

# Spatial Data in R

## R for Stata Users

Luiza Andrade, Leonardo Viotti & Rob Marty

November-December 2018



## 1 GIS Overview

## 2 Load and Plot Spatial Object

## 3 Mapping with ggplot: Polygons

## 4 Mapping with ggplot: Points

## 5 Mapping with ggplot: Lines

## 6 Basemap

## 7 Interactive Map

## 8 Spatial Operations

# Outline

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

# GIS Overview

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.

Example Attributes for Point Data

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

Example Attributes for Line Data

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

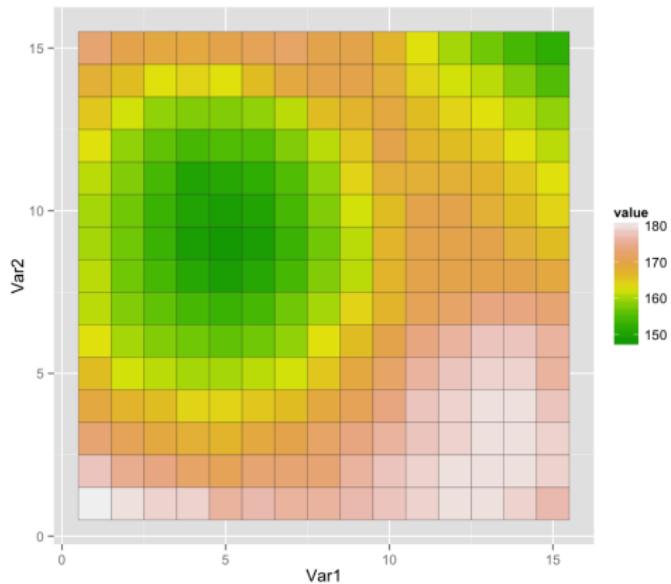
Example Attributes for Polygon Data

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

Next

# 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
  - 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. This distorts the earth in some way (shape, area, distance or direction)
  - Using to the World Mercator projection, the World Bank is located 4680364.64 north and -8576320.73 east.



Geographic (3D)



Projected (2D)

# Outline

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

## Load a Shapefile

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.

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

# Sometimes the above doesn't work. In this case, try setting
# directory to the folder where the shpefile is:
# setwd(finalData)
# worldmap <- readOGR(dsn=". ", layer="worldmap")
```

# Plot Shapefile

```
plot(worldmap)
```



# Plot Shapefile

We can subset the spatialdataframe just like any other dataframe.

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



# Spatial Data Frame

Here, we've loaded a `SpatialPolygonsDataFrame`. (There's equivalent object types for points and lines). A spatial dataframe is comprised of a list that includes different parts of the shapefile. We access these parts using the `@` symbol.

```
# Coordinate Reference System  
worldmap@proj4string
```

```
# Dataframe  
worldmap@data
```

# Outline

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

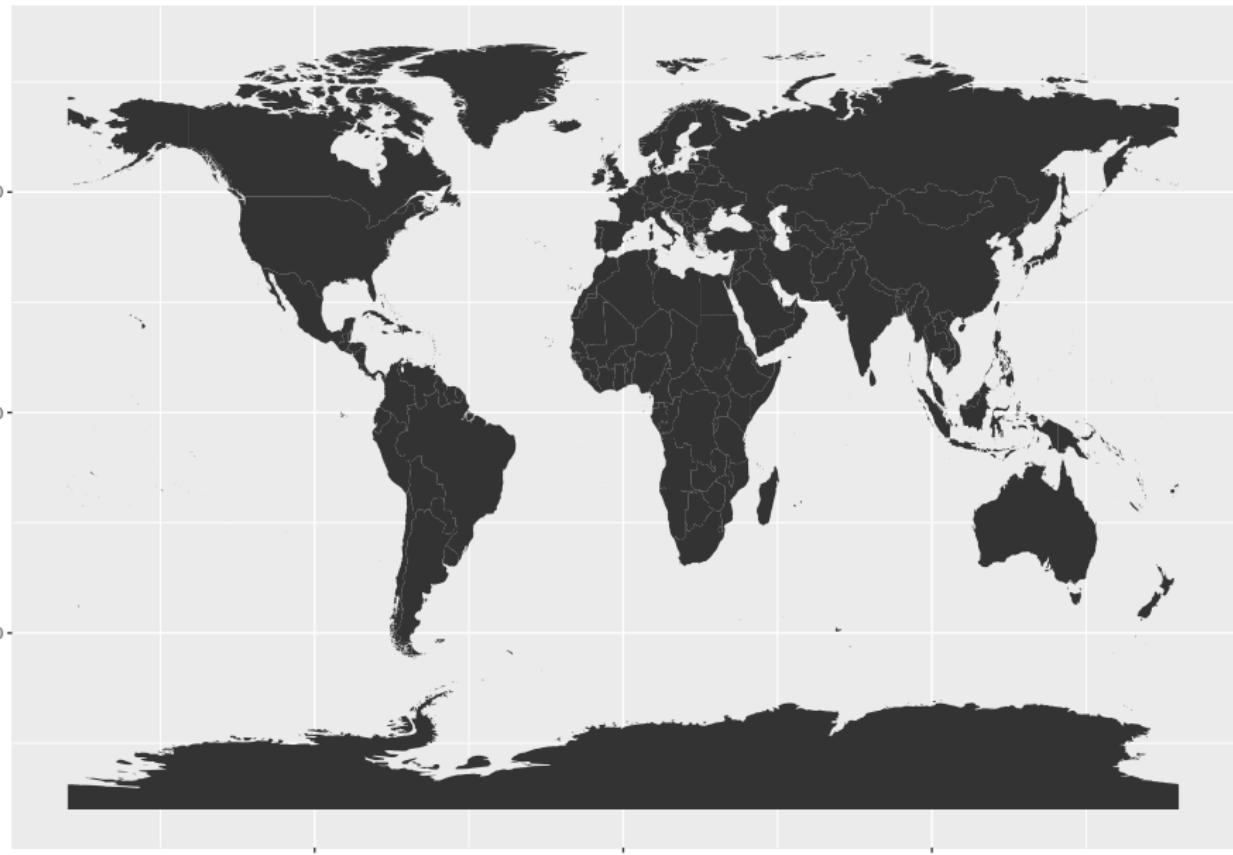
# Making a Map with ggplot

## Basics of mapping with ggplot

- ➊ ggplot has methods for mapping spatial objects. We use geom\_polygon for polygons and geom\_path for lines.
- ➋ 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.
- ➌ 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)
```

```
##      long     lat order  hole piece group id
## 1 -69.89914 12.45201     1 FALSE     1   0.1  0
## 2 -69.89568 12.42301     2 FALSE     1   0.1  0
## 3 -69.94216 12.43852     3 FALSE     1   0.1  0
## 4 -70.00415 12.50050     4 FALSE     1   0.1  0
## 5 -70.06613 12.54699     5 FALSE     1   0.1  0
## 6 -70.05086 12.59709     6 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
```

