Spatial Data in R

R for Stata Users

Luiza Andrade, Leonardo Viotti & Rob Marty

April 2019

Introduction

In this section we'll introduce handling spatial datasets in R. We'll cover how to make (1) maps using `ggplot()`, (2) maps with basemaps using `ggmap()` and (3) interactive maps with `leaflet()`. In addition, we'll introduce some operations that can be done on spatial objects, such as calculating distances. We'll rely on the `rgdal()`, `rgeos()` and `raster()` packages for loading and manipulating spatial data.

Note: An alternative (and newer) way of working with spatial data in R is to use the `sf()` package. We won't be using this today, but when googling around you may notice this package being used.

Introduction

Before we start, let's make sure we're all set:

1. Start a fresh session.
2. Make sure to load `rgdal`, `broom`, `ggmap`, `raster`, `leaflet`, `rworldmap`, `rgeos`, `tidyverse` and `ggrepel` packages. If they are not installed, install them.

Introduction

Here's a shortcut if you missed the last session:

```
# Install packages
install.packages(c("rgdal", "broom", "ggmap",
                   "raster", "leaflet", "rworldmap",
                   "rgeos", "tidyverse", "ggrepel"))

# Load packages
library(tidyverse)
library(rgdal)
library(broom)
library(ggmap)
library(raster)
library(leaflet)
library(rworldmap)
library(rgeos)
library(ggrepel)

#  Set folder paths
projectFolder <- "YOUR/FOLDER/PATH"
finalData     <- file.path(projectFolder,
                           "DataWork", "DataSets", "Final")
```

Outline

Load Spatial Object

We'll load the map data using the the readOGR function from the rgdal package. The argument for dsn is the folder where the shapefile is and layer is the name of the shapefile.

`readOGR()`

- **dsn**: data source name (interpretation varies by driver - for some drivers, dsn is a file name, but may also be a folder)
- **layer**: layer name (varies by driver, may be a file name without extension)

Load Spatial Object

```
library(rgdal)
worldmap <-  readOGR(dsn = finalData, layer = "worldmap")
```

Potentital issues

1. Can't find file? Try

```
worldmap <- readOGR(file.path(finalData, "worldmap.shp"))
```

If it doesn't work, try setting the working directory to the folder where the shapefile is (usually works better with Macs)

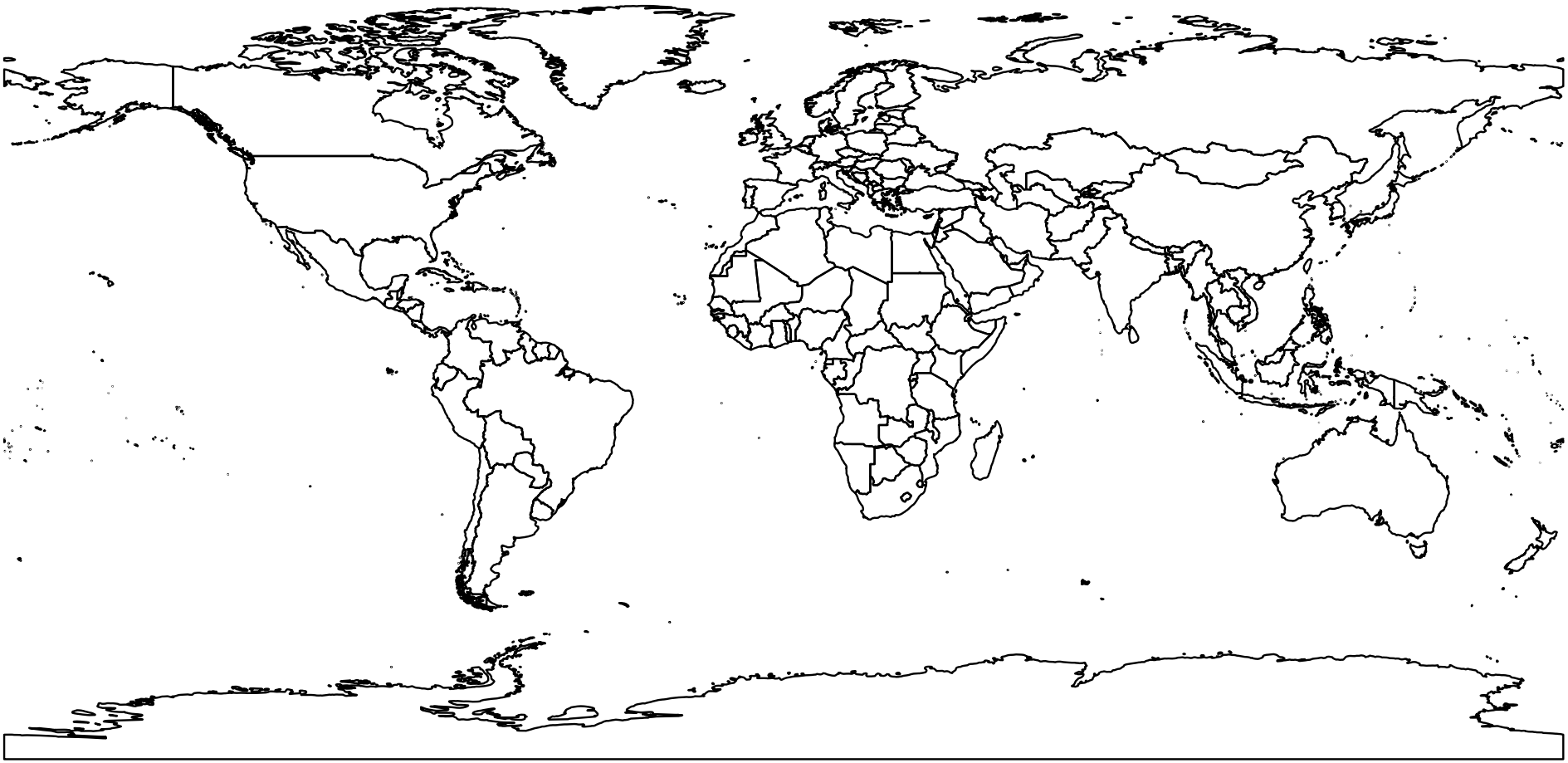
```
setwd(finalData)
worldmap <- readOGR(dsn = ".", layer = "worldmap")
```

2. Can't find gdal? Use the following code

```
library(maptools)
worldmap <- readShapeSpatial(file.path(finalData,"worldmap.shp"))
prj <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
crs(worldmap) <- CRS(prj)
```

Plot Spatial Object

```
plot(worldmap)
```



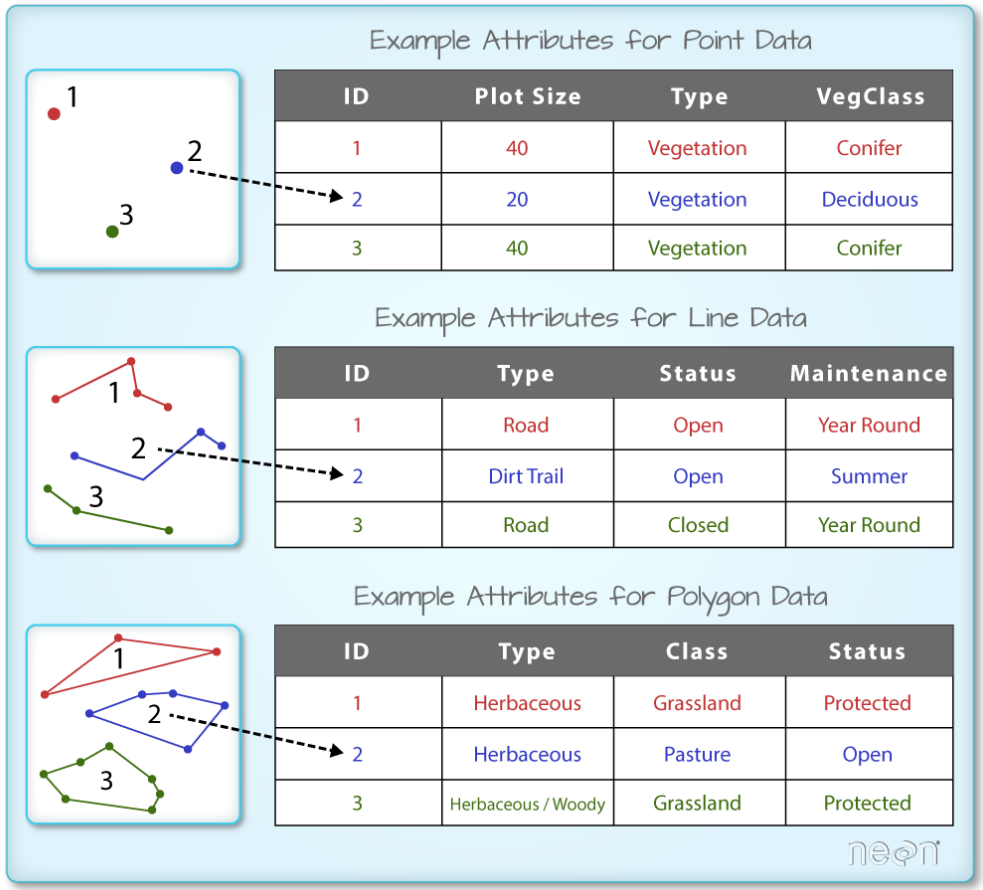
Plot Spatial Object

We can subset the the spatial object just like any other dataframe.

```
plot(worldmap[1:42,])
```

GIS Data: Shapefiles

One of the main GIS data types is vector data. Vectors (also called shapefiles) consist of points, lines and polygons. These shapes are attached to a dataframe, where each row corresponds to a different spatial element.



ID	Plot Size	Type	VegClass
1	40	Vegetation	Conifer
2	20	Vegetation	Deciduous
3	40	Vegetation	Conifer

ID	Type	Status	Maintenance
1	Road	Open	Year Round
2	Dirt Trail	Open	Summer
3	Road	Closed	Year Round

ID	Type	Class	Status
1	Herbaceous	Grassland	Protected
2	Herbaceous	Pasture	Open
3	Herbaceous / Woody	Grassland	Protected

GIS Data: Shapefiles

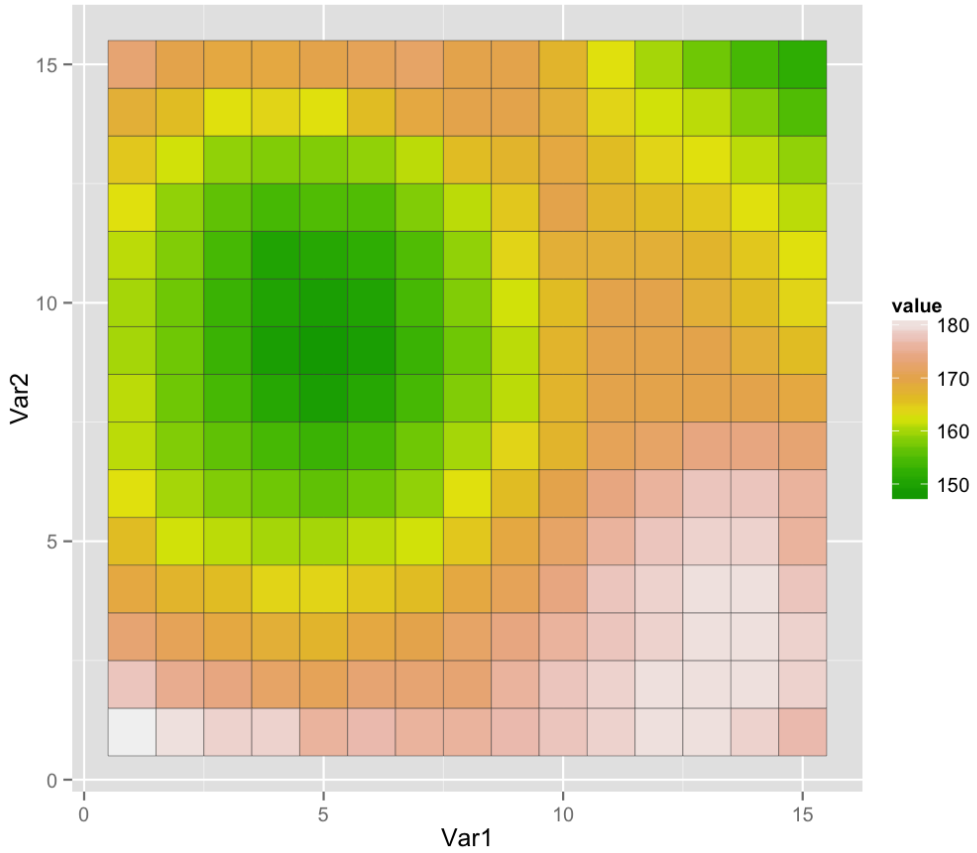
Let's look at the data in our shapefile.

```
# Dataframe
head(worldmap@data)
```

```
##          ADMIN        REGION  POP_EST  GDP_MD_                     region_1
## 0        Aruba South America   103065   2258.0                         <NA>
## 1  Afghanistan          Asia 28400000  22270.0                Southern Asia
## 2       Angola        Africa 12799293 110300.0          Sub-Saharan Africa
## 3     Anguilla South America    14436    108.9                         <NA>
## 4      Albania        Europe  3639453  21810.0 Central and Eastern Europe
## 5        Aland        Europe    27153       NA                         <NA>
##    year hppy_rn hppy_sc  gdp_pc  family  health freedom trst_g_ genrsty
## 0    NA      NA      NA      NA      NA      NA      NA      NA      NA
## 1 2015     154   3.360 0.38227 0.11037 0.17344 0.16430 0.07112 0.31268
## 2 2015     141   3.866 0.84731 0.66366 0.04991 0.00589 0.08434 0.12071
## 3    NA      NA      NA      NA      NA      NA      NA      NA      NA
## 4 2015     109   4.655 0.95530 0.50163 0.73007 0.31866 0.05301 0.16840
## 5    NA      NA      NA      NA      NA      NA      NA      NA      NA
##    dystp_r
## 0       NA
## 1  2.14558
## 2  2.09459
## 3       NA
## 4  1.92816
```

GIS Data: Raster Data

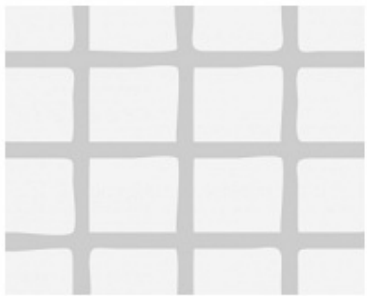
The second main GIS data type is raster data. Rasters are spatially-referenced grids, where each grid has a value associated with it.



