# Mapping

**Materials for an *In Development* Course on GIS & Mapping**

Andrew Grogan-Kaylor

2024-11-22

# Table of contents

# List of Figures

# List of Tables

# 1 Introduction to Mapping and GIS

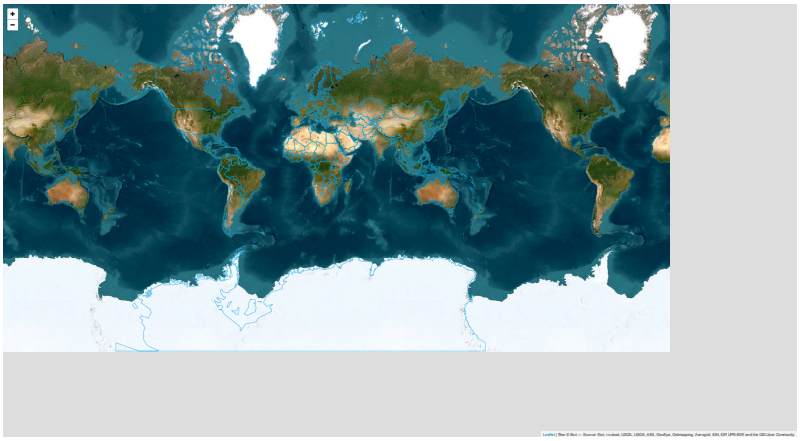


Figure 1.1: Countries of the World

Mapping and GIS have become an increasingly important part of social research, advocacy, and program and policy development. This website contains materials for an *in development* course on *Mapping and GIS*.

The content of this site is built around the use of R software, though many of the concepts are applicable across GIS and mapping applications such as R, Stata, ArcGIS and QGIS.

# Part I

# Introduction

# 2 Possible Course Outline

> **!** Mapping is FUN **and** CHALLENGING
>
> Mapping is fun. Many people enjoy looking at maps, and find them informative and helpful. Many people find it very fun to learn how to make maps. At the same time, making even basic maps with geographic data can be challenging. Making an aesthetically pleasing and useful map can be very challenging. We will work hard in this course to learn these important skills. At the same time, we will have fun doing so!

## 2.1 5 Week MiniCourse

1. Introduction
    - Introduction to the Course
    - Introduction to Shapefiles (and Possibly `sf` Objects) As GIS Data
    - Introduction to Appropriate Software: R / RStudio / `sf` / `ggplot`; or ArcGIS; or QGIS; or Tableau
    - Quick Mapping Exercise
2. Basic GIS Skills
    - Symbology
    - Joining by Attribute
3. Data With Geographic Coordinates & Geographic Concepts
    - Data With Latitude and Longitude
    - Coordinate Reference Systems
4. Geocoding
5. Lab Day for Final Projects

## 2.2 More Advanced Topics

1. Basemaps e.g. `leaflet` (Chapter 22)
2. Projections, projecting and re-projecting geographic data (Chapter 5)
3. Geoprocessing (Spatial joins; Spatial Selection; Selection by Attribute)

## 2.3 Full Semester Course

- Begin with a coding (R) approach to the topics listed in Section 2.1.
- Revisit these topics, and discuss each topic in more depth, with drag and drop software: ArcGIS; QGIS; Tableau. Build in more advanced topics (Section 2.2).
- Finish up with a *Mapping Showcase* constructing useful maps with aesthetically pleasing and clearly understandable design elements.

# 3 Introduction to R

- Introductory content on R can be found here: https://globalfamilies.quarto.pub/global-families-project/quick-intro-R.html.
- Introductory content on ggplot for graphing and data visualization can be found here: https://globalfamilies.quarto.pub/global-families-project/quick-intro-ggplot2.html.

# Part II

# Geographical and GIS Concepts

# 4 Latitude and Longitude

## 4.1 Introduction

Latitude and longitude are coordinates for locating objects on earth.

- Latitude represents distance from the *equator*.
  - 0 latitude is at the equator.
  - +90 latitude, or 90N, is at the North Pole.
  - -90 latitude, or 90S, is at the South Pole.
- Longitude represents distance from the *prime meridian*.
  - 0 longitude is at the prime meridian.
  - +180 and -180, or 180E and 180W, meet at the other side of the world.

## 4.2 Call the Libraries

```
library(plotly)
```

## 4.3 Generate Some Random Coordinates

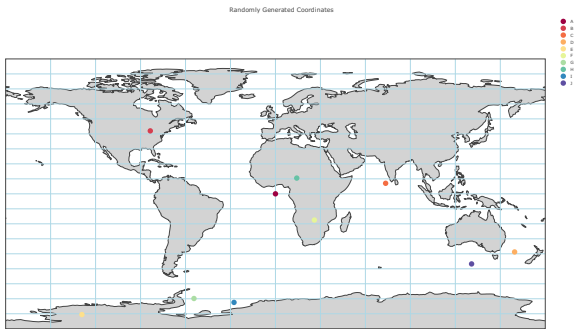
```
set.seed(3846) # random seed

N <- 10 # number of points

# latitude from -90 to +90
```

```r
latitude <- runif(N, min = -90, max = 90)

# longitude from + -180 to + 180

longitude <- runif(N, min = -180, max = 180)

# 1st point reset to 0, 0

latitude[1] <- 0 # equator

longitude[1] <- 0 # prime meridian

latitude[2] <- 42 # Ann Arbor-ish

longitude[2] <- -83.5 # Ann Arbor-ish

# label

label <- LETTERS[1:N] # label with letters of alphabet

# dataframe

mydata <- data.frame(latitude, longitude, label)

mydata # replay
```

```
     latitude  longitude label
1    0.000000    0.00000     A
2   42.000000  -83.50000     B
3    7.023357   73.49444     C
4  -38.775631  159.53317     D
5  -80.547165 -129.12572     E
6  -17.472905   25.95547     F
7  -69.826706  -54.36372     G
8   10.498416   14.20623     H
9  -72.440487  -27.65036     I
10 -46.753424  130.83572     J
```

## 4.4 Map The Coordinates

```
g <- list(lonaxis = list(showgrid = T, # geographic parameters
                         gridcolor = "lightblue"),
          lataxis = list(showgrid = T,
                         gridcolor = "lightblue"),
          showland = TRUE,
          landcolor = toRGB("lightgrey"))

mymap <- plot_geo(mydata) %>%
  add_markers(x = ~longitude,
              y = ~latitude,
              color = ~label,
              colors = "Spectral",
              marker = list(size = 15)) %>%
  layout(title = "Randomly Generated Coordinates",
         geo = g)

mymap # replay
```



# 5 Map Projections

Map projections exist because we are trying to take the round globe of the earth, and project it onto a 2 dimensional surface. Because a spherical globe can not be projected onto a flat surface without *some* distortion, different projections make different choices about the kind of distortion involved.


Figure 5.1: A Globe And A Flat Map

> **Note**
>
> This page is mostly a conceptual overview, and not code-focused. However, the code is provided for the sake of transparency and teaching. It is not necessary to understand the code here. But I learned a lot from: https://plotly.com/r/#maps.

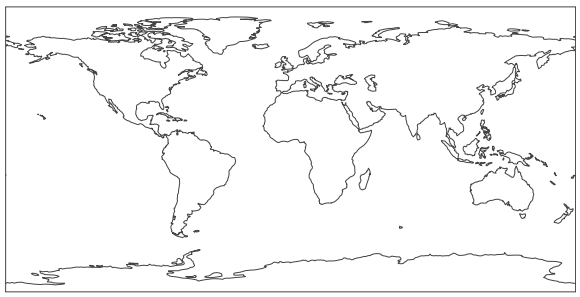
## 5.1 Set Up The Map

## 5.2 Call `plotly` Library

```
library(plotly)
```

## 5.3 A Basic Map

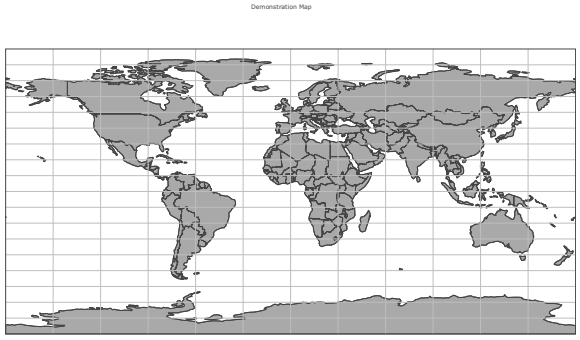
```
# a very basic map could be created with:

library(plotly)

mymap0 <- plot_geo() # create basic map; read into mymap0

mymap0 # replay
```



## 5.4 A More Advanced Map

> **Note**
>
> Again, it is not necessary to understand the code to understand the conceptual ideas of this page. The code below–especially the first code chunk–is admittedly a little complicated, mostly because I added options to get the map to look exactly the way that I wanted.

```
mymap <- plot_geo() %>%
  layout(title = "Demonstration Map",
         geo = list(showland = TRUE, # show land
                    landcolor = toRGB("darkgrey"), # land color
                    showcountries = TRUE, # show countries
                    showocean = FALSE, # show ocean
                    oceancolor = "lightblue", # ocean color
                    lataxis = list(showgrid = TRUE, # latitude options
                                   gridcolor = toRGB("grey")),
                    lonaxis = list(showgrid = TRUE, # longitude options
                                   gridcolor = toRGB("grey"))))

mymap # replay
```



## 5.5 Map Projections

### 5.5.1 Globe (Orthographic)

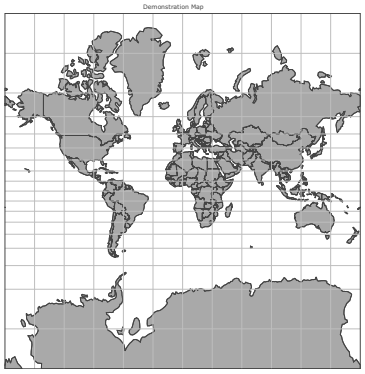
An *orthographic* projection reprsents the globe with 3 dimensional accuracy.

```
mymap %>%
  layout(geo = list(projection = list(type = 'orthographic')))
```


