9911.png
## Source : http://tile.stamen.com/terrain/14/14977/9911.png
## Source : http://tile.stamen.com/terrain/14/14978/9911.png
## Source : http://tile.stamen.com/terrain/14/14979/9911.png
## Source : http://tile.stamen.com/terrain/14/14980/9911.png
## Source : http://tile.stamen.com/terrain/14/14981/9911.png
```

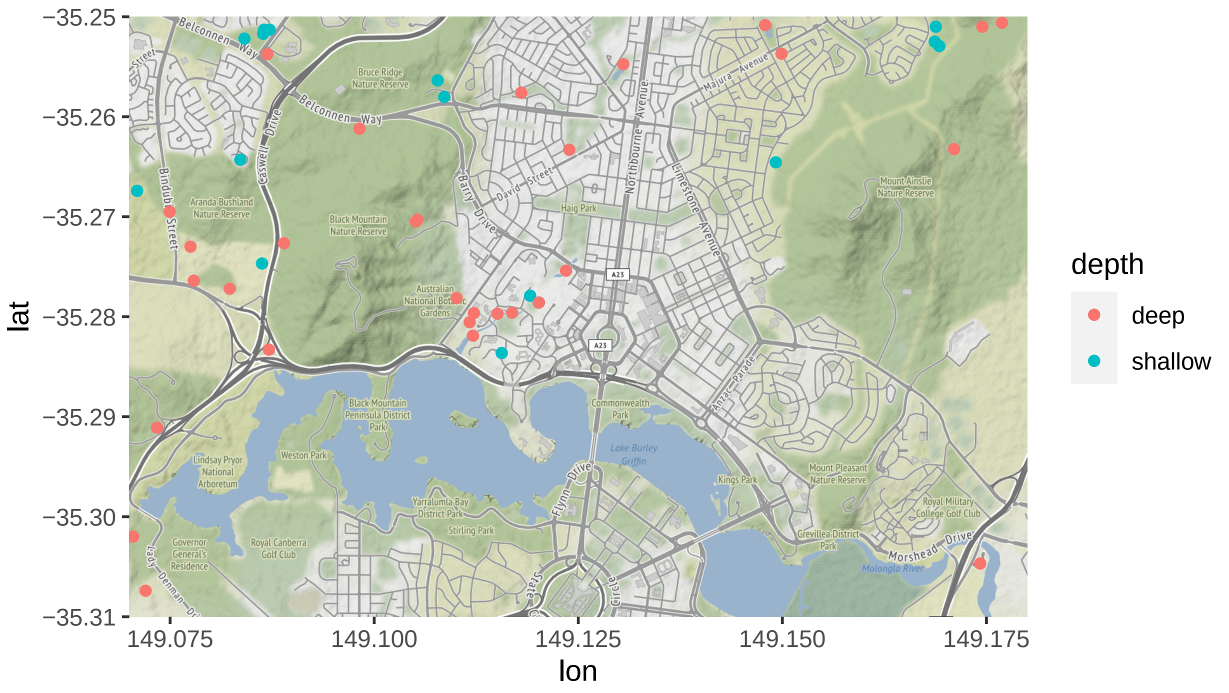
```
ggmap(m4)
```



Given this smaller map we can add the sites. Note that sites outside the region get truncated and you get a warning about this.

```
ggmap(m4)+
  geom_point(data = sites,
             aes(x=longitude,y=latitude,col=depth))
```

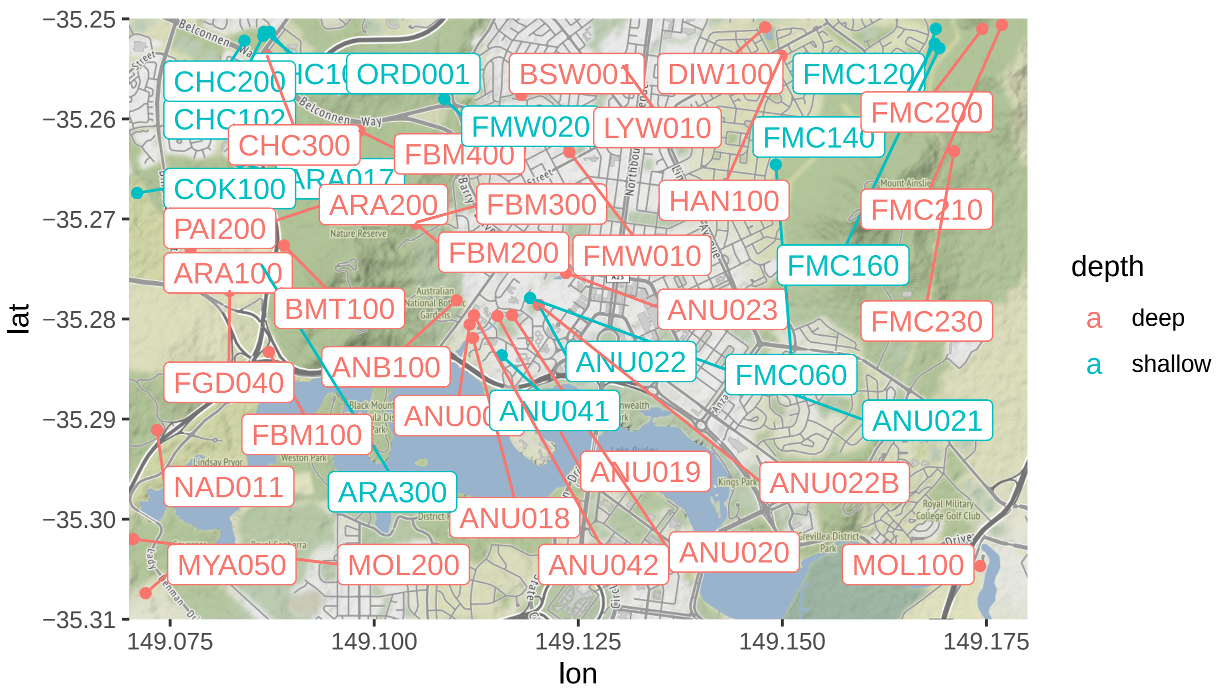
```
## Warning: Removed 274 rows containing missing values (geom_point).
```



We can truncate these sites so we no longer get this warning. We will do this with filter() from dplyr. We would also like to label some sites. A nice package for this is ggrepel – repel because it makes the labels float away from each other (i.e., they repel each other).

