Geospatial data analysis - Part 1

Note: This resource is adapted from the book: "Spatial Data Science with R and"terra" (https://rspatial.org/index.html 2019-2023. License: CC BY-SA 4.0.)

Spatial objects are typically represented using vector data, which includes information about the *shape* or *geometry* of the objects, along with *additional variables*. For instance, a vector dataset may describe the boundaries of countries (geometry) and also store attributes like country names and population size. Similarly, it can include the geometry of roads in an area, along with attributes such as road type and names.

I. Vector data

A simple representation of spatial data

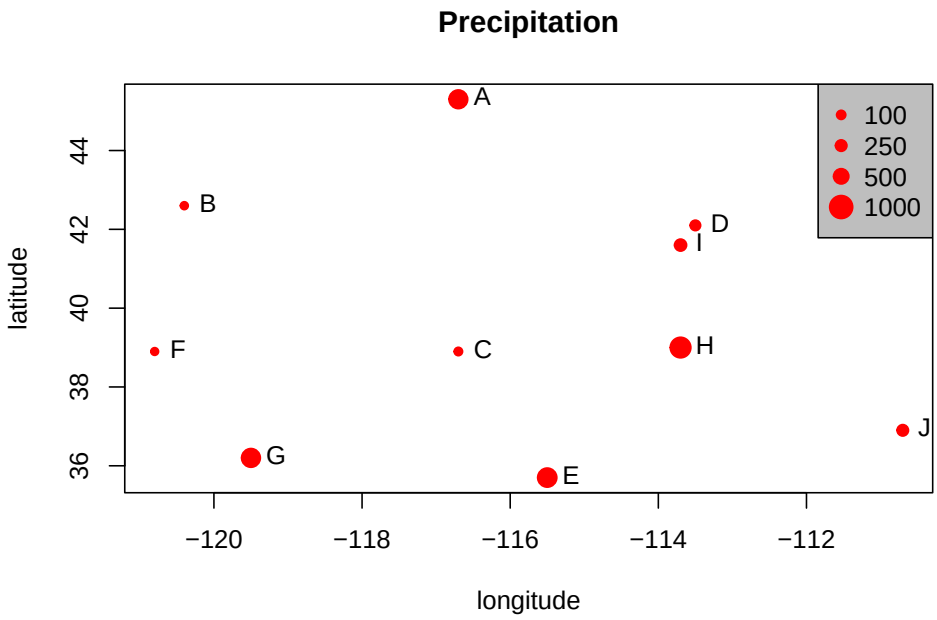
In the example below we make a very simple map using only R base (no geo-spatial packages yet). Note that a *map* is special type of plot (like a scatter plot, barplot, etc.). A *map* is a plot of geospatial data that also has labels and other graphical objects such as a scale bar or legend. The spatial data itself should not be referred to as a map.

```
name <- LETTERS[1:10]
longitude <- c(-116.7, -120.4, -116.7, -113.5, -115.5,
               -120.8, -119.5, -113.7, -113.7, -110.7)
latitude <- c(45.3, 42.6, 38.9, 42.1, 35.7, 38.9,
              36.2, 39, 41.6, 36.9)
stations <- cbind(longitude, latitude)
# Simulated rainfall data
set.seed(0)
precip <- round((runif(length(latitude))*10)^3)
```

A map of point locations is not that different from a basic x-y scatter plot. Below there is a plot (a map in this case) that shows the location of the weather stations, and the size of the dots is proportional to the amount of precipitation. The point size is set with argument `cex`.