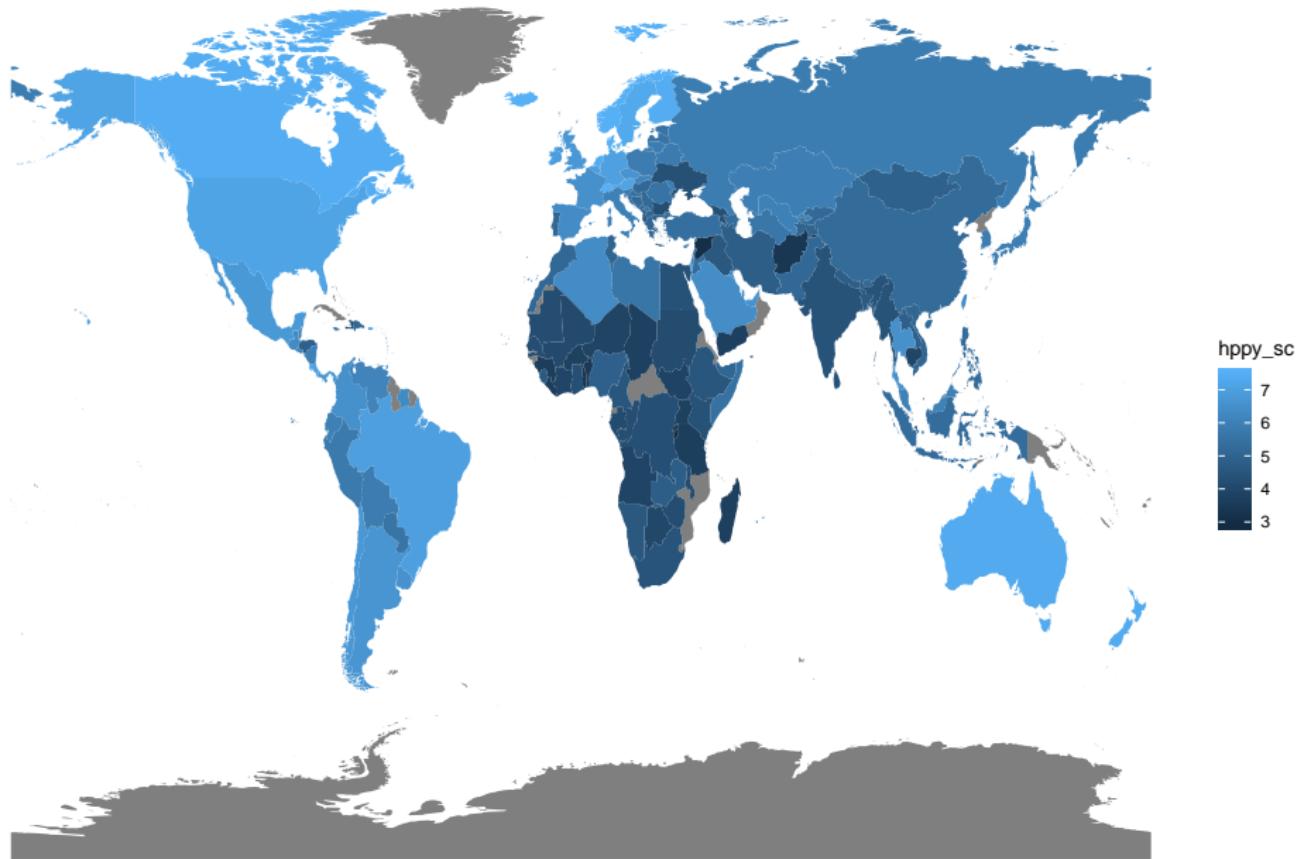
# Making a Map with ggplot

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

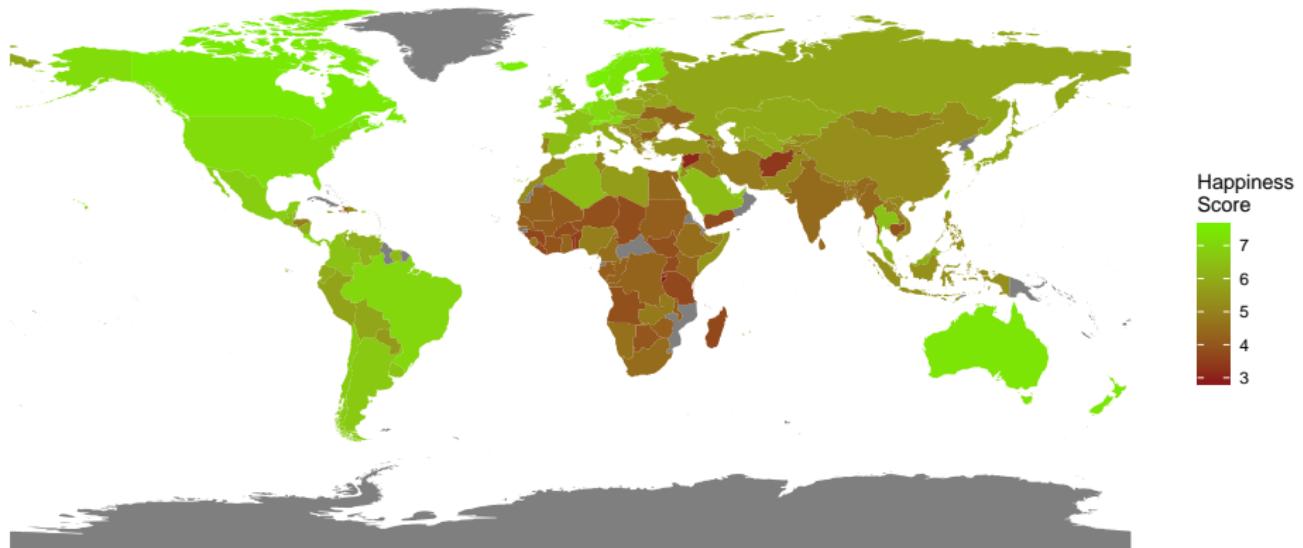


# Making a Prettier Map with ggplot

```
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="Population") +
  scale_colour_gradient(low = "firebrick4",
                        high = "chartreuse2")
```

