

Spatial Data

R for Advanced Stata Users

Luiza Andrade, Rob Marty, Rony Rodriguez-Ramirez, Luis Eduardo San Martin, Leonardo Viotti
DIME | The World Bank

13 April 2021



Table of contents

1. Loading and exploring spatial data
2. Spatial data structure
3. Simple features
4. Visualizing spatial data
5. Why projections matters
6. Basic spatial operations

Setting the stage

Setting the stage

Install new packages

```
install.packages(c("sf",  
                  "rworldmap",  
                  "ggmap",  
                  "wesanderson"),  
                dependencies = TRUE)
```

And load them

```
library(here)  
library(tidyverse)  
library(sf)  
library(rworldmap)  
library(ggmap)  
library(wesanderson)
```

Setting the stage

Datasets we will use today

```
# Load data
whr_panel  <- read_rds(here("DataWork",
                           "DataSets",
                           "Final",
                           "whr_panel.RDS"))

wb_projects <- read_csv(here("DataWork",
                             "DataSets",
                             "Final",
                             "wb_projects.csv"))
```

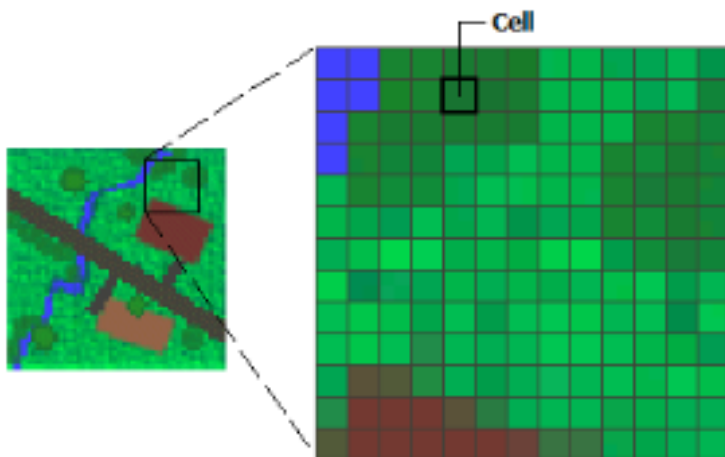
Introduction

Introduction

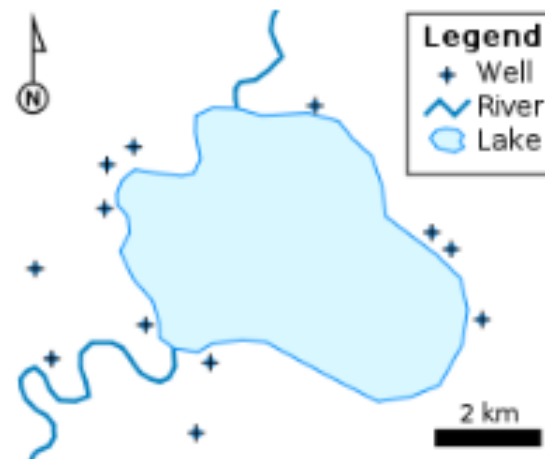
There are two main types of spatial data: vector and raster data.

- **Raster:** spatially-referenced grids where each cell has one value.
- **Vectors or shapefiles:** spatial-referenced objects consisting of points, lines and polygons. These shapes are attached to a dataframe, where each row corresponds to a different spatial element.

```
knitr::include_graphics("img/raster.png")
```



```
knitr::include_graphics("img/shapefile.png")
```



Introduction

- This session could be a whole course on its own, but we only have an hour and half.
- To narrow our subject, we will focus on only one type of spatial data, shapefiles.
- This is the most common type of spatial data that non-GIS experts will encounter in their work.
- We will focus mostly on how to visualize spatial data, although we will also cover some simple geometry operations.
- We will use the `sf` package, which is the tidyverse-compatible package for geospatial data in R.
- If you want to know more about geospatial data in R, we recommend the book <https://geocompr.robinlovelace.net/>, by Robin Lovelace, Jakub Nowosad, and Jannes Muenchow.

