Mapping Session

Juliano Palacios Abrantes

05/11/2019

Map sources

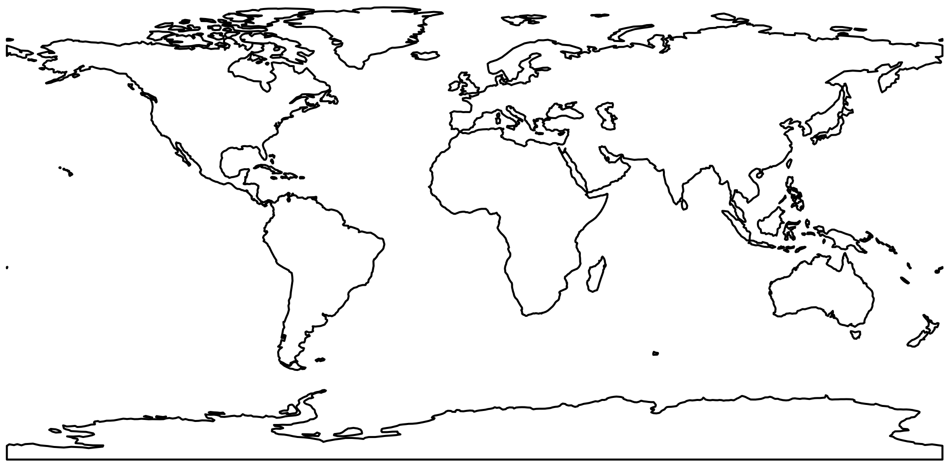
There are tons of free map data sources and a bunch of R packages that integrate it well. One of the main sources I use is Natural Earth 10-m-resolution coastline shapefile. You can visit the website to find different versions or types of files, and they mainly come in three resolution formats: 10m (fine), 50m (medium), 110m (coarse). The resolution translates directly into mapping and processing speed with finer resolution taking much longer. As I'm working through lots of examples, I'll use the coarser/lower resolution maps for most of these.

Natural Earth also has its own R package called `rnaturalearth`

```r
land110 <- ne_download(scale = 110,
                        type = 'land',
                        category = 'physical')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/private/var/folders/83/gdppt_ds3s5dmgtvxtj9mvjw0000gn/T/RtmpodaJ7j", layer: "ne_110m_land"
## with 127 features
## It has 3 fields
```

```r
plot(land110)
```



If you want to fill in the land, you should use a land shapefile. If you want only the outline of the continents/islands, use a coastline file.

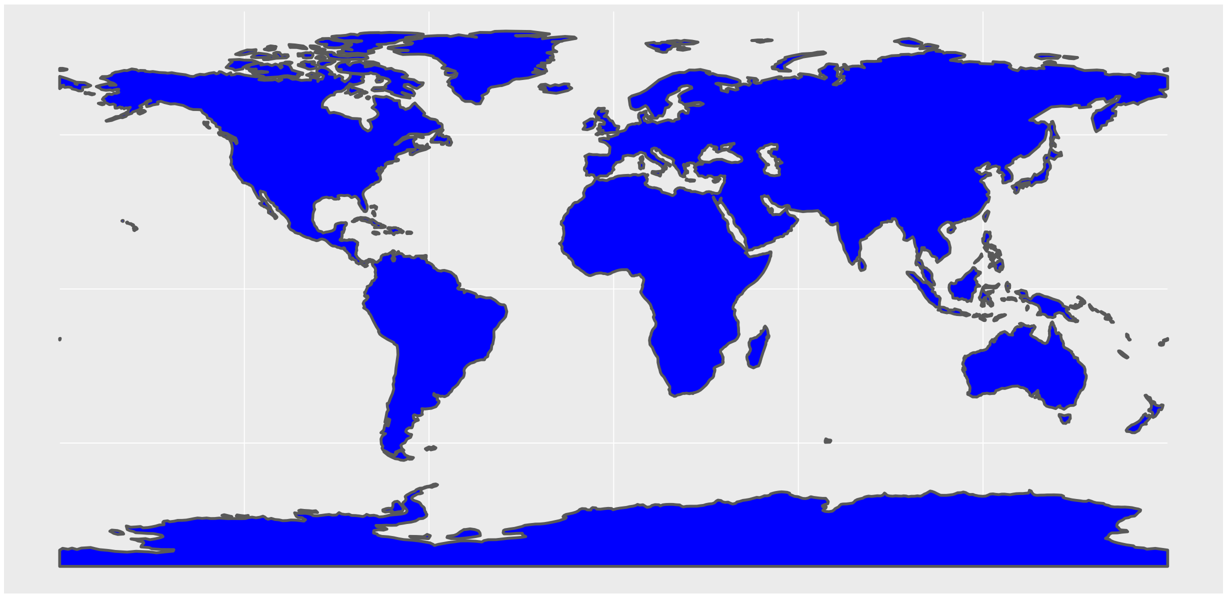
```r
coast110 <- ne_download(scale = 110,
                         type = 'coastline',
                          category = 'physical')
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/private/var/folders/83/gdppt_ds3s5dmgtvxtj9mvjw0000gn/T/RtmpodaJ7j", layer: "ne_110m_coastl
## with 134 features
## It has 3 fields
## Integer64 fields read as strings:  scalerank
```

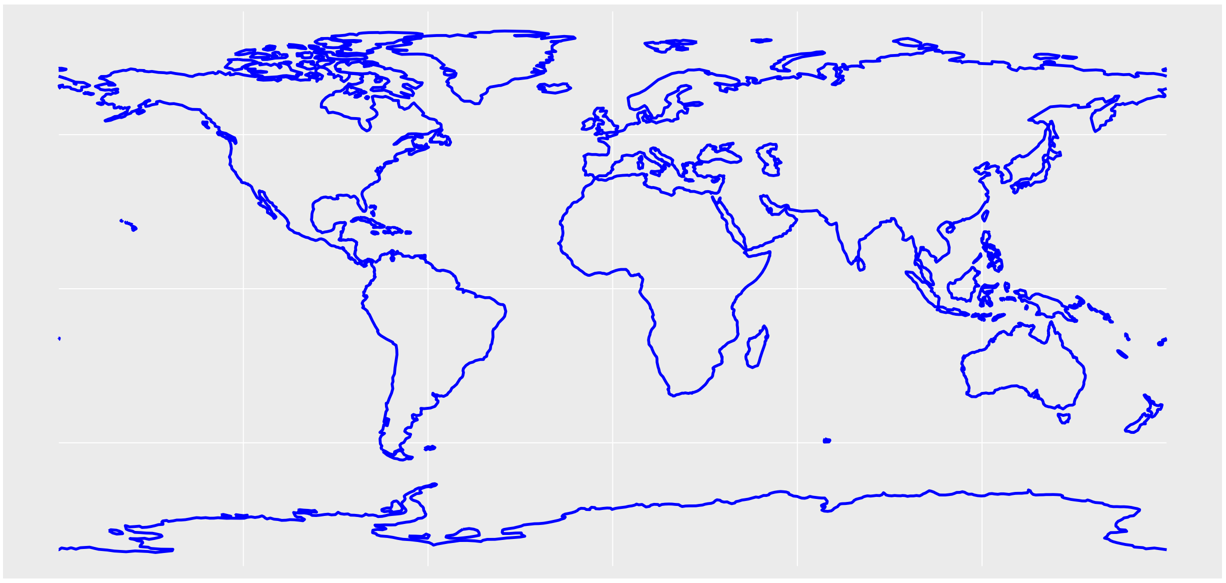
```r
sp::plot(coast110)
```

They originally look the same, but if you convert it to a "simple features" object (sf), you can see the difference.

```r
land110_sf <- st_as_sf(land110)

# Land shapefile
ggplot(land110_sf) +
  geom_sf(fill="blue")
```



```r
coast110_sf <- st_as_sf(coast110)

# Coastal lines shapefile
ggplot(coast110_sf) +
  geom_sf(color="blue")
```