# Making a Prettier Map with ggplot



# Map of Africa

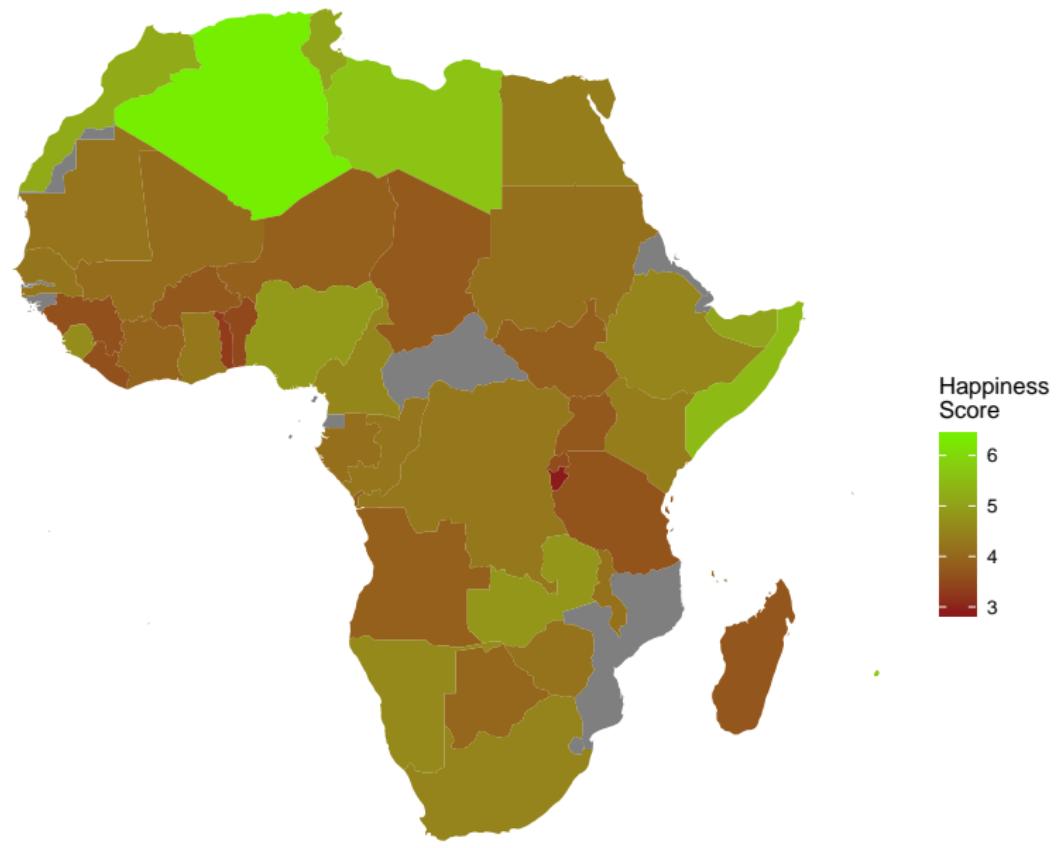
## Exercise 1

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

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

# 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)

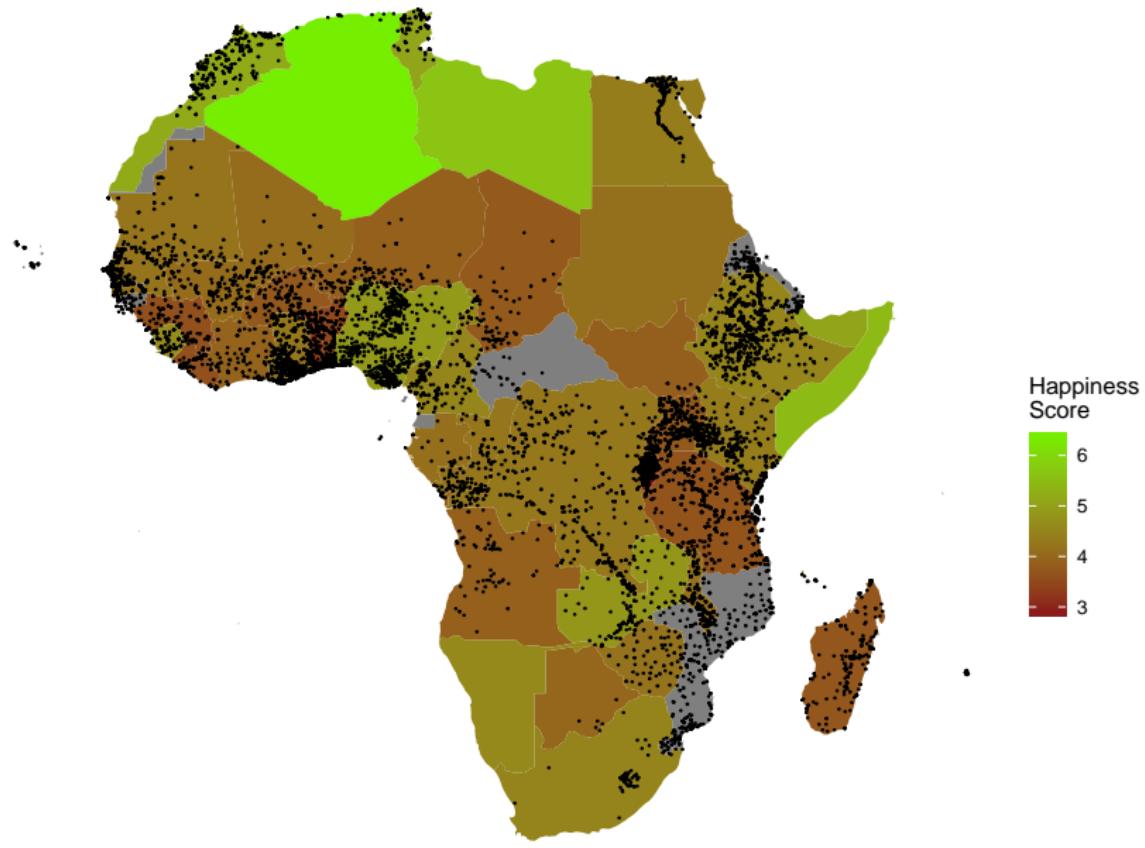
## [1] "project_location_id"   "place_name"           "latitude"
## [4] "longitude"             "project_title"        "start_actual_isodate"
## [7] "end_actual_isodate"    "recipients"          "ad_sector_names"
## [10] "commitments"          "disbursements"
```

## 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.

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

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

# Add trunk roads to map

## Exercise 2

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\nnScore") +
  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 can use the `spTransform()` function to reproject the data.

The below code shows two different ways of specifying the projection. The first way writes out the full projection. The second way uses a shorter way of writing the projection using an *epsg code*. EPSG codes reference commonly used projections.

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

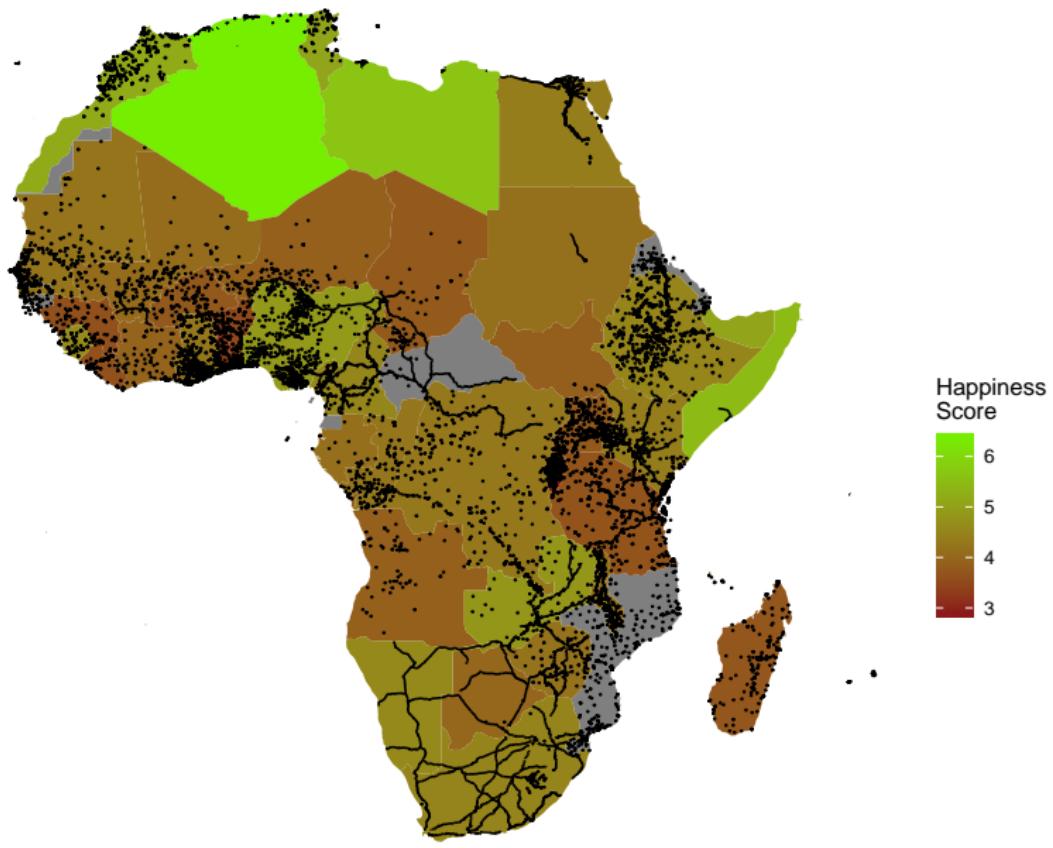
# Way 2
trunk_roads_reproj <- spTransform(trunk_roads, CRS("+init=epsg:4326"))
```