Loading a shapefile with sf

The first thing we will do in this session is to recreate this data set:

```
worldmap <-  
  st_read(here("DataWork",  
              "DataSets",  
              "Final",  
              "worldmap.shp"))
```

```
## Reading layer `worldmap' from data source `D:\Documents\RA Jobs\DIME\trainings\dime-r-training\DataWork\DataSets\Final'  
## Simple feature collection with 244 features and 15 fields  
## Geometry type: MULTIPOLYGON  
## Dimension:      XY  
## Bounding box:   xmin: -180 ymin: -89.9989 xmax: 180 ymax: 83.5996  
## Geodetic CRS:   WGS 84
```

```
plot(worldmap)
```

Exploring the data

Creating a polygon shapefile

Loading spatial data

Load a built-in map using the `rworldmap` package

```
worldmap <- getMap(resolution="low")
```

Look at the data structure

```
View(worldmap)
```

This object is a list with three main components:

- Data
- Polygons
- Projection

Spatial data structure: the data

- The data portion of a shapefile is a data frame like any other in R.
- To access it, we need to refer to the data element in our list by typing `objectname@data`.

Exercise

Explore the dataset in `worldmap` using the functions `head()` and `names()`.

```
names()
```

Spatial data structure: the data

- The data portion of a shapefile is a data frame like any other in R.
- To access it, we need to refer to the data element in our list by typing `objectname@data`.

Exercise

Explore the dataset in `worldmap` using the functions `head()` and `names()`.

```
names(worldmap@data)
```

```
## [1] "ScaleRank"    "LabelRank"    "FeatureCla"    "SOVEREIGNT"    "SOV_A3"
## [6] "ADM0_DIF"     "LEVEL"        "TYPE"          "ADMIN"         "ADM0_A3"
## [11] "GEOU_DIF"     "GEOUNIT"      "GU_A3"         "SU_DIF"        "SUBUNIT"
## [16] "SU_A3"        "NAME"         "ABBREV"        "POSTAL"        "NAME_FORMA"
## [21] "TERR_"        "NAME_SORT"    "MAP_COLOR"     "POP_EST"       "GDP_MD_EST"
## [26] "FIPS_10_"     "ISO_A2"       "ISO_A3"        "ISO_N3"        "ISO3"
## [31] "LON"          "LAT"          "ISO3.1"        "ADMIN.1"       "REGION"
## [36] "continent"    "GEO3major"    "GEO3"          "IMAGE24"       "GLOCAF"
## [41] "Stern"        "SRESmajor"    "SRES"          "GBD"           "AVOIDnumeric"
```

Spatial data structure: the data

```
head(worldmap@data)
```

```
##      ScaleRank LabelRank      FeatureCla      SOVEREIGNT SOV_A3 ADM0_DIF LEVEL
## 1          3          3 Admin-0 countries Netherlands NL1      1      2
## 2          1          1 Admin-0 countries Afghanistan AFG      0      2
## 3          1          1 Admin-0 countries      Angola AGO      0      2
## 4          1          1 Admin-0 countries United Kingdom GB1      1      2
## 5          1          1 Admin-0 countries      Albania ALB      0      2
## 6          3          3 Admin-0 countries      Finland FI1      1      2
##
##      TYPE      ADMIN ADM0_A3 GEOU_DIF      GEOUNIT GU_A3 SU_DIF
## 1      Country      Aruba ABW      0      Aruba ABW      0
## 2 Sovereign country Afghanistan AFG      0 Afghanistan AFG      0
## 3 Sovereign country      Angola AGO      0      Angola AGO      0
## 4      Dependency      Anguilla AIA      0      Anguilla AIA      0
## 5 Sovereign country      Albania ALB      0      Albania ALB      0
## 6      Country      Aland ALD      0      Aland ALD      0
##
##      SUBUNIT SU_A3      NAME ABBREV POSTAL      NAME_FORMA
## 1      Aruba ABW      Aruba Aruba AW      <NA>
## 2 Afghanistan AFG Afghanistan Afg. AF Islamic State of Afghanistan
## 3      Angola AGO      Angola Ang. AO      Republic of Angola
## 4      Anguilla AIA      Anguilla Ang. AI      <NA>
## 5      Albania ALB      Albania Alb. AL      Republic of Albania
## 6      Aland ALD      Aland Aland AI      Eland Islands
##
##      TERR_      NAME_SORT MAP_COLOR POP_EST GDP_MD_EST FIPS_10_ ISO_A2 ISO_A3 ISO_N3
## 1 Neth.      Aruba      9 103065 2258.0 <NA> AW ABW 533
## 2 <NA> Afghanistan      7 28400000 22270.0 <NA> AF AFG 4
## 3 <NA>      Angola      1 12799293 110300.0 <NA> AO AGO 24
## 4 U.K.      Anguilla      3 14436 108.9 <NA> AI AIA 660
## 5 <NA>      Albania      6 3639453 21810.0 <NA> AL ALB 8
## 6 Fin.      Aland      6 27153 NA <NA> -99 ALA 248
##
##      ISO3      LON      LAT ISO3.1      ADMIN.1      REGION      continent
```