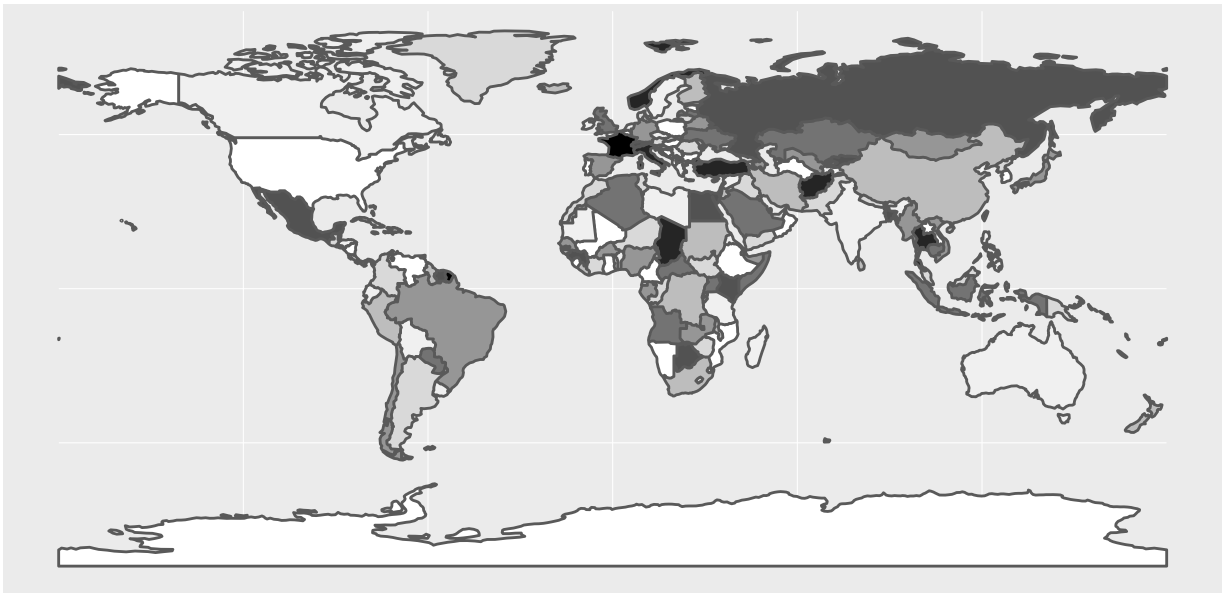


Alternatively, you might want country level data like in making a world map.

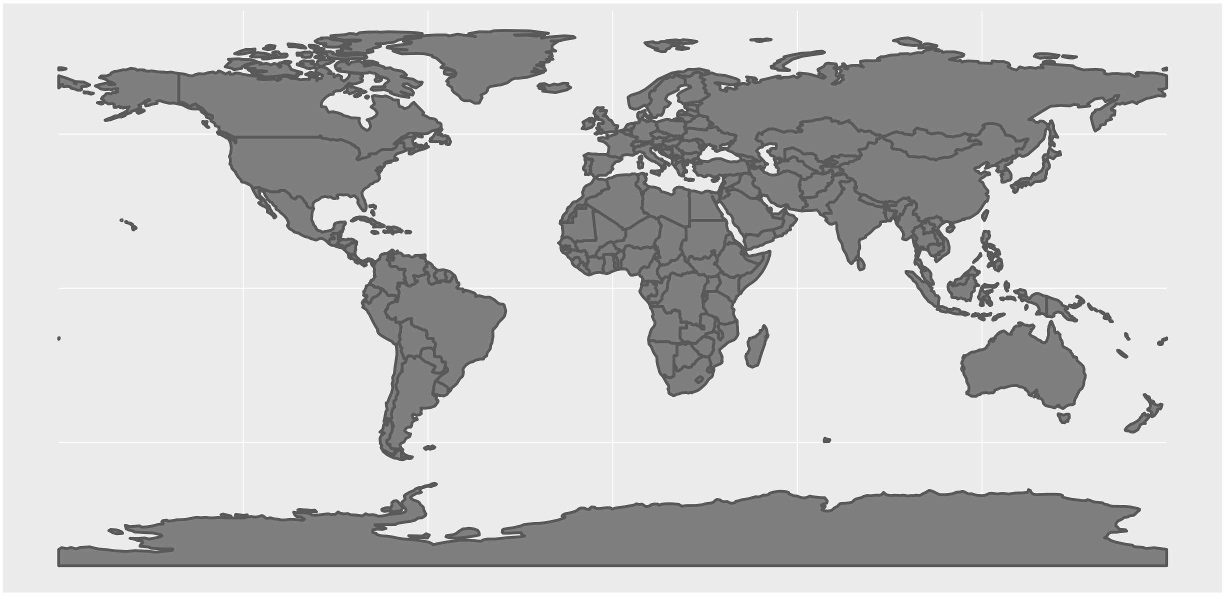
```
country110 <- ne_countries() #natural earth countries

x <- ne_download(scale=110,
                 type="countries",
                 category="cultural",
                 returnclass="sf")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/private/var/folders/83/gdppt_ds3s5dmgtvxtj9mvjw0000gn/T/RtmpodaJ7j", layer: "ne_110m_admin_
## with 177 features
## It has 94 fields
## Integer64 fields read as strings:  POP_EST NE_ID
```

```
country110_sf <- st_as_sf(country110)

ggplot(country110_sf) +
  geom_sf(aes(fill=as.factor(mapcolor9))) +
  theme(legend.position="none") +
  scale_fill_brewer(palette="Greys")
```



```
ggplot(country110_sf) +
  geom_sf(fill="grey50") +
  theme(legend.position="none")
```



But, it's hard to see what's going on at a smaller level with this world level data. You have two options:

1. Zoom in to a subset of the current map.
2. Find a smaller scale map that is of your area of interest.

Another good source for country level data including subunits is http://www.diva-gis.org/gdata