Coordinate Reference Systems

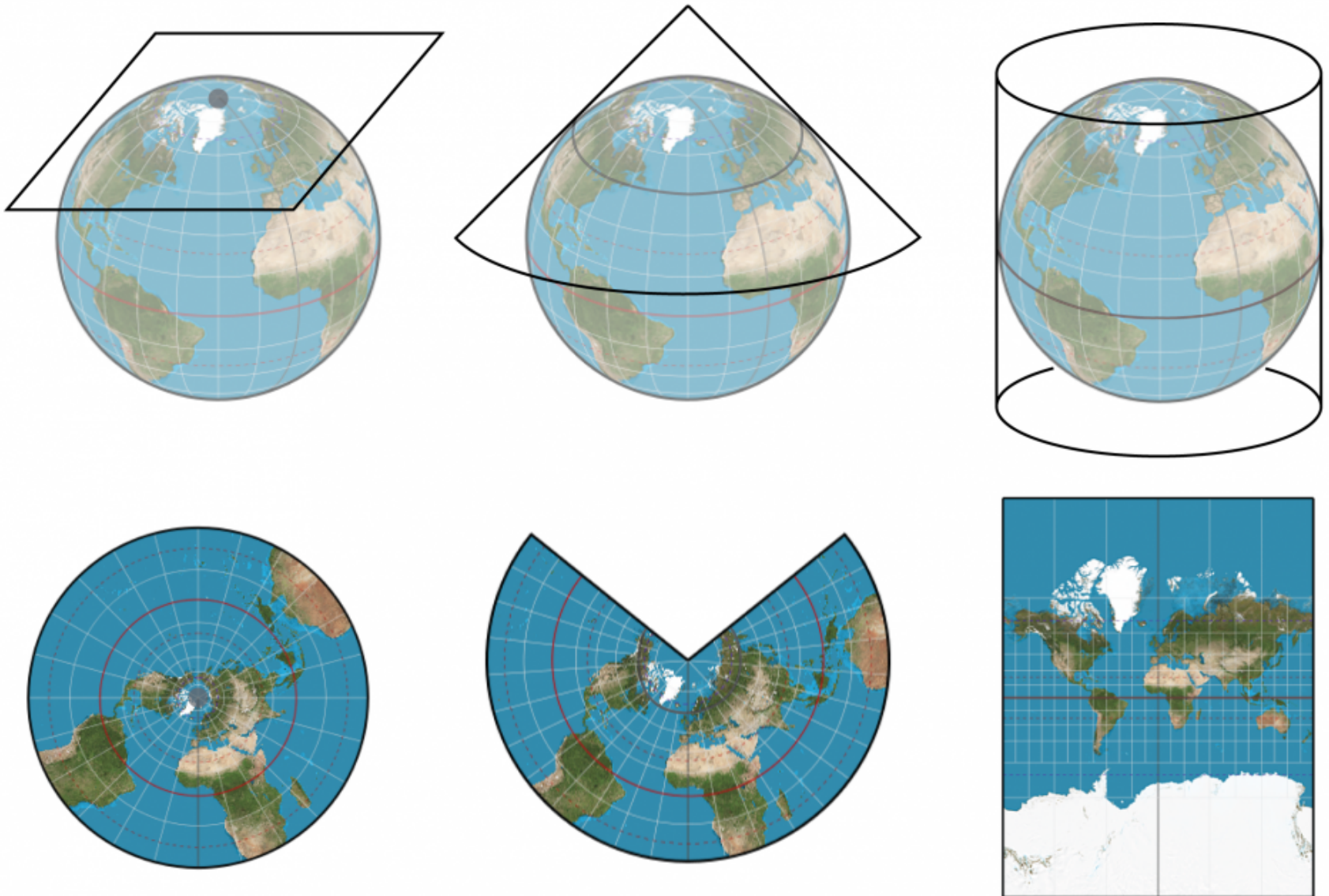
Coordinate reference systems map pairs of numbers to a location.

- **Geographic Coordinate Systems** live on a sphere; here, the units are in decimal degrees (latitude = angle from equator; longitude = angle from prime meridian)
  - Using the WGS84 coordinate system the World Bank MC building is located at 38.89 degrees latitude and -77.04 degrees longitude.
- **Projected Coordinate Systems** project the earth onto a flat surface (units here are typically in meters from some reference point).
  - Using to the World Mercator projection, the World Bank is located 4680364.64 north and -8576320.73 east.



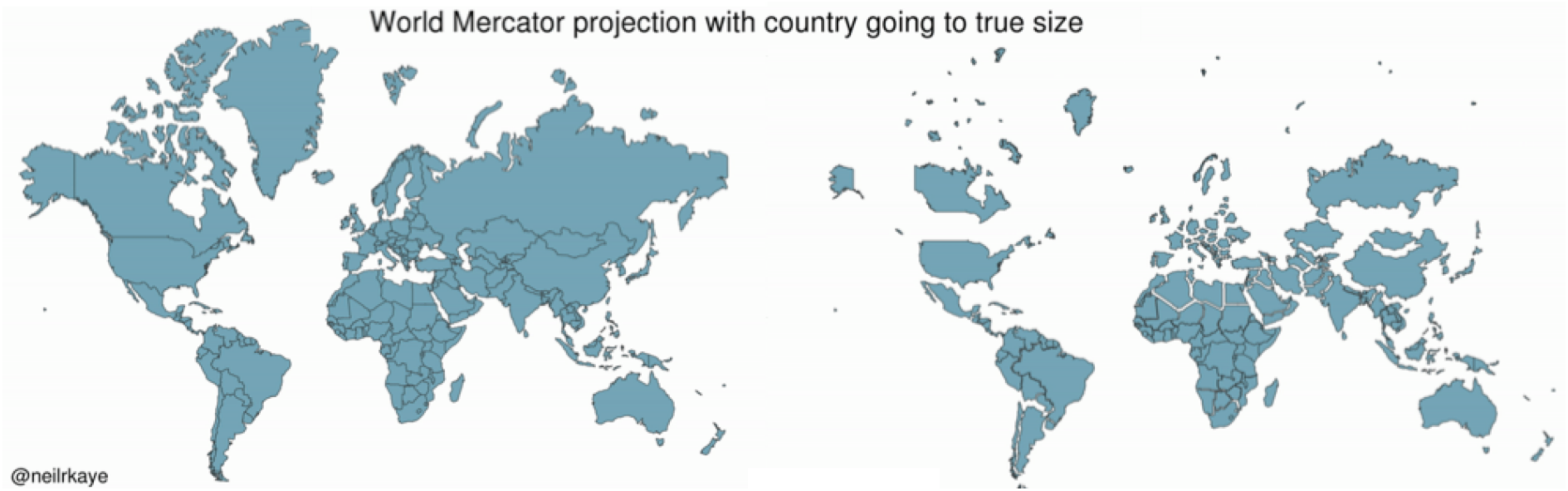
Coordinate Reference Systems

Making The Earth Flat



Coordinate Reference Systems

Projecting the earth onto a flat surface distorts distorts the earth in some way (shape, area, distance or direction).



Coordinate Reference Systems

Let's check the coordinate reference system of our shapefile

```
# Coordinate Repference System
worldmap@proj4string
```

```
## CRS arguments:
##  +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

And lets look at the values of the coordinates for the center points of each country.

```
# Grab center of polygons
worldmap_center <- coordinates(worldmap)
head(worldmap_center)
```

```
##         [,1]      [,2]
## 0 -69.98267  12.52089
## 1  66.00473  33.83523
## 2  17.56443 -12.33195
## 3 -63.06498  18.22397
## 4  20.04983  41.14245
## 5  19.94399  60.23134
```

Coordinate Reference Systems

We can change the projection using `spTransform()`. Lets change to the mercator projection, which preserves angles and directions (distorting distance and area).

EPSG codes reference commonly used projections and provide a shorthand to referencing a projection. epsg:3857 is a shorthand for the mercator projection.

```
# Reproject
worldmap_reproj <- spTransform(worldmap, crs("+init=epsg:3857"))
```

And lets look at some coordinates. These look very different than before!

```
# Look at some coordinates
worldmap_reproj_center <- coordinates(worldmap_reproj)
head(worldmap_reproj_center)
```

```
##        [,1]     [,2]
## 0 -7790436  1405048
## 1  7352021  4017160
## 2  1955500 -1390000
## 3 -7020362  2063781
## 4  2231825  5035200
## 5  2220164  8451485
```

Coordinate Reference Systems

When plotting, the map now looks different too.

```
plot(worldmap_reproj[!(worldmap_reproj$REGION %in% "Antarctica"),])
```

Outline

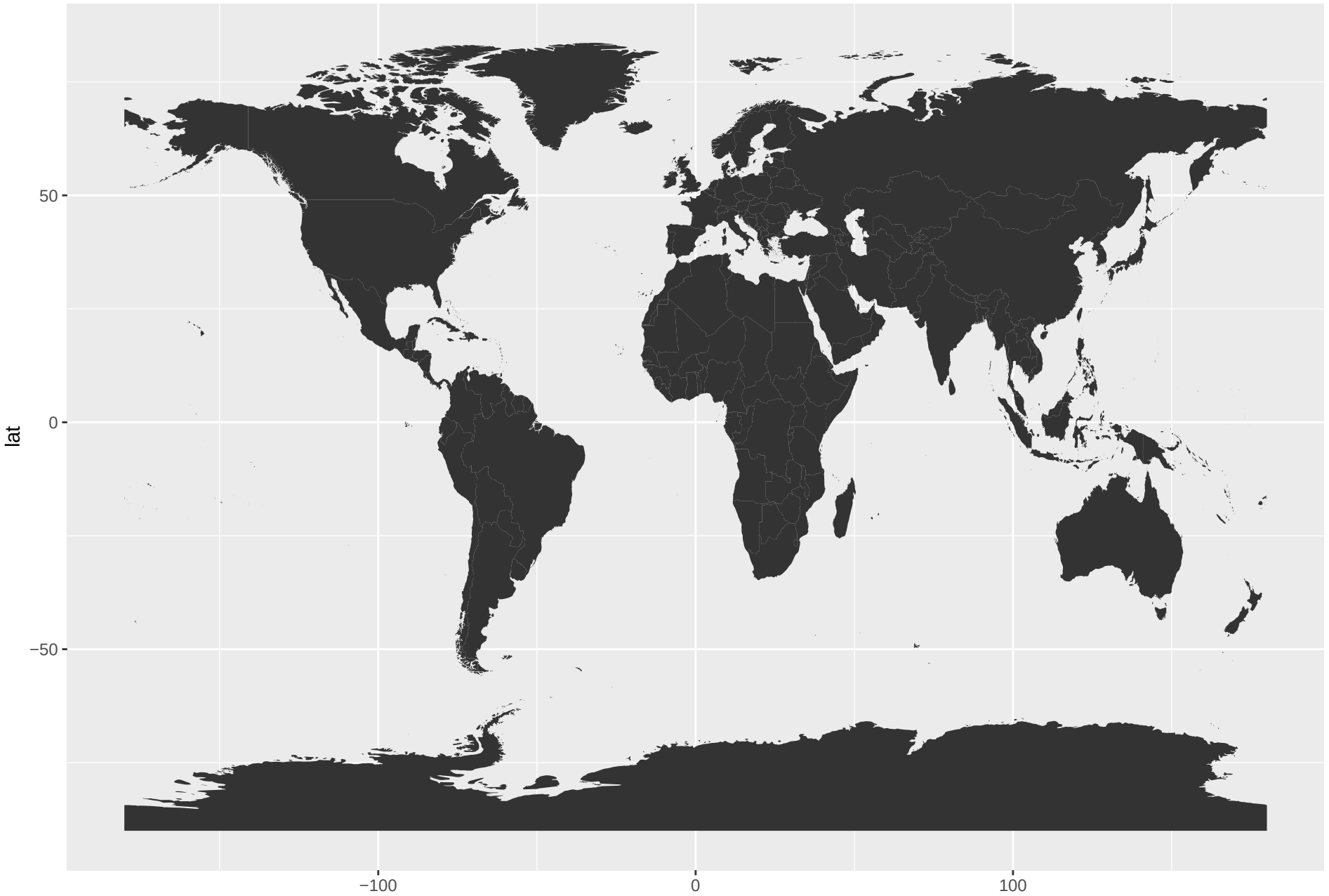
Making a Map with ggplot

Basics of mapping with ggplot

1. ggplot has methods for mapping spatial objects. We use geom_polgyon for polygons and geom_path for lines.
2. In aes() for spatial objects, we always put x=lon, y=lat and group=group, even though our spatial object may not have these variables. We do this because ggplot converts the spatial object into a dataframe that has these variables. ggplot can't directly use shapefiles, but converts them into a dataframe that it can interpret.

```
library(ggplot2)
ggplot() +
  geom_polygon(data=worldmap, aes(x=long, y=lat, group=group))
```

Making a Map with ggplot



Making a Map with ggplot

Now, let's color the map by a variable in the dataset.

- We may be tempted to use the below code, but it won't work.
- When ggplot converts the spatial object into a dataframe, it doesn't keep any of our variables – it only keeps spatial information.

```
# THIS CODE WILL NOT WORK :'(
ggplot() +
  geom_polygon(data=worldmap, aes(x = long, y = lat,
                                  group = group,
                                  fill = hppy_sc))
```

Converting Spatial Object into Dataframe for ggplot

To use our variables when mapping a spatial object with ggplot, we need to convert the spatial object into a dataframe and add our variables ourselves.

- We use the tidy function from the broom package to convert our shapefile into a dataframe.
- The `tidy()` function will create an "id" variable where values are the row.names from the shapefile. So we can later merge data from the shapefile to the new dataframe, we create an id variable in the shapefile that are the row.names.

```
library(broom)
worldmap$id <- row.names(worldmap)
worldmap_tidy <- tidy(worldmap)
```

```
head(worldmap_tidy, 3)
```

```
## # A tibble: 3 x 7
##    long   lat order hole  piece group id
##   <dbl> <dbl> <int> <lgl> <chr> <chr> <chr>
## 1 -69.9  12.5     1 FALSE 1     0.1   0
## 2 -69.9  12.4     2 FALSE 1     0.1   0
## 3 -69.9  12.4     3 FALSE 1     0.1   0
```

Merge data back to object

Now, merge the "tidy" dataframe with the data from the shapefile by the id variable. Now we have a dataframe that ggplot can understand with our data!