Spatial data structure: the data

We can treat the data in a geospatial object just like any other data frame

Exercise

Use the command `select()` from tidyverse's `dplyr` package to keep only the following variables in the `worldmap` data: `ADMIN`, `REGION`, `continent`, `POP_EST`, `GDP_MD_EST`.

```
worldmap@data <-  
  worldmap@data %>%  
  select(ADMIN, REGION, continent, POP_EST, GDP_MD_EST)
```


Spatial data structure: the data

We can treat the data in a geospatial object just like any other data frame

Exercise

Explore the `worldmap` data using `summary()`.

```
summary()
```

Spatial data structure: the data

We can treat the data in a geospatial object just like any other data frame

Exercise

Explore the `worldmap` data using `summary()`.

```
summary(worldmap@data)
```

```
##           ADMIN           REGION           continent
## Afghanistan : 1 Europe :65 Africa : 57
## Aland : 1 Africa :57 Antarctica : 1
## Albania : 1 Asia :45 Australia : 26
## Algeria : 1 South America:44 Eurasia :110
## American Samoa: 1 Australia :26 North America: 3
## Andorra : 1 (Other) : 4 South America: 44
## (Other) :238 NA's : 3 NA's : 3
## POP_EST GDP_MD_EST
## Min. :0.000e+00 Min. : 0
## 1st Qu.:2.507e+05 1st Qu.: 2329
## Median :4.489e+06 Median : 20775
## Mean :2.793e+07 Mean : 292888
## 3rd Qu.:1.557e+07 3rd Qu.: 116050
## Max. :1.339e+09 Max. :14260000
## NA's :1 NA's :6
```

Spatial data structure: polygons

```
plot(worldmap)
```

Spatial data structure: projection

```
worldmap@proj4string
```

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



[Click here to see why Josh and CJ are confused](#)

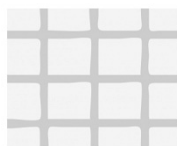
Spatial data structure: projection

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 meters north and -8576320.73 meters east.