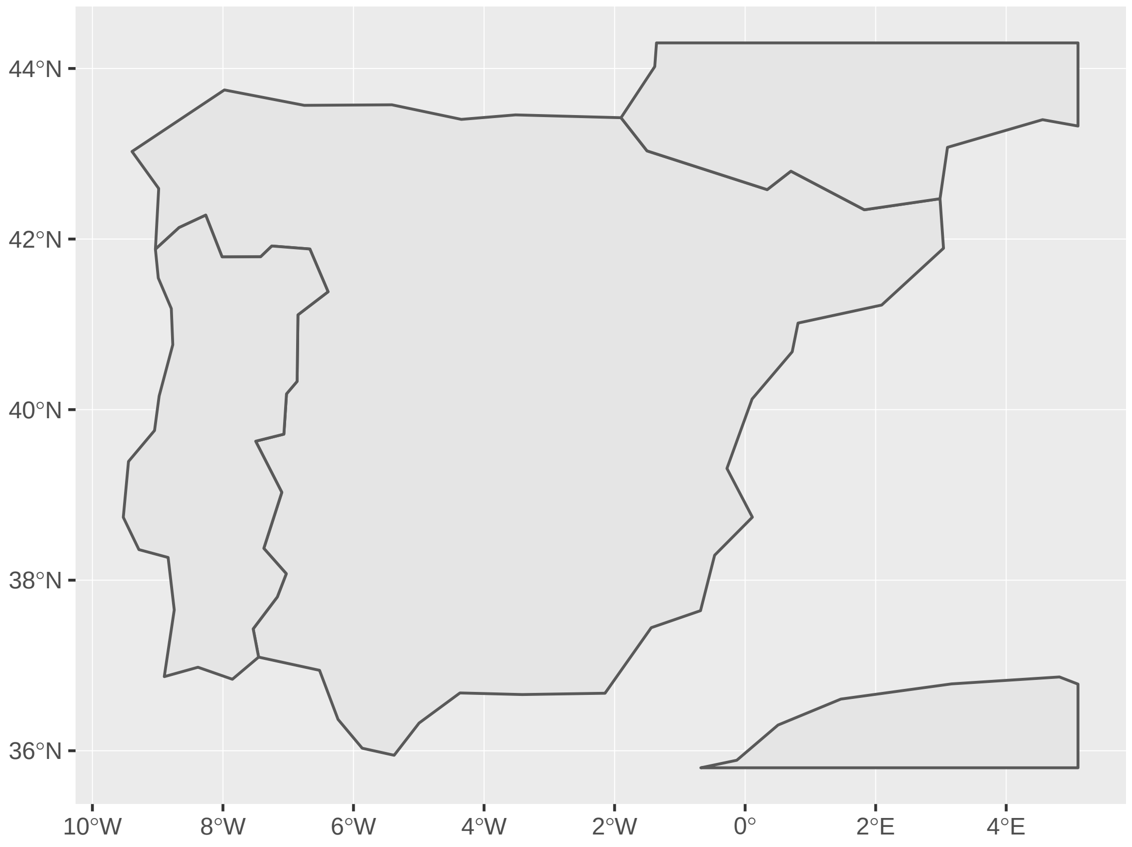
```
dims <- c(xmin=-9.9,
          xmax=5.10,
          ymin=35.8,
          ymax=44.3)
```

```r
Spain <- st_crop(x=country110_sf,
                 y=dims)
```

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

```
## Warning: attribute variables are assumed to be spatially constant
## throughout all geometries
```

```r
Spain %>% ggplot() +
  geom_sf()
```



Notice any problems?

1. Islands are missing (Raquel does not like this!)
2. Coastline is very smooth.

3. Generally unrealistic.

```r
country10_sf <- ne_download(scale=10,
                            type="countries",
                            category="cultural",
                            returnclass="sf")
```

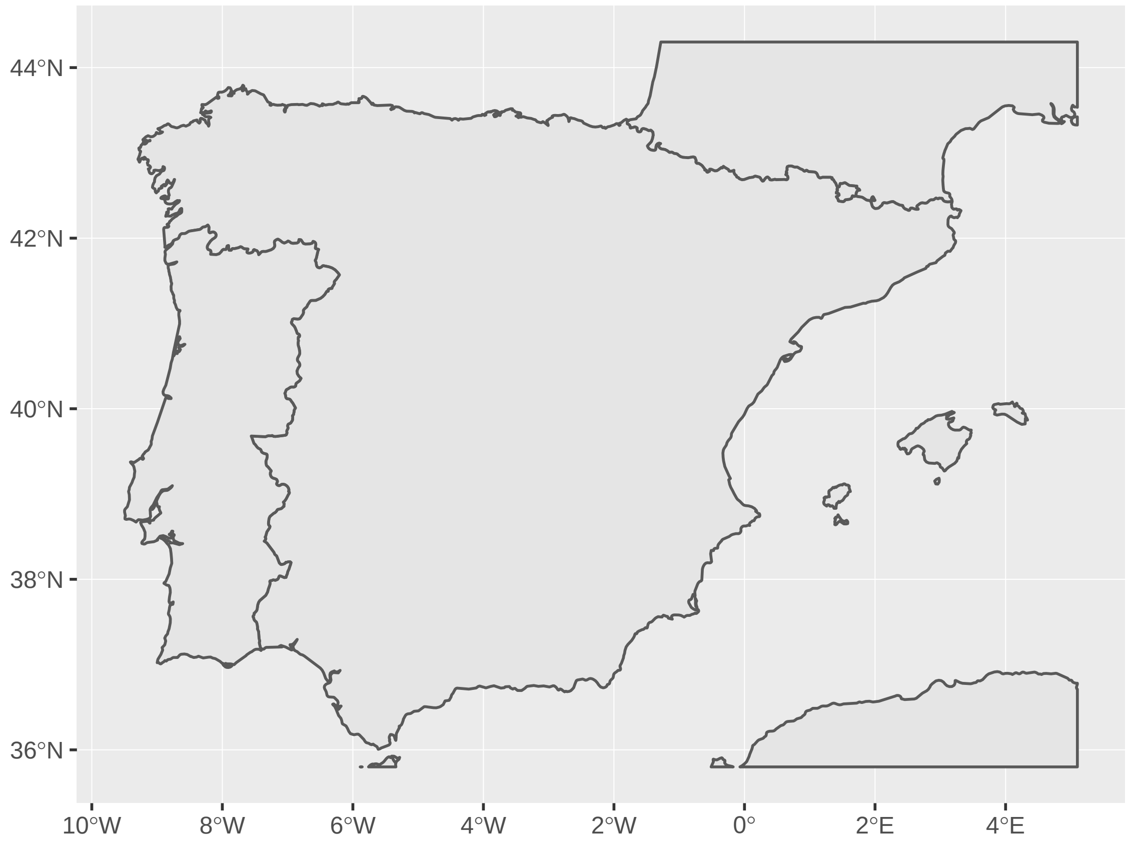
```
## OGR data source with driver: ESRI Shapefile
## Source: "/private/var/folders/83/gdppt_ds3s5dmgtvxtj9mvjw0000gn/T/RtmpodaJ7j", layer: "ne_10m_admin_(
## with 255 features
## It has 94 fields
## Integer64 fields read as strings:  POP_EST NE_ID
```

```r
Spain <- st_crop(x=country10_sf, y=dims)
```

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

```
## Warning: attribute variables are assumed to be spatially constant
## throughout all geometries
```