## 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_reproj, 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

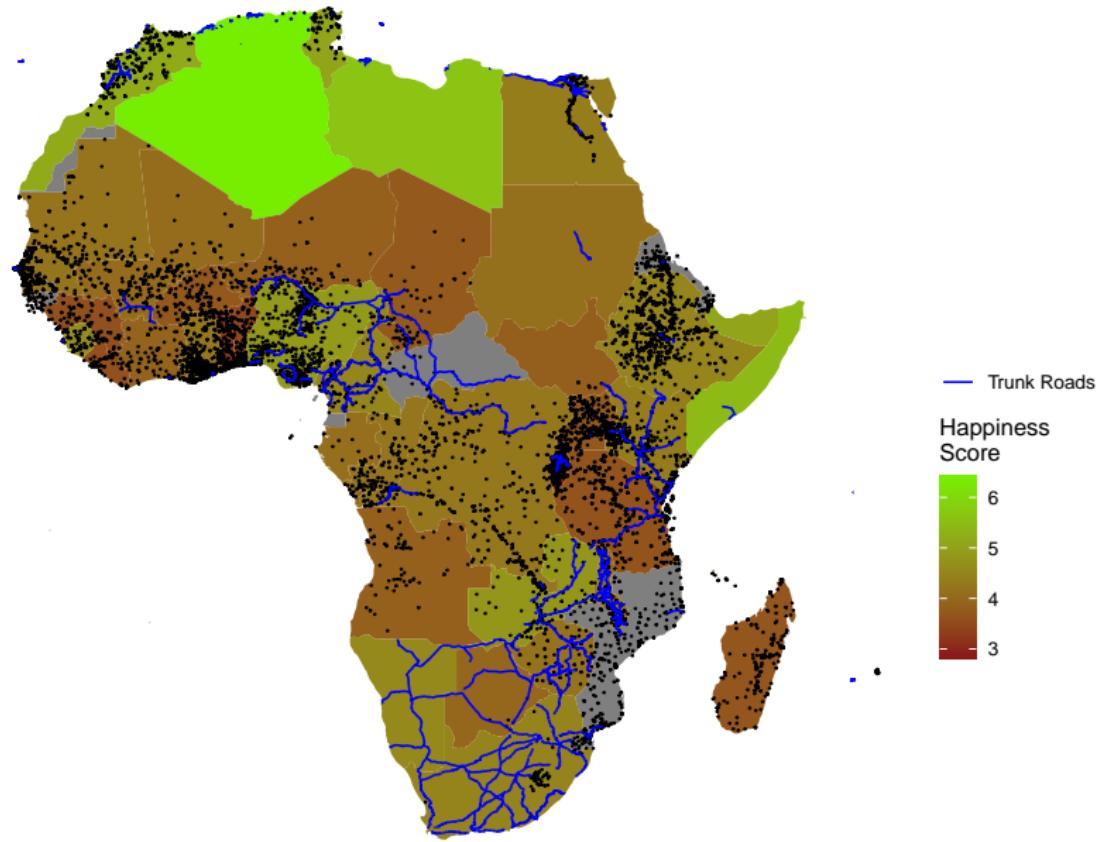


# Add Roads to Legend

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_reproj, aes(x=long, y=lat,
                                         group=group,
                                         color="Trunk Roads")) +
  theme_void() +
  coord_quickmap() + # Removes distortions
  labs(fill="Happiness\nScore",
       color="") +
  scale_fill_gradient(low = "firebrick4",
                      high = "chartreuse2") +
  scale_color_manual(values=c("blue"))
```

# Add Roads to Legend



# Outline

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

## Basemap

We can also plot our spatial data on basemaps. One of the packages that enables us 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 = 12,
                               maptype = "toner-lite")

ggmap(nairobi_map)
```