```
worldmap_tidy <- merge(worldmap_tidy, worldmap@data, by="id")
head(worldmap_tidy, 3)
```

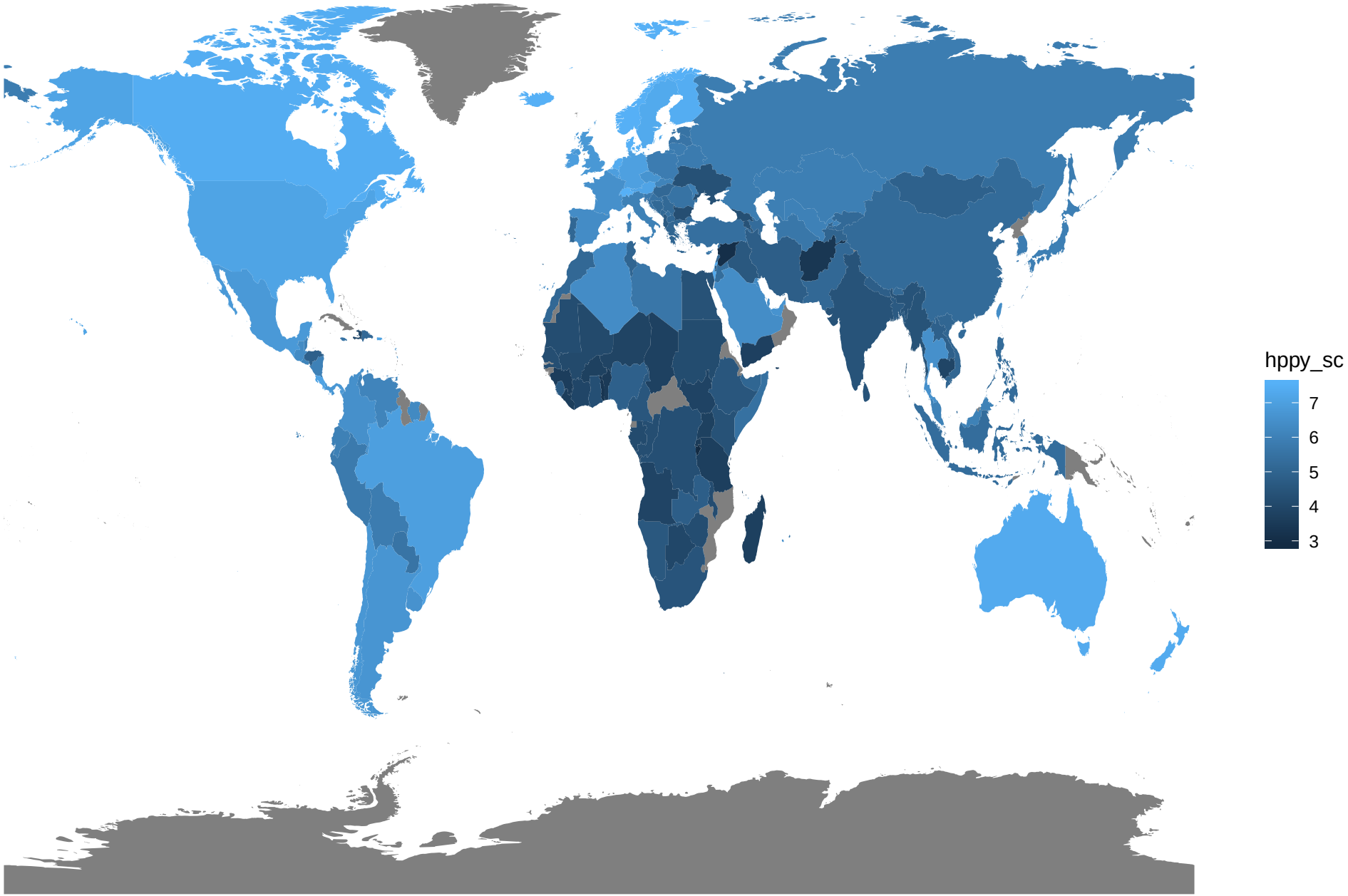
```
##   id      long      lat order  hole piece group ADMIN        REGION
## 1  0 -69.89914 12.45201     1 FALSE     1   0.1 Aruba South America
## 2  0 -69.89568 12.42301     2 FALSE     1   0.1 Aruba South America
## 3  0 -69.94216 12.43852     3 FALSE     1   0.1 Aruba South America
##   POP_EST GDP_MD_ region_1 year hppy_rn hppy_sc gdp_pc family health
## 1  103065    2258     <NA>   NA      NA      NA     NA     NA     NA
## 2  103065    2258     <NA>   NA      NA      NA     NA     NA     NA
## 3  103065    2258     <NA>   NA      NA      NA     NA     NA     NA
##   freedom trst_g_ genrsty dystp_r
## 1      NA      NA      NA      NA
## 2      NA      NA      NA      NA
## 3      NA      NA      NA      NA
```

Making a Map with ggplot

Now, using this tidy dataframe, we can make a map.

```
library(ggplot2)

ggplot() +
  geom_polygon(data=worldmap_tidy, aes(x=long, y=lat,
                                       group=group,
                                       fill = hppy_sc)) +
  theme_void()
```

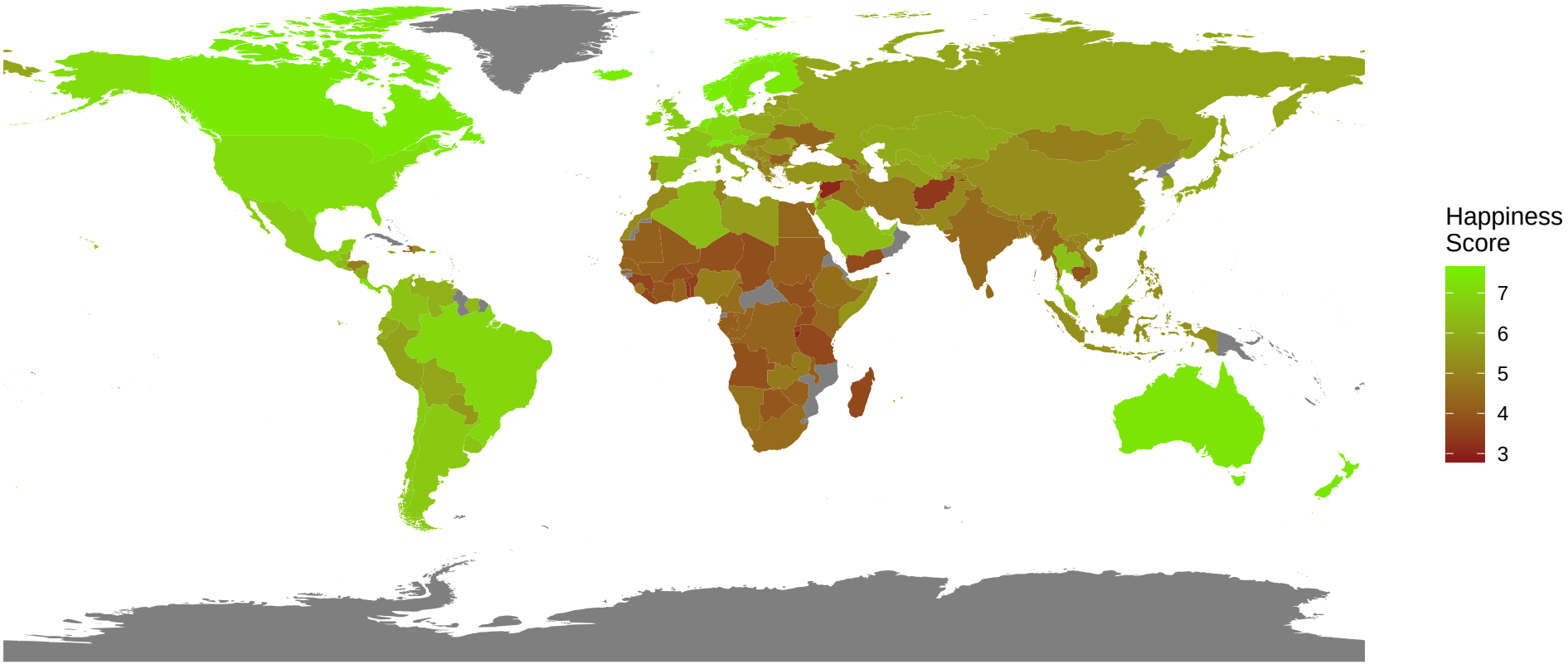
Making a Map with ggplot



Making a Prettier Map with ggplot

```
library(ggplot2)

ggplot() +
  geom_polygon(data=worldmap_tidy, aes(x=long, y=lat,
                                       group=group,
                                       fill = hppy_sc)) +
  theme_void() +
  coord_quickmap() + # Removes distortions
  labs(fill="Happiness\nScore") +
  scale_fill_gradient(low = "firebrick4",
                      high = "chartreuse2")
```

Making a Prettier Map with ggplot



Map of Africa

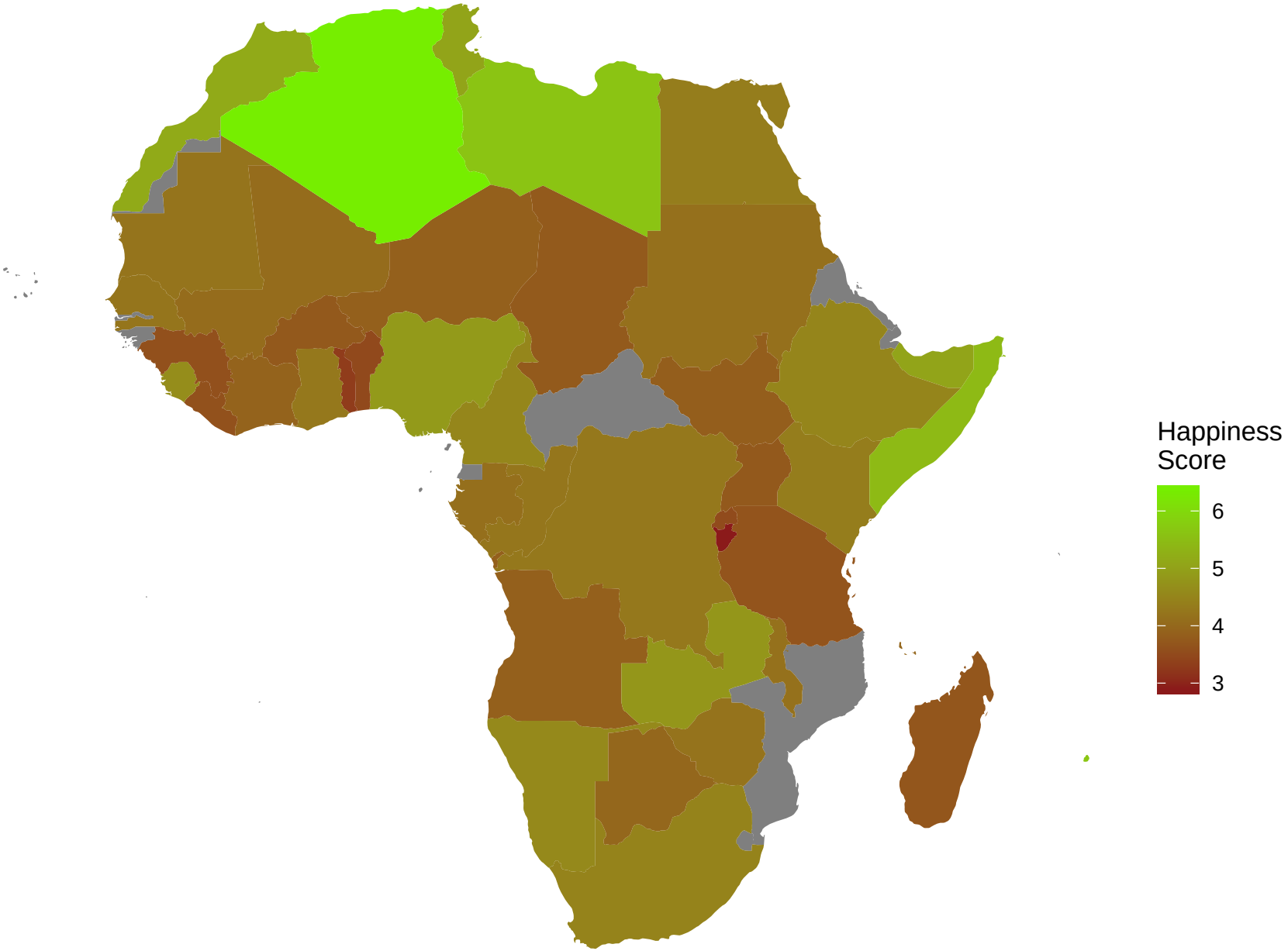
Exercise 1

Make a map just of Africa. Hint: Subset world_tidy using the *REGION* variable. You can use the `subset()` function to subset a dataframe. Forget how the function works? Type `help(subset)` or `?subset` for the help file.

Map of Africa

```
africa_tidy <- subset(worldmap_tidy, REGION == "Africa")

ggplot() +
  geom_polygon(data=africa_tidy, aes(x=long, y=lat,
                                     group=group,
                                     fill = hppy_sc)) +
  theme_void() +
  coord_quickmap() + # Removes distortions
  labs(fill="Happiness\nScore") +
  scale_fill_gradient(low = "firebrick4",
                      high = "chartreuse2")
```

Map of Africa



Outline

Add Points to Map: Does the WB allocate projects to happy or sad countries?

We have data on the locations of World Bank projects from AidData. Let's plot projects on our map to see if the World Bank tends to allocate projects to happy or sad countries.

```
wb_projects <- read.csv(file.path(finalData, "wb_projects.csv"))
names(wb_projects)
```

```
##  [1] "project_location_id"  "place_name"           "latitude"
##  [4] "longitude"            "project_title"        "start_actual_isodate"
##  [7] "end_actual_isodate"   "recipients"           "ad_sector_names"
## [10] "commitments"          "disbursements"
```

Add Points to Map

For ggplot, we don't need to convert the dataframe to a spatial object. ggplot wants a dataframe, and that's what we have.

Exercise 2

Add World Bank project locations to the previous map.

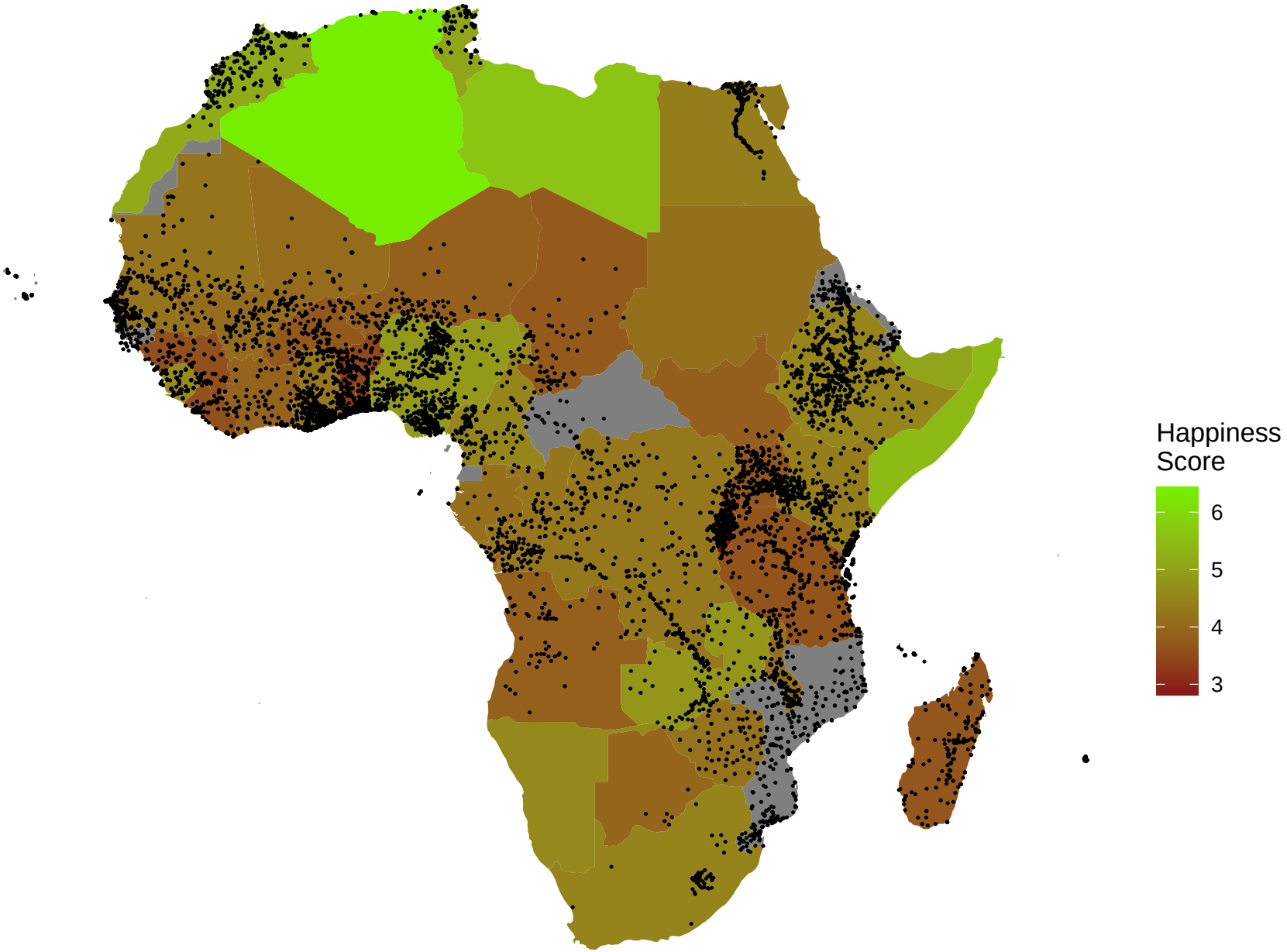
Hints:

1. Use geom_points() to add points data
2. Set data=wb_projects
3. In aes(), set x=longitude and y=latitude

Add Points to Map