### 5.5.2 Mercator

A *Mercator* projection reprsents the earth with perpendicular latitude and longitude. This projection can be helpful in some kinds of navigation, but areas of landmasses are distorted, especially as one approaches the poles.

```
mymap %>%
  layout(geo=list(projection = list(type = 'mercator')))
```




### 5.5.3 Mollweide

The *Mollweide* projection is an *equal area* projection. As a consequence, latitude and longitude lines are not perpendicular, and the shapes of some landmasses may appear to be distorted.

```
mymap %>%
  layout(geo=list(projection = list(type = 'mollweide')))
```

### 5.5.4 Robinson

The *Robinson* projection is an attempt to compromise between equal areas and a natural looking map.

```
mymap %>%
  layout(geo=list(projection = list(type = 'robinson')))
```

# 6 Coordinate Reference Systems (CRS)

## 6.1 Coordinate Reference Systems

> "Map projections try to portray the surface of the earth or a portion of the earth on a flat piece of paper or computer screen. A coordinate reference system (CRS) then defines, with the help of coordinates, how the two-dimensional, projected map in your GIS is related to real places on the earth. The decision as to which map projection and coordinate reference system to use, depends on the regional extent of the area you want to work in, on the analysis you want to do and often on the availability of data."
> From qgis.org

## 6.2 Call Libraries

```
library(sf) # simple (spatial) features

library(ggplot2) # beautiful graphs
```

## 6.3 Open `wrld_simpl` Shapefile

```
world <- read_sf("./shapefiles/wrld_simpl/wrld_simpl.shp")
```

```r
head(world) # show the top (head) of the data
```

```
Simple feature collection with 6 features and 10 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -61.88722 ymin: -18.01639 xmax: 50.37499 ymax: 42.61805
Geodetic CRS:  +proj=longlat +datum=WGS84 +no_defs
# A tibble: 6 x 11
  FIPS  ISO2  ISO3     UN NAME                AREA REGION SUBREGION    LON   LAT
  <chr> <chr> <chr> <int> <chr>              <int>  <int>     <int>  <dbl> <dbl>
1 AC    AG    ATG      28 Antigua and Barb~     44     19        29 -61.8   17.1
2 AG    DZ    DZA      12 Algeria           238174      2        15   2.63  28.2
3 AJ    AZ    AZE      31 Azerbaijan          8260    142       145  47.4   40.4
4 AL    AL    ALB       8 Albania             2740    150        39  20.1   41.1
5 AM    AM    ARM      51 Armenia             2820    142       145  44.6   40.5
6 AO    AO    AGO      24 Angola            124670      2        17  17.5  -12.3
# i 1 more variable: geometry <MULTIPOLYGON [°]>
```

## 6.4 Find Out the CRS of `wrld_simpl`

As with many global data sets (and many other data sets), `wrld_simpl` uses *World Geodetic System 1984*.

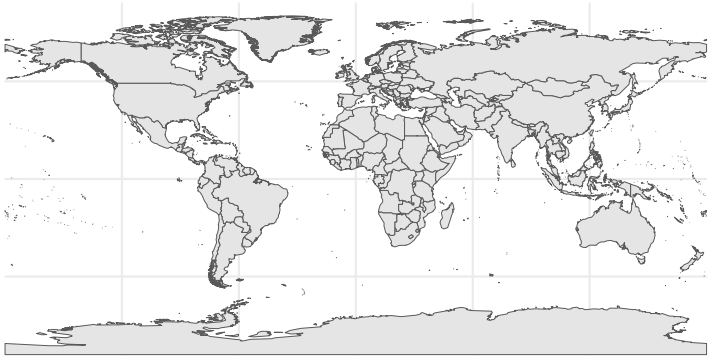
```r
st_crs(world)
```

```
Coordinate Reference System:
  User input: unknown
  wkt:
GEOGCRS["unknown",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1]],
        ID["EPSG",6326]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["Degree",0.0174532925199433]],
    CS[ellipsoidal,2],
        AXIS["longitude",east,
            ORDER[1],
```

```
        ANGLEUNIT["Degree",0.0174532925199433]],
    AXIS["latitude",north,
        ORDER[2],
        ANGLEUNIT["Degree",0.0174532925199433]]]
```

## 6.5 Plot The `wrld_simpl` Data

```
ggplot(world) +
  geom_sf() +
  theme_minimal()
```



# Part III

# GIS Data

# 7 Simple Features (`sf`)

R's new preferred way of handling spatial data seems to be *simple features* (`sf`).

Use `library(sf)` to read, write and manipulate *simple features*.

Other forms of data such as *shapefiles* (Chapter 8) often need to be read as simple features.

Mapping routines such as `ggplot` (Chapter 17) will also often want data to be in the form of *simple features*.

# 8 Shapefiles

## 8.1 Introduction

Shapefiles are a spatial data format originally developed by ESRI.

Shapefiles come in three major types:

- **points** , to represent point features such as individual or agency locations. At larger scales, cities or towns might also be points.
- **lines** , to represent line features such as roads, trails, or rivers.
- **polygons** , to represent polygon features such as outlines of cities, states, or countries.

## 8.2 Shape*files* Are Actually *Collections* of Files

Shape*files* are actually a *set* or *collection* of associated files, all with the same name, and all in the same directory, but different suffixes.

R–and many other software programs–generally reference the `*.shp` file of the shapefile.

```
list.files("./shapefiles/a2trees")
```

```
[1] "AA_Trees.cpg"     "AA_Trees.dbf"     "AA_Trees.prj"     "AA_Trees.sbn"
[5] "AA_Trees.sbx"     "AA_Trees.shp"     "AA_Trees.shp.xml" "AA_Trees.shx"
```

## 8.3 Call Libraries

```
library(ggplot2) # beautiful graphs

library(sf) # simple (spatial) features
```

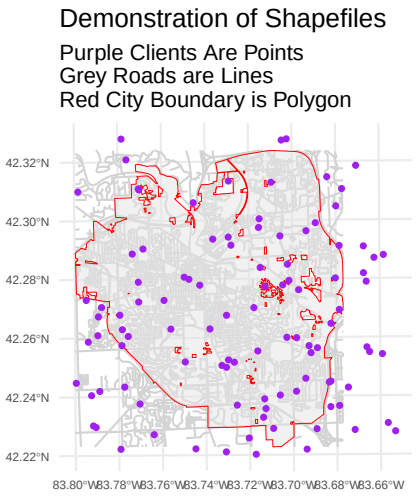
## 8.4 Open Shapefiles

```
city_boundary <- read_sf("./shapefiles/AA_City_Boundary/AA_City_Boundary.shp")

buildings <- read_sf("./shapefiles/AA_Building_Footprints/AA_Building_Footprints.shp")

# trees <- read_sf("./shapefiles/a2trees/AA_Trees.shp")

# parks <- read_sf("./shapefiles/AA_Parks/AA_Parks.shp")

# university <- read_sf("./shapefiles/AA_University/AA_University.shp")

clients <- read_sf("./shapefiles/clients/clients.shp")

WashtenawRoads <- read_sf("./shapefiles/Roads/RoadCenterlines.shp")

AnnArborRoads <- st_crop(WashtenawRoads,
                         city_boundary) # crop to only get A2 roads

# watersheds <- read_sf("./shapefiles/watersheds/Watersheds.shp")
```

## 8.5 Use ggplot to Map the Shapefiles

```
ggplot(city_boundary) + # initial sf data
  # geom_sf(data = buildings,
  #         fill = "lightgrey") +
  geom_sf(data = AnnArborRoads, # first layer: Ann Arbor roads
          color = "lightgrey") +
```

```
  geom_sf(color = "red", # second layer: city boundary
          alpha = .5) +
  geom_sf(data = clients, # third layer: clients
          size = 1,
          color = "purple") +
  labs(title = "Demonstration of Shapefiles",
       subtitle = "Purple Clients Are Points \nGrey Roads are Lines \nRed City Boundary is Poly
  theme_minimal() +
  theme(axis.text = element_text(size = rel(.5)))
```



# 9 Shapefiles on This Site

For user convenience, these shapefiles are available on this site.


Show 10 entries

Search:

| | shapefile |
|---|---|
| 1 | a2010.zip |
| 2 | AA_Building_Footprints.zip |
| 3 | AA_City_Boundary.zip |
| 4 | AA_Parks.zip |
| 5 | AA_Univesity.zip |
| 6 | Roads.zip |
| 7 | STATES.ch_2016_us_state_500k.zip |
| 8 | waterbody.zip |
| 9 | wdl_simpl.zip |

Showing 1 to 9 of 9 entries

Previous 1 Next

# 10 Inspecting Geographic Data Files

Often, when we open a new geographic data file, we don't know what fields of data it contains.

It is useful to get an idea of what data our geographic data file already contains in order to know whether or not we may need to merge in other data (Chapter 18).

`names` and `head` are two useful commands for learning about new geographic data files.

As an example, I am going to use the country data from the `rnaturalearth` library (Chapter 14).