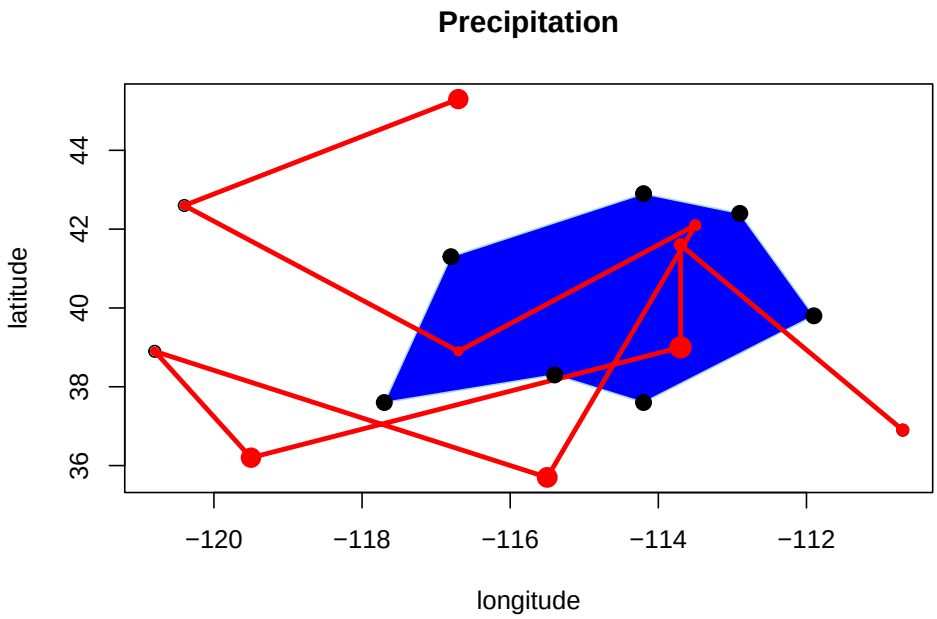
```
psize <- 1 + precip/500
plot(stations, cex=psize, pch=20, col='red', main='Precipitation')
# add names to plot
text(stations, name, pos=4)
# add a legend
breaks <- c(100, 250, 500, 1000)
```

```
legend.psize <- 1+breaks/500
legend("topright", legend=breaks, pch=20, pt.cex=legend.psize
       , col='red', bg='gray')
```



Numeric vectors representing locations can be used to draw simple maps. It also shows how points can (and typically are) represented by pairs of numbers. A line and a polygon can be represented by a number of these points. Polygons need to "closed", that is, the first point must coincide with the last point, but the `polygon` function took care of that for us.

```
lon <- c(-116.8, -114.2, -112.9, -111.9, -114.2, -115.4, -117.7)
lat <- c(41.3, 42.9, 42.4, 39.8, 37.6, 38.3, 37.6)
x <- cbind(lon, lat)
plot(stations, main='Precipitation')
polygon(x, col='blue', border='light blue')
lines(stations, lwd=3, col='red')
points(x, cex=2, pch=20)
points(stations, cex=psize, pch=20, col='red', main='Precipitation')
```



`SpatVector` objects from scratch (Terra package):

The main vector data types are points, lines and polygons. Geometry of these data structures consists of sets of coordinate pairs (x, y). The simplest case are called points. Each point has one coordinate pair, and *n* associated variables (e.g. a point representing a coffee shop has its coordinates and it can also have the coffee shop name, number of stories, etc.).

The `terra` package defines a set of *classes* with names that start with `Spat` to represent spatial data.

For vector data, the relevant class is `SpatVector`. These classes represent geometries as well as attributes (variables) describing the geometries.

- Create the data

```
longitude <- c(-116.7, -120.4, -116.7, -113.5, -115.5
               , -120.8, -119.5, -113.7, -113.7, -110.7)
latitude <- c(45.3, 42.6, 38.9, 42.1, 35.7, 38.9
              , 36.2, 39, 41.6, 36.9)
lonlat <- cbind(longitude, latitude)
```

- Load the `terra` package from the library. Use the `vect` function to create a `SpatVector` object (by promoting the `lonlat` matrix).

```
library(terra)
pts <- vect(lonlat)
class (pts)
```

```
## [1] "SpatVector"
## attr(,"package")
## [1] "terra"
```

- The object has the coordinates we supplied, but also an `extent`. This spatial extent was computed from the coordinates. There is also an "empty" coordinate reference system (CRS)

```
pts
```

```
##  class       : SpatVector
##  geometry    : points
##  dimensions  : 10, 0  (geometries, attributes)
##  extent      : -120.8, -110.7, 35.7, 45.3  (xmin, xmax, ymin, ymax)
##  coord. ref. :
```

```
geom(pts)
```

```
##       geom part      x    y hole
##  [1,]    1    1 -116.7 45.3    0
##  [2,]    2    1 -120.4 42.6    0
##  [3,]    3    1 -116.7 38.9    0
##  [4,]    4    1 -113.5 42.1    0
##  [5,]    5    1 -115.5 35.7    0
##  [6,]    6    1 -120.8 38.9    0
##  [7,]    7    1 -119.5 36.2    0
##  [8,]    8    1 -113.7 39.0    0
##  [9,]    9    1 -113.7 41.6    0
## [10,]   10    1 -110.7 36.9    0
```