# Basemap

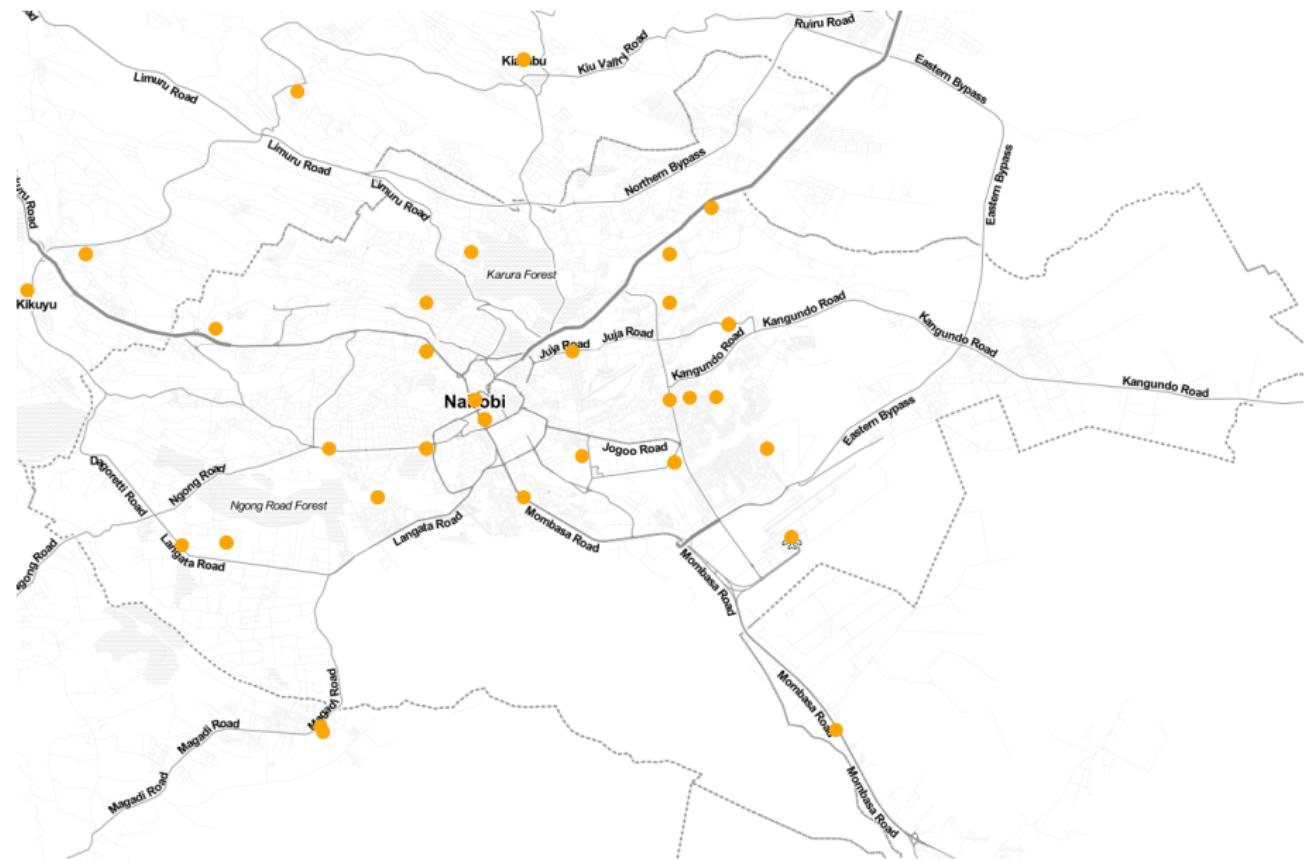


# Basemap

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

```
ggmap(nairobi_map) +
  geom_point(data=wb_projects, aes(x=longitude,
                                    y=latitude),
             color="orange",
             size=3)
```

# Basemap



# Outline

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

# Convert dataframe to spatialdataframe

Before making an interactive map, let's convert our dataframe of World Bank projects to a spatialdataframe. We don't necessarily need to do this, but it'll make our code simpler 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
coordinates(wb_projects) <- ~longitude+latitude

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

# +init=epsg:4326 is short for:
# +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

# Convert dataframe to spatialdataframe

We can directly plot spatialdataframes

```
plot(wb_projects)
```



# 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.

To make things simpler, subset the data to projects in Kenya. Leaflet works similarly to ggplot where you sequentially define different elements.