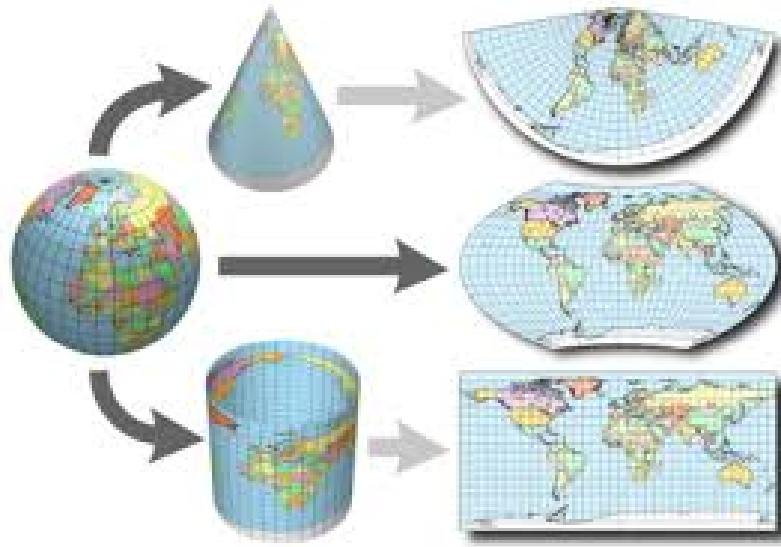
Geographic (3D)



Projected (2D)

Spatial data structure: projection

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



Projections are also the main source of error when plotting spatial data: if two spatial objects have different reference systems, plotting them together will result in quite weird maps

Projecting spatial data in sf

Creating an sf object

The `sf` package deals with spatial data in a special way: it allows us to treat it as a regular data frame, while maintaining its spatial component.

`st_as_sf(x)`

Transforms objects into `sf` objects

- `...`: the object to be transformed

Exercise

Turn the `worldmap` object into an `sf` object.

```
st_as_sf()
```


Creating an sf object

The `sf` package deals with spatial data in a special way: it allows us to treat it as a regular data frame, while maintaining its spatial component.

`st_as_sf(x)`

Transforms objects into `sf` objects

- `...`: the object to be transformed

Exercise

Turn the `worldmap` object into an `sf` object.

```
worldmap <-  
  st_as_sf(worldmap)
```

Creating an sf object

```
class(worldmap)
```

```
## [1] "sf"          "data.frame"
```

```
summary(worldmap)
```

```
##           ADMIN           REGION           continent
## Afghanistan : 1 Europe :65 Africa : 57
## Aland : 1 Africa :57 Antarctica : 1
## Albania : 1 Asia :45 Australia : 26
## Algeria : 1 South America:44 Eurasia :110
## American Samoa: 1 Australia :26 North America: 3
## Andorra : 1 (Other) : 4 South America: 44
## (Other) :238 NA's : 3 NA's : 3
## POP_EST GDP_MD_EST geometry
## Min. :0.000e+00 Min. : 0 MULTIPOLYGON :244
## 1st Qu.:2.507e+05 1st Qu.: 2329 epsg:NA : 0
## Median :4.489e+06 Median : 20775 +proj=long...: 0
## Mean :2.793e+07 Mean : 292888
## 3rd Qu.:1.557e+07 3rd Qu.: 116050
```

Creating an sf object

```
plot(worldmap)
```

Projections in sf

Here are two useful `sf` commands:

`st_crs(x)`

Displays the current projection of an `sf` object

`st_transform(x, crs)`

Projects object **x** using projection **crs**

Projections in sf

Exercise

Create two objects derived from `worldmap`, but with different projections:

- Use the Mollweid projection (`crs = "+proj=moll"`) to create `worldmap_moll`
- Use the Mercator projection (`crs = "EPSG:3857"`) to create `worldmap_mercator`

```
worldmap_moll <-  
worldmap_mercator <-
```

Projections in sf

Exercise

Create two objects derived from `worldmap`, but with different projections:

- Use the Mollweid projection (`crs = "+proj=moll"`) to create `worldmap_moll`
- Use the Mercator projection (`crs = "EPSG:3857"`) to create `worldmap_mercator`

```
worldmap_moll <-  
  worldmap %>%  
  st_transform("+proj=moll")  
  
worldmap_mercator <-  
  worldmap %>%  
  st_transform("EPSG:3857")
```

Projections in sf

```
worldmap_moll %>%  
  select(REGION) %>%  
  plot()  
  
worldmap_mercator %>%  
  select(REGION) %>%  
  filter(REGION != "Antarctica") %>%  
  plot()
```

Why did I use `select` above?