- Recreate the object, and now provide a CRS.

```
crdref <- "+proj=longlat +datum=WGS84"
pts <- vect(lonlat, crs=crdref)
pts
```

```
##  class       : SpatVector
##  geometry    : points
##  dimensions  : 10, 0  (geometries, attributes)
##  extent      : -120.8, -110.7, 35.7, 45.3  (xmin, xmax, ymin, ymax)
##  coord. ref. : +proj=longlat +datum=WGS84 +no_defs
```

- You can use a `data.frame` with the same number of rows as there are geometries to define the attributes (variables) of a `SpatVector` object by using the `atts` parameter. The following example creates a simulated data for precipitation and add it to the `SpatVector`.

```
precipvalue <- runif(nrow(lonlat), min=0, max=100)
df <- data.frame(ID=1:nrow(lonlat), precip=precipvalue)
ptv <- vect(lonlat, atts=df, crs=crdref)
ptv
```

```
##  class       : SpatVector
##  geometry    : points
##  dimensions  : 10, 2  (geometries, attributes)
##  extent      : -120.8, -110.7, 35.7, 45.3  (xmin, xmax, ymin, ymax)
##  coord. ref. : +proj=longlat +datum=WGS84 +no_defs
##  names       :    ID precip
##  type        : <int>  <num>
##  values      :     1  6.179
##                    2   20.6
##                    3  17.66
```

- You can also create a `SpatVector` **from** a data frame. The following code creates a vector (geometry: points) with the coordinates of cities in Ontario - Canada. We use the `maps` package that has useful information about cities and maps of the world (see https://cran.r-project.org/web/packages/maps/maps.pdf).

```
require(maps)
data(canada.cities)
head(canada.cities)
```

```
##            name country.etc    pop   lat    long capital
## 1 Abbotsford BC          BC 157795 49.06 -122.30       0
## 2      Acton ON          ON   8308 43.63  -80.03       0
## 3 Acton Vale QC          QC   5153 45.63  -72.57       0
## 4    Airdrie AB          AB  25863 51.30 -114.02       0
## 5    Aklavik NT          NT    643 68.22 -135.00       0
## 6    Albanel QC          QC   1090 48.87  -72.42       0
```

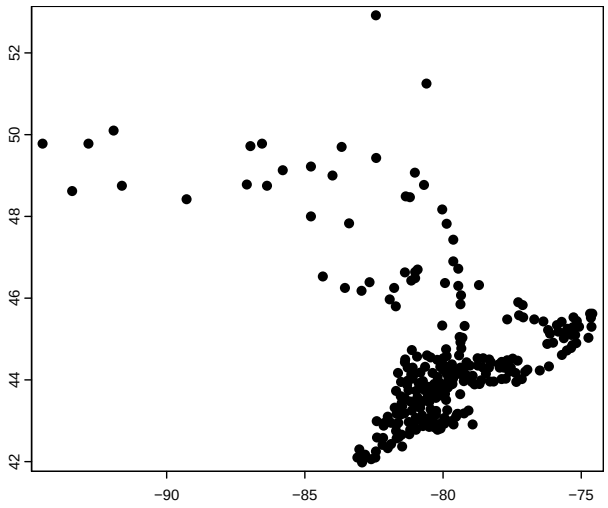
```
# get just the cities for Ontario
library(dplyr)
citiesOnt <- canada.cities%>%
  filter(country.etc =="ON")

#promote this dataframe to SpatVector
citiesOnt <- vect(citiesOnt,geom=c("long","lat"))
citiesOnt
```

```
##  class       : SpatVector
##  geometry    : points
##  dimensions  : 258, 4  (geometries, attributes)
##  extent      : -94.49, -74.6, 41.98, 52.92  (xmin, xmax, ymin, ymax)
##  coord. ref. :
##  names       :          name country.etc   pop capital
##  type        :         <chr>       <chr> <int>   <int>
##  values      :      Acton ON          ON  8308       0
##                Alexandria ON          ON  3604       0
##                  Alliston ON          ON 10353       0
```

```
class(citiesOnt)
```