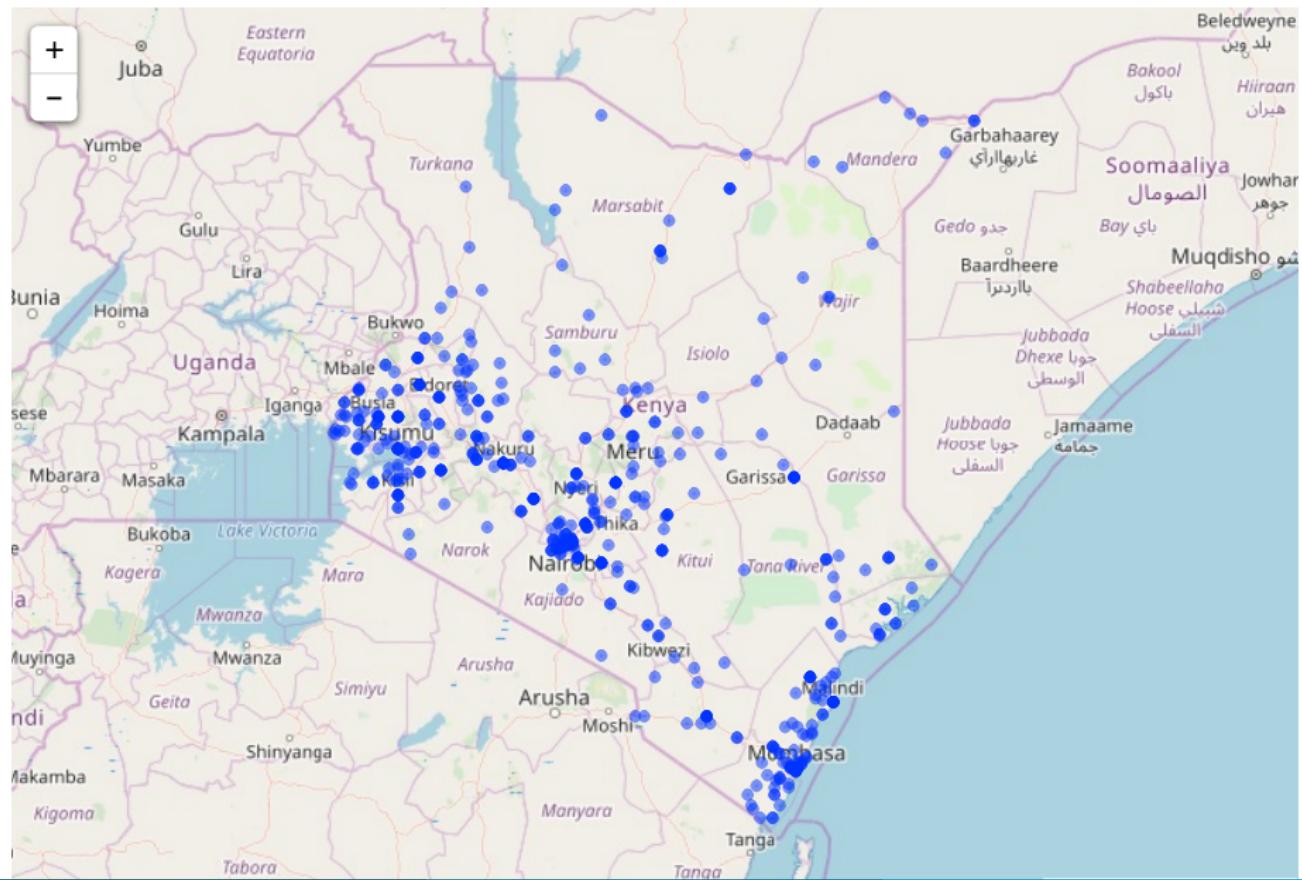
However, instead of using + between elements, we use a pipe: %>%

```
library(leaflet)

kenya_projects <- subset(wb_projects, recipients == "Kenya")

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

# Interactive Map

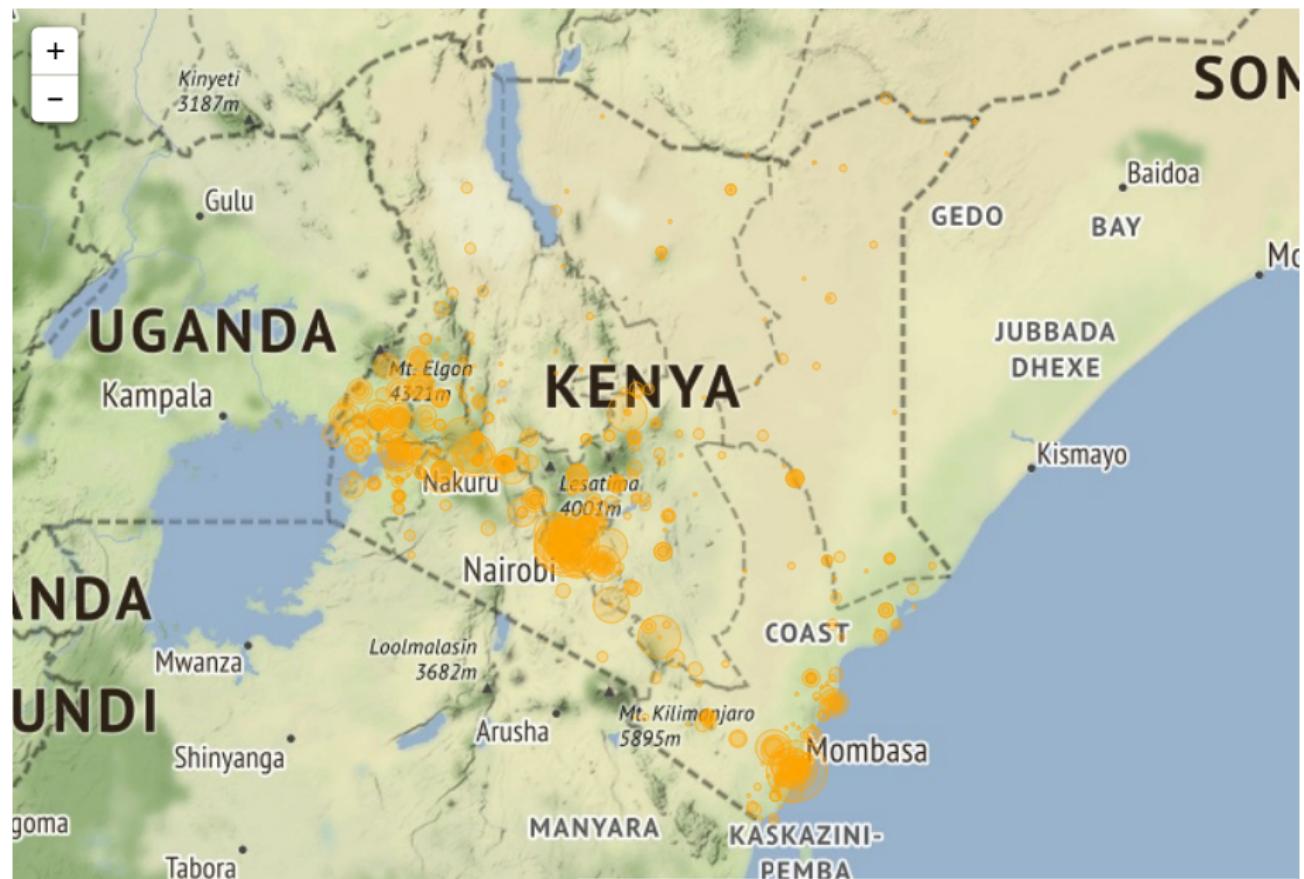


# Better Interactive Map

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 3

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.

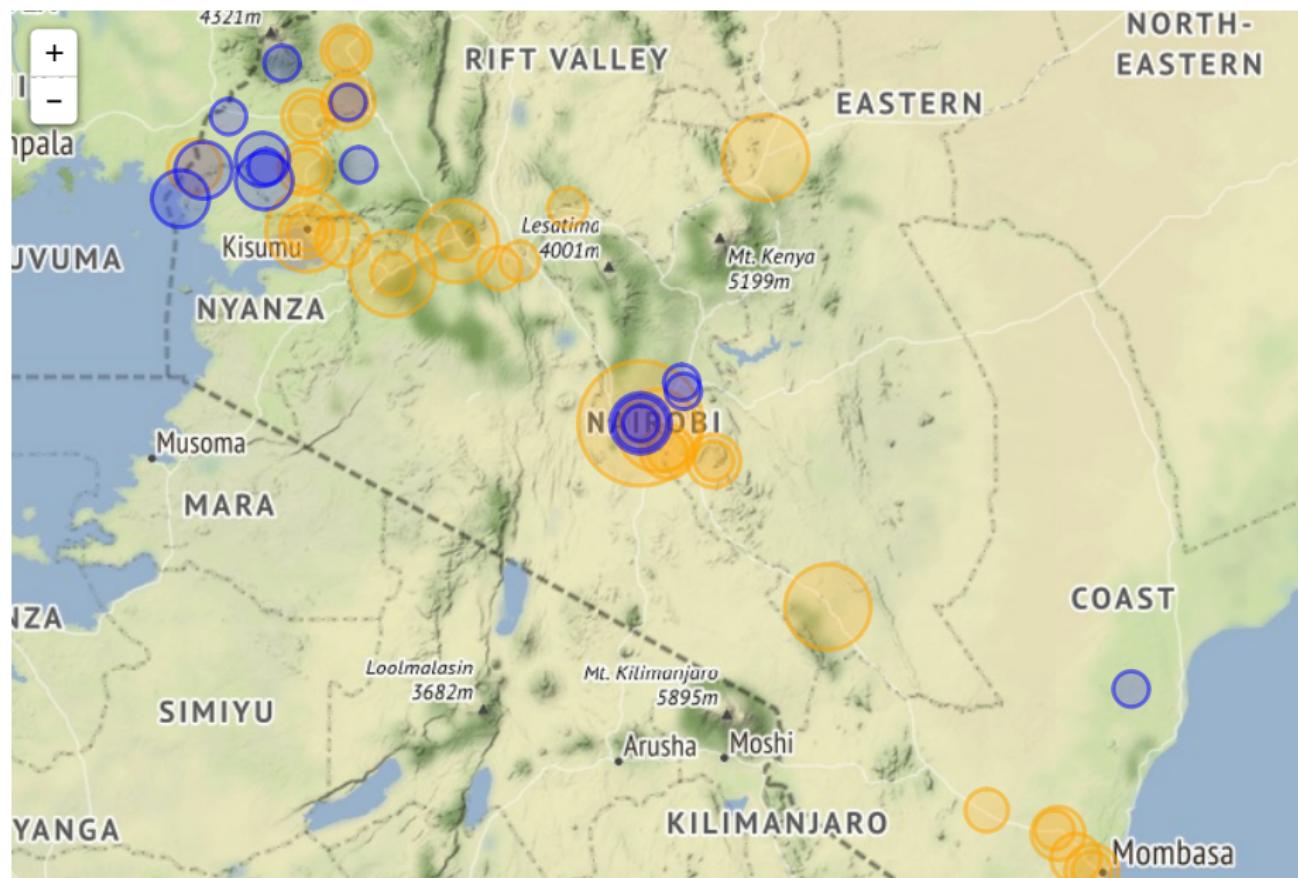
Hints:

- ① Use the "ad\_sector\_names" variable and the subset function to subset the dataset. For example: `subset(kenya_projects, ad_sector_names == "Transport and storage")`
- 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 sto  
                        popup = ~project_title,  
                        radius = ~sqrt(commitments)*5,  
                        weight = 2.5,  
                        color = "orange") %>%  
  addCircles(data=subset(kenya_projects,  
                        ad_sector_names == "Water supply and  
                        popup = ~project_title,  
                        radius = ~sqrt(commitments)*5,  
                        weight = 2.5,  
                        color = "blue") %>%  
  addProviderTiles(providers$Stamen.Terrain)
```