## 10.1 Data

I use `ne_countries()` to create the `mapdata` dataset as an `sf` object (Chapter 7).

```
library(rnaturalearth)

mapdata <- ne_countries(scale = "medium",
                        returnclass = "sf")
```

## 10.2 names

`names` gives us the names of all of the fields of the data.

```
names(mapdata)
```

```
  [1] "featurecla" "scalerank"  "labelrank"  "sovereignt" "sov_a3"
  [6] "adm0_dif"   "level"      "type"       "tlc"        "admin"
 [11] "adm0_a3"    "geou_dif"   "geounit"    "gu_a3"      "su_dif"
 [16] "subunit"    "su_a3"      "brk_diff"   "name"       "name_long"
 [21] "brk_a3"     "brk_name"   "brk_group"  "abbrev"     "postal"
 [26] "formal_en"  "formal_fr"  "name_ciawf" "note_adm0"  "note_brk"
 [31] "name_sort"  "name_alt"   "mapcolor7"  "mapcolor8"  "mapcolor9"
 [36] "mapcolor13" "pop_est"    "pop_rank"   "pop_year"   "gdp_md"
 [41] "gdp_year"   "economy"    "income_grp" "fips_10"    "iso_a2"
 [46] "iso_a2_eh"  "iso_a3"     "iso_a3_eh"  "iso_n3"     "iso_n3_eh"
 [51] "un_a3"      "wb_a2"      "wb_a3"      "woe_id"     "woe_id_eh"
 [56] "woe_note"   "adm0_iso"   "adm0_diff"  "adm0_tlc"   "adm0_a3_us"
 [61] "adm0_a3_fr" "adm0_a3_ru" "adm0_a3_es" "adm0_a3_cn" "adm0_a3_tw"
 [66] "adm0_a3_in" "adm0_a3_np" "adm0_a3_pk" "adm0_a3_de" "adm0_a3_gb"
 [71] "adm0_a3_br" "adm0_a3_il" "adm0_a3_ps" "adm0_a3_sa" "adm0_a3_eg"
 [76] "adm0_a3_ma" "adm0_a3_pt" "adm0_a3_ar" "adm0_a3_jp" "adm0_a3_ko"
 [81] "adm0_a3_vn" "adm0_a3_tr" "adm0_a3_id" "adm0_a3_pl" "adm0_a3_gr"
 [86] "adm0_a3_it" "adm0_a3_nl" "adm0_a3_se" "adm0_a3_bd" "adm0_a3_ua"
 [91] "adm0_a3_un" "adm0_a3_wb" "continent"  "region_un"  "subregion"
 [96] "region_wb"  "name_len"   "long_len"   "abbrev_len" "tiny"
[101] "homepart"   "min_zoom"   "min_label"  "max_label"  "label_x"
[106] "label_y"    "ne_id"      "wikidataid" "name_ar"    "name_bn"
[111] "name_de"    "name_en"    "name_es"    "name_fa"    "name_fr"
[116] "name_el"    "name_he"    "name_hi"    "name_hu"    "name_id"
[121] "name_it"    "name_ja"    "name_ko"    "name_nl"    "name_pl"
[126] "name_pt"    "name_ru"    "name_sv"    "name_tr"    "name_uk"
[131] "name_ur"    "name_vi"    "name_zh"    "name_zht"   "fclass_iso"
[136] "tlc_diff"   "fclass_tlc" "fclass_us"  "fclass_fr"  "fclass_ru"
[141] "fclass_es"  "fclass_cn"  "fclass_tw"  "fclass_in"  "fclass_np"
[146] "fclass_pk"  "fclass_de"  "fclass_gb"  "fclass_br"  "fclass_il"
[151] "fclass_ps"  "fclass_sa"  "fclass_eg"  "fclass_ma"  "fclass_pt"
[156] "fclass_ar"  "fclass_jp"  "fclass_ko"  "fclass_vn"  "fclass_tr"
[161] "fclass_id"  "fclass_pl"  "fclass_gr"  "fclass_it"  "fclass_nl"
[166] "fclass_se"  "fclass_bd"  "fclass_ua"  "geometry"
```