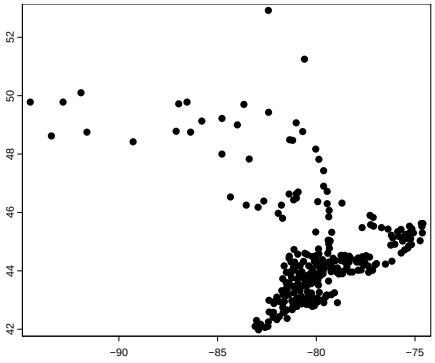
```
## [1] "SpatVector"
## attr(,"package")
## [1] "terra"
```

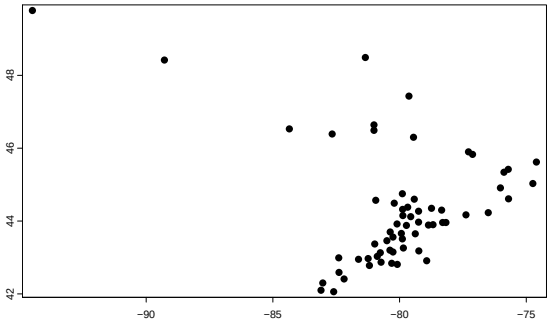
```
plot(citiesOnt)
```



- Attributes in a `SpatVector` can be used for filtering. In the following example, the second plot keeps only cities in Ontario with a population larger than 10000.

```
par(mar = c(4, 4, .1, .1))
plot(citiesOnt)
plot(citiesOnt[citiesOnt$pop>10000])
```





Lines and polygons

- Use `type = "lines"` for lines and `type = "polygons"` for polygons. Both are created connecting the points in the provided order. If `type = "polygons"` the resulting geometry is a closed figure.

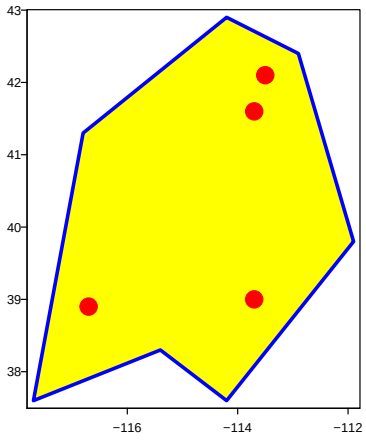
```
lon <- c(-116.8, -114.2, -112.9, -111.9, -114.2, -115.4, -117.7)
lat <- c(41.3, 42.9, 42.4, 39.8, 37.6, 38.3, 37.6)
lonlat <- cbind(lon, lat)
lonlat
```

```
##         lon  lat
## [1,] -116.8 41.3
## [2,] -114.2 42.9
## [3,] -112.9 42.4
## [4,] -111.9 39.8
## [5,] -114.2 37.6
## [6,] -115.4 38.3
## [7,] -117.7 37.6
```

```
pols <- vect(lonlat, type="polygons", crs=crdref)
pols
```

```
##  class       : SpatVector
##  geometry    : polygons
##  dimensions  : 1, 0  (geometries, attributes)
##  extent      : -117.7, -111.9, 37.6, 42.9  (xmin, xmax, ymin, ymax)
##  coord. ref. : +proj=longlat +datum=WGS84 +no_defs
```

```
plot(pols, las=1)
plot(pols, border='blue', col='yellow', lwd=3, add=TRUE)
points(pts, col='red', pch=20, cex=3)
```



Reading and writing spatial vector files

The `shapefile` is the most commonly used file format for vector data.

NOTE: A SHAPEFILE IS REALLY A SET OF AT LEAST THREE (IDEALLY FOUR) FILES, WITH ALL THE SAME NAME, BUT DIFFERENT EXTENSION. FOR SHAPEFILE X YOU *MUST* HAVE, IN THE SAME DIRECTORY, THESE THREE FILES: X.SHP, X.SHX, X.DBF, AND IDEALLY ALSO X.PRJ

- Let's include the borders from a shape file to the cities for Ontario. We have downloaded the shape file used in this example from Ontario Ministry of Natural Resources and Forestry at https://geohub.lio.gov.on.ca/maps/mnr-district.
- Use the `vect` function to read the file. Following code assumes that the `r` script has been saved in the same location as `MNR_District` folder (i.e. both are in the same parent folder).