## Two layers on map



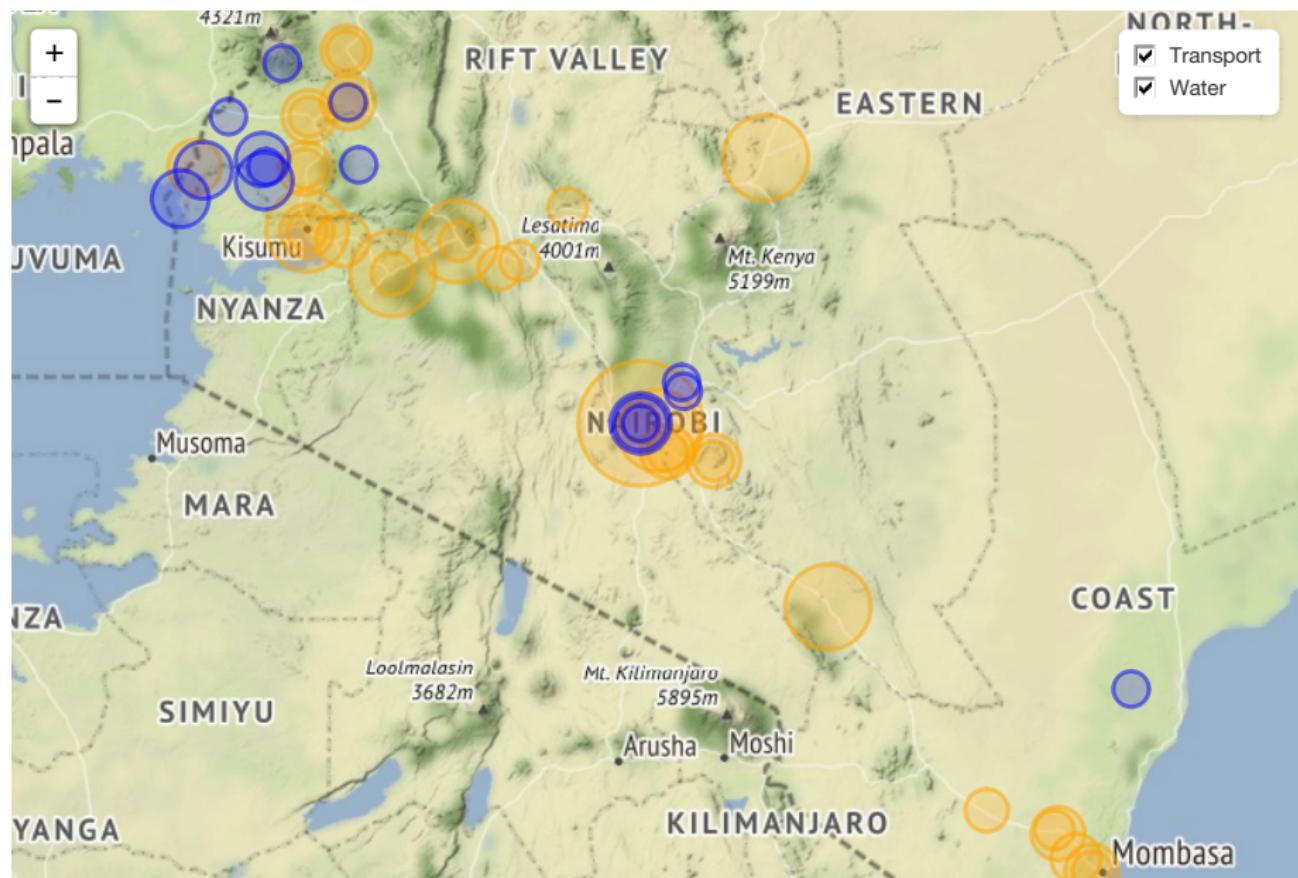
# Add Layers to Interactive Map

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

- ➊ When defining the layer (eg, in `addCircles`), add "group=" and give the group a name.
- ➋ 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 to Interactive Map