```
sitesClose <- sites %>% dplyr::filter(
                     longitude > xmin2,
                     longitude < xmax2,
                     latitude > ymin2,
                     latitude < ymax2)

library(ggrepel)

ggmap(m4) +
  geom_point(data = sitesClose, aes(x=longitude,y=latitude,col=depth))+
  geom_label_repel(
    data = sitesClose,
               aes(x=longitude,
                   y=latitude,
                   col=depth,
                   label=site.code),
    box.padding = 0.9)
```

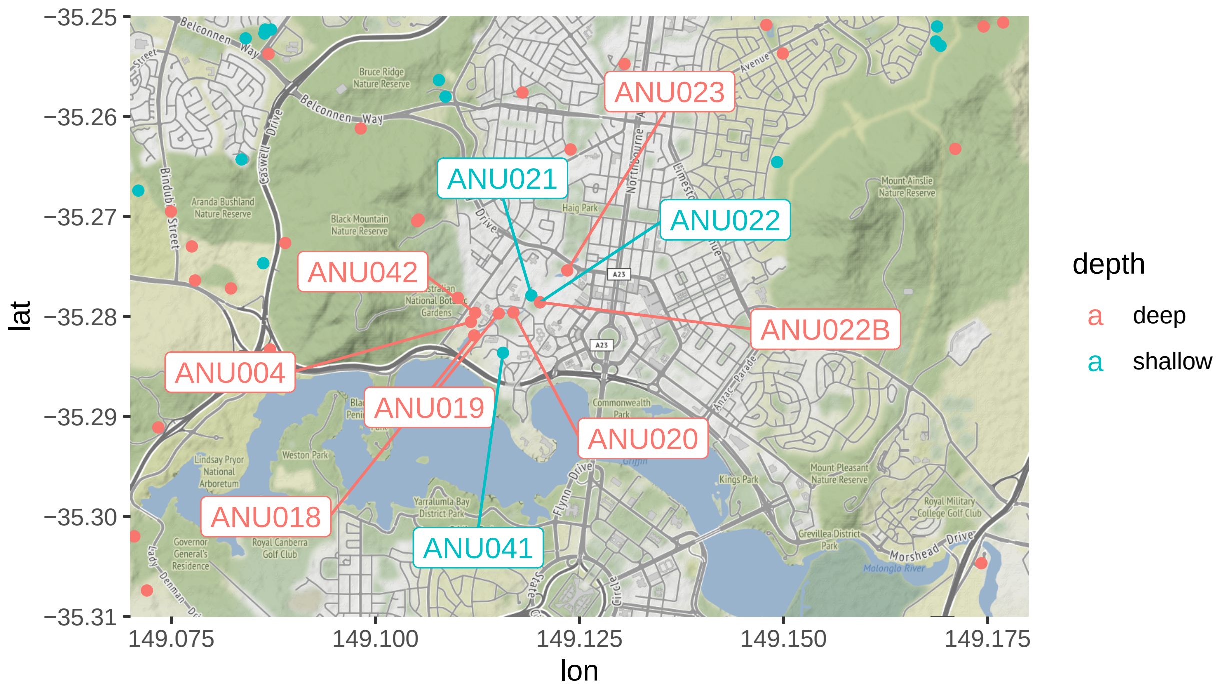
```
## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning in min(x): no non-missing arguments to min; returning Inf
## Warning in max(x): no non-missing arguments to max; returning -Inf
```



This is a bit cluttered because there are so many sites. Lets pick just the sites at ANU. We can do this again using filter, and getting those sites that start with 'ANU'.

```r
ggmap(m4) +
  geom_point(data = sitesClose, aes(x=longitude,y=latitude,col=depth))+
  geom_label_repel(
    data = dplyr::filter(sitesClose,
                         stringr::str_detect(site.code, 'ANU')),
                aes(x=longitude,
                    y=latitude,
                    col=depth,
                    label=site.code),
    box.padding = 0.9)
```

```
## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf

## Warning in min(x): no non-missing arguments to min; returning Inf

## Warning in max(x): no non-missing arguments to max; returning -Inf
```

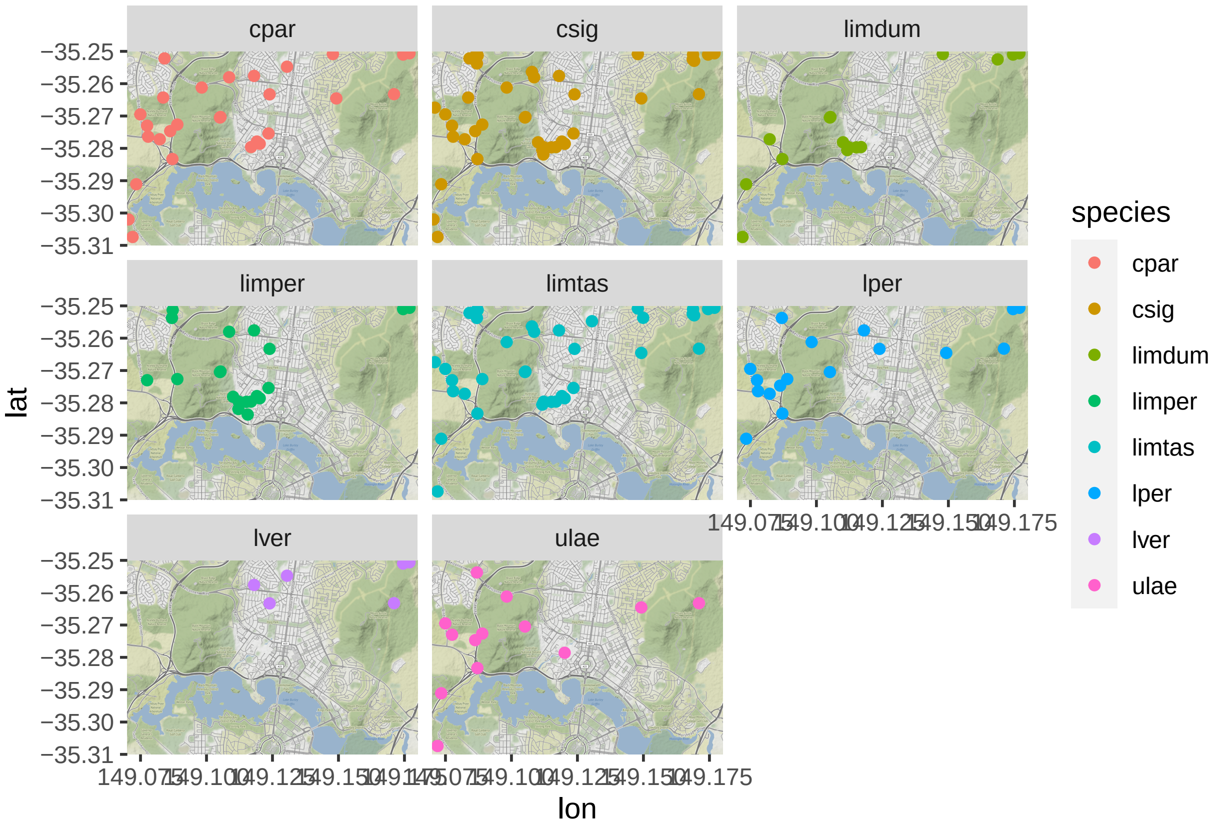


Extension / practice

Use pivot to plot the sites that each species is present in as facets.