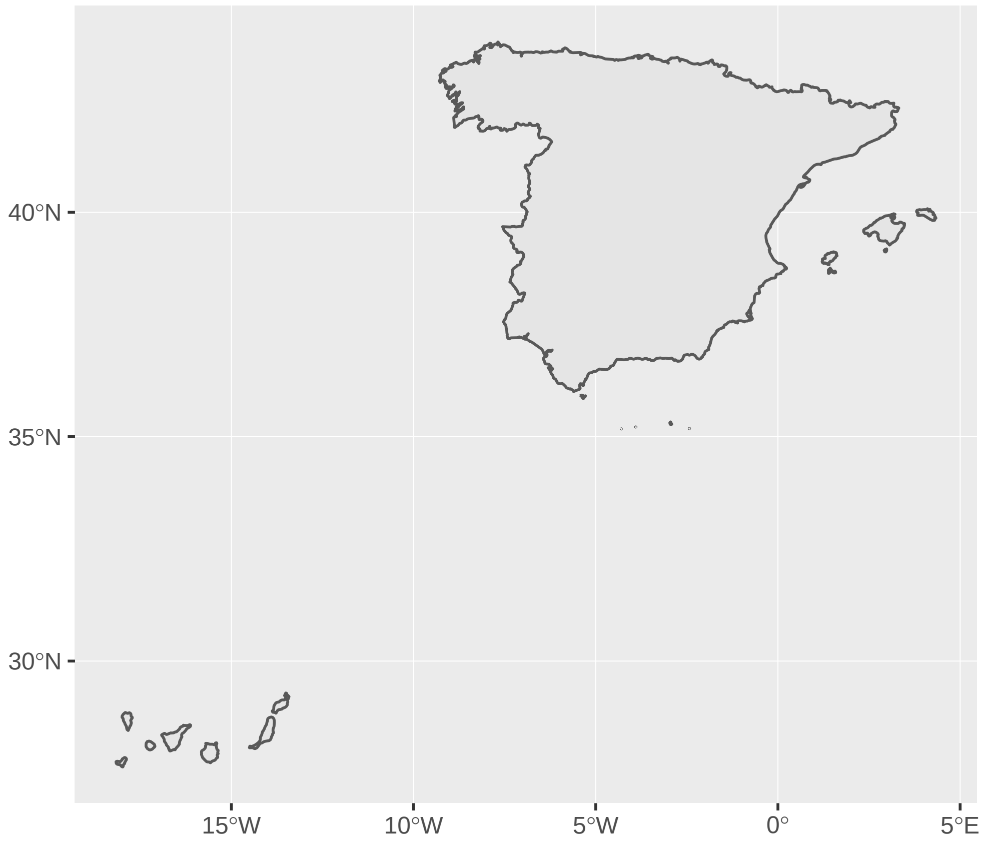
```r
Spain %>% ggplot() +
   geom_sf()
```



1. Where is GRAN Canaria!?
2. You still have other countries/land in the map

Naturally, you can also just filter out the country you want because the `sf` package works very nicely with `dplyr`

```r
# head(country10_sf)
Spain <- country10_sf %>%
  filter(NAME == "Spain")

ggplot(Spain) +
  geom_sf()
```

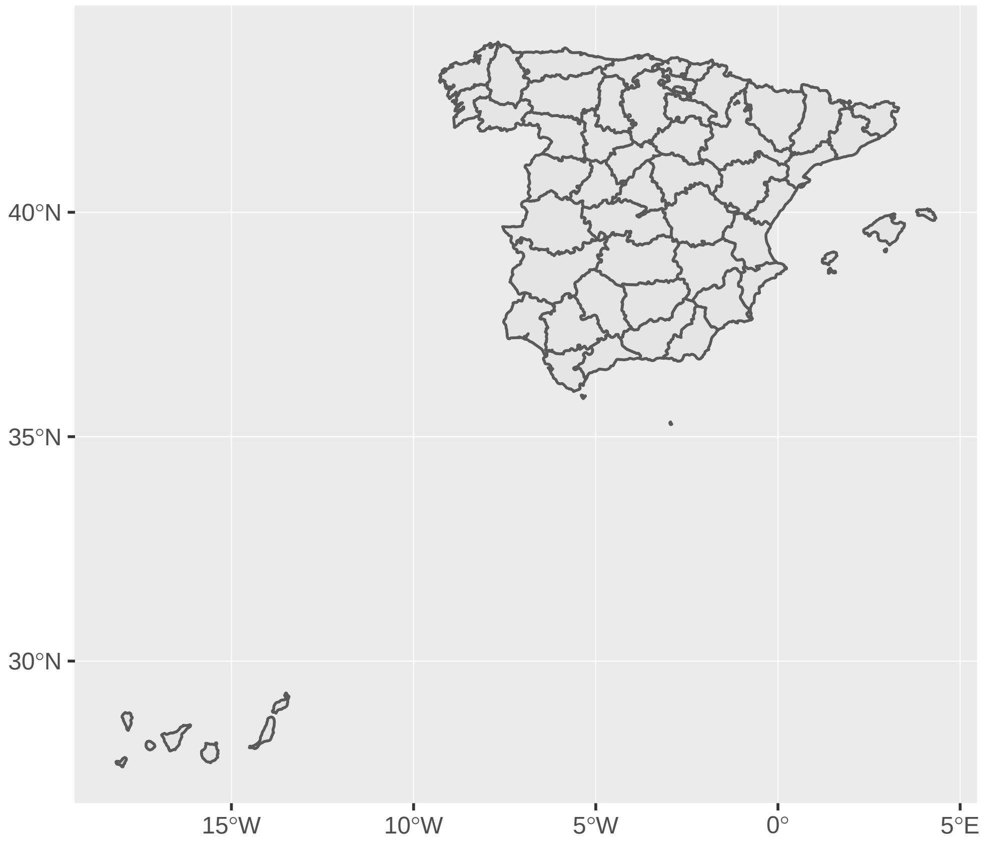


Still, is to coerce I need to know the regions within Spain…

Ways to fix this: 1. Use the higher resolution world map when working at this scale 2. Use a finer resolution regional map.

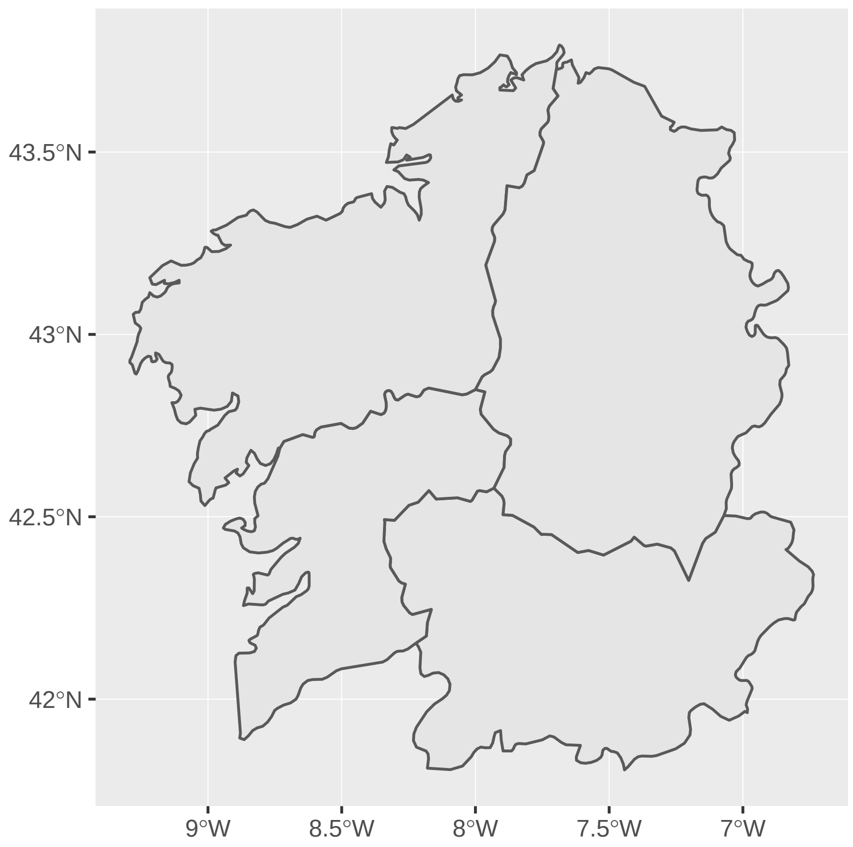
Regional

```r
# ?ne_download
states <- ne_states()
states_sf <- st_as_sf(states)

states_sf_es <- states_sf %>%
  filter(iso_a2=="ES") #Filter out to one country as otherwise it takes forever.

ggplot(states_sf_es) +
  geom_sf()
```