# Outline

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

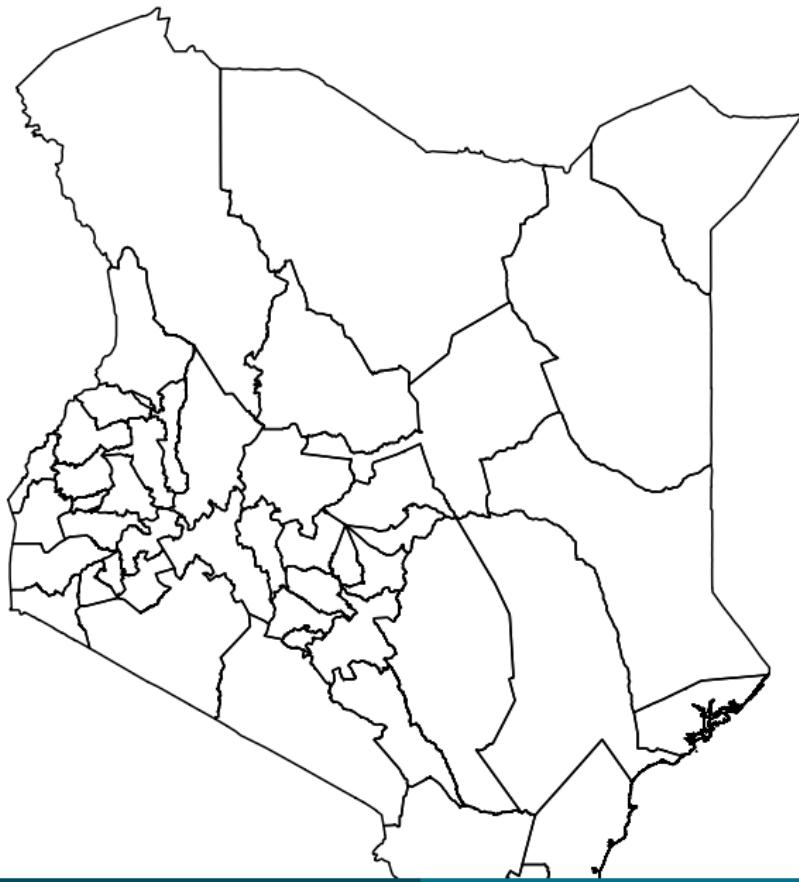
# Grabbing Administrative Data

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 `crop()` from the *sp* package.

```
trunk_roads_kenya <- crop(trunk_roads, extent(ken_adm1))

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)
```

# Outline

- 1 GIS Overview
- 2 Load and Plot Spatial Object
- 3 Mapping with ggplot: Polygons
- 4 Mapping with ggplot: Points
- 5 Mapping with ggplot: Lines
- 6 Basemap
- 7 Interactive Map
- 8 Spatial Operations
- 9 Appendix

## 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

We have World Bank projects and the Africa polygon dataset. Let's say we want to add variables to the World Bank projects from the Africa polygon dataset.

For this, we can use the `over()` function. We can use this to extract data from the polygons to each point.

```
# 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)
```

```
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
```

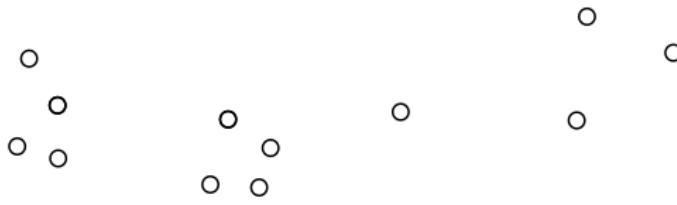
# Buffer Spatial Objects

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

o



o



o

# 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

7/29/03

6/8/10

7/29/03  
6/8/10  
6/8/10

7/29/03

7/29/03

7/29/03

7/29/03

7/29/03

# 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

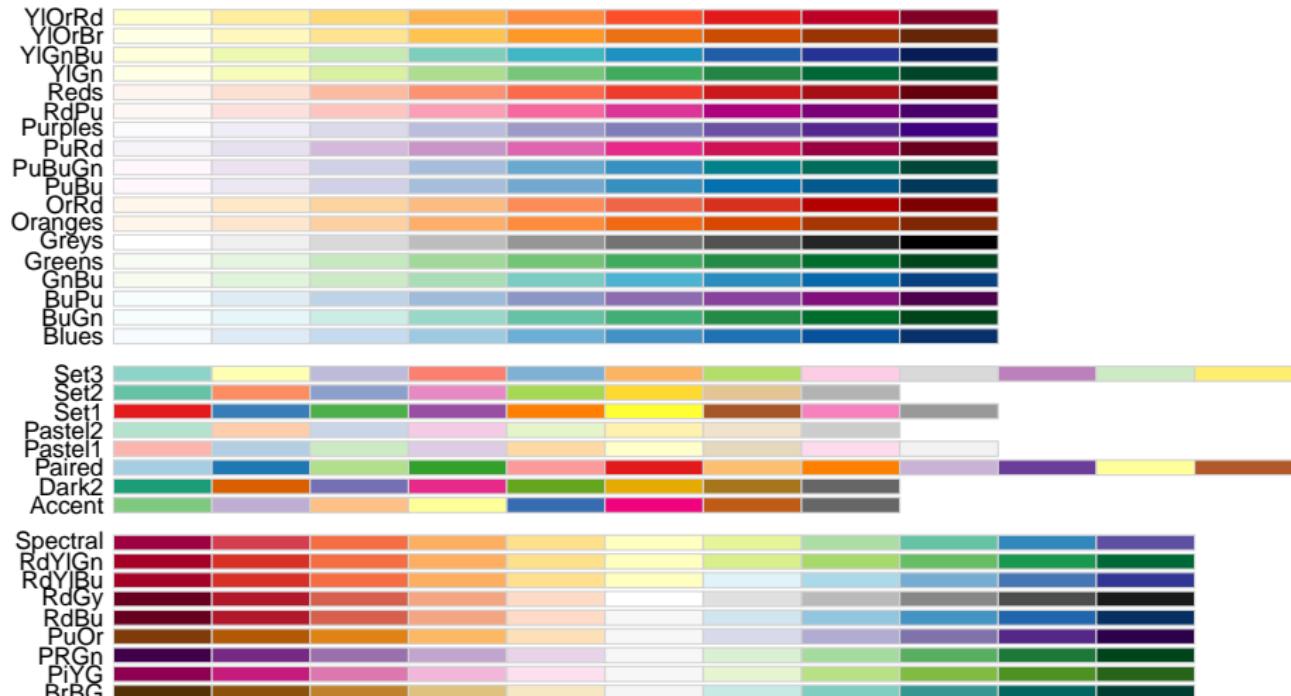
- ① `scale_*_distiller(palette)`: [For continuous variables]
- ② `scale_*_brewer(palette)`: [For discrete variables]

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

- ① - [Rspatial](<http://rspatial.org/>) provides tutorials for many topics in GIS.
- ② - [Nick Eubank Tutorials](<http://www.nickeubank.com/gis-in-r/>) – another great set of tutorials.
- ③ - [Spatial Features](<https://r-spatial.github.io/sf/articles/sf1.html>). 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.
- ④ - [Here](<https://gist.github.com/sboysel/fc661f26ef51eae6377b>) Provides useful links to a bunch of other resources.