## 10.3 head

`head` shows us the first several rows of data.

```
head(mapdata)
```

```
Simple feature collection with 6 features and 168 fields
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: -73.36621 ymin: -22.40205 xmax: 109.4449 ymax: 41.9062
Geodetic CRS:  WGS 84
       featurecla scalerank labelrank sovereignt sov_a3 adm0_dif level
1 Admin-0 country         1         3   Zimbabwe    ZWE        0     2
2 Admin-0 country         1         3     Zambia    ZMB        0     2
3 Admin-0 country         1         3      Yemen    YEM        0     2
4 Admin-0 country         3         2    Vietnam    VNM        0     2
5 Admin-0 country         5         3  Venezuela    VEN        0     2
6 Admin-0 country         6         6    Vatican    VAT        0     2
              type tlc     admin adm0_a3 geou_dif   geounit gu_a3 su_dif
1 Sovereign country   1  Zimbabwe     ZWE        0  Zimbabwe   ZWE      0
2 Sovereign country   1    Zambia     ZMB        0    Zambia   ZMB      0
3 Sovereign country   1     Yemen     YEM        0     Yemen   YEM      0
4 Sovereign country   1   Vietnam     VNM        0   Vietnam   VNM      0
5 Sovereign country   1 Venezuela     VEN        0 Venezuela   VEN      0
6 Sovereign country   1   Vatican     VAT        0   Vatican   VAT      0
    subunit su_a3 brk_diff      name name_long brk_a3  brk_name brk_group
1  Zimbabwe   ZWE        0  Zimbabwe  Zimbabwe    ZWE  Zimbabwe      <NA>
2    Zambia   ZMB        0    Zambia    Zambia    ZMB    Zambia      <NA>
3     Yemen   YEM        0     Yemen     Yemen    YEM     Yemen      <NA>
4   Vietnam   VNM        0   Vietnam   Vietnam    VNM   Vietnam      <NA>
5 Venezuela   VEN        0 Venezuela Venezuela    VEN Venezuela      <NA>
6   Vatican   VAT        0   Vatican   Vatican    VAT   Vatican      <NA>
  abbrev postal                       formal_en
1  Zimb.     ZW             Republic of Zimbabwe
2 Zambia     ZM               Republic of Zambia
3   Yem.     YE                Republic of Yemen
4  Viet.     VN    Socialist Republic of Vietnam
5   Ven.     VE Bolivarian Republic of Venezuela
6   Vat.      V        State of the Vatican City
                           formal_fr              name_ciawf note_adm0 note_brk
1                               <NA>               Zimbabwe      <NA>     <NA>
2                               <NA>                 Zambia      <NA>     <NA>
3                               <NA>                  Yemen      <NA>     <NA>
4                               <NA>                Vietnam      <NA>     <NA>
```

```
5 República Bolivariana de Venezuela               Venezuela      <NA>     <NA>
6                              <NA> Holy See (Vatican City)      <NA>     <NA>
          name_sort name_alt mapcolor7 mapcolor8 mapcolor9 mapcolor13  pop_est
1          Zimbabwe     <NA>         1         5         3          9 14645468
2            Zambia     <NA>         5         8         5         13 17861030
3       Yemen, Rep.     <NA>         5         3         3         11 29161922
4           Vietnam     <NA>         5         6         5          4 96462106
5     Venezuela, RB     <NA>         1         3         1          4 28515829
6 Vatican (Holy See) Holy See         1         3         4          2      825
  pop_rank pop_year gdp_md gdp_year                   economy
1       14     2019  21440     2019    5. Emerging region: G20
2       14     2019  23309     2019  7. Least developed region
3       15     2019  22581     2019  7. Least developed region
4       16     2019 261921     2019    5. Emerging region: G20
5       15     2019 482359     2014    5. Emerging region: G20
6        2     2019    -99     2019 2. Developed region: nonG7
              income_grp fips_10 iso_a2 iso_a2_eh iso_a3 iso_a3_eh iso_n3
1          5. Low income      ZI     ZW        ZW    ZWE       ZWE    716
2  4. Lower middle income      ZA     ZM        ZM    ZMB       ZMB    894
3  4. Lower middle income      YM     YE        YE    YEM       YEM    887
4  4. Lower middle income      VM     VN        VN    VNM       VNM    704
5  3. Upper middle income      VE     VE        VE    VEN       VEN    862
6 2. High income: nonOECD      VT     VA        VA    VAT       VAT    336
  iso_n3_eh un_a3 wb_a2 wb_a3   woe_id woe_id_eh                  woe_note
1       716   716    ZW   ZWE 23425004  23425004 Exact WOE match as country
2       894   894    ZM   ZMB 23425003  23425003 Exact WOE match as country
3       887   887    RY   YEM 23425002  23425002 Exact WOE match as country
4       704   704    VN   VNM 23424984  23424984 Exact WOE match as country
5       862   862    VE   VEN 23424982  23424982 Exact WOE match as country
6       336   336   -99   -99 23424986  23424986 Exact WOE match as country
  adm0_iso adm0_diff adm0_tlc adm0_a3_us adm0_a3_fr adm0_a3_ru adm0_a3_es
1      ZWE      <NA>      ZWE        ZWE        ZWE        ZWE        ZWE
2      ZMB      <NA>      ZMB        ZMB        ZMB        ZMB        ZMB
3      YEM      <NA>      YEM        YEM        YEM        YEM        YEM
4      VNM      <NA>      VNM        VNM        VNM        VNM        VNM
5      VEN      <NA>      VEN        VEN        VEN        VEN        VEN
6      VAT      <NA>      VAT        VAT        VAT        VAT        VAT
  adm0_a3_cn adm0_a3_tw adm0_a3_in adm0_a3_np adm0_a3_pk adm0_a3_de adm0_a3_gb
1        ZWE        ZWE        ZWE        ZWE        ZWE        ZWE        ZWE
2        ZMB        ZMB        ZMB        ZMB        ZMB        ZMB        ZMB
3        YEM        YEM        YEM        YEM        YEM        YEM        YEM
```

```
4        VNM        VNM        VNM        VNM        VNM        VNM        VNM
5        VEN        VEN        VEN        VEN        VEN        VEN        VEN
6        VAT        VAT        VAT        VAT        VAT        VAT        VAT
  adm0_a3_br adm0_a3_il adm0_a3_ps adm0_a3_sa adm0_a3_eg adm0_a3_ma adm0_a3_pt
1        ZWE        ZWE        ZWE        ZWE        ZWE        ZWE        ZWE
2        ZMB        ZMB        ZMB        ZMB        ZMB        ZMB        ZMB
3        YEM        YEM        YEM        YEM        YEM        YEM        YEM
4        VNM        VNM        VNM        VNM        VNM        VNM        VNM
5        VEN        VEN        VEN        VEN        VEN        VEN        VEN
6        VAT        VAT        VAT        VAT        VAT        VAT        VAT
  adm0_a3_ar adm0_a3_jp adm0_a3_ko adm0_a3_vn adm0_a3_tr adm0_a3_id adm0_a3_pl
1        ZWE        ZWE        ZWE        ZWE        ZWE        ZWE        ZWE
2        ZMB        ZMB        ZMB        ZMB        ZMB        ZMB        ZMB
3        YEM        YEM        YEM        YEM        YEM        YEM        YEM
4        VNM        VNM        VNM        VNM        VNM        VNM        VNM
5        VEN        VEN        VEN        VEN        VEN        VEN        VEN
6        VAT        VAT        VAT        VAT        VAT        VAT        VAT
  adm0_a3_gr adm0_a3_it adm0_a3_nl adm0_a3_se adm0_a3_bd adm0_a3_ua adm0_a3_un
1        ZWE        ZWE        ZWE        ZWE        ZWE        ZWE        -99
2        ZMB        ZMB        ZMB        ZMB        ZMB        ZMB        -99
3        YEM        YEM        YEM        YEM        YEM        YEM        -99
4        VNM        VNM        VNM        VNM        VNM        VNM        -99
5        VEN        VEN        VEN        VEN        VEN        VEN        -99
6        VAT        VAT        VAT        VAT        VAT        VAT        -99
  adm0_a3_wb     continent region_un          subregion
1        -99        Africa    Africa     Eastern Africa
2        -99        Africa    Africa     Eastern Africa
3        -99          Asia      Asia       Western Asia
4        -99          Asia      Asia South-Eastern Asia
5        -99 South America  Americas      South America
6        -99        Europe    Europe    Southern Europe
                  region_wb name_len long_len abbrev_len tiny homepart
1         Sub-Saharan Africa        8        8          5  -99        1
2         Sub-Saharan Africa        6        6          6  -99        1
3 Middle East & North Africa        5        5          4  -99        1
4        East Asia & Pacific        7        7          5    2        1
5  Latin America & Caribbean        9        9          4  -99        1
6      Europe & Central Asia        7        7          4    4        1
  min_zoom min_label max_label   label_x    label_y      ne_id wikidataid
1        0       2.5       8.0  29.92544 -18.911640 1159321441       Q954
2        0       3.0       8.0  26.39530 -14.660804 1159321439       Q953
```

```
3        0       3.0       8.0  45.87438  15.328226 1159321425       Q805
4        0       2.0       7.0 105.38729  21.715416 1159321417       Q881
5        0       2.5       7.5 -64.59938   7.182476 1159321411       Q717
6        0       5.0      10.0  12.45342  41.903323 1159321407       Q237
    name_ar       name_bn      name_de      name_en             name_es
1                      Simbabwe     Zimbabwe            Zimbabue
2                         Sambia       Zambia              Zambia
3                           Jemen        Yemen               Yemen
4                        Vietnam      Vietnam             Vietnam
5                      Venezuela    Venezuela           Venezuela
6                 Vatikanstadt Vatican City Ciudad del Vaticano
   name_fa          name_fr    name_el       name_he   name_hi   name_hu
1              Zimbabwe Z                         Zimbabwe
2                 Zambie     Z                        Zambia
3                   Yémen     Т                             Jemen
4               Viêt Nam    B                       Vietnám
5              Venezuela B                        Venezuela
6       Cité du Vatican   B                       Vatikán
    name_id            name_it    name_ja     name_ko      name_nl   name_pl
1  Zimbabwe           Zimbabwe               Zimbabwe  Zimbabwe
2    Zambia             Zambia                    Zambia    Zambia
3     Yaman              Yemen                       Jemen     Jemen
4   Vietnam            Vietnam                   Vietnam   Wietnam
5 Venezuela          Venezuela             Venezuela Wenezuela
6   Vatikan Città del Vaticano           Vaticaanstad   Watykan
    name_pt   name_ru       name_sv   name_tr   name_uk    name_ur
1  Zimbábue            Zimbabwe  Zimbabve
2    Zâmbia               Zambia   Zambiya
3     Iémen                Jemen     Yemen
4  Vietname              Vietnam   Vietnam    '
5 Venezuela           Venezuela Venezuela
6  Vaticano        Vatikanstaten   Vatikan
       name_vi  name_zh name_zht      fclass_iso tlc_diff      fclass_tlc
1      Zimbabwe         Admin-0 country     <NA> Admin-0 country
2        Zambia           Admin-0 country     <NA> Admin-0 country
3         Yemen             Admin-0 country     <NA> Admin-0 country
4      Việt Nam             Admin-0 country     <NA> Admin-0 country
5     Venezuela        Admin-0 country     <NA> Admin-0 country
6 Thành Vatican           Admin-0 country     <NA> Admin-0 country
  fclass_us fclass_fr fclass_ru fclass_es fclass_cn fclass_tw fclass_in
1      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
```

```
2      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
3      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
4      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
5      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
6      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
  fclass_np fclass_pk fclass_de fclass_gb fclass_br fclass_il fclass_ps
1      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
2      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
3      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
4      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
5      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
6      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
  fclass_sa fclass_eg fclass_ma fclass_pt fclass_ar fclass_jp fclass_ko
1      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
2      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
3      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
4      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
5      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
6      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
  fclass_vn fclass_tr fclass_id fclass_pl fclass_gr fclass_it fclass_nl
1      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
2      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
3      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
4      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
5      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
6      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>      <NA>
  fclass_se fclass_bd fclass_ua                      geometry
1      <NA>      <NA>      <NA> MULTIPOLYGON (((31.28789 -2...
2      <NA>      <NA>      <NA> MULTIPOLYGON (((30.39609 -1...
3      <NA>      <NA>      <NA> MULTIPOLYGON (((53.08564 16...
4      <NA>      <NA>      <NA> MULTIPOLYGON (((104.064 10....
5      <NA>      <NA>      <NA> MULTIPOLYGON (((-60.82119 9...
6      <NA>      <NA>      <NA> MULTIPOLYGON (((12.43916 41...
```

# 11 Symbology

## 11.1 Introduction

Shapefiles (Chapter 8) are a standard format for storing geographic data.

Shapefiles generally come in three types: points ; lines ; and polygons .

# 12 Symbology

*Symbology* is the idea of using shapefile attributes to encode *quantitative (continuous)* or *qualitative (discrete; categorical)* information, such as income or program participation.

## 12.1 Symbology

Table 12.1: Symbology

| Shapefile Type | Symbology |
|---|---|
| Points | Size:<br>Discrete Color:<br>Continuous Color: |
| Lines | Width:<br>Discrete Color:<br>Continuous Color:<br>Pattern: |
| Polygons | Discrete Color:<br>Continuous Color:<br>Pattern: |

## 12.2 A Note About the Use of Color

Color palettes can be either qualitative (discrete; categorical) or quantitative (continuous) in nature.

> **Note**
>
> The code used to generate these color palettes is shown for reference, but may not be the exact code needed when

these colors are used in maps.

### 12.2.1 Base R

```
barplot(rep(1,10), col = heat.colors(10), axes = FALSE)
```

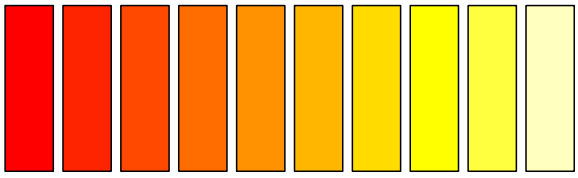


Figure 12.1: Heatmap Colors

```
barplot(rep(1,10), col = topo.colors(10), axes = FALSE)
```

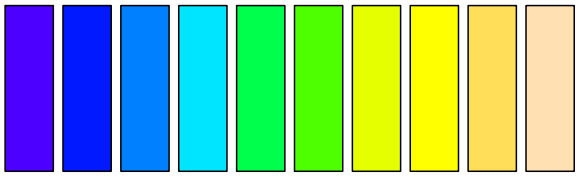


Figure 12.2: Topographical Colors

```r
barplot(rep(1,10), col = terrain.colors(10), axes = FALSE)
```

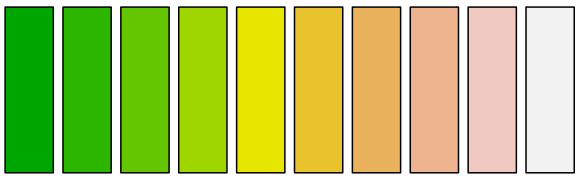


Figure 12.3: Terrain Colors

### 12.2.2 RColorBrewer

```r
library(RColorBrewer) # A library for color palettes
```

```r
display.brewer.pal(name = "Blues", n = 9)
```


Figure 12.4: Blues Palette

```
display.brewer.pal(name = "Spectral", n = 9)
```

Spectral (divergent)

Figure 12.5: Spectral Palette

```
display.brewer.pal(name = "Set1", n = 9)
```

Set1 (qualitative)

Figure 12.6: Set1 Palette

### 12.2.3 Viridis

More details can be found at https://cran.r-project.org/web/packages/viridis/vignettes/intro-to-viridis.html