But really, what I really care about is Galicia…

```
states_sf_es %>%
  filter(region == "Galicia") %>%
  ggplot() +
  geom_sf()
```



You can also load your own shapefile

```
Path_sf <- here("Sessions/Spatial Analysis/Data/FAO_AREAS")

# The File
Name_sf <- "FAO_AREAS.shp"

# Load it!
FAO_Areas_sf <- st_read(dsn = Path_sf,
                        layer =file_path_sans_ext(Name_sf))
```

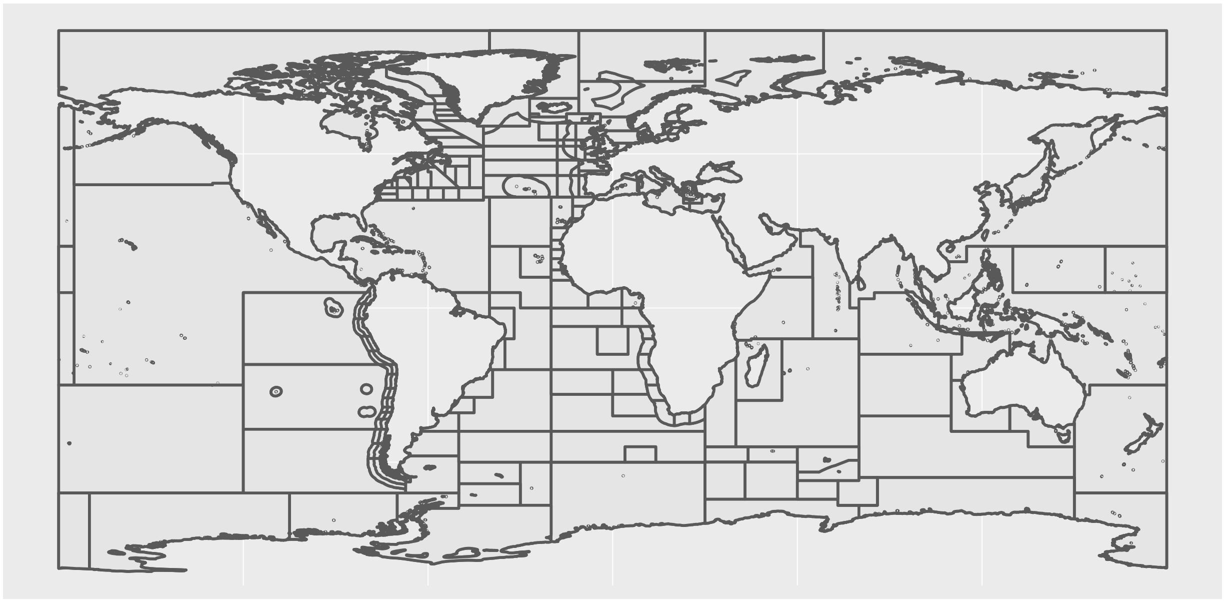
```
## Reading layer `FAO_AREAS' from data source `/Users/carmelia/GitHub/FOL_Studygroup/Sessions/Spatial A
## Simple feature collection with 324 features and 16 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -180 ymin: -85.4703 xmax: 180 ymax: 89.99
## epsg (SRID):    4326
## proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

```
# Explor it
names(FAO_Areas_sf)
```

```
##  [1] "FID"        "F_CODE"     "F_LEVEL"    "F_STATUS"   "OCEAN"
##  [6] "SUBOCEAN"   "F_AREA"     "F_SUBAREA"  "F_DIVISION" "F_SUBDIVIS"
## [11] "F_SUBUNIT"  "ID"         "NAME_EN"    "NAME_FR"    "NAME_ES"
## [16] "SURFACE"    "geometry"
```