```
sites_sp <- pivot_longer(sites, 7:14,names_to = 'species',values_to = 'presence')
sites_sp <- filter(sites_sp,presence ==1)

ggmap(m4) +
  geom_point(data = sites_sp,aes(x=longitude,y=latitude,col=species))+
  facet_wrap(~species)
```

```
## Warning: Removed 1321 rows containing missing values (geom_point).
```



Brief aside: leaflet

One thing similar thing you can do is use the package leaflet. It allows you to do interactive maps (e.g., that people can zoom/move around in the normal way you interact with maps). It is nice and pretty similar. But you maybe can't put one of these in a paper – maybe in a website? Look it up if you are interested. More info here: https://rstudio.github.io/leaflet/

```
library(leaflet)
m <- leaflet() %>%
  addTiles() %>%
  addMarkers(lng = sitesClose$longitude, lat = sitesClose$latitude,
             popup = sitesClose$site.code)
m
```

Part 2: More general map data

Let's start by considering different data that you might be interested in. For example, if you are interested in any way in Australian populations the best source is likely the ABS. The ABS provides a lot of data including digital boundaries for different regional classifications.

https://www.abs.gov.au/websitedbs/D3310114.nsf/home/Digital+Boundaries

Another type of data one might be interested in is climate data. An easily accessible dataset is from worldclim.

https://www.worldclim.org/data/worldclim21.html

These two examples show the two general formats that spatial data are available in: shapefiles (such as the polygons describing geographic regions in the ABS data) and rasters (i.e., the gridded format that worldclim comes in).

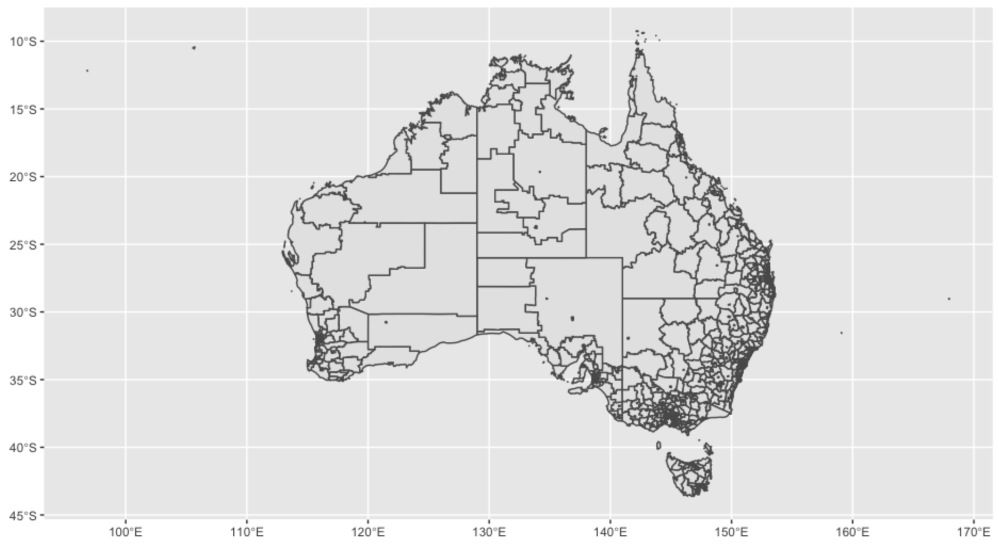
Because these files are quite big and annoying I will pre-clean them.

sf

We start by reading in the full map from the ABS. It is huge – if I try to plot it at its original resolution my computer crashes. So we can simplify it (i.e., decrease the resolution)

DON'T ACTUALLY DO THIS IN THE WORKSHOP!!

```
aus_sa2 <- sf::st_read('~/Documents/temp/1270055001_sa2_2016_aust_shape/SA2_2016_AUST.shp')
str(aus_sa2)

library(rmapshaper)
aus_sa2s <- ms_simplify(aus_sa2,keep = 0.01, keep_shapes=TRUE)

ggplot(aus_sa2s)+geom_sf()
```



It takes a long time to plot because the boundaries provided by the ABS are extremely high resolution and there are a lot of them.

A sf object is basically a tibble (or data frame) but where one of the columns is a complex geometric type rather than the usual numeric or string. We can treat it like a data.frame and use filter again to get the bits we want – just the ACT today.

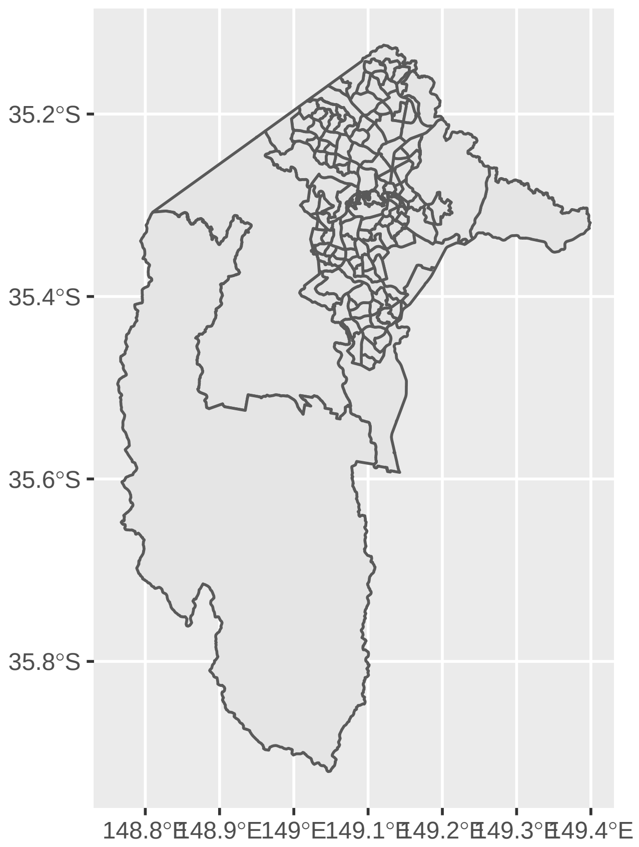
```
library(sf)
act_sa2 <- dplyr::filter(aus_sa2,STE_NAME16 == 'Australian Capital Territory')
st_write(act_sa2,'act_sa2.shp')
```

This shapefile is much smaller / easier to work with. We can read it in and plot it.