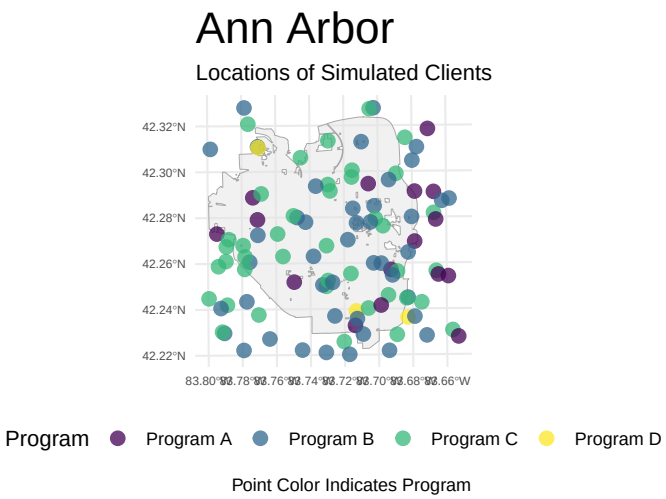
```
library(scales)
```

```
show_col(viridis_pal()(9),
         ncol = 9, # 9 columns
         labels = FALSE) # no labels
```

Figure 12.7: Viridis Palette

# 13 Demonstration Map

```
library(ggplot2) # graphing and mapping

library(sf) # simple features

city_boundary <- read_sf("./shapefiles/AA_City_Boundary/AA_City_Boundary.shp")

clients <- read_sf("./shapefiles/clients/clients.shp")

ggplot(city_boundary) +
  geom_sf(color = "darkgrey", alpha = .5) +
  geom_sf(data = clients,
          aes(color = program), # color = program
          size = 3, # size
          alpha = .75) + # transparency
  labs(title = "Ann Arbor",
       subtitle = "Locations of Simulated Clients",
       caption = "Point Color Indicates Program") +
  scale_color_viridis_d(name="Program") + # nice viridis colors
  theme_minimal() +
  theme(plot.title = element_text(size = rel(2)),
        axis.text = element_text(size = rel(.5)),
        legend.position = "bottom")
```


Ann Arbor
Locations of Simulated Clients
42.32°N
42.30°N
42.28°N
42.26°N
42.24°N
42.22°N
Program
Program A
Program B
Program C
Program D
Point Color Indicates Program

# 14 Using Data From `rnaturalearth`

`rnaturalearth` is a source library for data various types of global mapping data.

## 14.1 Call Libraries

```
library(rnaturalearth) # natural earth data

library(ggplot2) # beautiful maps

library(dplyr) # data wrangling

library(sf) # simple (spatial) features
```

## 14.2 Get `mapdata` As `sf`

`ne_countries` stands for *Natural Earth Countries*.

I use `ne_countries()` to create the `mapdata` dataset as an `sf` object (Chapter 7).

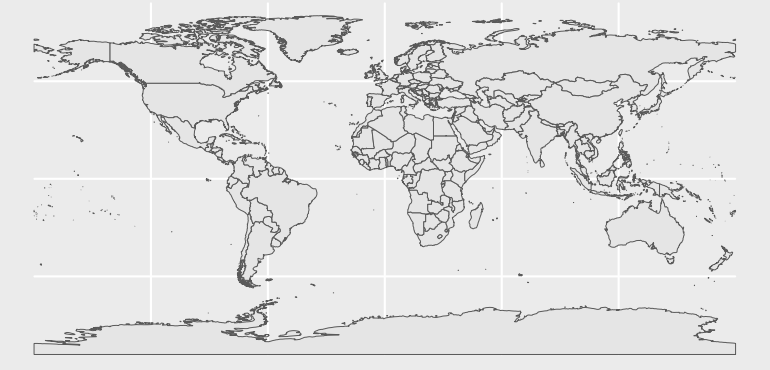
```
mapdata <- ne_countries(scale = "medium", # medium scale
                        returnclass = "sf") # as sf object
```

## 14.3 Map

### 14.3.1 Simple Basic Map

I make a simple map of this `sf` object with `ggplot` (Chapter 17).

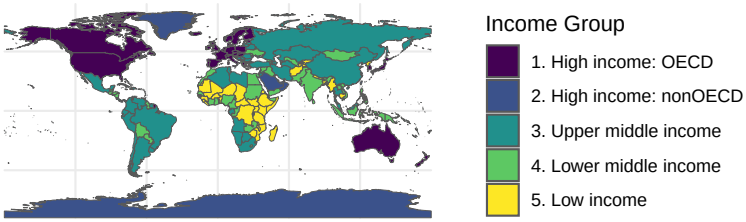
```
ggplot(mapdata) + # the data I am mapping
  geom_sf() # the geometry I am using
```



### 14.3.2 More Complicated Map

```
ggplot(mapdata) + # the sf data that I am mapping
  geom_sf(aes(fill = income_grp)) + # what goes on the map: FILL
  scale_fill_viridis_d(name = "Income Group", # beautiful colors
                       option = "viridis") +
  labs(title = "Countries of the World") + # labels
  theme_minimal() # minimal theme
```

Countries of the World



# 15 Using Data From WDI

## 15.1 Background

The World Bank collects statistical information from countries around the world. A particularly useful data set is the **W**orld **D**evelopment **I**ndicators (WDI) which are country level statistical information from around the world.

Using `library(WDI)` you can download indicator data directly from the World Bank and read it into a data set.

## 15.2 Call Libraries

```
library(WDI) # for accessing World Bank data

library(dplyr) # data wrangling
```

## 15.3 Get Some Data From the World Development Indicators (WDI)

```
# get names of specific indicators from WDI Data Catalog

WorldBankData <- WDI(country="all",
                     indicator=c("SI.POV.GINI", # Gini
                                 "NY.GDP.PCAP.CD", # GDP
                                 "SE.ADT.LITR.ZS", # adult literacy
                                 "SP.DYN.LE00.IN", # life expectancy
                                 "SP.POP.TOTL", # population
```

```
                                  "SN.ITK.DEFC.ZS"), # undernourishment
                      start = 2023,
                      end = 2023,
                      extra = TRUE)


save(WorldBankData, file="WorldBankData.RData")
```

# 16 Rename Some Variables

```
# think about renaming some variables with more intuitive names
# e.g....

# rename some variables with dplyr (just copy and paste your indicators)

WorldBankData <- dplyr::rename(WorldBankData,
                        GDP = NY.GDP.PCAP.CD,
                        adult_literacy = SE.ADT.LITR.ZS,
                        life_expectancy = SP.DYN.LE00.IN,
                        population = SP.POP.TOTL,
                        Gini = SI.POV.GINI,
                        undernourishment = SN.ITK.DEFC.ZS)

save(WorldBankData, file="WorldBankData.RData")
```

## 16.1 Look At The Data

```
load("WorldBankData.RData") # load the data

head(WorldBankData)
```

```
                     country iso2c iso3c year status lastupdated Gini      GDP
1                 Afghanistan    AF   AFG 2023         2024-10-24   NA       NA
2 Africa Eastern and Southern    ZH   AFE 2023         2024-10-24   NA 1672.506
3  Africa Western and Central    ZI   AFW 2023         2024-10-24   NA 1584.333
4                      Albania    AL   ALB 2023         2024-10-24   NA 8367.776
5                      Algeria    DZ   DZA 2023         2024-10-24   NA 5260.206
6               American Samoa    AS   ASM 2023         2024-10-24   NA       NA
  adult_literacy life_expectancy population undernourishment
```

```
1             NA              NA   42239854               NA
2       73.27511              NA  739108306               NA
3       60.50555              NA  502789511               NA
4             NA              NA    2745972               NA
5             NA              NA   45606480               NA
6             NA              NA      43914               NA
                     region   capital longitude latitude             income
1                South Asia     Kabul   69.1761  34.5228         Low income
2                Aggregates                                     Aggregates
3                Aggregates                                     Aggregates
4      Europe & Central Asia    Tirane   19.8172  41.3317 Upper middle income
5 Middle East & North Africa   Algiers   3.05097  36.7397 Lower middle income
6        East Asia & Pacific Pago Pago  -170.691 -14.2846 Upper middle income
         lending
1            IDA
2     Aggregates
3     Aggregates
4           IBRD
5           IBRD
6 Not classified
```

# Part IV

# Mapping With `ggplot`

# 17 Making Maps with ggplot

## 17.1 Call the libraries

```
library(ggplot2) # beautiful graphs

library(dplyr) # data wrangling

library(sf) # simple (spatial) features

library(readr) # import csv
```

## 17.2 Use read_sf To Open Shapefiles

Getting the directory and filename right is important.

```
city_boundary <- read_sf("./shapefiles/AA_City_Boundary/AA_City_Boundary.shp")

buildings <- read_sf("./shapefiles/AA_Building_Footprints/AA_Building_Footprints.shp")

trees <- read_sf("./shapefiles/a2trees/AA_Trees.shp")

parks <- read_sf("./shapefiles/AA_Parks/AA_Parks.shp")

university <- read_sf("./shapefiles/AA_University/AA_University.shp")

WashtenawRoads <- read_sf("./shapefiles/Roads/RoadCenterlines.shp")

AnnArborRoads <- st_crop(WashtenawRoads,
                         city_boundary) # crop to only get A2 roads
```