Projections in sf

Projections in sf

Visualizing polygons

Combining non-spatial and spatial data

- To create the `worldmap` shapefile that you have in your final data folder, we combined the data in `whr_panel` and the polygon in `worldmap`. Given what we have seen, this is as simple as joining two data sets

We need to make a few adjustment to the data so the join works:

```
worldmap <-  
  worldmap %>%  
  mutate(country = as.character(ADMIN),  
    country = str_replace_all(country, "United States of America", "United States"),  
    country = str_replace_all(country, "Northern Cyprus", "North Cyprus"),  
    country = str_replace_all(country, "Hong Kong S.A.R.", "Hong Kong"),  
    country = str_replace_all(country, "Republic of Serbia", "Serbia"),  
    country = str_replace_all(country, "Somaliland", "Somaliland Region"),  
    country = str_replace_all(country, "West Bank", "Palestinian Territories"),  
    country = str_replace_all(country, "Democratic Republic of the Congo", "Congo (Kinshasa)",  
    country = str_replace_all(country, "Republic of the Congo", "Congo (Brazzaville)",  
    country = str_replace_all(country, "United Republic of Tanzania", "Tanzania"))  
  
whr_panel <-  
  whr_panel %>%  
  filter(year == 2015)
```

Then we can join them:

```
worldmap <-  
  worldmap %>%  
  left_join(whr_panel)
```

Visualizing polygons

- `ggplot` has a special geometry for `sf`: `geom_sf`
- `geom_sf` takes into account the spatial features to maintain proportions

```
ggplot(worldmap) +  
  geom_sf()
```

Visualizing polygons

Visualizing polygons

Exercise

Use the `fill` aesthetics inside `geom_sf` to show the happiness score in the map.

```
ggplot(worldmap) +  
  geom_sf()
```

Visualizing polygons

Exercise

Use the `fill` aesthetics inside `geom_sf` to show the happiness score in the map.

```
ggplot(worldmap) +  
  geom_sf(aes(fill = happiness_score))
```

Visualizing polygons

Visualizing polygons

Exercise

Use the `fill` aesthetics inside `geom_sf` to show the happiness score in the map.

```
ggplot(worldmap %>%  
  filter(REGION != "Antarctica")) +  
  geom_sf(aes(fill = happiness_score)) +  
  labs(fill = "Happiness Score") +  
  scale_fill_gradient(low = "blue",  
                     high = "yellow") +  
  theme_void() +  
  theme(legend.position = "top")
```

Visualizing polygons

Visualizing points

Visualizing points

When you have GPS coordinates, using `ggplot` to map them is very easy: use `geom_points` and link `x` to the longitude variable and `y` to the latitude variable.

Exercise

Create a scatter plot of the projects in the `wb_projects` dataset.

```
ggplot() +  
  geom_point(aes(x = ,  
                 y = ))
```

Visualizing points

When you have GPS coordinates, using `ggplot` to map them is very easy: use `geom_points` and link `x` to the longitude variable and `y` to the latitude variable.

Exercise

Create a scatter plot of the projects in the `wb_projects` dataset.

```
ggplot() +  
  geom_point(data = wb_projects,  
            aes(x = longitude,  
                y = latitude))
```

Visualizing points

Visualizing points

```
ggplot() +  
  geom_point(data = wb_projects,  
            aes(x=longitude,  
                y=latitude),  
            size = .1) + # Smaller dots  
  coord_quickmap() + # Correct distortion  
  theme_void() # Clean background
```

Visualizing points

Adding a basemap

The package `ggmap` allows us to layers as a basemap. The code is the same as `ggplot`, except here we start the code with `ggmap()` instead of `ggplot()`.