```
act_sa2 <- sf::st_read('./data/ABS_SA2_ACT/act_sa2.shp')
```

```
## Reading layer `act_sa2' from data source `/Users/u1094337/Dropbox/2020/teaching/maps/maps_initial_dra
## replacing null geometries with empty geometries
## Simple feature collection with 133 features and 12 fields (with 2 geometries empty)
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 148.7628 ymin: -35.92076 xmax: 149.3993 ymax: -35.12442
## geographic CRS: GDA94
```

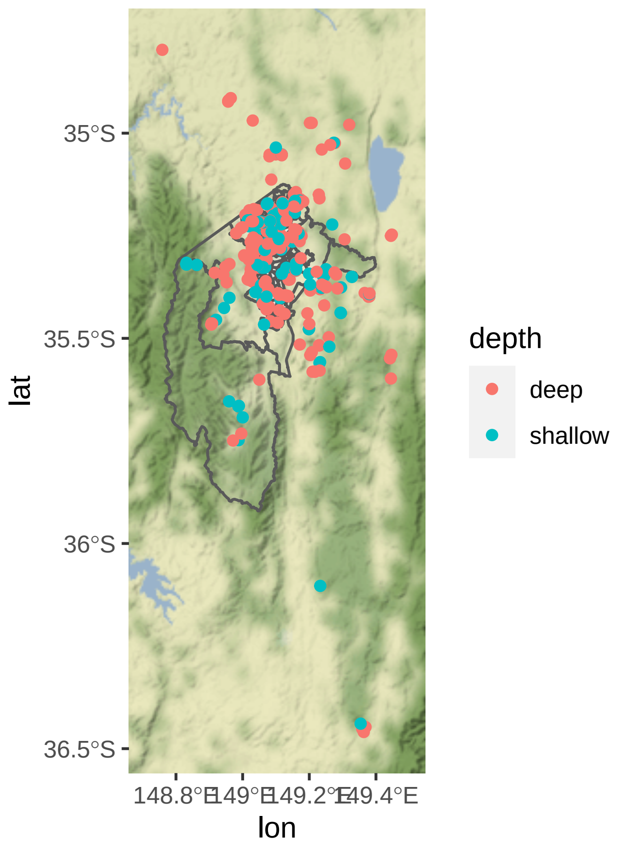
```
ggplot(act_sa2)+geom_sf()
```



And, it plays well with our other maps. The thing you need to be careful about is setting inherit.aes=FALSE, because otherwise it tries to use the same columns as ggmap and things go badly.

```
ggmap(m3)+
  geom_sf(data = act_sa2,alpha=0.1,inherit.aes=FALSE) +
  geom_point(data = sites, aes(x=longitude,y=latitude,col=depth))
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



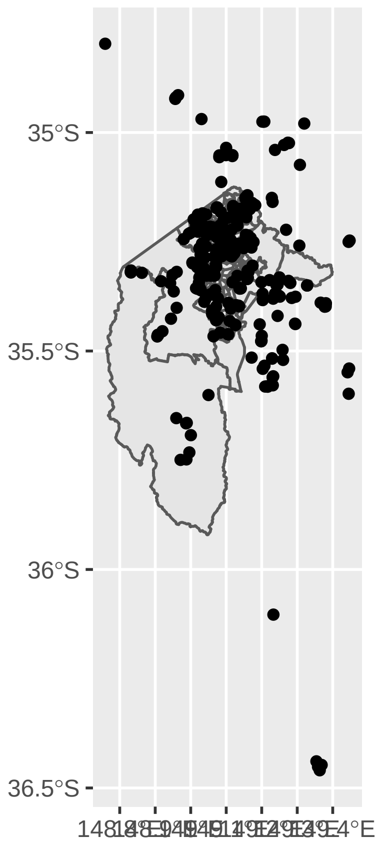
The most common challenge for me when deailing with maps is the coordinate reference scheme (CRS). The CRS describes the coordinates are organised / what units they come in / etc., which is necessary because the world is spherical(ish) but maps are flat. Objects in a geographic format (like sf objects) will always have a CRS. You can check what it is using st_crs(). This information came from the shapefile we read in.

```
sf::st_crs(act_sa2)
```

```
## Coordinate Reference System:
##   User input: GDA94
##   wkt:
## GEOGCRS["GDA94",
##     DATUM["Geocentric Datum of Australia 1994",
##         ELLIPSOID["GRS 1980",6378137,298.257222101,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     USAGE[
##         SCOPE["unknown"],
##         AREA["Australia - GDA"],
##         BBOX[-60.56,93.41,-8.47,173.35]],
##     ID["EPSG",4283]]
```

To use a bit more of the sf package, it would help to have our sites in a sf object rather than just in a tibble. We can use st_as_af() to do this – we need to specify the "geometric" bit of the tibble, as well as the CRS. We use 4283 because it is what act_sa2 is in and it is ideal for them to be the same.

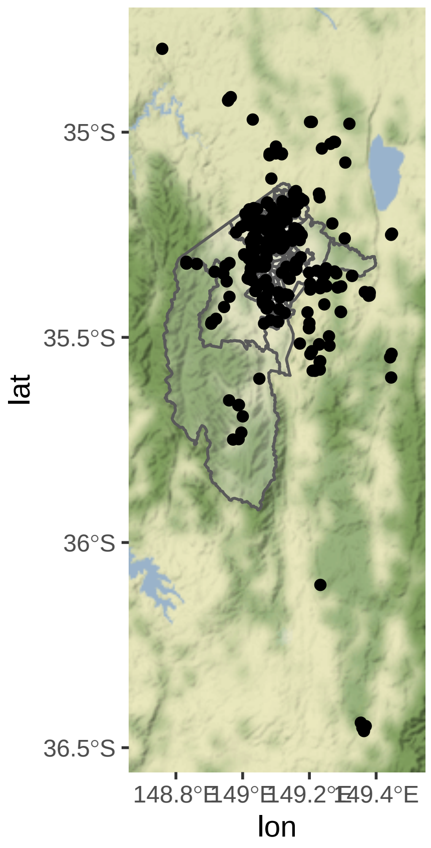
```
sites_sf <- sf::st_as_sf(sites,coords = c('longitude','latitude'),crs=4283)

ggplot(act_sa2)+
  geom_sf() + geom_sf(data = sites_sf)
```