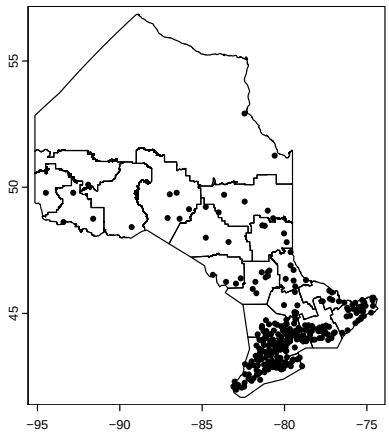
```
library(terra)
filename <- paste0(getwd(),"/MNR_District/MNR_District.shp")
basename(filename)
```

```
## [1] "MNR_District.shp"
```

```
ontarioShape <- vect(filename)
ontarioShape
```

```
##  class       : SpatVector
##  geometry    : polygons
##  dimensions  : 18, 9  (geometries, attributes)
##  extent      : -95.15368, -74.31952, 41.67656, 56.85937  (xmin, xmax, ymin, ymax)
##  source      : MNR_District.shp
##  coord. ref. : lon/lat WGS 84 (EPSG:4326)
##  names       : OGF_ID        DISTRICT_N REGION_NAM GEOMETRY_U EFFECTIVE_
##  type        :  <int>             <chr>      <chr>      <chr>      <chr>
##  values      :      1   Aurora Midhurs~   Southern 2023/10/08 2023/10/08
##                     2 Pembroke District   Southern 2023/10/08 2023/10/08
##                     3   Far North Dist~  Northwest 2023/10/08 2023/10/08
##  SYSTEM_DAT OBJECTID SHAPEAREA SHAPELEN
##       <chr>    <int>     <int>    <int>
##  2023/10/08        1         0        0
##  2023/10/08        2         0        0
##  2023/10/08        3         0        0
```

```
plot(ontarioShape)
points(citiesOnt)
```



- Use `writeVector()` to save a vector file. You need to add argument `overwrite=TRUE` if you want to overwrite an existing file. The following code saves a `shapefile` with the `SpatVector citiesOnt` inside a folder (named as **"ct"** in this example) in your current working directory.

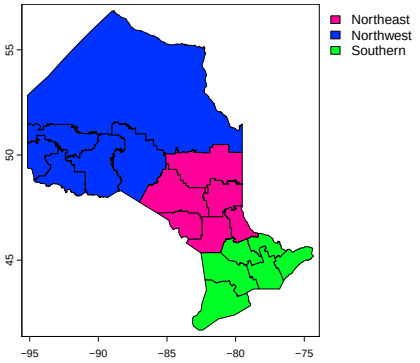
Note: The **"ct"** folder must exists in advance. You may need to create it, for example, using the file explorer of windows.

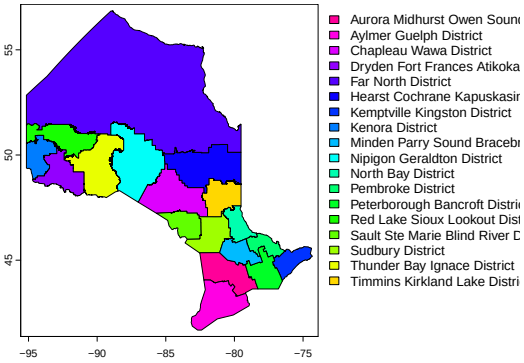
```
# get current working directory
currentDir <- getwd()
# save the shapefile in a previosly created folder inside your working directory
outfile <- paste0(currentDir,"/ct/citiesOntario.shp")
writeVector(citiesOnt, outfile, overwrite=TRUE)
```

Vector data manipulation

- `terra::plot()` method let you add colors to your map by simply using a categorical variable as parameter. You can use a categorical attribute to assign colors to the corresponding areas in the map.

```
library(terra)
plot(ontarioShape,"REGION_NAM")
plot(ontarioShape,"DISTRICT_N")
```