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

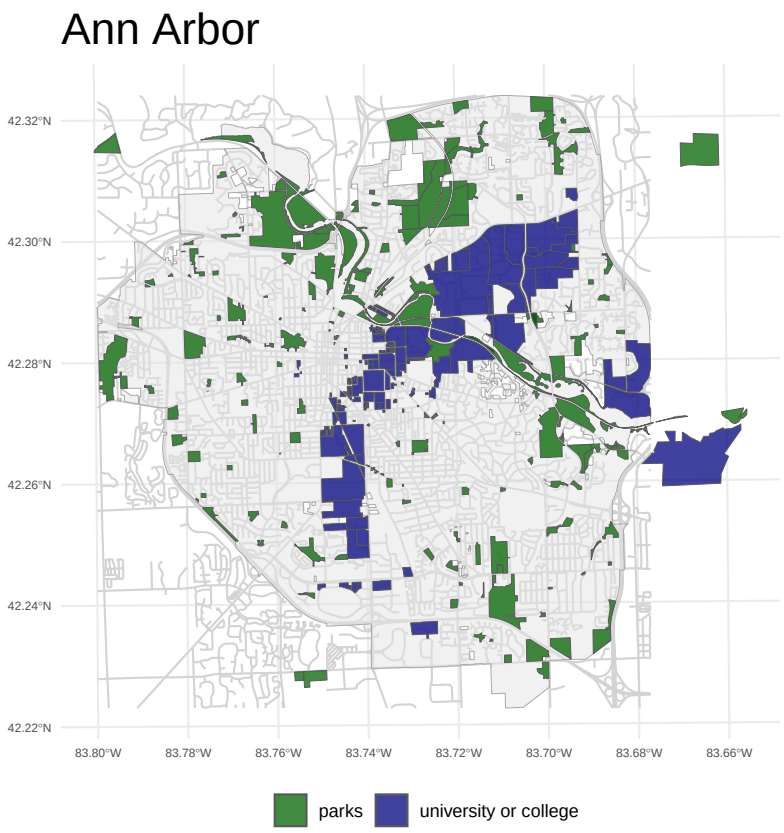
```
# watersheds <- read_sf("../shapefiles/watersheds/Watersheds.shp")
```

## 17.3 Use ggplot to Make The Map

```
# NB RE Macs: the plotting device on Macs can be very slow
# we notice this with all the detail that is involved in maps
# maps can be REALLY slow on Macs
# so--inconveniently--we write directly to PDF on a Mac
# and don't see the graph in our RStudio window
# we have to manually open the PDF to see the created map

# Apparently, the first layer is important for setting the CRS of the map

# pdf("./mapping/ggplot-map-test.pdf") # open PDF device (uncomment on Mac)

# dev.off() # turn off PDF device (uncomment on Mac)
```

```
ggplot(city_boundary) +
  # geom_sf(data = buildings,
  #         fill = "lightgrey") +
  geom_sf(data = AnnArborRoads,
          color = "lightgrey") +
  geom_sf(color = "darkgrey", alpha = .5) +
  geom_sf(data = university,
          aes(fill = "university or college"),
          alpha = .75) +
  geom_sf(data = parks,
          alpha = .75,
          aes(fill = "parks")) +
  # geom_sf(data = trees,
  #         size = .1,
  #         color = "darkgreen") +
  labs(title = "Ann Arbor") +
  scale_color_viridis_d() +
  scale_fill_manual(name="",
```

```
                    values = c("darkgreen", "navy")) +
  theme_minimal() +
  theme(plot.title = element_text(size = rel(2)),
        axis.text = element_text(size = rel(.5)),
        legend.position = "bottom")
```



# 18 Merge Shapefiles With External Data

## 18.1 Introduction

A common task in mapping is that we have a *shapefile* (Chapter 8) or `sf` object (Chapter 7) of map data, but we want to merge in some *external data* from another source so that we can map that *external data*.

Often we want to use different colors to map that external data (Chapter 11).

Here, I use an `sf` object (Chapter 7) of countries of the world (Chapter 14), and merge that data with data from the World Bank World Development Indicators (Chapter 15).

This tutorial builds upon another tutorial on Mapping with `ggplot`(Chapter 17)

## 18.2 Call Libraries

```
library(rnaturalearth) # natural earth data

library(sf) # simple (spatial) features

library(ggplot2) # beautiful plots

library(dplyr) # data wrangling and joins
```

## 18.3 Get Map Data on Countries of the World

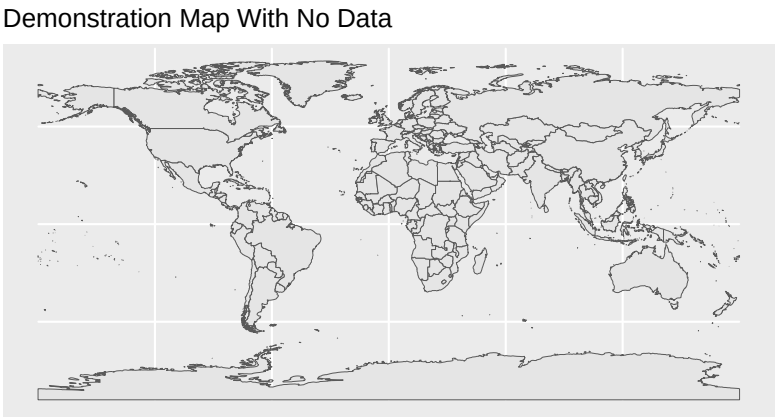
I am using the `rnaturalearth` package to get map data on countries of the world. I read this data into an object called `world`.

```
mapdata <- ne_countries(scale = "medium", # medium scale
                        returnclass = "sf") # as sf object
```

## 18.4 Make a Map Without Data

I map the data with `ggplot`, and the special `geom`, `geom_sf`.

```
ggplot(mapdata) +
  geom_sf() +
  labs(title = "Demonstration Map With No Data")
```




## 18.5 Get External Data

Here I load the World Bank Data (Chapter 15).

```
load("WorldBankData.Rdata")

head(WorldBankData) # replay data set
```

```
                    country iso2c iso3c year status lastupdated Gini      GDP
1               Afghanistan    AF   AFG 2023         2024-10-24   NA       NA
2 Africa Eastern and Southern    ZH   AFE 2023         2024-10-24   NA 1672.506
3  Africa Western and Central    ZI   AFW 2023         2024-10-24   NA 1584.333
4                   Albania    AL   ALB 2023         2024-10-24   NA 8367.776
5                   Algeria    DZ   DZA 2023         2024-10-24   NA 5260.206
6              American Samoa    AS   ASM 2023         2024-10-24   NA       NA
  adult_literacy life_expectancy population undernourishment
1             NA              NA   42239854               NA
2       73.27511              NA  739108306               NA
3       60.50555              NA  502789511               NA
4             NA              NA    2745972               NA
5             NA              NA   45606480               NA
6             NA              NA      43914               NA
                    region   capital longitude latitude              income
1                South Asia     Kabul   69.1761  34.5228          Low income
2                Aggregates                                    Aggregates
3                Aggregates                                    Aggregates
4      Europe & Central Asia    Tirane   19.8172  41.3317 Upper middle income
5 Middle East & North Africa   Algiers   3.05097  36.7397 Lower middle income
6        East Asia & Pacific Pago Pago  -170.691 -14.2846 Upper middle income
         lending
1            IDA
2     Aggregates
3     Aggregates
4           IBRD
5           IBRD
6 Not classified
```

## 18.6 Join Data to Shapefile

I use `left_join` from the `dplyr` package to merge the spatial data in `world` with `externaldata`.

`left_join` is a function that keeps all observations in the data on the left (the shapefile), and only those matching observations in the data on the right (the external data), which is usually what I want in mapping.

I need a unique identifier for my rows of data, so here I use `iso_a3`, a unique 3 letter identifier for countries of the world.

First I need to make a copy of a variable in `WorldBankData` with a new name so that the identifiers will match exactly.

```
WorldBankData$iso_a3 <- WorldBankData$iso3c
```

Then I merge the data using `left_join`.

```
newdata <- left_join(mapdata, # map data
                     WorldBankData, # table of indicators
                     by = "iso_a3") # join by
```

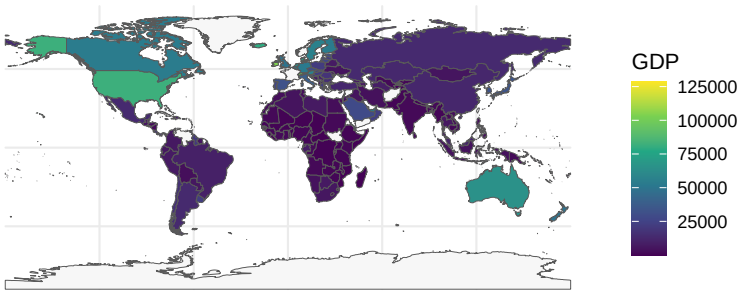
## 18.7 Make a Map With The Data

Once I have the merged data, it is easy to map it with `ggplot` and `geom_sf`. Note that I need to specify an `aes`thetic for `geom_sf`. Here `GDP` is the *fill* color for countries on the map.