Since geom_sf automatically plots the geometric object we don't have to specify its aes.

```
ggmap(m3)+
  geom_sf(data=act_sa2,inherit.aes = FALSE,alpha=0.2) +
  geom_sf(data = sites_sf,inherit.aes = FALSE)
```

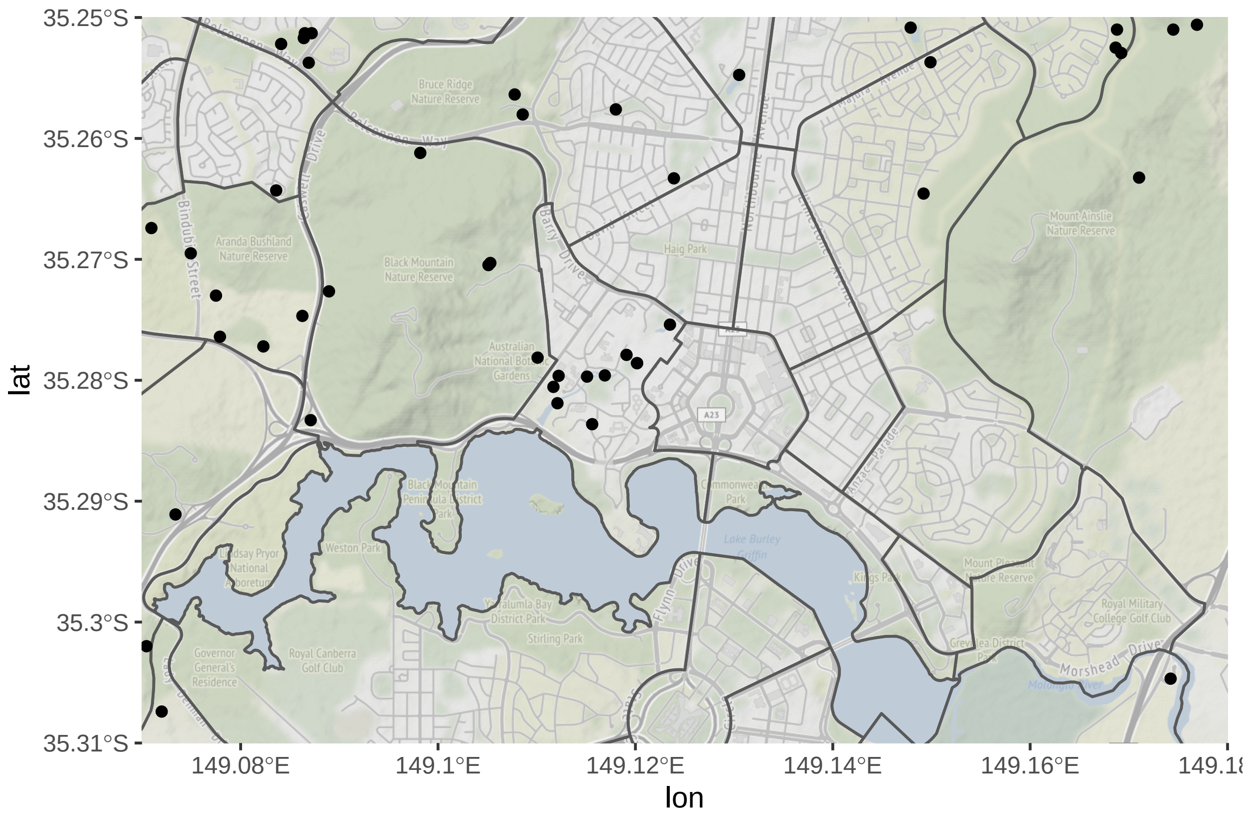
```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



Lets go back to ANU.

```
ggmap(m4)+
  geom_sf(data=act_sa2,inherit.aes = FALSE,alpha=0.5) +
  geom_sf(data = sites_sf,inherit.aes = FALSE)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```


35.25°S
35.26°S
35.27°S
35.28°S
35.29°S
35.3°S
35.31°S
lat
149.08°E
149.1°E
149.12°E
149.14°E
149.16°E
149.1
lon
Bruce Ridge Nature Reserve
Black Mountain Nature Reserve
Aranda Bushland Nature Reserve
Australian National Botanic Gardens
Haig Park
Mount Ainslie Nature Reserve
Lake Burley Griffin
Commonwealth Park
Weston Park
Yarralumla Bay District Park
Stirling Park
Royal Canberra Golf Club
Governor General's Residence
Mount Pleasant Nature Reserve
Royal Military College Golf Club
Morshead Drive
Kings Park
Flynn Drive
Barry Drive
Bindubi Street
Caswell Drive

What I would like to do is a calculation that involves our spatial objects. An obvious thing to try would be to count the number of sites in each SA2 region. This type of tool might be useful if we wanted to create some kind of statistical model. To do this, we will use a spatial join (st_join) – what this does is, for each point, asks which SA2 it is located inside, and concatenates the information from that SA2 to that row in the sites data. We will then count how many rows have a given SA2, and join *that* information back to the SA2 data. And then use the counts to colour the plot.

```
sites_in_sa2 <- sf::st_join(sites_sf,act_sa2)
```

```
## although coordinates are longitude/latitude, st_intersects assumes that they are planar
```

```
library(dplyr)

site_counts <- sites_in_sa2 %>% tibble::as_tibble() %>%
  group_by(SA2_NAME16) %>% summarise(count=n())
```