- To extract the attributes (data.frame) from a `SpatVector`, use:

```
d <- as.data.frame(ontarioShape)
head(d)
```

```
##   OGF_ID                          DISTRICT_N REGION_NAM GEOMETRY_U EFFECTIVE_
## 1      1 Aurora Midhurst Owen Sound District   Southern 2023/10/08 2023/10/08
## 2      2                   Pembroke District   Southern 2023/10/08 2023/10/08
## 3      3                  Far North District  Northwest 2023/10/08 2023/10/08
## 4      4        Kemptville Kingston District   Southern 2023/10/08 2023/10/08
## 5      5     Red Lake Sioux Lookout District  Northwest 2023/10/08 2023/10/08
## 6      6             Chapleau Wawa District  Northeast 2023/10/08 2023/10/08
##   SYSTEM_DAT OBJECTID SHAPEAREA SHAPELEN
## 1 2023/10/08        1         0        0
## 2 2023/10/08        2         0        0
## 3 2023/10/08        3         0        0
## 4 2023/10/08        4         0        0
## 5 2023/10/08        5         0        0
## 6 2023/10/08        6         0        0
```

- Use `geom()` to extract the geometry as a matrix (although this is rarely needed).

```
g <- geom(ontarioShape)
head(g)
```

```
##      geom part         x        y hole
## [1,]    1    1 -82.56028 45.35729    0
## [2,]    1    1 -81.07638 45.36194    0
## [3,]    1    1 -81.07795 45.35870    0
## [4,]    1    1 -80.50045 44.99963    0
## [5,]    1    1 -80.50039 44.99962    0
## [6,]    1    1 -79.98732 44.93258    0
```

- A "well-known text" is a standard text representation of geometry objects. Set the parameter `wkt` to `TRUE` to extract the geometry as "well-known text".

```
g <- geom(ontarioShape, wkt=TRUE)
substr(g[1], 1, 50)
```

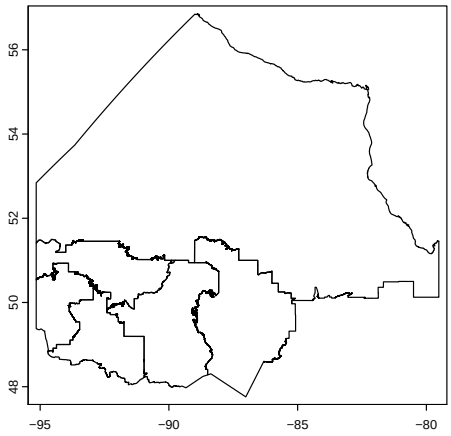
```
## [1] "POLYGON ((-82.560277 45.357293, -81.076382 45.3619"
```

- `SpatVector` variables can be accessed as variables in data frames. You can use variables to retrieve a sub-area.

```
ontarioShape$REGION_NAM
```

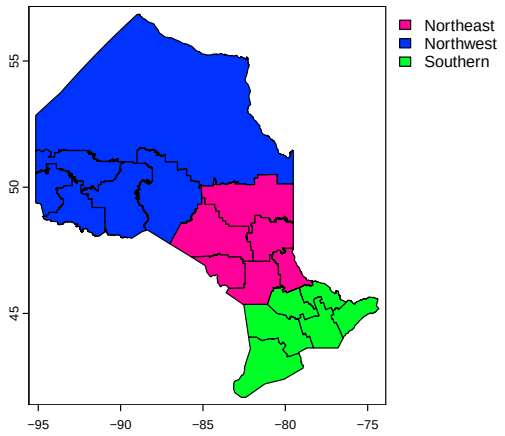
```
##  [1] "Southern"  "Southern"  "Northwest" "Southern"  "Northwest" "Northeast"
##  [7] "Northwest" "Northeast" "Southern"  "Northeast" "Northeast" "Southern"
## [13] "Northwest" "Northeast" "Northeast" "Southern"  "Northwest" "Northwest"
```

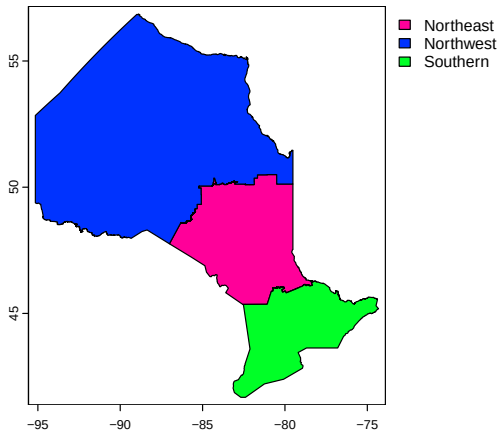
```
ontarioNWShape <- ontarioShape[ontarioShape$REGION_NAM=="Northwest"]
plot(ontarioNWShape)
```