```
ggplot() +
  geom_polygon(data=africa_tidy, aes(x=long, y=lat,
                                     group=group,
                                     fill = hppy_sc)) +
  geom_point(data=wb_projects, aes(x=longitude,
                                   y=latitude),
             size=.1) +
  theme_void() +
  coord_quickmap() + # Removes distortions
  labs(fill="Happiness\nScore") +
  scale_fill_gradient(low = "firebrick4",
                      high = "chartreuse2")
```

Add Points to Map



Outline

Add trunk roads to map

Exercise 3

Let's add roads to our map. In the final data folder, there's a file called troads.shp, which is a shapefile of trunk roads in Africa.

Hints:

1. Load the data using readOGR. readOGR has two arguments: dsn (the file path to the shapefile) and layer (the name of the shapefile)
2. To add roads to the map, do we need to use the tidy function to convert the spatial object to a dataframe?
3. In ggplot, geom_path is used to handle spatial lines. Remember, in aesthetics (aes) we need to add x=long, y=lat, and group=group.

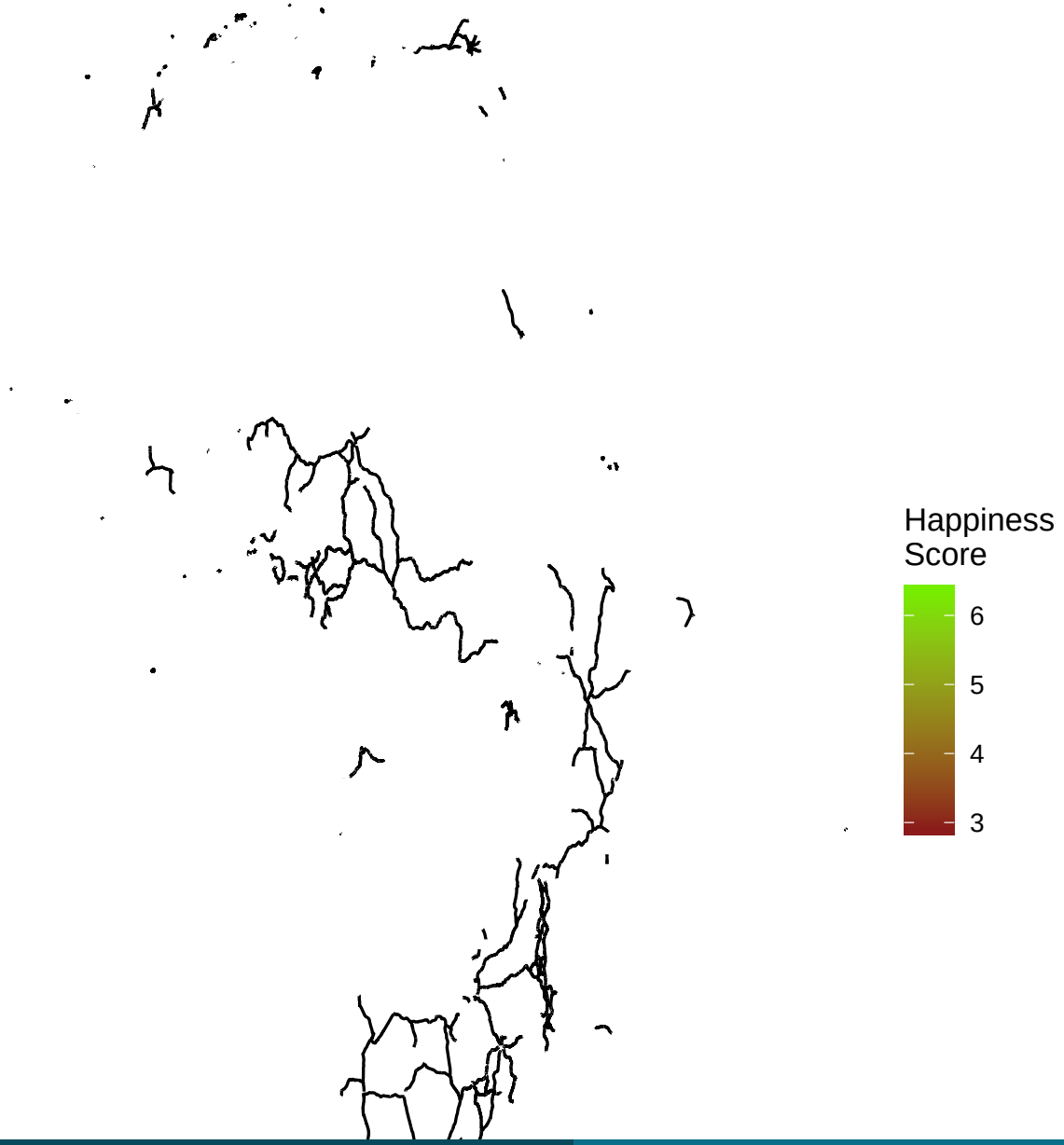
Add trunk roads to map

Lets load the data and add it to our map.

```
trunk_roads <- readOGR(dsn=finalData, layer="troads")

ggplot() +
  geom_polygon(data=africa_tidy, aes(x=long, y=lat,
                                     group=group,
                                     fill = hppy_sc)) +
  geom_path(data=trunk_roads, aes(x=long, y=lat,
                                  group=group)) +
  geom_point(data=wb_projects, aes(x=longitude,
                                   y=latitude),
             size=.1) +
  theme_void() +
  coord_quickmap() + # Removes distortions
  labs(fill="Happiness\nScore") +
  scale_fill_gradient(low = "firebrick4",
                      high = "chartreuse2")
```

Add trunk roads to map

We see the roads, but where did the rest of the map go?



Check Projections: They Don't Match

Let's check the projections of the worldmap data and our trunk roads.

```
# World Map Projection
worldmap@proj4string
```

```
## CRS arguments:
##  +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

```
# Trunk Roads Projection
trunk_roads@proj4string
```

```
## CRS arguments:
##  +proj=utm +zone=30 +a=6378249.145 +b=6356514.96582849 +units=m
## +no_defs
```

Reproject Data

We use the `spTransform()` function to reproject the data.

The below code shows two different ways of specifying the same projection.

1. The first way writes out the full projection, explicitly defining parameters such as proj, datum, etc.
2. The second way uses an epsg code. EPSG codes reference commonly used projections and provide a shorthand to referencing a projection.

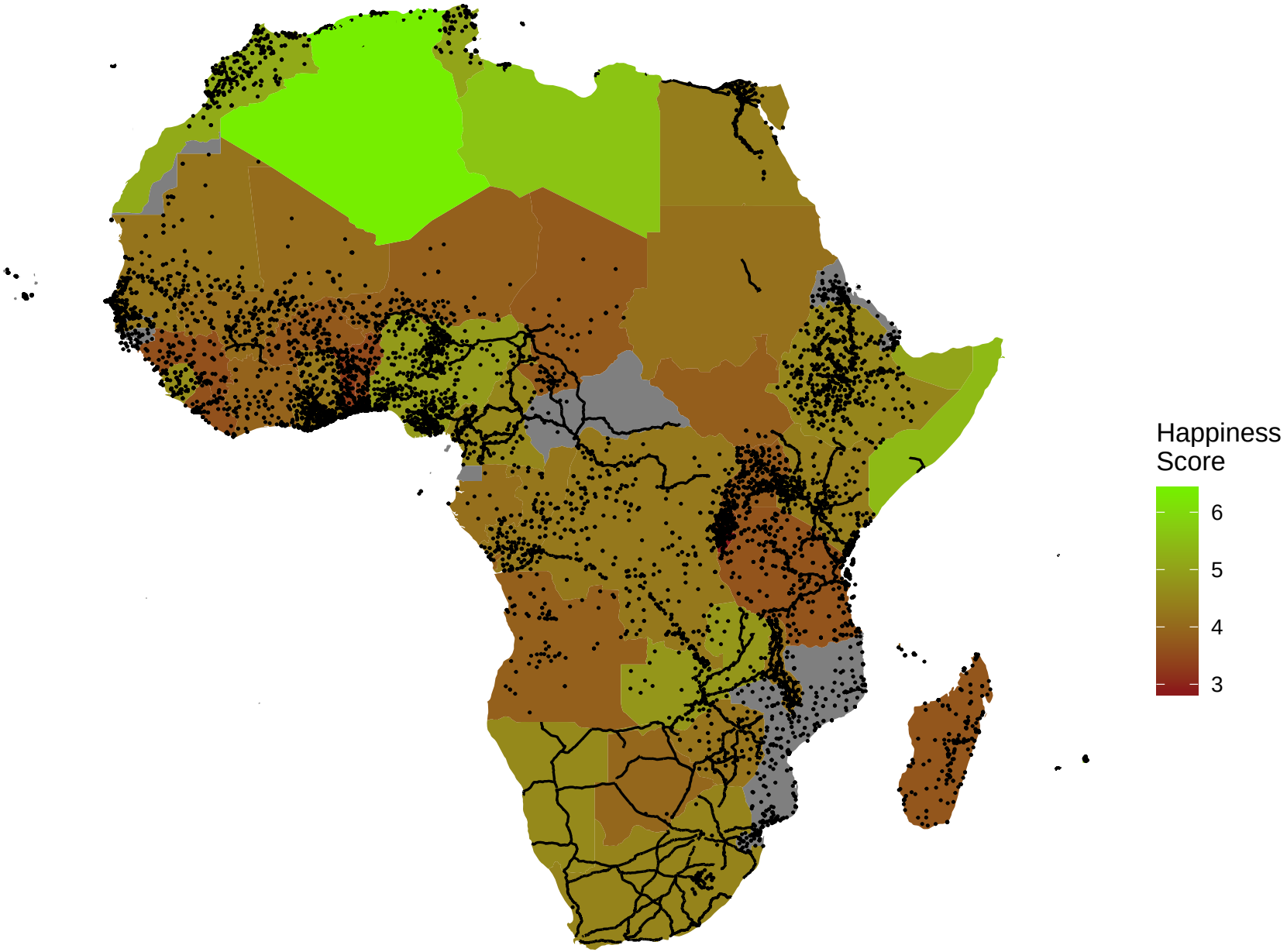
```
# Way 1
prj <- "+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
trunk_roads_reprj <- spTransform(trunk_roads, CRS(prj))

# Way 2
trunk_roads_reprj <- spTransform(trunk_roads, CRS("+init=epsg:4326"))
```

Add trunk roads to map

Now, plot the map using the reprojected version of the trunk roads.

```
ggplot() +
  geom_polygon(data=africa_tidy, aes(x=long, y=lat,
                                     group=group,
                                     fill = hppy_sc)) +
  geom_path(data=trunk_roads_reprj, aes(x=long, y=lat,
                                        group=group)) +
  geom_point(data=wb_projects, aes(x=longitude,
                                   y=latitude),
             size=.1) +
  theme_void() +
  coord_quickmap() + # Removes distortions
  labs(fill="Happiness\nScore") +
  scale_fill_gradient(low = "firebrick4",
                      high = "chartreuse2")
```

Add trunk roads to map

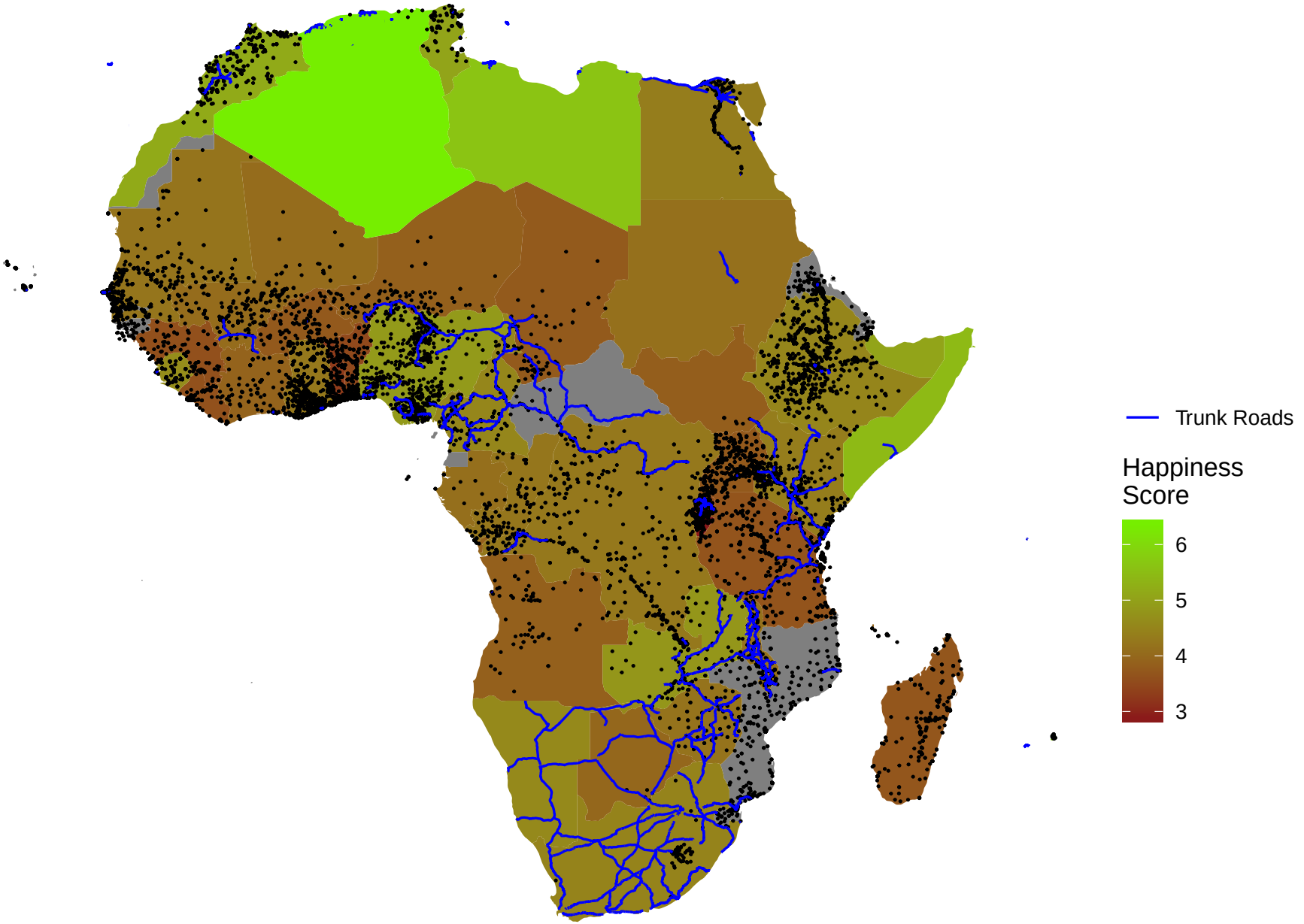


Add Roads to Legend

ggplot only puts objects in the legend where some element is in the aesthetics. To add "trunk roads" in the legend, add a color element in the aesthetics then manually define the color.

```
ggplot() +
  geom_polygon(data=africa_tidy, aes(x=long, y=lat,
                                     group=group,
                                     fill = hppy_sc)) +
  geom_point(data=wb_projects, aes(x=longitude,
                                   y=latitude),
             size=.1) +
  geom_path(data=trunk_roads_reprj, aes(x=long, y=lat,
                                  group=group,
                                  color="Trunk Roads")) +
  theme_void() +
  coord_quickmap() + # Removes distortions
  labs(fill="Happiness\nScore",
       color="") + # REMOVE TITLE ABOVE TRUNK ROAD IN LEGEND
  scale_fill_gradient(low = "firebrick4",
                      high = "chartreuse2") +
  scale_color_manual(values=c("blue")) # MANUALY DEFINE COLOR HERE
```

Add Roads to Legend



Outline

Basemap

We can also plot our spatial data on basemaps. One of the packages that enables use to do this is the ggmap package. In the function, *maptype* changes the type of basemap we use. Check the helpfile to see what other basemaps can be used help(get_stamenmap).

```
library(ggmap)

nairobi <- c(left = 36.66,
             bottom = -1.44,
             right = 37.10,
             top = -1.15)
nairobi_map <- get_stamenmap(nairobi,
                             zoom = 11,
                             maptype = "toner-lite")

ggmap(nairobi_map)
```

Basemap


−1.15
−1.20
−1.25
−1.30
−1.35
−1.40
lat
36.7
36.8
36.9
37.0
lon
Limuru Road
Kiambu
Kiu Valley Road
Ruiru Road
Eastern Bypass
Northern Bypass
Karura Forest
Kikuyu
Juja Road
Kangundo Road
Nairobi
Jogoo Road
Dagoretti Road
Ngong Road
Ngong Road Forest
Langata Road
Mombasa Road
Magadi Road

Basemap

We can add layers to basemap. The code is the same as ggplot, except here we start the code with ggmap() instead of gplot().