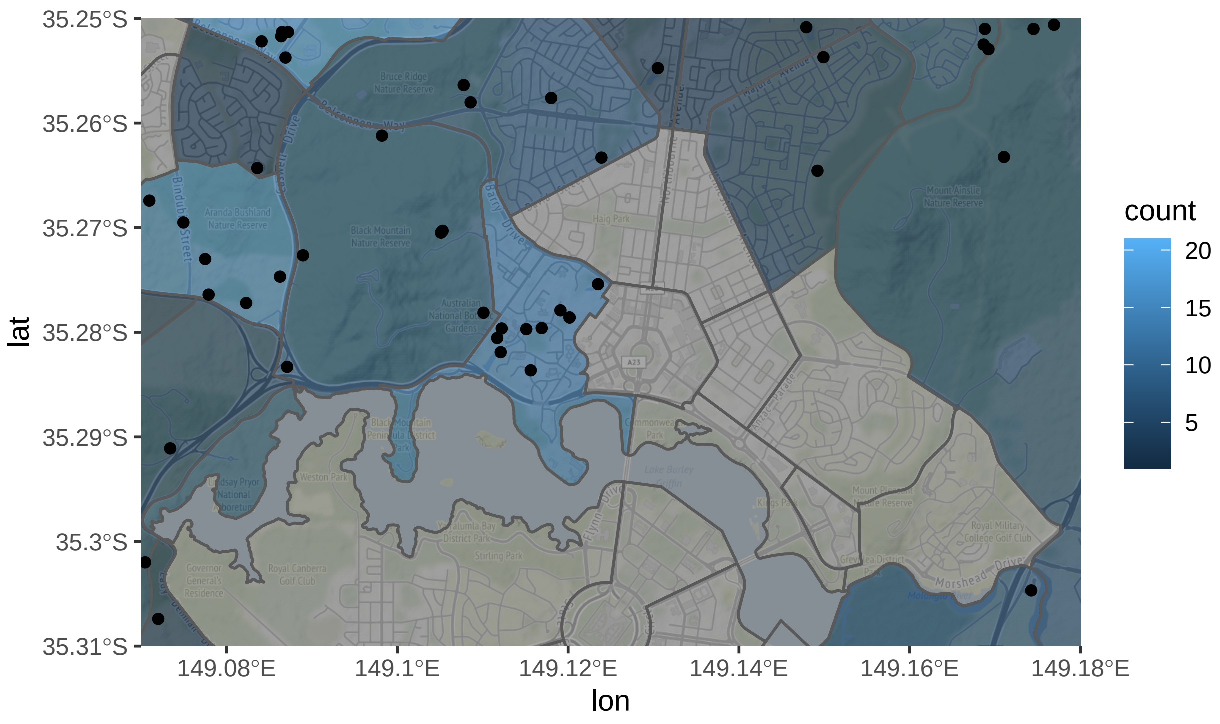
```
## `summarise()` ungrouping output (override with `.groups` argument)
```

```
act_sa2t <- dplyr::left_join(act_sa2,site_counts)
```

```
## Joining, by = "SA2_NAME16"
```

```
ggmap(m4)+
  geom_sf(data=act_sa2t,inherit.aes = FALSE,alpha=0.7,aes(fill=count)) +
  geom_sf(data = sites_sf,inherit.aes = FALSE)
```

```
## Coordinate system already present. Adding new coordinate system, which will replace the existing one
```



```
#note the NA ones are empty
```

There are a lot more things you could do with these types of spatial data (polygons, points, lines), and the sf package has a lot of tools to do so. The things I want you to take away from this are

- use the sf package when you have shapefiles,
- geom_sf lets you easily plot these in ggplot,
- sf objects act like data frames – so you can do the normal data manipulation things like filter, join, etc..
- you have to be a little careful with the CRS.

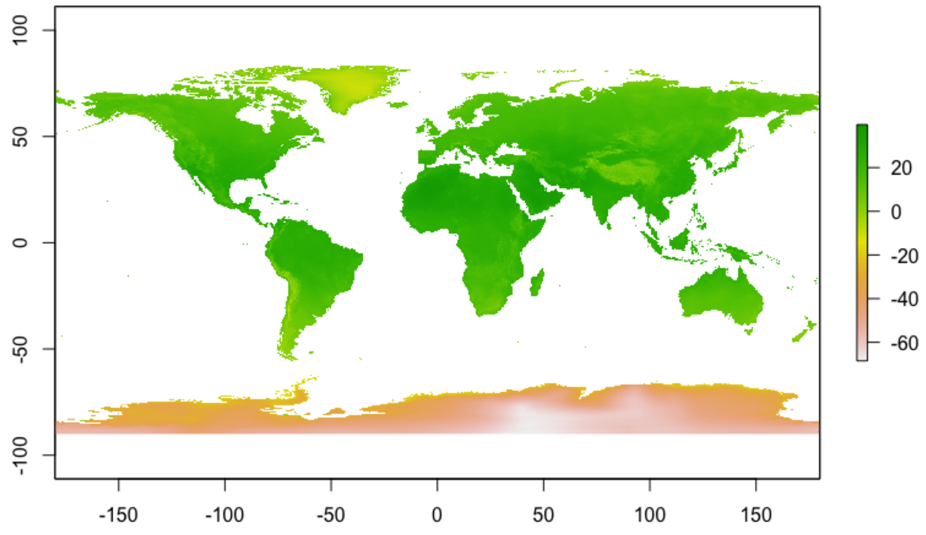
raster

When spatial data are continuously varying, they typically come in raster form. A raster is basically a picture, where every pixel is an observation. They can be huge because they include a lot of information – just the monthly average temperatures from worldclim were 4GB. It has close to a billion rows.

```
library(raster)
library(rgdal) #needs to be installed

WORLD_RASTER <- raster('~/Documents/temp/wc2.1_30s_tavg/wc2.1_30s_tavg_07.tif')
plot(WORLD_RASTER)
str(WORLD_RASTER)
```

Fortunately we can trim it to our area of interest, and save just that bit, which is much easier to work with.

```
e <- as(extent(xmin2, xmax2, ymin2, ymax2), 'SpatialPolygons')
crs(e) <- "+proj=longlat +datum=WGS84 +no_defs"
r <- crop(WORLD_RASTER, e)

plot(r)

rf <- writeRaster(r, filename=file.path("./CBR_July_temp.tif"), format="GTiff", overwrite=TRUE)
```



When you read in a raster object it has a base plot() command but doesn't play nicely with ggplot. Note that raster is a lot like sf in that its a hugely powerful package that can do a lot of stuff – you can do calculations on rasters, sample them, reproject them, etc. etc. (I'm not an expert).

```
library(raster)
```

```
## Loading required package: sp

##
## Attaching package: 'raster'

## The following object is masked from 'package:dplyr':
##
##     select

## The following object is masked from 'package:tidyr':
##
##     extract
```