- Use the `aggregate()` function to aggregate (dissolve) polygons that have the same value for an attribute of interest. Let's compare the map from `ontarioShape` before and after aggregate it by region.

```
ontarioShapeAgg <- aggregate(ontarioShape, by='REGION_NAM')
plot(ontarioShape,"REGION_NAM")
plot(ontarioShapeAgg,"REGION_NAM")
```





Simple maps with `rnaturalearth` package

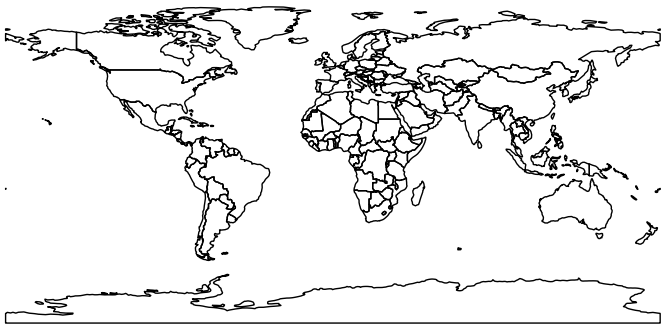
`rnaturalearth` is an R package to hold and facilitate interaction with Natural Earth map data. Natural Earth is a public domain map dataset including vector country and other administrative boundaries.

- `rnaturalearthdata`, `rnaturalearthhires` packages need to be installed in order to use all `rnaturalearth` capabilities.

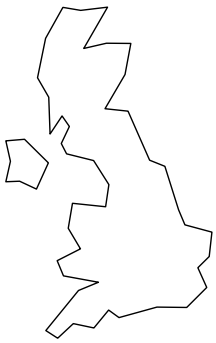
```
if(!require(rnaturalearthdata))
  install.packages("rnaturalearthdata")

if(!require(rnaturalearthhires))
  install.packages("rnaturalearthhires"
                   , repos = "https://ropensci.r-universe.dev"
                   , type = "source")

library(rnaturalearth)
```

- Use `ne_countries()` for country (admin-0) boundaries. You can control several parameters of the map such as `scale`, and `type`.

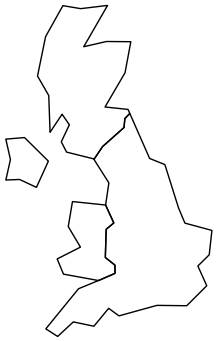
```
    # world at small scale (low resolution)
    terra::plot(ne_countries(type = "countries", scale = "small"))
```



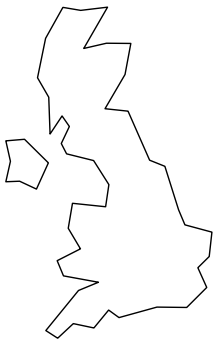
```r
# countries, UK undivided
terra::plot(ne_countries(country = "united kingdom", type = "countries"))
```



```r
# map_units, UK divided into England, Scotland, Wales and Northern Ireland
terra::plot(ne_countries(country = "united kingdom", type = "map_units"))
```



```
# countries, small scale
terra::plot(ne_countries(country = "united kingdom", scale = "small"))
```

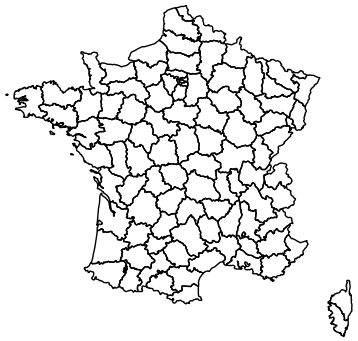


```
# countries, medium scale
terra::plot(ne_countries(country = "united kingdom", scale = "medium"))
```

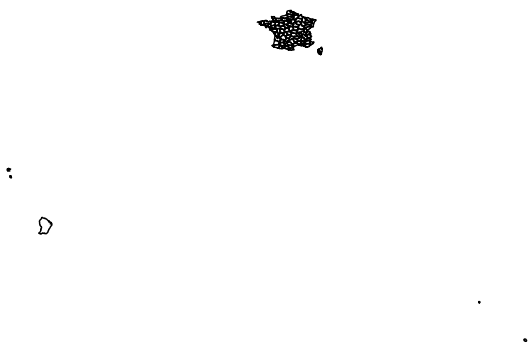
```
    # countries, large scale
    terra::plot(ne_countries(country = "united kingdom", scale = "large"))
```