> Data could also be mapped with another package like `leaflet` (Chapter 22).

```
ggplot(newdata) +
  geom_sf(aes(fill = GDP)) + # adding a fill aesthetic
  scale_fill_viridis_c(na.value = "grey97", # value for NA
                       option = "viridis") + # viridis colors
  labs(title = "Demonstration Map With Merged Data") +
  theme_minimal() # better theme
```

Demonstration Map With Merged Data



# 19 Making Maps With ggplot Using Location Data

## 19.1 Call Libraries

```
library(readr) # read CSV

library(dplyr) # data wrangling

library(sf) # simple features

library(ggplot2) # maps
```

## 19.2 Use read_csv to Read Text File with Client Data

```
clients <- read_csv("./location-data/clients.csv")
```

## 19.3 Only Clients in Ann Arbor Area

```
clients <- clients %>%
  filter(latitude <= 42.33 &
           latitude >= 42.22 &
           longitude >= -83.8 &
           longitude <= -83.65)
```

## 19.4 Convert Clients to `sf` Object While Indicating *Coordinate Reference System* (CRS)

```
point <- st_as_sf(clients,
                  coords = c("longitude", "latitude"),
                  crs = 4269) # A2 is NAD1983

# write to shapefile

st_write(point,
         "./shapefiles/clients/clients.shp",
         append = FALSE) # replace; don't append
```

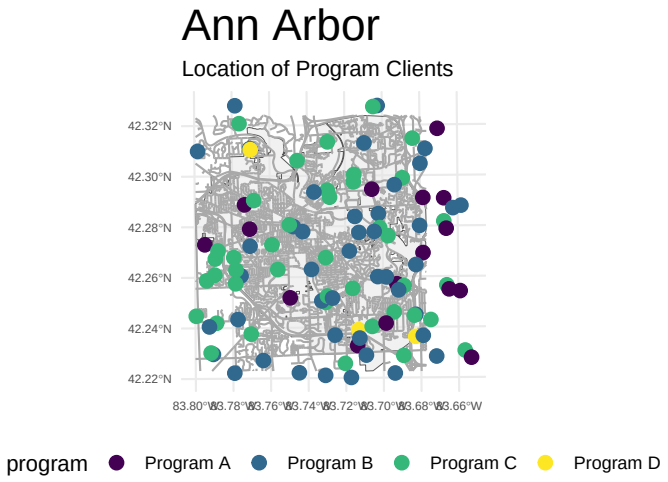
## 19.5 Read in Shapefile(s)

```
city_boundary <- read_sf("./shapefiles/AA_City_Boundary/AA_City_Boundary.shp")

WashtenawRoads <- read_sf("./shapefiles/Roads/RoadCenterlines.shp")

AnnArborRoads <- st_crop(WashtenawRoads,
                         city_boundary) # crop to only get A2 roads
```

## 19.6 Map

```
ggplot(city_boundary) +
  geom_sf(alpha = .5) +
  geom_sf(data = AnnArborRoads,
          color = "darkgrey") +
  geom_sf(data = point,
          aes(color = program),
          size = 3) +
labs(title = "Ann Arbor",
```

```
    subtitle = "Location of Program Clients") +
scale_color_viridis_d() +
scale_fill_viridis_d() +
theme_minimal() +
theme(plot.title = element_text(size = rel(2)),
      axis.text = element_text(size = rel(.5)),
      legend.position = "bottom")
```



# Part V

# More Advanced GIS Concepts

# 20 Geocoding

## 20.1 Call Libraries

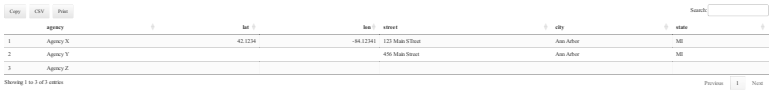
```
library(tidygeocoder) # geocoding
```

```
Warning: package 'tidygeocoder' was built under R version 4.4.2
```

```
library(dplyr) # for %>% operator

library(readr) # import CSV

library(DT) # nice tables
```

## 20.2 Get Data To Be Geocoded

```
simulated_address_data <- read_csv("simulated-address-data/simulated-address-data.csv")

DT::datatable(simulated_address_data,
              extensions = 'Buttons',
              options = list(
                dom = 'Bfrtip',
                buttons = c('copy',
                            'csv',
                            'print'))) # nice table
```




## 20.3 Concatenate Addresses

```
simulated_address_data$address <- paste(simulated_address_data$street,
                                        ", ",
                                        simulated_address_data$city,
                                        ", ",
                                        simulated_address_data$state)

DT::datatable(simulated_address_data,
              extensions = 'Buttons',
              options = list(
                dom = 'Bfrtip',
                buttons = c('copy',
                            'csv',
                            'print'))) # nice table
```

| | agency | lat | lon | street | city | state | address |
|---|---|---|---|---|---|---|---|
| 1 | Agency X | 42.1234 | -84.12341 | 123 Main STreet | Ann Arbor | MI | 123 Main STreet , Ann Arbor , MI |
| 2 | Agency Y | | | 456 Main Street | Ann Arbor | MI | 456 Main Street , Ann Arbor , MI |
| 3 | Agency Z | | | | | | NA , NA , NA |

## 20.4 Geocode

> **Caution**
>
> ArcGIS geocoding has LOW success rate with this data. You will want to find a process with HIGH success rate.

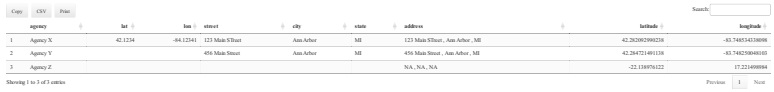
> **Tip**
>
> You could also try batchgeo -> KML -> Latitude/Longitude

```
geocoded_data <- simulated_address_data %>%
  tidygeocoder::geocode(address,
                        method = 'arcgis',
                        lat = latitude,
                        long = longitude)
```

```
Passing 3 addresses to the ArcGIS single address geocoder
```

```
Query completed in: 2.2 seconds
```

```
DT::datatable(geocoded_data,
              extensions = 'Buttons',
              options = list(
                dom = 'Bfrtip',
                buttons = c('copy',
                            'csv',
                            'print'))) # nice table
```



| | agency | lat | lon | street | city | state | address | latitude | longitude |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Agency X | 42.1234 | -84.12341 | 123 Main STreet | Ann Arbor | MI | 123 Main STreet , Ann Arbor , MI | 42.2820929992238 | -83.7485343338098 |
| 2 | Agency Y | | | 456 Main Street | Ann Arbor | MI | 456 Main Street , Ann Arbor , MI | 42.2847214911338 | -83.7482590048103 |
| 3 | Agency Z | | | | | | NA , NA , NA | -22.1380976122 | 17.2214698884 |

Copy CSV Print Search:

Showing 1 to 3 of 3 entries Previous 1 Next

**❗ Geocoding Can Make Errors!**

NB that for whatever reason–perhaps because there is no address information–the geocoder has made a mistake: Agency Z has been placed in the Southern Hemisphere. Geocoding can be an error prone process and requires careful inspection of your tabular and mapped data.
A geocoder may also be unable to geocode some of your data. Low success rates are not uncommon, and you may have to work hard to ensure that the majority, or all, of your data are geocoded.

Geocoded data can then be mapped using procedures outlined in Chapter 19.

# 21 cartogram

A *cartogram* is a map where the areas of different regions are distorted (increased in size; decreased in size) by the value of some quantitative variable.

## 21.1 Call Libraries

```
library(rnaturalearth) # natural earth data

library(ggplot2) # beautiful maps

library(dplyr) # data wrangling

library(sf) # simple (spatial) features

library(cartogram) # cartograms!
```

## 21.2 Remove Scientific Notation

```
options(scipen = 999) # high 'penalty' for scientific notation
```

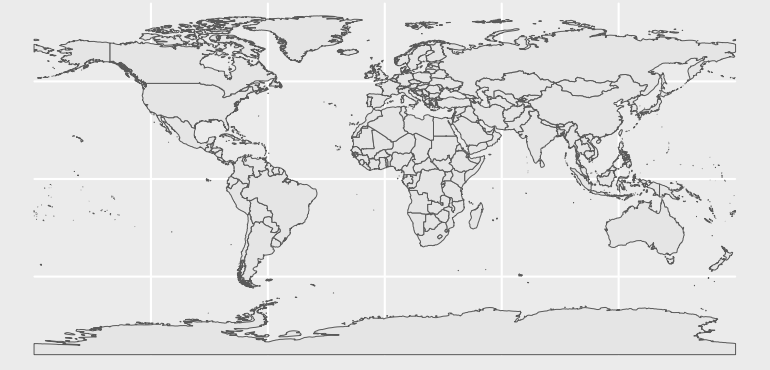
## 21.3 Get Map Data From rnaturalearth

```
mapdata <- ne_countries(scale = "medium", # medium scale
                        returnclass = "sf")  # as sf object
```

## 21.4 Make A Basic Map

We make a basic map, reading it into an object called `mymap`. We then *replay* `mymap`.

```
mymap <- ggplot(mapdata) + # the data I am mapping
  geom_sf() # the geometry I am using

mymap # replay my map
```



## 21.5 Make And Plot The Cartogram

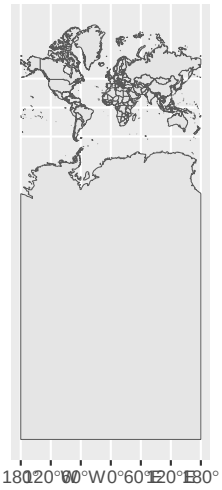
### 21.5.1 Project The Cartogram Data

> Tip
>
> `cartogram` requires *projected* data (Chapter 5), so we need to project the data with `st_transform`. A number of projections, including the *Mercator* and *Mollweide* projections are possibilities. You may need to experiment with a number of projections to see which ones work best in any particular cartogram.

```
mapdata_proj <- st_transform(mapdata,
                             3857) # Mercator

# mapdata_proj <- st_transform(mapdata,
#                              crs = "+proj=moll") # Mollweide
```

### 21.5.2 Plot The Projected Data

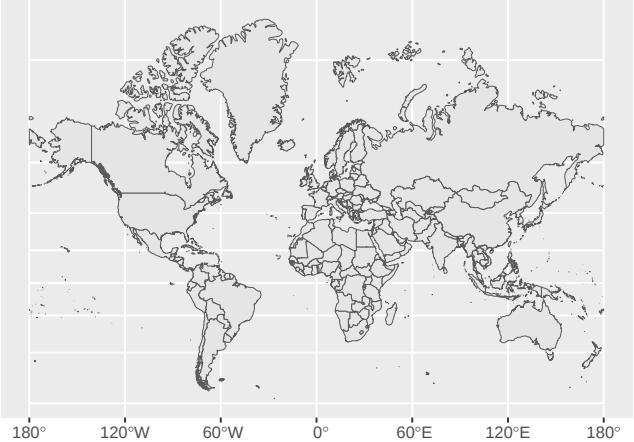
```
ggplot(mapdata_proj) +
  geom_sf() # plot projected data
```




Why Does Antarctica Look So Strange? How To Fix This?