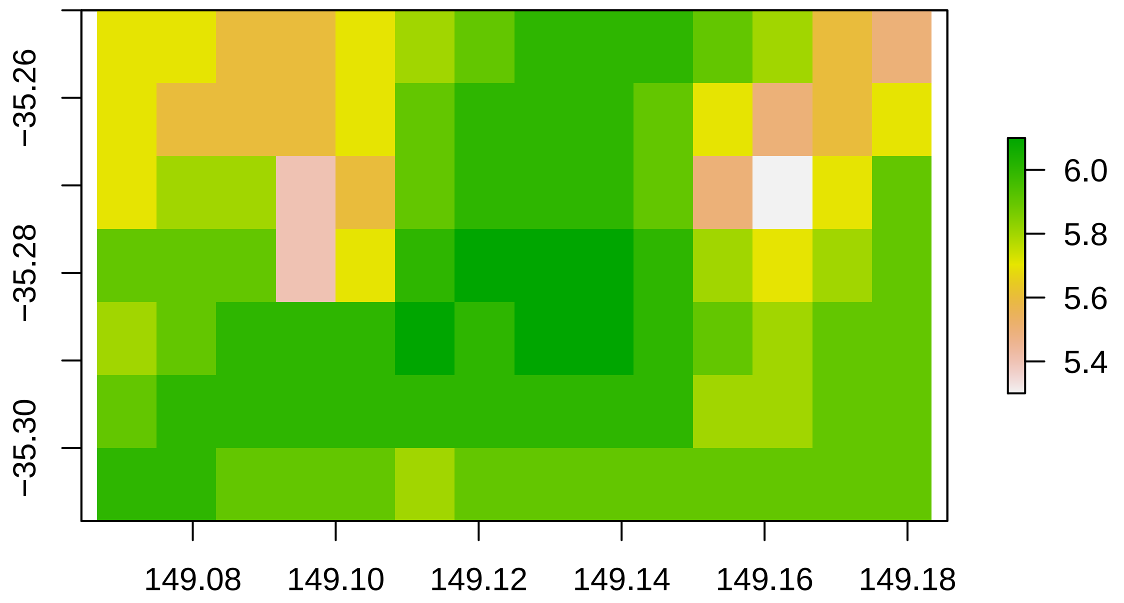
```
library(rgdal)
```

```
## rgdal: version: 1.5-12, (SVN revision 1018)
## Geospatial Data Abstraction Library extensions to R successfully loaded
## Loaded GDAL runtime: GDAL 3.1.1, released 2020/06/22
## Path to GDAL shared files: /Library/Frameworks/R.framework/Versions/4.0/Resources/library/rgdal/gdal
## GDAL binary built with GEOS: TRUE
## Loaded PROJ runtime: Rel. 6.3.1, February 10th, 2020, [PJ_VERSION: 631]
## Path to PROJ shared files: /Library/Frameworks/R.framework/Versions/4.0/Resources/library/rgdal/proj
## Linking to sp version:1.4-2
## To mute warnings of possible GDAL/OSR exportToProj4() degradation,
## use options("rgdal_show_exportToProj4_warnings"="none") before loading rgdal.
```

```
CBR_temp <- raster('./data/CBR_July_temp/CBR_July_temp.tif')
str(CBR_temp)
```

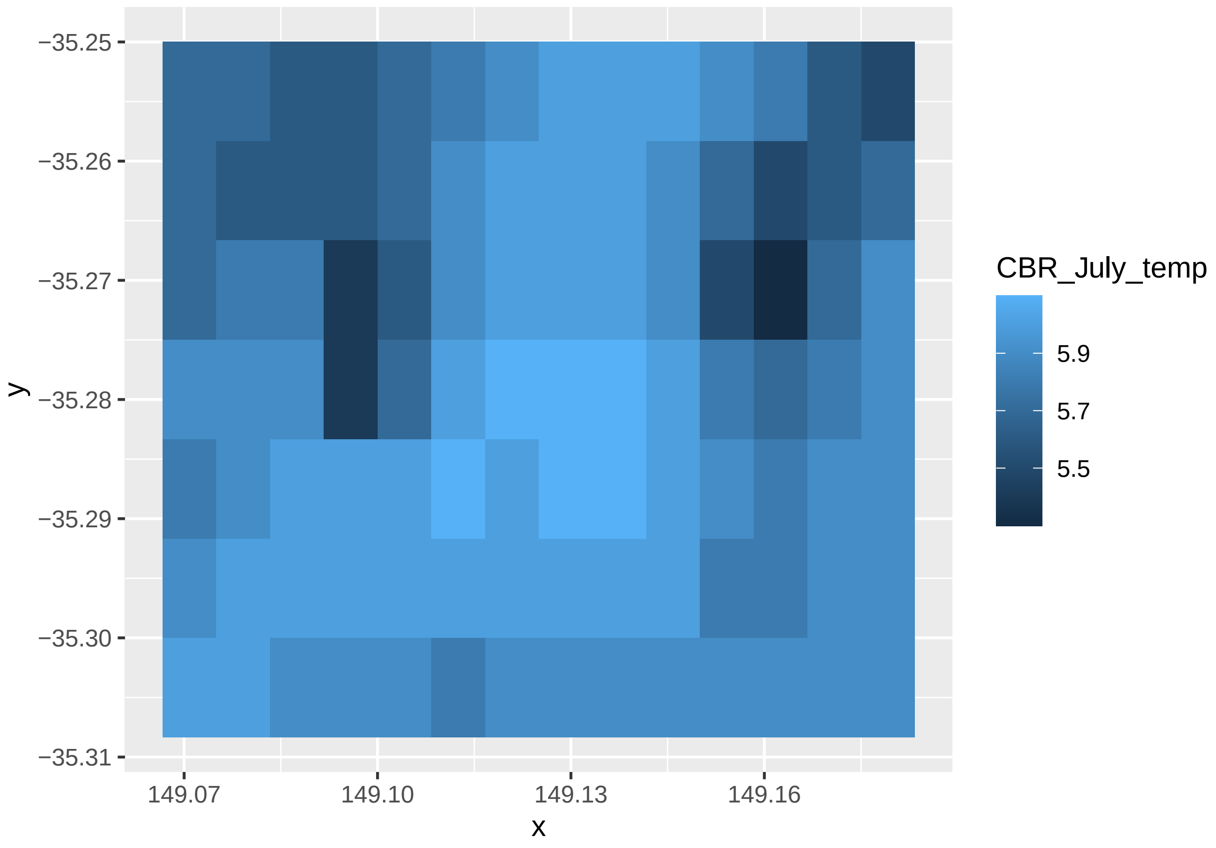
```
## Formal class 'RasterLayer' [package "raster"] with 12 slots
##   ..@ file    :Formal class '.RasterFile' [package "raster"] with 13 slots
##   .. .. ..@ name        : chr "/Users/u1094337/Dropbox/2020/teaching/maps/maps_initial_draft/data/CB
##   .. .. ..@ datanotation: chr "FLT4S"
##   .. .. ..@ byteorder   : chr "little"
##   .. .. ..@ nodatavalue : num -Inf
##   .. .. ..@ NAchanged   : logi FALSE
##   .. .. ..@ nbands      : int 1
##   .. .. ..@ bandorder   : chr "BIL"
##   .. .. ..@ offset      : int 0
##   .. .. ..@ toptobottom : logi TRUE
##   .. .. ..@ blockrows   : int 7
##   .. .. ..@ blockcols   : int 14
##   .. .. ..@ driver      : chr "gdal"
##   .. .. ..@ open        : logi FALSE
##   ..@ data    :Formal class '.SingleLayerData' [package "raster"] with 13 slots
##   .. .. ..@ values    : logi(0)
##   .. .. ..@ offset    : num 0
##   .. .. ..@ gain      : num 1
##   .. .. ..@ inmemory  : logi FALSE
##   .. .. ..@ fromdisk  : logi TRUE
##   .. .. ..@ isfactor  : logi FALSE
##   .. .. ..@ attributes: list()
##   .. .. ..@ haveminmax: logi TRUE
##   .. .. ..@ min       : num 5.3
##   .. .. ..@ max       : num 6.1
##   .. .. ..@ band      : int 1
##   .. .. ..@ unit      : chr ""
##   .. .. ..@ names     : chr "CBR_July_temp"
##   ..@ legend  :Formal class '.RasterLegend' [package "raster"] with 5 slots
##   .. .. ..@ type      : chr(0)
##   .. .. ..@ values    : logi(0)
##   .. .. ..@ color     : logi(0)
##   .. .. ..@ names     : logi(0)
##   .. .. ..@ colortable: logi(0)
##   ..@ title   : chr(0)
##   ..@ extent  :Formal class 'Extent' [package "raster"] with 4 slots
##   .. .. ..@ xmin: num 149
##   .. .. ..@ xmax: num 149
##   .. .. ..@ ymin: num -35.3
##   .. .. ..@ ymax: num -35.2
##   ..@ rotated : logi FALSE
##   ..@ rotation:Formal class '.Rotation' [package "raster"] with 2 slots
##   .. .. ..@ geotrans: num(0)
##   .. .. ..@ transfun:function ()
##   ..@ ncols   : int 14
##   ..@ nrows   : int 7
##   ..@ crs     :Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. ..@ projargs: chr "+proj=longlat +datum=WGS84 +no_defs"
##   ..@ history : list()
##   ..@ z       : list()
```

```
plot(CBR_temp)
```