- Use `ne_states()` for boundaries within countries (admin-1). Use `geounit` or `country` parameter to define the territory.

```
# states geounit='france'
terra::plot(ne_states(geounit = "france"))
```

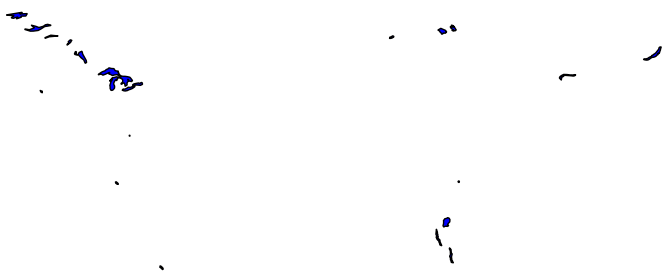


```
# states country='france'
terra::plot(ne_states(country = "france"))
```

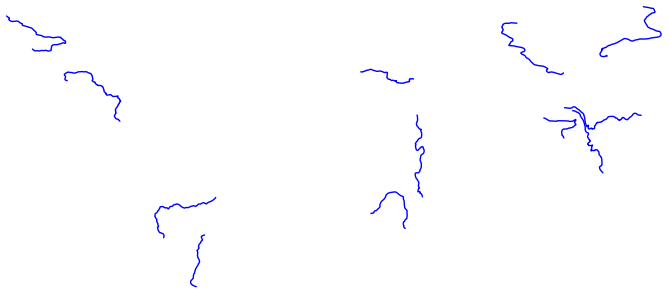


- `rnaturalearth` has the `ne_download()` function to facilitate download of other vector and raster maps. Each Natural Earth dataset is characterised on the website according to `scale`, `type` and `category`. rnaturalearth allows you to specify `scale`, `type` and `category` and will construct the url and download the corresponding file.

```
# lakes
lakes110 <- ne_download(scale = 110, type = "lakes", category = "physical")
terra::plot(lakes110, col = "blue")
```



```r
# rivers
rivers110 <- ne_download(scale = 110
                         , type = "rivers_lake_centerlines"
                         , category = "physical")
terra::plot(rivers110, col = "blue")
```



NOTE: A LIST OF VECTOR LAYERS AVAILABLE VIA `NE_DOWNLOAD(TYPE=[LAYER_NAME], SCALE=)` CAN BE FOUND AT:
HTTPS://CRAN.R-PROJECT.ORG/WEB/PACKAGES/RNATURALEARTH/VIGNETTES/RNATURALEARTH.HTML

- ***Spatial vectors*** returned by `rnaturalearth` are objects from `sp` (outdated - default) or `sf` packages. Use the `vect` function from the `terra` package to convert them to `SpatVector`

objects. This let you use `terra` capabilities in vectors from `rnaturalearth` (e.g. color the map by using a categorical variable)

```
colombiaShape <- vect(ne_states(country = "colombia"))
colombiaShape
```

```
##  class       : SpatVector
##  geometry    : polygons
##  dimensions  : 34, 121  (geometries, attributes)
##  extent      : -81.7237, -66.87506, -4.236484, 13.57836  (xmin, xmax, ymin, ymax)
##  coord. ref. : +proj=longlat +datum=WGS84 +no_defs
##  names       :      featurecla scalerank adm1_code diss_me iso_3166_2 wikipedia
##  type        :           <chr>     <int>     <chr>   <int>      <chr>     <chr>
##  values      : Admin-1 states~         5  COL-1406    1406     CO-NAR        NA
##                Admin-1 states~         5  COL-1407    1407     CO-PUT        NA
##                Admin-1 states~         5  COL-1415    1415     CO-CHO        NA
##  iso_a2 adm0_sr     name name_alt (and 111 more)
##   <chr>   <int>    <chr>    <chr>
##      CO       5   Nariño       NA
##      CO       1 Putumayo       NA
##      CO       5    Chocó       NA
```

```
d <- as.data.frame(colombiaShape)

# Using the "name" feature to colour Colombian departments
# (Colombian administrative areas)
terra::plot(colombiaShape, "name")
```