In some projections, especially the *Mercator* projection, Antartica looks strange.
The key is to run this `dplyr` code to remove Antarctica.

```
mapdata_proj <- mapdata_proj %>%
  dplyr::filter(! continent == "Antarctica")

ggplot(mapdata_proj) +
  geom_sf() # plot projected data
```




### 21.5.3 Make The Cartogram Data

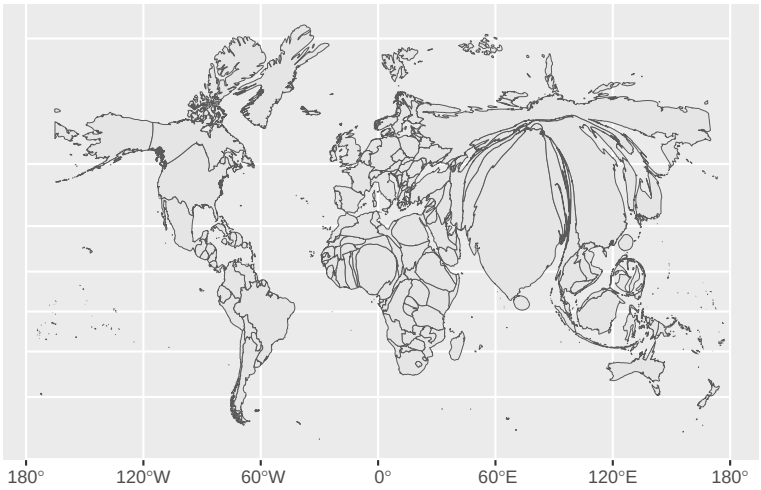
> **Tip**
>
> Each iteration takes a **LONG** time. Fewer iterations help the time, but each iteration contributes to the *distortion*, and makes a more `cartogram`-like *cartogram*. Because this is the most time intensive step, I time the creation of the cartogram with `Sys.time`.

```
start_time <- Sys.time() # time this step

mapdata_cartogram <- cartogram_cont(mapdata_proj,
                                    "pop_est",
                                    itermax = 7)

end_time <- Sys.time()

end_time - start_time
```

```
Time difference of 37.56932 secs
```

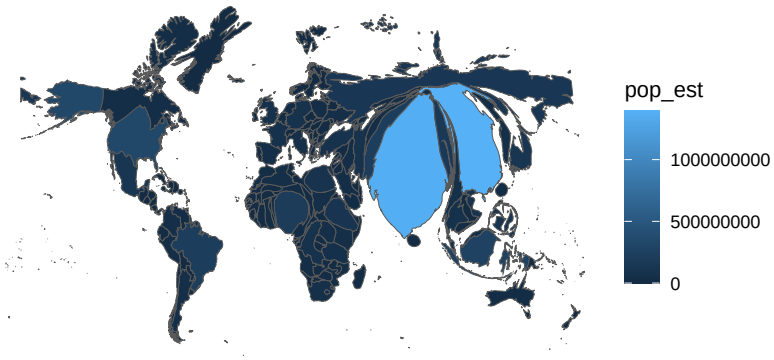
### 21.5.4 Basic Cartogram

```
ggplot(mapdata_cartogram) +
  geom_sf()
```



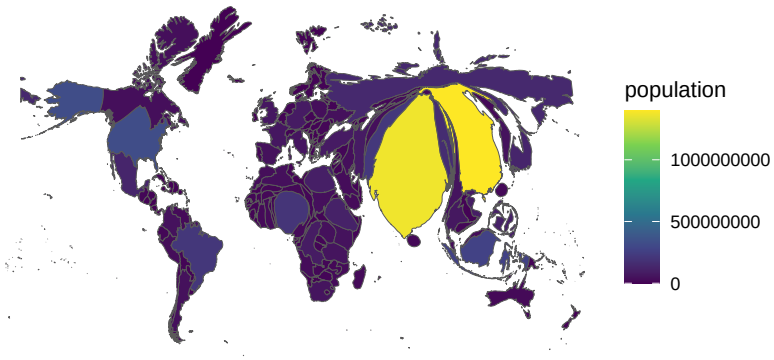

### 21.5.5 Basic Cartogram With `fill` Color

```
ggplot(mapdata_cartogram) +
  geom_sf(aes(fill = pop_est)) + # fill is population estimate
  theme_void()
```




### 21.5.6 Cartogram With Better (`viridis`) Colors

```
ggplot(mapdata_cartogram) +
  geom_sf(aes(fill = pop_est)) + # fill is population estimate
  scale_fill_viridis_c(name = "population",
                       option = "viridis") + # beautiful colors
  theme_void()
```



# Part VI

# Other R Libraries for Mapping

# 22 Mapping With `leaflet`

## 22.1 Call Libraries

```
library(leaflet) # web based maps
```

```
Warning: package 'leaflet' was built under R version 4.4.2
```

```
library(sf) # simple (spatial) features

library(readr) # import csv

library(dplyr) # data wrangling

library(here) # where am I?
```

```
Warning: package 'here' was built under R version 4.4.2
```

```
library(pander) # nice tables

# setwd(here()) # set the working directory
```

## 22.2 Get Simulated Client Data

```
clients <- read_csv("./location-data/clients.csv")

pander(head(clients)) # top of client data
```

Table 22.1: Table continues below

| ID | age | gender | race_ethnicity | family_income | program |
|---|---|---|---|---|---|
| 2892 | 23 | Male | African American | 42359 | Program B |
| 1971 | 39 | Female | Asian American | 66500 | Program C |
| 4728 | 26 | Female | Asian American | 52726 | Program C |
| 1020 | 24 | Male | Latinx | 52911 | Program D |
| 4429 | 36 | Female | Asian American | 50287 | Program C |
| 3136 | 33 | Male | African American | 45570 | Program C |

| mental_health_T1 | mental_health_T2 | latitude | longitude |
|---|---|---|---|
| 95.25 | 106.8 | 42.16 | -83.6 |
| 82.64 | 96.3 | 42.29 | -83.88 |
| 80.49 | 98.72 | 42.14 | -83.78 |
| 93.82 | 91.67 | 42.24 | -83.68 |
| 83.37 | 99.69 | 42.18 | -83.64 |
| 75.28 | 92.9 | 42.21 | -83.7 |

## 22.3 Only Clients In Ann Arbor Area

```
clients <- clients %>%
  filter(latitude <= 42.35 &
           latitude >= 42.2 &
           longitude >= -83.8 &
           longitude <= -83.65)
```

## 22.4 Read in Shapefiles

```
parks <- read_sf("./shapefiles/AA_Parks/AA_Parks.shp")

university <- read_sf("./shapefiles/AA_University/AA_University.shp")

city_boundary <- read_sf("./shapefiles/AA_City_Boundary/AA_City_Boundary.shp")
```

# 23 Transform CRS

> "Map projections try to portray the surface of the earth or a portion of the earth on a flat piece of paper or computer screen. A coordinate reference system (CRS) then defines, with the help of coordinates, how the two-dimensional, projected map in your GIS is related to real places on the earth. The decision as to which map projection and coordinate reference system to use, depends on the regional extent of the area you want to work in, on the analysis you want to do and often on the availability of data." From qgis.org
>
> see https://stackoverflow.com/questions/66471147/how-to-plot-sp-object-as-sf-in-r-leaflet

```
university <- st_transform(university, 4326) # transform CRS

parks <- st_transform(parks, 4326) # transform CRS

city_boundary <- st_transform(city_boundary, 4326) # transform CRS
```

## 23.1 Leaflet Map

### 23.1.1 Color Palette

```
pal <- colorFactor(c("red", "blue", "orange", "green"),
                   domain = levels(as.factor(clients$program)))
```

### 23.1.2 Map

```
leaflet(clients) %>%
  setView(lng = mean(clients$longitude),
          lat = mean(clients$latitude),
          zoom = 12) %>%
  # addTiles() %>% # Open StreetMap
  addProviderTiles(providers$CartoDB.Positron) %>%
  addCircleMarkers(~longitude,
             ~latitude,
             popup = ~paste("Client ID:", as.character(ID)),
             label = ~paste("Client ID:", as.character(ID)),
             color = ~pal(program),
             clusterOptions = markerClusterOptions()) %>%
  addLegend("bottomright",
            pal = pal,
            values = ~program,
            title = "Program") %>%
  # addPolygons(data = parks, color = "green") %>%
  # addPolygons(data = university, color = "blue") %>%
  addPolygons(data = city_boundary,
              color = "red",
              fillOpacity = 0.0)
```


+
−
MARION TOWNSHIP
Genoa Charter Township
Brighton
Hamburg Township
Pinckney
Portage Lake
Dexter Township
Webster Township
Dexter
Program
Program A
Program B
Program C
Program D
Leaflet | © OpenStreetMap contributors © CARTO

# 24 Mapping With `plotly`

## 24.1 Call Libraries

```
library(plotly)
```

```
Warning: package 'plotly' was built under R version 4.4.2
```

## 24.2 Set Geographic Parameters

```
g <- list(showland = TRUE,
          showcountries = TRUE,
          landcolor = toRGB("forestgreen"), # land color
          showocean = TRUE, # show ocean
          oceancolor = "lightblue", # ocean color
          # projection = list(type = 'robinson'),
          projection = list(type = 'orthographic',
                    rotation = list(lon = 0,
                                    lat = 0,
                                    roll = 0)))
```

## 24.3 Make a Map

```
plot_geo() %>%
  layout(title = "Demonstration Map",
         geo = g)
```


## 24.4 scattermapbox

```
plot_ly(type = "scattermapbox") %>%
  layout(
    mapbox = list(
      style = 'open-street-map',
      zoom = 6.0, # zoom
      center = list(lon = -83, lat = 42))) # centered on SE Michigan
```