To let it talk to ggplot the easiest way is to convert to a data.frame and use geom_raster(). Note that geom_raster() isn't specific to this type of file, rather it is a ggplot default and you can use it for any standard data.frame.

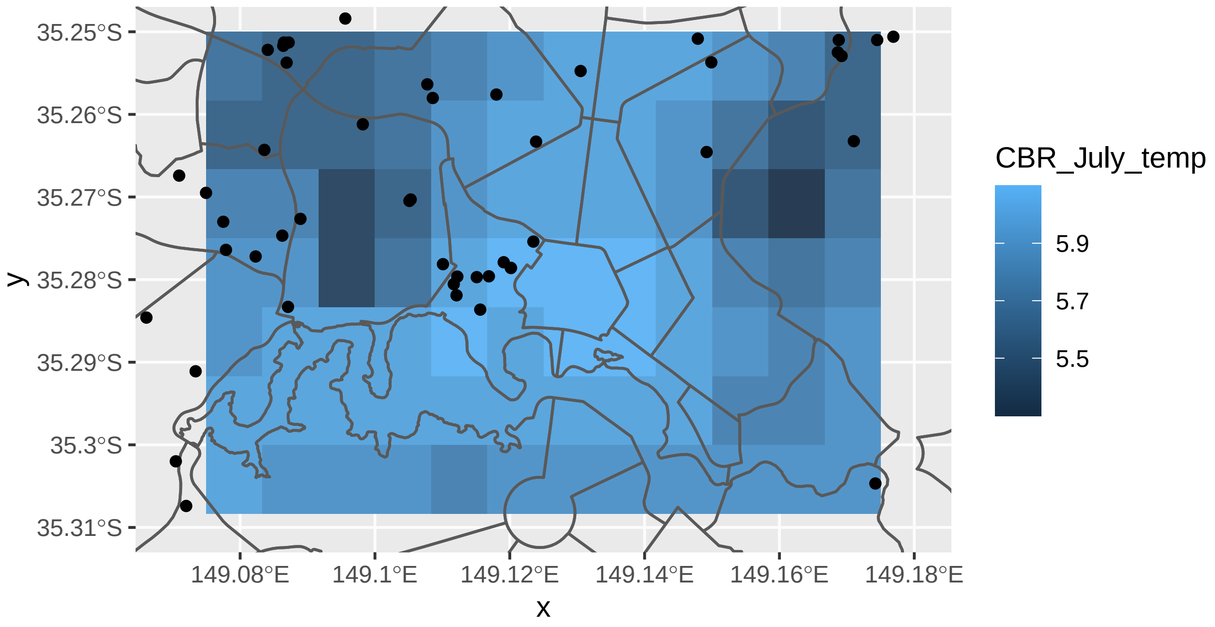
```
ctDF <- as.data.frame(CBR_temp,xy=TRUE)

ggplot(ctDF,aes(x=x,y=y,fill=CBR_July_temp))+
  geom_raster()
```



And since now everything talks to ggplot, we can combine this with our earlier work with sf. Note that we have to add the xlim() and ylim() here.

```r
ggplot(ctDF,aes(x=x,y=y,fill=CBR_July_temp))+
  geom_raster()+
  geom_sf(data = act_sa2,inherit.aes = FALSE,alpha=0.1)+
  geom_sf(data = sites_sf,inherit.aes = FALSE)+
  xlim(xmin2,xmax2) + ylim(ymin2,ymax2)
```

```
## Warning: Removed 14 rows containing missing values (geom_raster).
```



This was quite a brief survey. The takeaways from this section are:

- continously varying spatial data tend to come in raster formats, which are basically images,
- you can use the raster package to work with them. It is powerful but quite complex,
- you can convert them to data.frame and then use ggplot, which lets you continue to interact with all the other ggplot pieces.

More intense geospatial work is pretty complex and I'm definitely not an expert. I'd encourage you to explore further if you are interested.

Resources

Some useful packages, data, etc.:

maps package https://www.rdocumentation.org/packages/maps/versions/3.3.0 It includes a world map, regional maps in some locations (e.g., the USA), locations of most cities worldwide (including Australia)

Australian Bureau of Statistics maps: https://www.abs.gov.au/AUSSTATS/abs@.nsf/DetailsPage/1270.0.55.001July%202016?OpenDocument

or

https://data.gov.au/dataset/ds-dga-0ed76b6e-4f6e-4cc6-9f95-0b4e3729dc69/details?q=

Some other workshops you might like to peruse:

http://www.seascapemodels.org/data/data-wrangling-spatial-course.html https://github.com/dlab-berkeley/Geospatial-Fundamentals-in-R-with-sf

And some books:

Geocomputation for R https://geocompr.robinlovelace.net/

Leaflet for R https://rstudio.github.io/leaflet/

R for Data Science (more generally) https://r4ds.had.co.nz/