Exercise 4

Plot projects in Narobi on top of a basemap. As a challenge, make the size AND the color of the points correspond to aid commitments.
Hint: Set size and color to "commitments" in the aesthetics.

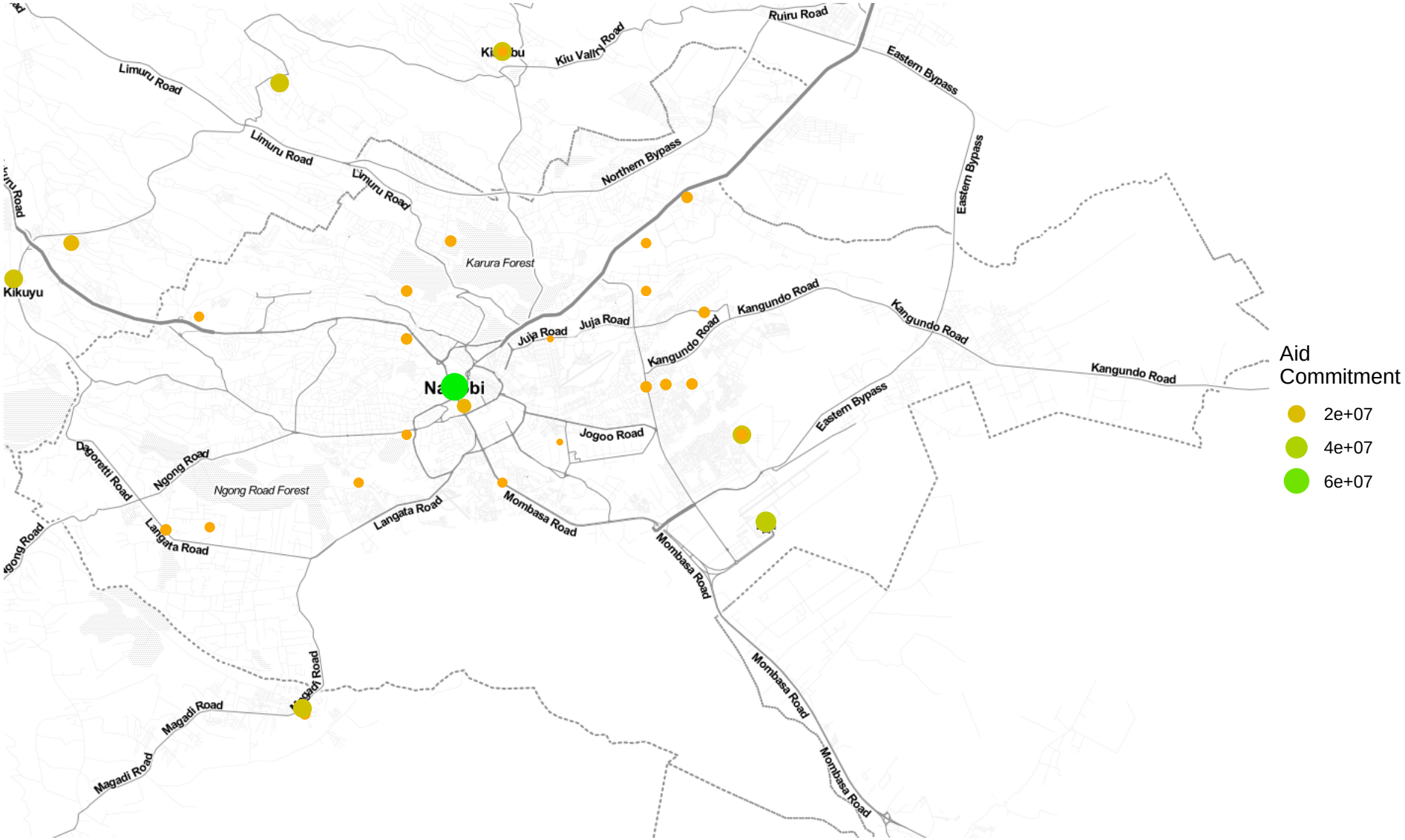
```
# Code where size/color does not correspond to commitments
kenya_projects <- subset(wb_projects, recipients == "Kenya")

ggmap(nairobi_map) +
    geom_point(data=kenya_projects, aes(x=longitude,
                                        y=latitude),
               color="orange") +
  theme_void()
```

Basemap

Here's the code to make point size and color correspond with commitments. To combine size and color to one legend item, add "guide=legend" to both scale_color_continuous and scale_size_continuous.

```
ggmap(nairobi_map) +
    geom_point(data=kenya_projects, aes(x=longitude,
                                        y=latitude,
                                        size=commitments,
                                        color=commitments)) +
  scale_color_continuous(guide = "legend",
                         low="orange", high="green2") +
  scale_size_continuous(guide = "legend") +
  labs(size = "Aid\nCommitment",
       color = "Aid\nCommitment") +
  theme_void()
```

Basemap


Limuru Road
Limuru Road
Limuru Road
Northern Bypass
Ruiru Road
Eastern Bypass
Eastern Bypass
Kiu Valley Road
Karura Forest
Kikuyu
Juja Road
Juja Road
Kangundo Road
Kangundo Road
Kangundo Road
Kangundo Road
Jogoo Road
Eastern Bypass
Dagoretti Road
Ngong Road
Ngong Road Forest
Langata Road
Langata Road
Mombasa Road
Mombasa Road
Mombasa Road
Mombasa Road
Magadi Road
Magadi Road
Aid Commitment
2e+07
4e+07
6e+07

Outline

Convert dataframe to spatialdataframe

So far we've been using points as a dataframe. However, we can also define points as a spatial points dataframe. Here we explicitly define which variables are the latitude and longitude and can define a projection.

We don't necessarily need to do this for interactive maps, but it'll make our code simplier later on.

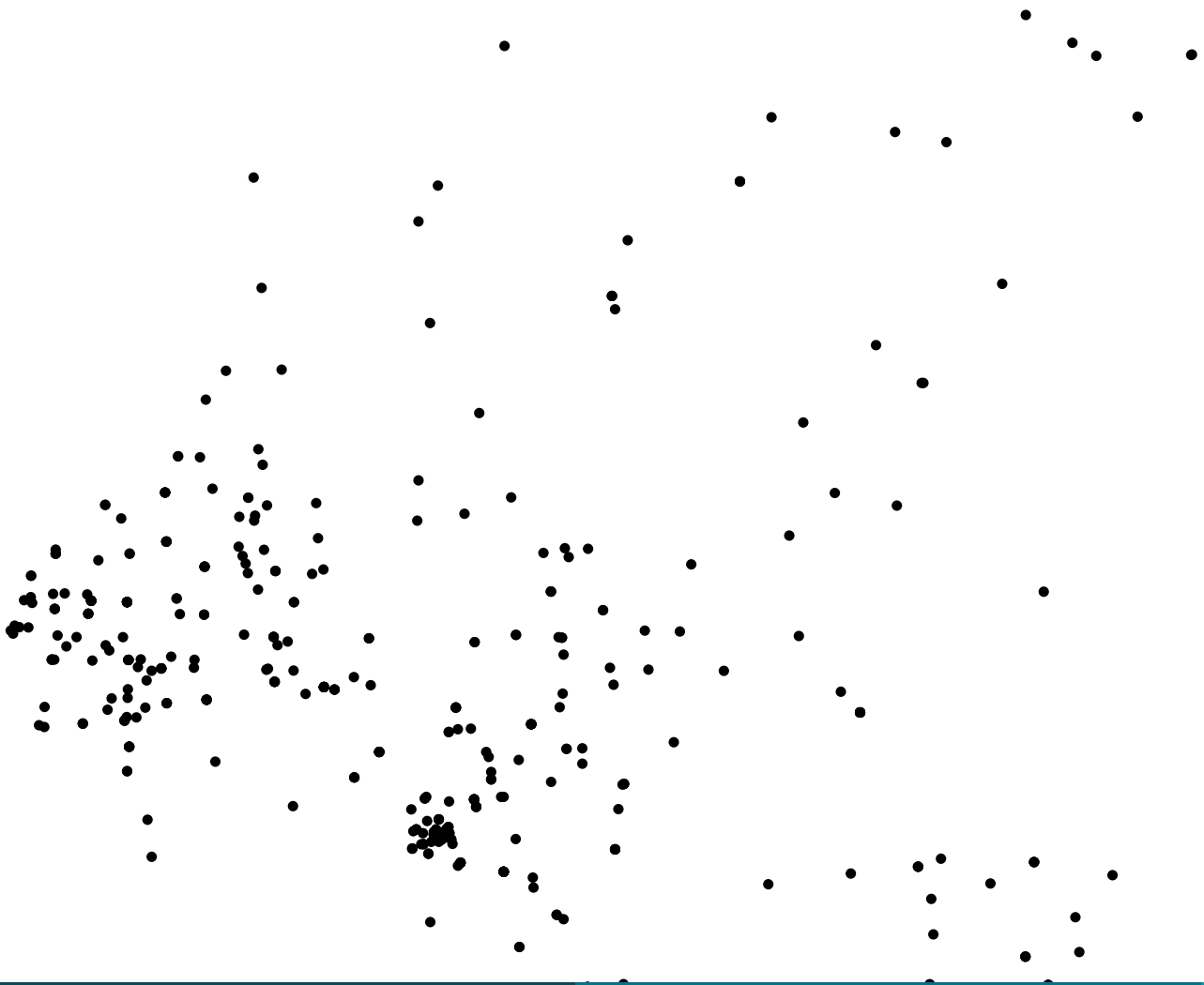
1. We first define which variables are the latitude and longitude variables using the `coordinates()` function.
2. Then, we define our coordinate reference system using the crs function.

```
library(raster)

# Create Spatial Points Dataframe
#    ~longitude+latitude is a formular, where ~ tells R to reference those
#    variables in the associated datasets; in this case, kenya_projects
coordinates(kenya_projects) <- ~longitude+latitude

# Define Projection
crs(kenya_projects) <- CRS("+init=epsg:4326")
```

Convert dataframe to spatialdataframe

We can directly plot spatialdataframes. (pch is symbol and cex is size).

```
plot(kenya_projects, pch=16, cex=.5)
```



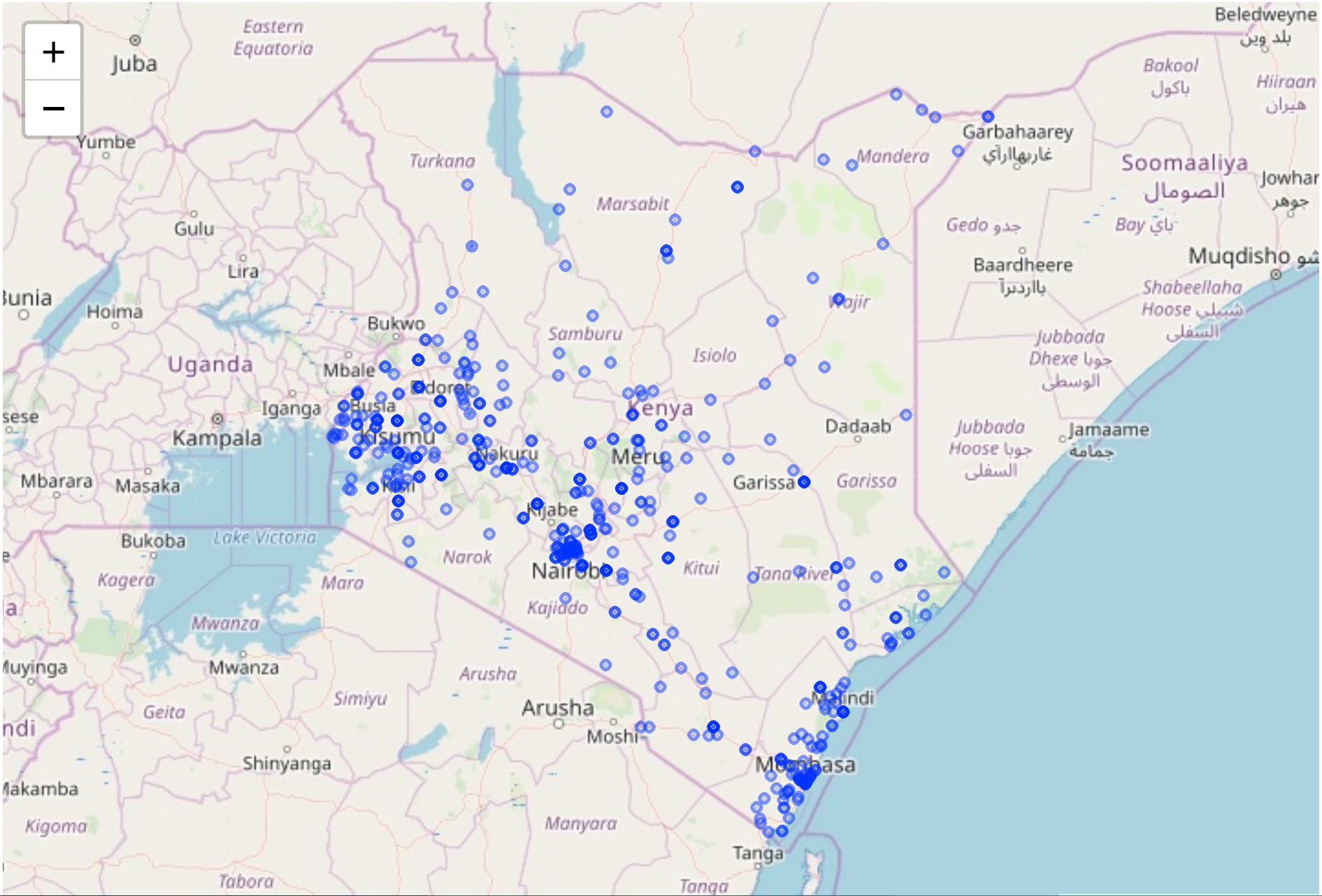
Interactive Map

We'll use leaflet to make an interactive map. Leaflet is a javascript library for interactive maps. The leaflet R package allows one to interact with this library in R.