Here is how we can retrieve basemaps:

```
# Create an object with Africa only
africa <-
  worldmap %>%
  filter(REGION == "Africa")

# Calculate which part of the world we want images for
# (this is called a bounding box)
africa_box <-
  st_bbox(africa)

# Save the basemap
africa_basemap <-
  get_stamenmap(as.vector(africa_box),
    zoom = 3, # The higher the zoom, the more details you get
    maptype = "watercolor")
```

Adding a basemap

```
ggmap(africa_basemap)
```

Customizing basemaps

 You can also use other image sources than Stamen Maps with the `get_map` function:

- Google Maps (`"google"`), OpenStreetMap (`"osm"`), Stamen Maps (`"stamen"`)

 Here are a few other map types you can use with `get_stamenmap`:

- `"terrain"`, `"terrain-background"`, `"terrain-labels"`, `"terrain-lines"`, `"toner"`, `"toner-2010"`, `"toner-2011"`, `"toner-background"`, `"toner-hybrid"`, `"toner-labels"`, `"toner-lines"`, `"toner-lite"`, `"watercolor"`)

 Finally, you can use the options `color` and `alpha` to change from black and white to color and increase transparency of the basemap.

Adding layers on top of a basemap

```
ggmap(africa_basemap) +  
  geom_point(data = wb_projects,  
            aes(x=longitude,  
                y=latitude),  
            size = .1) +  
  theme_void()
```

Note that with `ggmap` we don't need the option `coord_quickmap`

Adding layers on top of a basemap

Combining our two maps

Now, instead of a basemap, let's layer these points on top of our happiness score map.

```
ggplot() +  
  geom_sf(data = africa,  
          aes(fill = happiness_score)) +  
  geom_point(data = wb_projects,  
            aes(x = longitude,  
                y = latitude),  
            size = .1) +  
  labs(fill="Happiness\nScore") +  
  scale_fill_gradient(low = "blue",  
                    high = "yellow") +  
  theme_void()
```

Combining our two maps

Why projections matter

Why projections matter

```
# Use a different projection for our Africa map
africa_moll <-
  st_transform(africa,
               "+proj=moll")

# And create the same graph from the last slide
ggplot() +
  geom_sf(data = africa_moll,
          aes(fill = happiness_score)) +
  geom_point(data = wb_projects,
             aes(x=longitude,
                 y=latitude)) +
  labs(fill="Happiness\nScore") +
  scale_fill_gradient(low = "blue",
                     high = "yellow") +
  theme_void()
```

Why projections matter

Transforming GPS data into a shapefile

- As we saw earlier, shapefiles can contain points, polygons or lines.
- So far, we have only use the `wb_projects` coordinates as if they were numbers like any others.
- To be able to change the projection of `wb_projects`, we need to convert it into a spatial object.

`st_as_sf(x, coords, crs)`

Transforms objects into `sf` objects

- `...`: the object to be transformed
- `coords`: a vector with the names of the variables in the data that indicate longitude and latitude, in this order
- `crs`: the coordinate reference system of the points in the data

Transforming GPS data into a shapefile

Exercise

Turn the `wb_projects` object into an `sf` object.

```
st_as_sf(x,  
  coords = c("longitude_variable", "latitude_variable"),  
  crs = 4326) # Shortcut to WGS84, the coordinate reference system used by most GPS
```

Transforming GPS data into a shapefile

Exercise

Turn the `wb_projects` object into an `sf` object.

```
wb_projects <-  
  st_as_sf(wb_projects,  
    coords = c("longitude", "latitude"),  
    crs = 4326)
```

Matching projections

Exercise

Change the projection of the `wb_projects` object to Mollweid.

Tip: use the CRS shortcut `"+proj=moll"`

```
wb_projects_moll <-
```

Matching projections

Exercise

Change the projection of the `wb_projects` object to Mollweid.