```
# head(FAO_Areas_sf)
```

```r
# And plot it (will take a little bit)
ggplot(FAO_Areas_sf) +
  geom_sf()
```



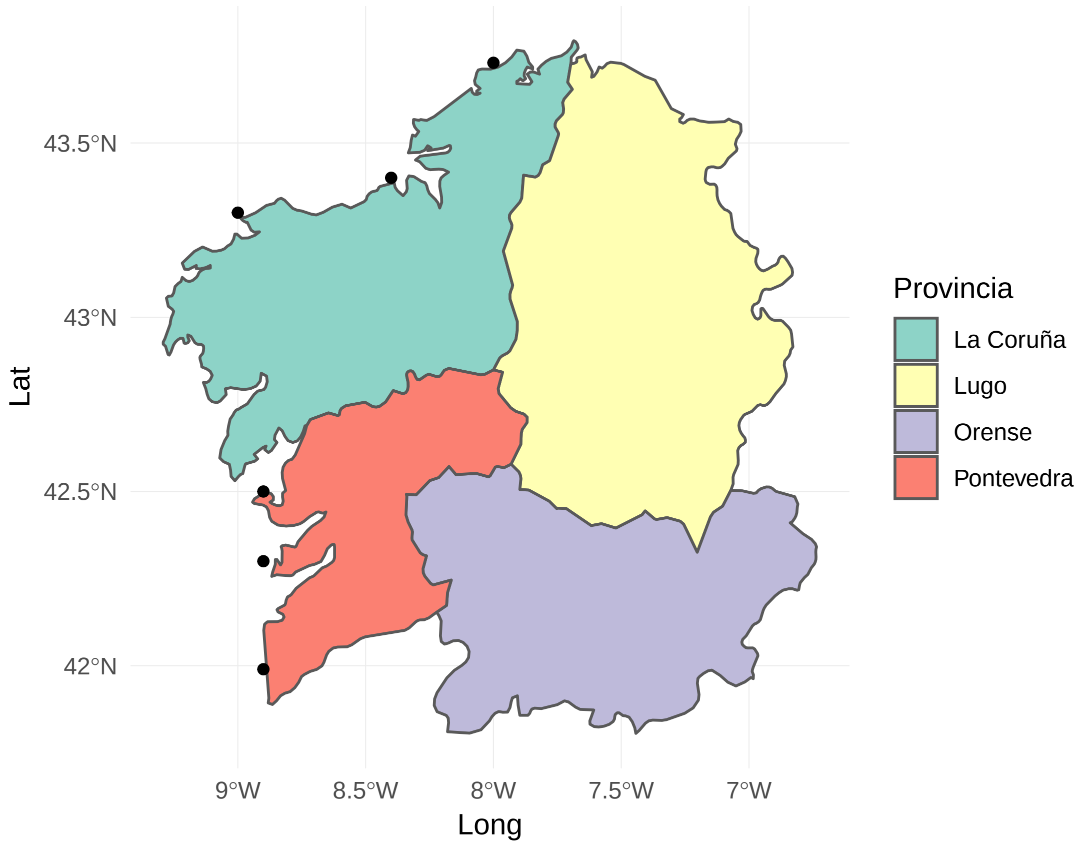
Use of maps in science

- Reference maps (classic figure 1)
- Raster maps
- Geographic entity maps
- Maps with data on top (dots, lines, etc.)

Reference Maps

Reference maps are often used to orient the audience to a study area they may be unfamiliar with. These are most effective when they situate it within some kind of knowledge that they have already. Thus, they often use a small scale and a large scale map.

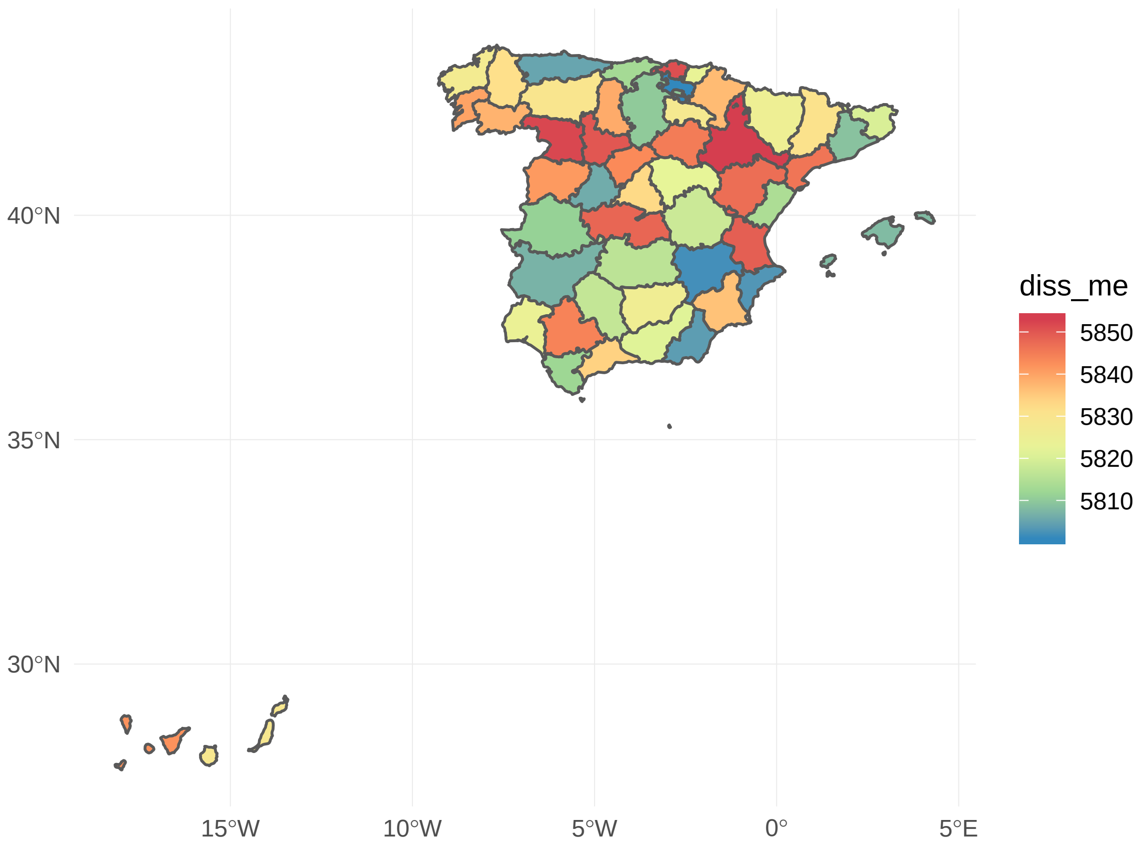
```r
# Random Samples along the coast

Samples <- data.frame(
  Lat = c(41.99,42.3,42.5,43.73,43.4,43.3),
  Long = c(-8.9,-8.9,-8.9,-8,-8.4,-9)
)


states_sf_es %>%
  filter(region == "Galicia") %>%
  ggplot() +
  geom_sf(
    aes(fill = name)
  ) +
  geom_point(data = Samples,
             aes(
              x = Long,
              y = Lat
            ),
            color = "black"
  ) +
  theme_minimal() +
  scale_fill_brewer("Provincia", palette = "Set3")
```



Raster Maps

Rasters are simplifications of spatial data according to a certain resolution size.

```
states_sf_es %>%
  ggplot() +
  geom_sf(
    aes(fill=diss_me)
  ) +
  scale_fill_distiller(palette = "Spectral") +
  theme_minimal()
```



Geographic Entity Maps

We've already made one of these that was filled on a map feature. The main point with these is having a column attached to your dataframe that you are interested in plotting. Then, the polygons are filled based on that.

```
#WARNING: EXAMPLE USES FAKE DATA
# head(states_sf_es)
names(states_sf_es)
```