Leaflet works similarly to ggplot where you sequentially define different elements. However, instead of using + between elements, we use a pipe: %>%

```
library(leaflet)

leaflet() %>%
  addCircles(data=kenya_projects) %>%
  addTiles()
```

Interactive Map

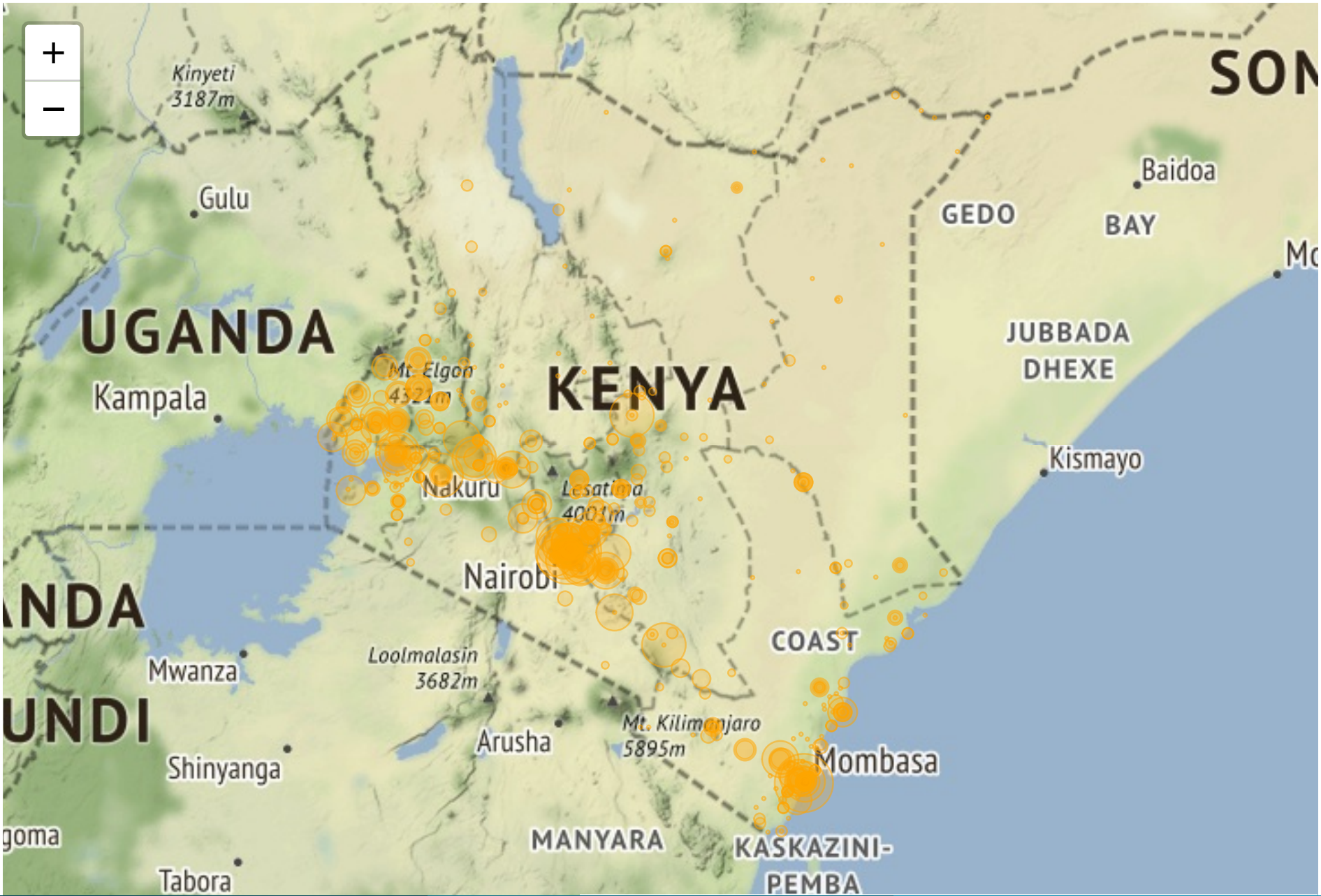


Better Interactive Map

Now, let's make text appear when we click a point, change the radius, change the basemap (click here for options) and change the color.

```
leaflet() %>%
  addCircles(data=kenya_projects,
             popup = ~project_title,
             radius = ~sqrt(commitments)*5,
             weight = 1,
             color = "orange") %>%
  addProviderTiles(providers$Stamen.Terrain)
```

Better Interactive Map



Two layers on map

Exercise 5

Add two layers to the interactive map, differentiating between two types of projects: "Transport and storage" and "Water supply and sanitation." Make sure to color the layers differently.

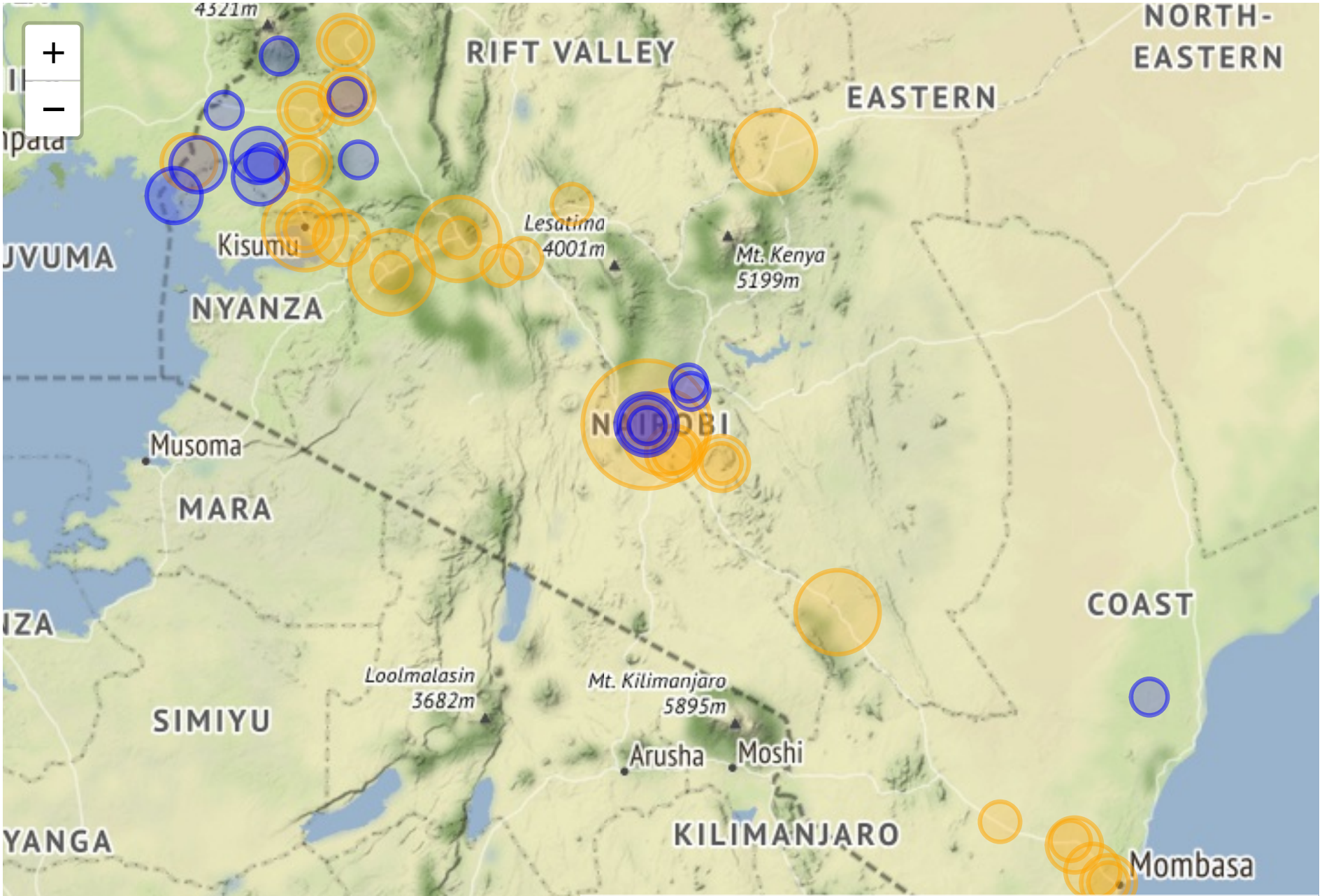
Hints:

1. Use the "ad_sector_names" variable and the subset function to subset kenya_projects.
2. To add multiple layers, just add another addCircles() layer.

Two layers on map

```
leaflet() %>%
  addCircles(data=subset(kenya_projects,
                         ad_sector_names == "Transport and storage"),
             popup = ~project_title,
             radius = ~sqrt(commitments)*5,
             weight = 2.5,
             color = "orange") %>%
    addCircles(data=subset(kenya_projects,
                           ad_sector_names == "Water supply and sanitation"),
             popup = ~project_title,
             radius = ~sqrt(commitments)*5,
             weight = 2.5,
             color = "blue") %>%
  addProviderTiles(providers$Stamen.Terrain)
```

Two layers on map



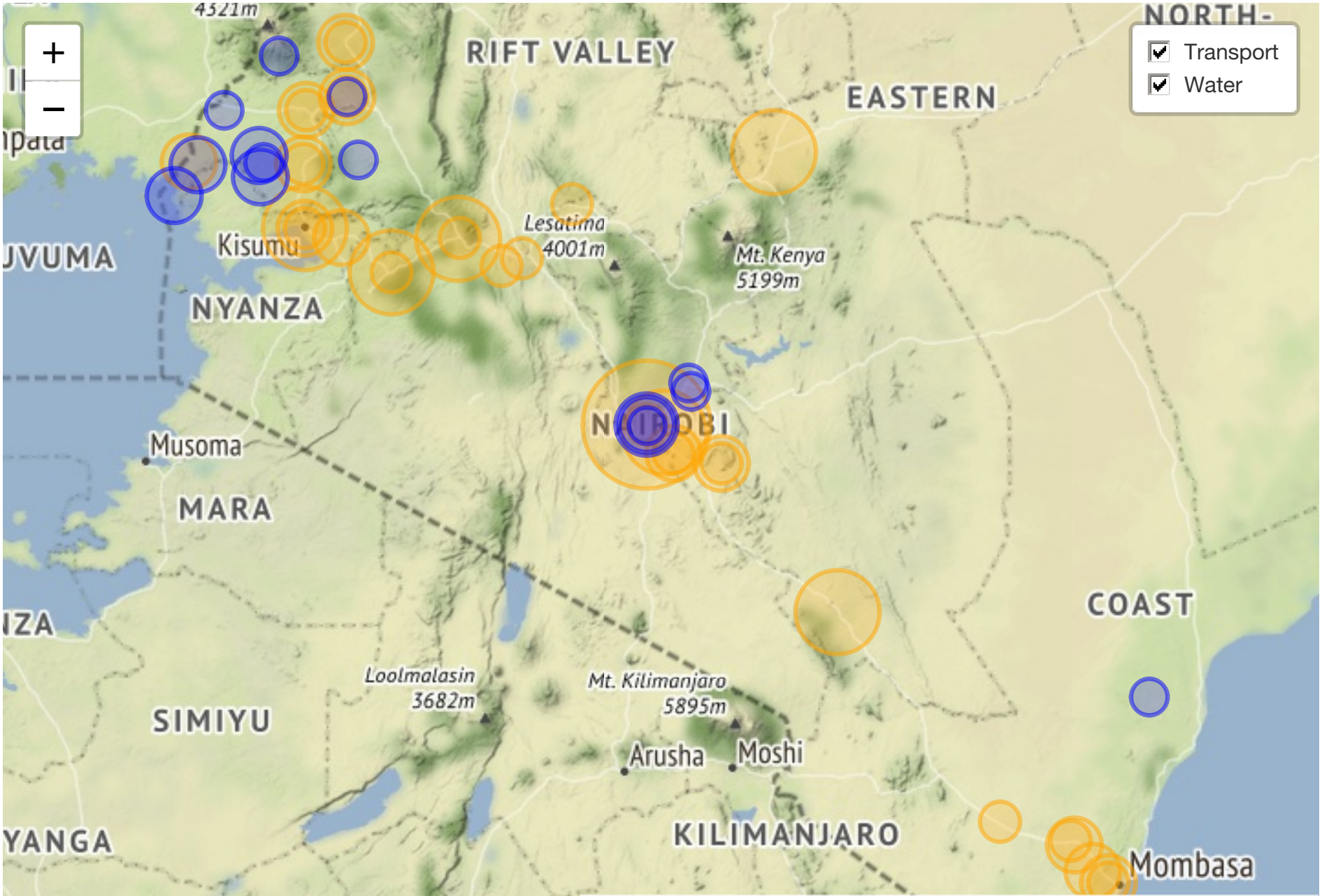
Add Layers and Buttons to Interactive Map

We can add buttons that cause layers to appear or dissappear in two steps:

1. When defining the layer (eg, in addCircles), add "group=" and give the group a name.
2. Add "addLayersControl", and as a parameter add "overlayGroups" with a list of the above groups.

```
leaflet() %>%
  addCircles(data=subset(kenya_projects,
                         ad_sector_names == "Transport and storage"),
             popup = ~project_title,
             radius = ~sqrt(commitments)*5,
             weight = 2.5,
             color = "orange",
             group = "Transport") %>%
    addCircles(data=subset(kenya_projects,
                           ad_sector_names == "Water supply and sanitation"),
             popup = ~project_title,
             radius = ~sqrt(commitments)*5,
             weight = 2.5,
             color = "blue",
             group = "Water") %>%
  addProviderTiles(providers$Stamen.Terrain) %>%
  addLayersControl(overlayGroups = c("Transport", "Water"),
                   options = layersControlOptions(collapsed = FALSE))
```

Add Layers and Buttons to Interactive Map



Outline

Grabbing Administrative Data

We can grab administrative layers from GADM (Database of Global Administrative Areas) using the getData function from the raster package.

```
library(raster)

ken_adm1 <- getData('GADM', country='KEN', level=1)
plot(ken_adm1)
```