```
##  [1] "featurecla" "scalerank"  "adm1_code"  "diss_me"    "iso_3166_2"
##  [6] "wikipedia"  "iso_a2"     "adm0_sr"    "name"       "name_alt"
## [11] "name_local" "type"       "type_en"    "code_local" "code_hasc"
## [16] "note"       "hasc_maybe" "region"     "region_cod" "provnum_ne"
## [21] "gadm_level" "check_me"   "datarank"   "abbrev"     "postal"
## [26] "area_sqkm"  "sameascity" "labelrank"  "name_len"   "mapcolor9"
## [31] "mapcolor13" "fips"       "fips_alt"   "woe_id"     "woe_label"
## [36] "woe_name"   "latitude"   "longitude"  "sov_a3"     "adm0_a3"
## [41] "adm0_label" "admin"      "geonunit"   "gu_a3"      "gn_id"
## [46] "gn_name"    "gns_id"     "gns_name"   "gn_level"   "gn_region"
## [51] "gn_a1_code" "region_sub" "sub_code"   "gns_level"  "gns_lang"
## [56] "gns_adm1"   "gns_region" "min_label"  "max_label"  "min_zoom"
## [61] "wikidataid" "name_ar"    "name_bn"    "name_de"    "name_en"
## [66] "name_es"    "name_fr"    "name_el"    "name_hi"    "name_hu"
## [71] "name_id"    "name_it"    "name_ja"    "name_ko"    "name_nl"
## [76] "name_pl"    "name_pt"    "name_ru"    "name_sv"    "name_tr"
## [81] "name_vi"    "name_zh"    "ne_id"      "geometry"
```

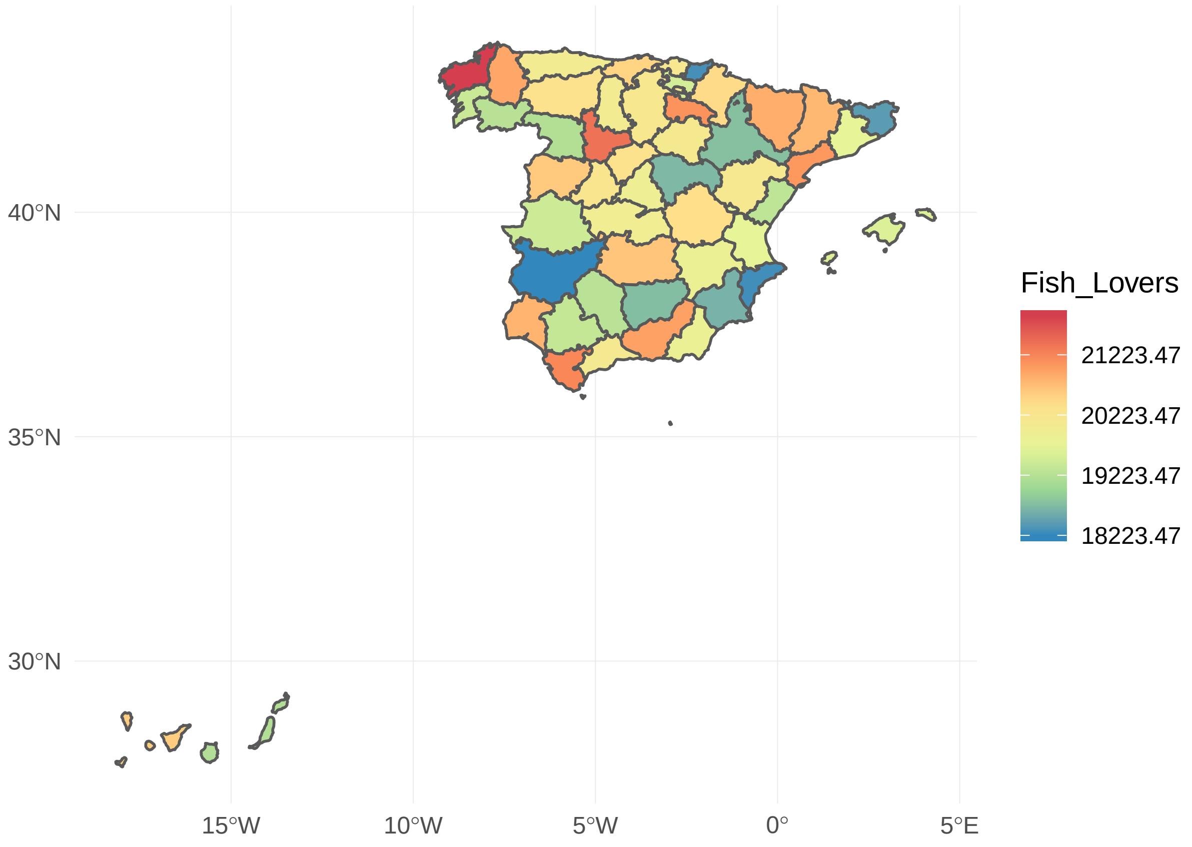
```r
# Create dummy dataset
df <- tibble(adm1_code= states_sf_es$adm1_code,
             Fish_Lovers = rnorm(nrow(states_sf_es), mean=20000, sd=1000))

# Join our data with the shapefile
states_sf_es <- left_join(states_sf_es,
                          df)
```

```
## Joining, by = "adm1_code"
```

```r
names(states_sf_es)
```

```
##  [1] "featurecla"  "scalerank"   "adm1_code"   "diss_me"     "iso_3166_2"
##  [6] "wikipedia"   "iso_a2"      "adm0_sr"     "name"        "name_alt"
## [11] "name_local"  "type"        "type_en"     "code_local"  "code_hasc"
## [16] "note"        "hasc_maybe"  "region"      "region_cod"  "provnum_ne"
## [21] "gadm_level"  "check_me"    "datarank"    "abbrev"      "postal"
## [26] "area_sqkm"   "sameascity"  "labelrank"   "name_len"    "mapcolor9"
## [31] "mapcolor13"  "fips"        "fips_alt"    "woe_id"      "woe_label"
## [36] "woe_name"    "latitude"    "longitude"   "sov_a3"      "adm0_a3"
## [41] "adm0_label"  "admin"       "geonunit"    "gu_a3"       "gn_id"
## [46] "gn_name"     "gns_id"      "gns_name"    "gn_level"    "gn_region"
## [51] "gn_a1_code"  "region_sub"  "sub_code"    "gns_level"   "gns_lang"
## [56] "gns_adm1"    "gns_region"  "min_label"   "max_label"   "min_zoom"
## [61] "wikidataid"  "name_ar"     "name_bn"     "name_de"     "name_en"
## [66] "name_es"     "name_fr"     "name_el"     "name_hi"     "name_hu"
## [71] "name_id"     "name_it"     "name_ja"     "name_ko"     "name_nl"
## [76] "name_pl"     "name_pt"     "name_ru"     "name_sv"     "name_tr"
## [81] "name_vi"     "name_zh"     "ne_id"       "Fish_Lovers" "geometry"
```

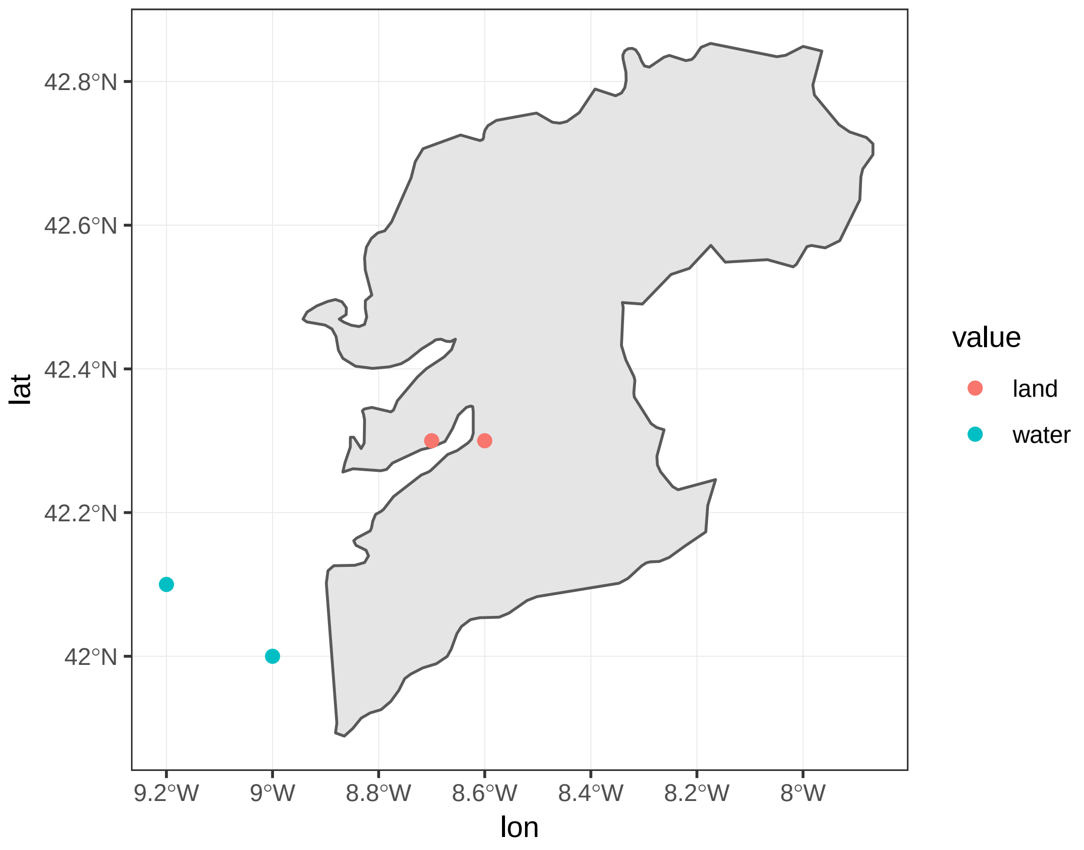
```r
states_sf_es %>%
  ggplot() +
  geom_sf(aes(fill=Fish_Lovers)) +
  scale_fill_distiller(palette = "Spectral",
                       breaks= seq(min(states_sf_es$Fish_Lovers),max(states_sf_es$Fish_Lovers), 1000)
                       ) +
  theme_minimal()
```



Maps with Data on Top

These are actually fairly simple to make in R. Make your map, add your data!

```
map <- states_sf_es %>%
  filter(name == "Pontevedra") %>%
  ggplot() +
  geom_sf()

df <- tibble(lon = c(-9, -9.2, -8.7,-8.6),
             lat = c(42, 42.1, 42.3, 42.3),
             value = c("water", "water", "land","land"))

map +
  geom_point(data=df,
             aes(x=lon,
                 y=lat,
                 color=value),
             size=2) +
  theme_bw()
```



Combining different map layers

Combining different layers on a map is *fairly* simple. You start with your base map layer, and then you add the different `geom_` layers as you like.

```
# made up catch data

df <- tibble(lon = c(runif(25, min=-20, max=-8.6),runif(25, min=2, max=20)),
             lat = runif(50, min=36, max=46),
             Catch = runif(50, min=100, max=1000)
             )



# The Spain's map croped to what I want to show
Spain <- ne_download(scale=10,
                     type="countries",
                     category="cultural",
                     returnclass="sf") %>%
  filter(NAME == "Spain")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "/private/var/folders/83/gdppt_ds3s5dmgtvxtj9mvjw0000gn/T/RtmpodaJ7j", layer: "ne_10m_admin_
## with 255 features
## It has 94 fields
## Integer64 fields read as strings:  POP_EST NE_ID
```

```
# Looks weird to only have a floating Spain
Others <- ne_download(scale=50,
                      type="countries",
                      category="cultural",
                      returnclass="sf") %>%
  filter(NAME %in% c("France","Portugal"))
```

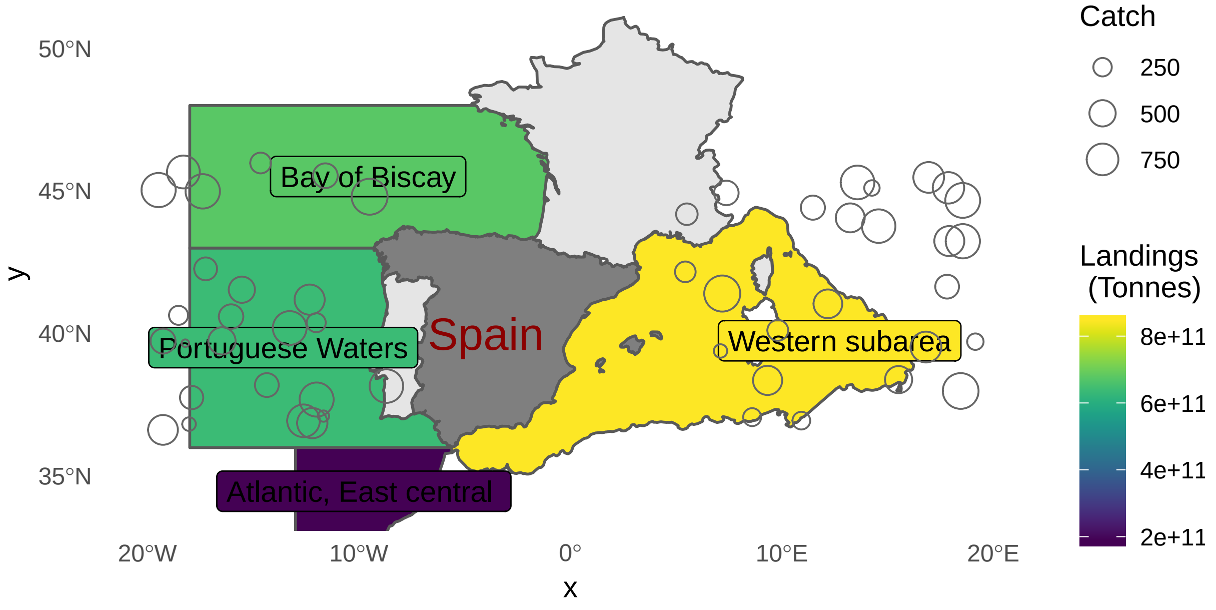
```
## OGR data source with driver: ESRI Shapefile
## Source: "/private/var/folders/83/gdppt_ds3s5dmgtvxtj9mvjw0000gn/T/RtmpodaJ7j", layer: "ne_50m_admin_
## with 241 features
## It has 94 fields
## Integer64 fields read as strings:  POP_EST NE_ID
```

```
# Then FAO Regions layer
FAO_Areas <- FAO_Areas_sf %>%
  filter(
    ID %in% c(98,72,278,21) # sub regions around spain
  ) %>%
  mutate(Name_Clear = gsub( " *\\(.*?\\) *", "", NAME_EN), # create nice names
         Name_Clear = gsub("\\/.*","",Name_Clear))
```

```
## Warning: The `printer` argument is deprecated as of rlang 0.3.0.
## This warning is displayed once per session.
```

```
# Now we make the plot
ggplot() +
  # Spain Land mass
  geom_sf(data = Spain, fill = "grey50") +
  # France and Portugal
  geom_sf(data = Others, fill = "grey90") +
  # FAO Regions by surface area
  geom_sf(data = FAO_Areas,
          aes(fill = SURFACE)
  ) +
  # Add lables to FAO Areas
  geom_sf_label(data = FAO_Areas,
                aes(label = Name_Clear,
                    fill = SURFACE)
  ) +
  # Add particular catch points
  geom_point(data = df,
             aes(
               x = lon,
               y = lat,
               size = Catch
             ),
             shape = 21,
             fill = "transparent",
             colour = "grey40"
  ) +
  # Add Spain's name
  annotate("text",
           x = -4,
           y = 40,
           label = "Spain",
            colour = "darkred",
            size = 6) +
  # Inlcude color-blind friendly scale
  scale_fill_viridis("Landings \n (Tonnes)",
                     option = "viridis") +
  theme_minimal() +
  theme(
    panel.grid.minor = element_blank(),
    panel.grid.major = element_line(color = "transparent")
  ) +
  # Crop the map to what we want to show
  ylim(33.9,50.3) +
  xlim(-20,20)
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may
## not give correct results for longitude/latitude data
```



Some extra points

st_simplify()

```
x <- st_simplify(states_sf_pe, dTolerance = 0.1)
ggplot(x) + geom_sf()

x <- st_simplify(states_sf, dTolerance = 0.1)

ggplot(x) + geom_sf()
```