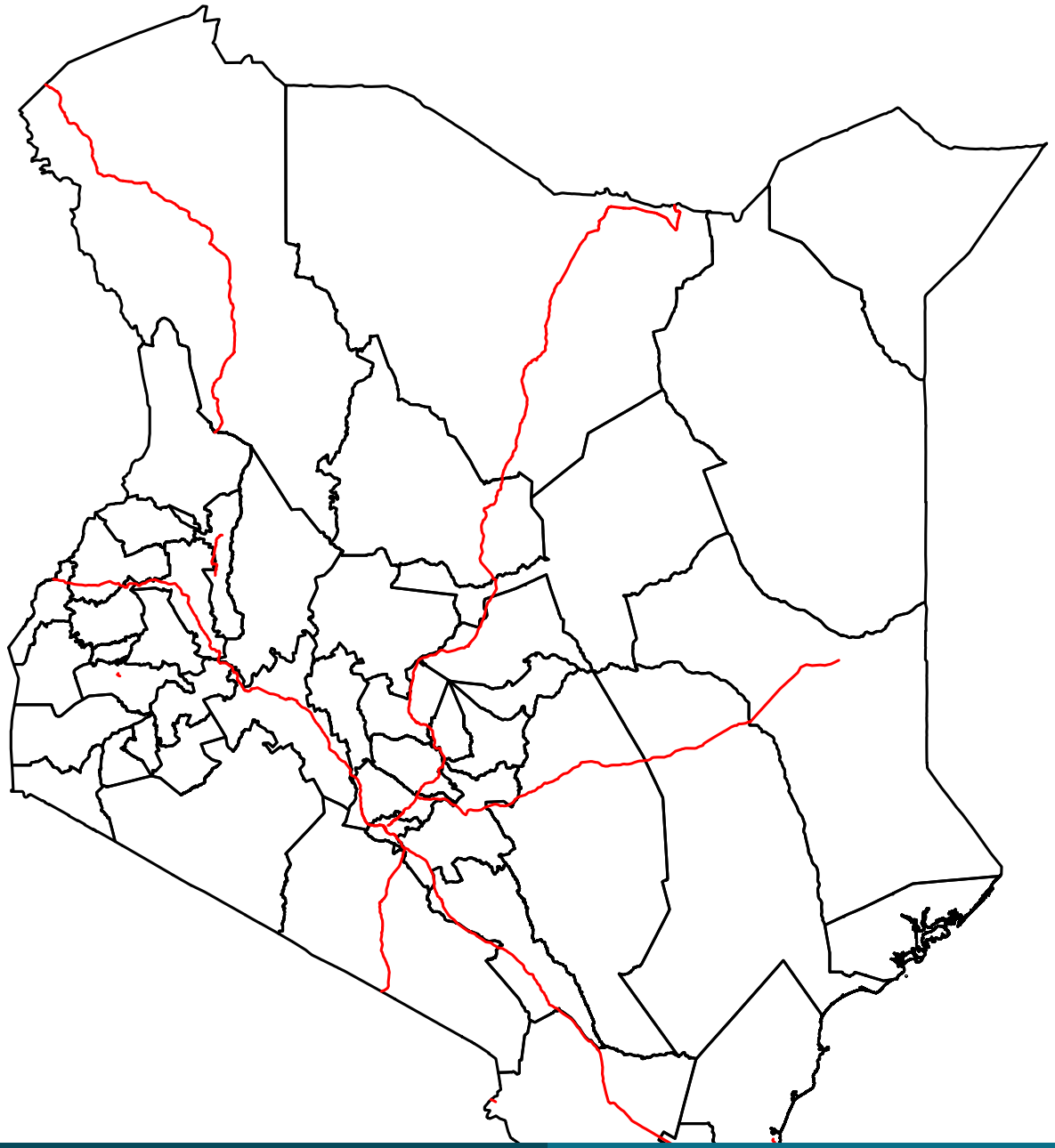
Grabbing Administrative Data

Cropping

Let's trim an object based on the spatial extent of another object. Let's say we only wanted trunk roads in Kenya. We can use gIntersection() from the *rgeos* package. The function converts the "SpatialLineDataFrame" to a "SpatialLine." We'll plot this later, and ggplot only works with "SpatialLineDataFrame" – so to convert the object back, we just add a variable – here, id.

```
library(rgeos)
trunk_roads_kenya <- gIntersection(trunk_roads_reprj,
                                   ken_adm1)
trunk_roads_kenya$id <- 1:length(trunk_roads_kenya)

plot(ken_adm1)
plot(trunk_roads_kenya, add=T, col="red")
```

Cropping



Distance Calculation

Let's calculate the distance of projects to trunk roads. gDistance calculates the distance in the units of the projection (here, decimal degrees)

```
library(rgeos)
# rows = roads; columns = projects
distance_matrix <- gDistance(kenya_projects, trunk_roads_kenya, byid=T)

# Min values of columns (projects)
kenya_projects$dist_road <- apply(distance_matrix, 2, min)
```

Homework: Map of World Bank Projects Near Roads

Exercise 6

Now that we've calculated the distance, make a map that includes the following characteristics:

1. Includes World Bank projects in Kenya that are within 10km of a trunk road (about 0.1 decimal degrees)
2. Includes trunk roads
3. Includes a legend item for both the projects and map
4. The size if the WB project points should correspond to aid commitments.
5. Includes the border of Kenya around the map (hint: use the getData() function from the raster package to get GADM data for ADM 0).

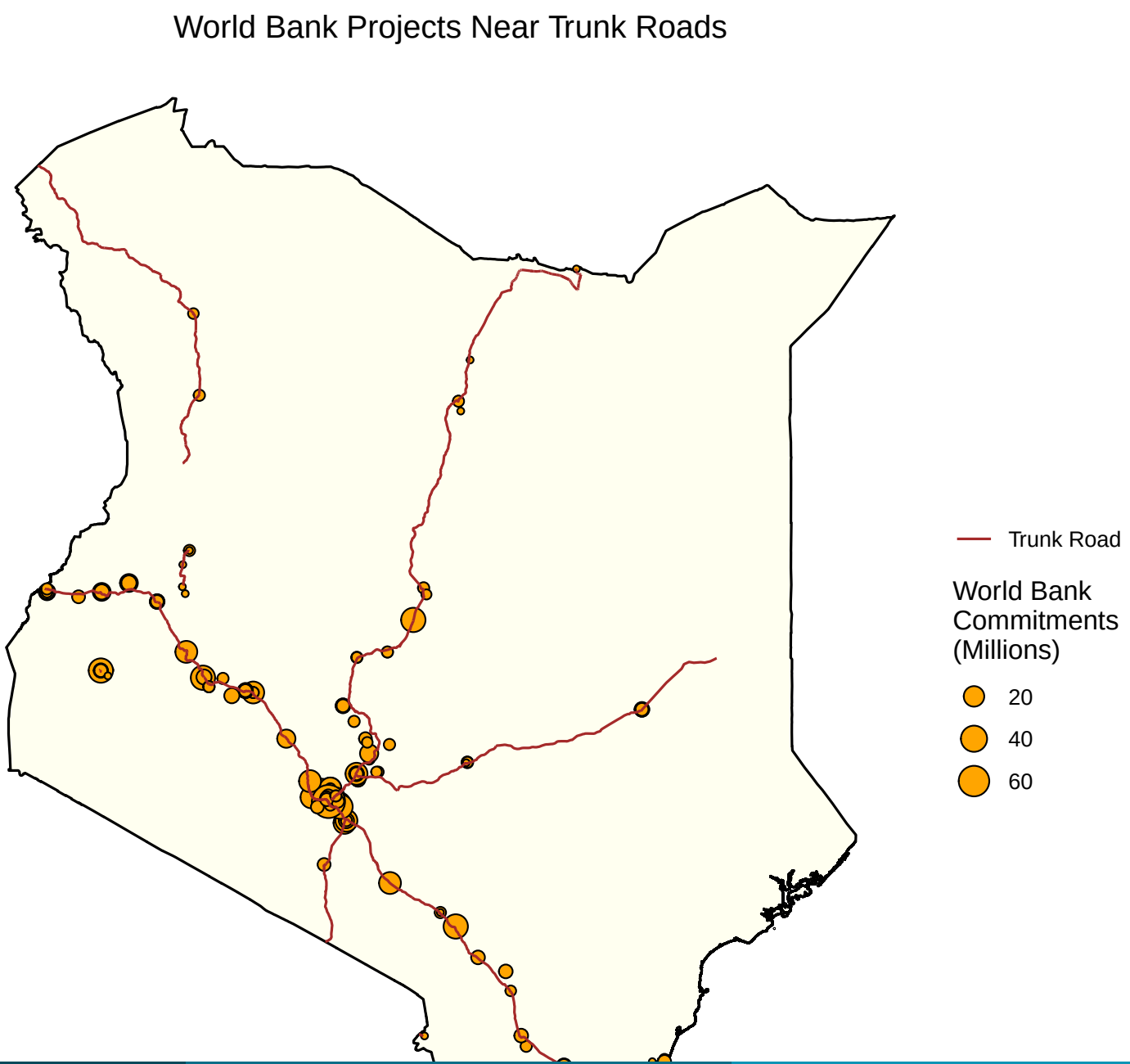
A tip:

ggplot() can't handle spatial points dataframes, so we have to convert back to a dataframe. But tidy doesn't work with spatial points dataframes either.

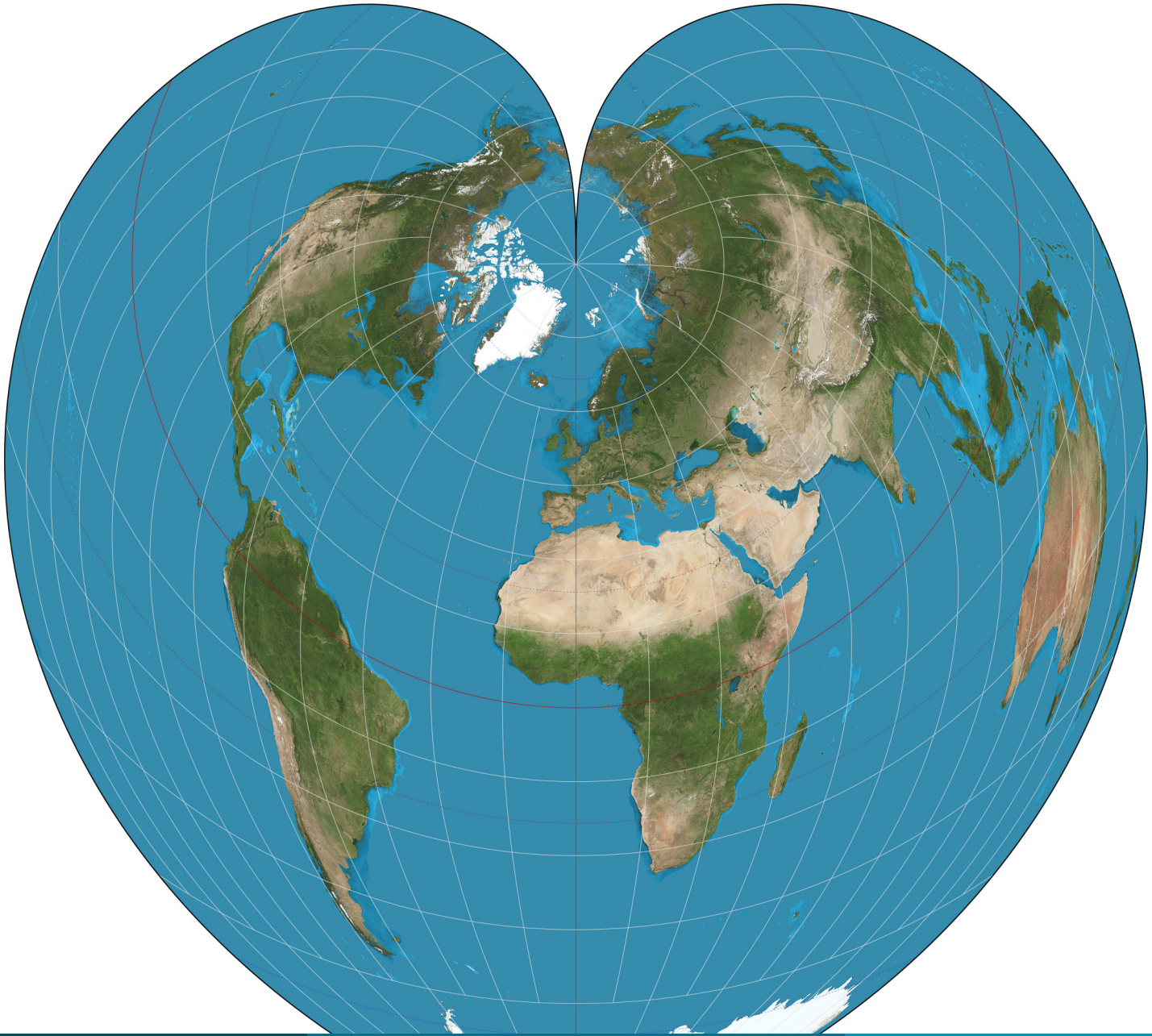
So we: (1) grab the data part of the spatial object, and (2) add coordinates back in (when converting a dataframe to a spatial dataframe, we loose the coordinates as variables)

```
kenya_projects_df <- kenya_projects@data
kenya_projects_df$long <- kenya_projects@coords[,1]
kenya_projects_df$lat <- kenya_projects@coords[,2]
```

Homework : Map of World Bank Projects Near Roads



Thank You!



Outline

Homework Solution

```
kenya_adm0 <- getData('GADM', country='KEN', level=0)

ggplot() +
  geom_polygon(data=kenya_adm0, aes(x=long, y=lat, group=group),
               fill="ivory", color="black") +
  geom_point(data=subset(kenya_projects_df, dist_road < 0.1),
                                        aes(x=long, y=lat,
                                        size=commitments/1000000),
             pch=21, # Adds border around points
             fill="orange",
             color="black") +
  geom_path(data=trunk_roads_kenya, aes(x=long, y=lat,
                                        group=group,
                                        color="Trunk Road")) +
  scale_color_manual(values="brown") +
  labs(color="", size="World Bank\nCommitments\n(Millions)",
       title = "World Bank Projects Near Trunk Roads") +
  coord_quickmap() +
  theme_void() +
  theme(plot.title = element_text(hjust = 0.5)) # Center Title
```

Distance Calculation: More Accurate

To get a more accurate distance we can use the geosphere package. geosphere uses algorithms that take into account the fact that the earth isn't flat, and that distances between decimal degrees changes depending on location.

```
library(geosphere)

# Returns dataframe that includes shortest distance from each point to the road
# NOTE: This may take a few minutes to run, so I'm only running on first 10
#       observations. Use the commented out version to run on all projects
# distance_df <- dist2Line(kenya_projects, trunk_roads_kenya)

distance_df <- dist2Line(kenya_projects[1:10,], trunk_roads_kenya)

head(distance_df, 5)
```

Calculate Areas

To calculate the area of a polygon, we can use:

1. gArea() from the rgeos package. Here, units will be in units of your coordinate system (eg., decimal degrees of meters).
2. areaPolygon from the geosphere package. This calculation takes into account the curvature of the earth and units will be in meters.

```
library(rgeos)
worldmap_area_v1 <- gArea(worldmap, byid=T)

library(geosphere)
worldmap_area_v2 <- areaPolygon(worldmap)
```

Export Spatial Object as Shapefile

Below shows code to export a spatial object as a shapefile

```
writeOGR(obj=roads_trunk_rpj,      # Spatial object
         dsn="~/Desktop",          # Folder to save shapefile
         layer="trunkroads",       # Name of shapefile
         driver="ESRI Shapefile", # Use ESRI shapefile format (.shp)
         overwrite_layer=T)        # Overwrite shapefile with same name
```

Over Function

Let's say we want to add variables to the World Bank projects from the Africa polygon dataset (e.g, the name of the country each project is in). For this, we can use the `over()` function. We can use this to extract data from the polygons to each point.

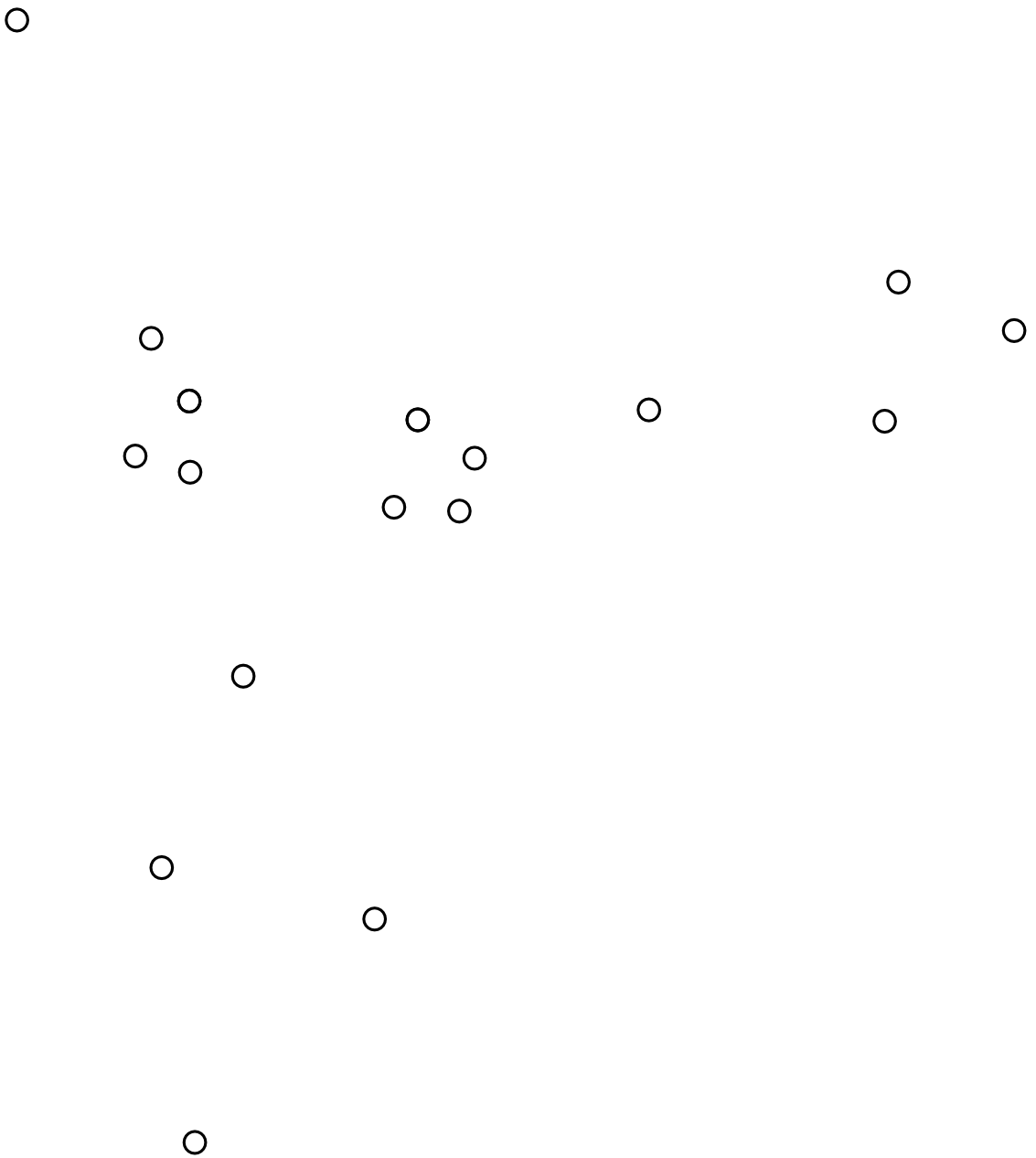
```
# Spatially define wb_projects
coordinates(wb_projects) <- ~longitude+latitude
crs(wb_projects) <- CRS("+init=epsg:4326")

# This yields a dataframe that has the same number of rows as wb_projects,
# but has data from worldmap.
wb_projects_OVER_worldmap <- over(wb_projects, worldmap)

# Add variables from worldmap to wb_projects
wb_projects$POP_EST <- wb_projects_OVER_worldmap$POP_EST
wb_projects$GDP <- wb_projects_OVER_worldmap$GDP_MD_
wb_projects$hppy_sc <- wb_projects_OVER_worldmap$hppy_sc
```

Buffer Spatial Objects

We can create buffers around spatial objects using `gBuffer()` from *rgeos*. This function works with points, lines or polygons. The width is in the units of your coordinate reference system. Consequently, units here are in decimal degrees. However, you could project your data so that units are in meters.

```
library(rgeos)
wb_projects_buff <- gBuffer(wb_projects, width=0.1, byid=T)
plot(wb_projects_buff[1:20,])
```

Buffer Spatial Objects

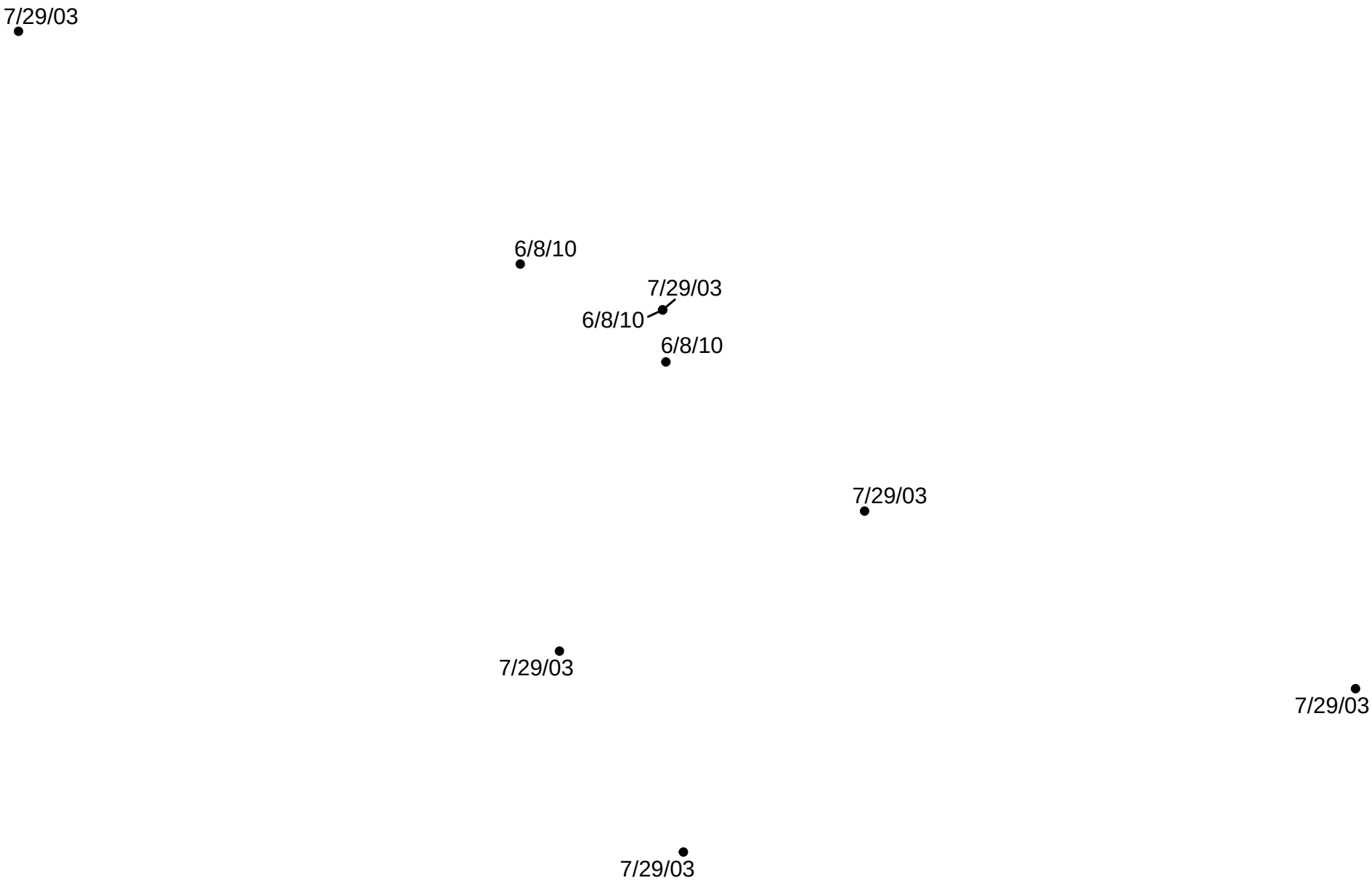


Add Text to Map

We can also add text to maps. Let's say we wanted to add the project start date to the maps– this comes from the start_actual_isodate variable, where data is stored as a string. To map text values, we use geom_text_repel() from the ggrepel package. You can also use geom_label_repel(), which will add a border around the text, or the "non-repel" versions: geom_text() or geom_label.

```
library(ggrepel)

wb_projects <- read.csv(file.path(finalData, "wb_projects.csv"))

ggplot() +
    geom_point(data=wb_projects[1:10,], aes(x=longitude,
                                            y=latitude)) +
    geom_text_repel(data=wb_projects[1:10,], aes(x=longitude,
                                                 y=latitude,
                                                 label=start_actual_isodate)) +
  theme_void()
```

Add Text to Map



Extract Centroid of Spatial Object

To obtain the centroid of a spatial object, such as the centroid of a polygon or line, use `gCentroid` from *rgeos*.

```
worldmap_centroids <- gCentroid(worldmap)
head(worldmap_centroids)
```

Useful Projections

Geographic Coordinate System

The most common geographic coordinate system is the World Geodetic System (WGS84).

```
"+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0"
```

Projected Coordinate Systems

Below are projections for equal area and equal distance. Replace [LATITUDE VALUE] and [LONGITUDE VALUE] with the center of the area that you're working with.
Equal Area: Lambert Azimuthal Equal Area

```
"+proj=laea +lat_0=[LATITUDE VALUE] +lon_0=[LONGITUDE VALUE]"
```

Equal Distance: Azimuthal Equidistant Projection

```
"+proj=aeqd +lat_0=[LATITUDE VALUE] +lon_0=[LONGITUDE VALUE]"
```

ggplot Colors Scale Summary

Below summarizes defining color scales or palettes in ggplot, with either continuous or discrete data. These are elements that will be included in your ggplot code.

Defining own color scale

1. scale_*_gradient(low, high): [For continuous variable] Manually define low/high colors
2. scale_*_gradientn(colors): [For continuous variable] Use a defined list of colors (e.g., c("purple","blue","yellow","white"))
3. scale_*_manual(colors): [For discrete variable] Use a defined list of colors, where the list is equal to the number of unique observations in the discrete variable.

Using pre-defined palettes

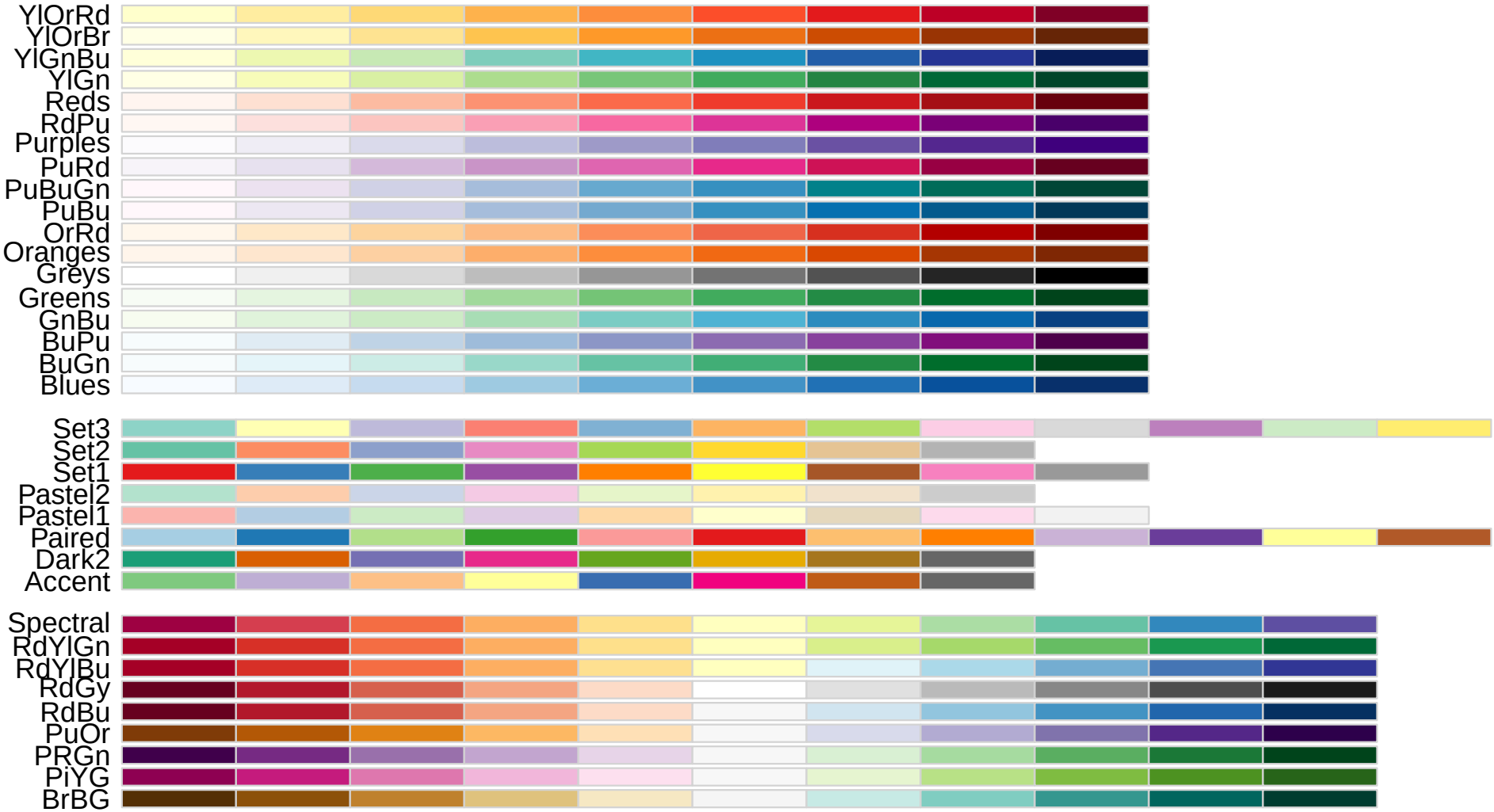
1. scale_*_distiller(palette): [For contiuous variables]
2. scale_*_brewer(palette): [For discrete variables]

Where * is either color or fill. Discrete variables = factor variables.

ggplot Colors Cale Summary

RColorBrewer provides a number of palettes to use.

```
library(RColorBrewer)
display.brewer.all()
```



Useful Resources

- Rspatial provides tutorials for many topics in GIS.
- Nick Eubank Tutorials – another great set of tutorials.
- Spatial Features. There's an entirely different way of defining spatial objects in R that we didn't cover, so see this website for an overview of "spatial features" that treat spatial objects more similarly to dataframes.
- This provides useful links to a bunch of other resources.