Tip: use the CRS shortcut `"+proj=moll"`

```
wb_projects_moll <-  
  st_transform(wb_projects,  
              "+proj=moll")
```

Combining plots with the same projection

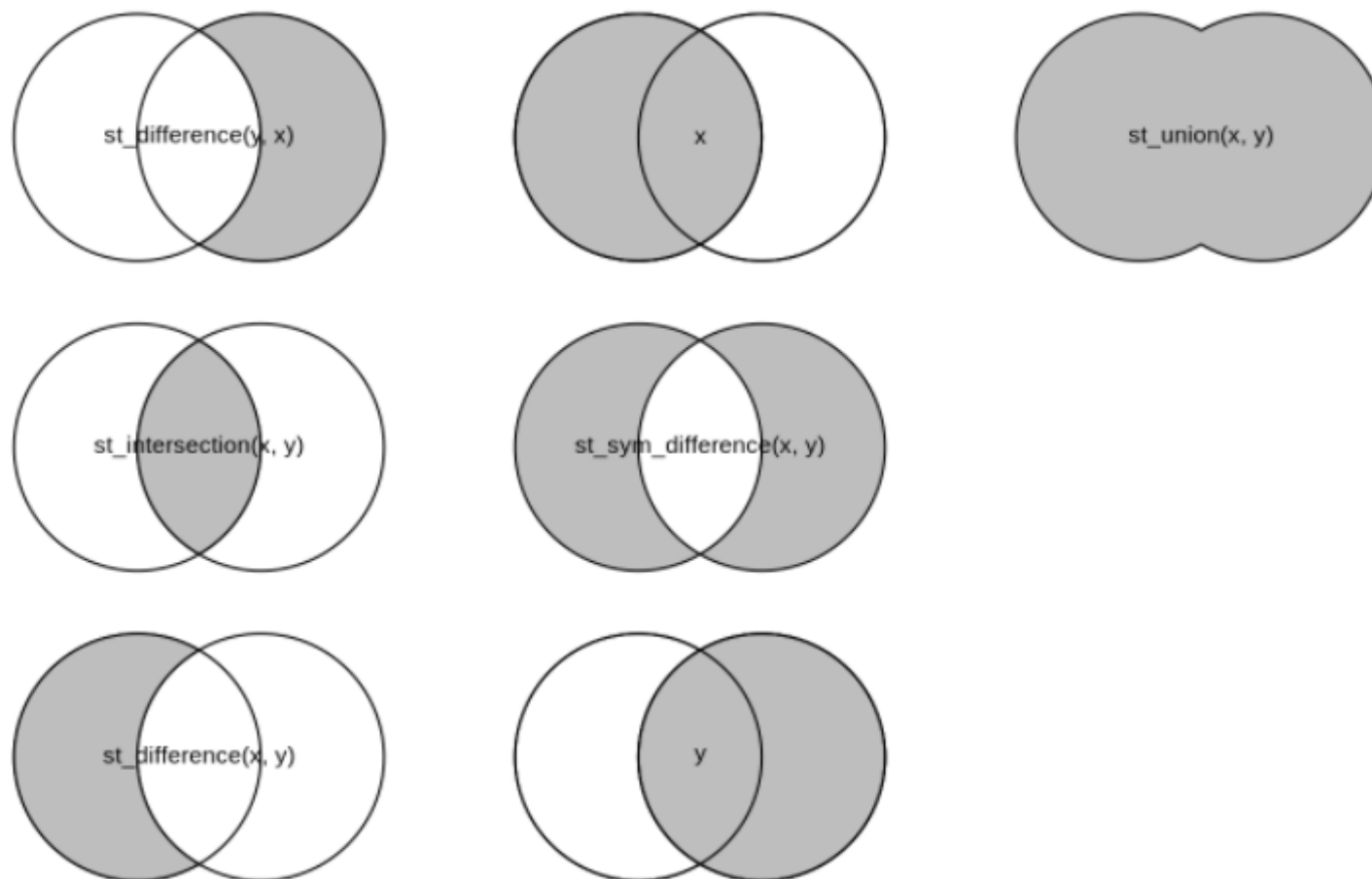
```
ggplot() +  
  geom_sf(data = africa_moll,  
          aes(fill = happiness_score)) +  
  geom_sf(data = wb_projects_moll) +  
  labs(fill="Happiness Score and WB Projects") +  
  scale_fill_gradient(low = "blue",  
                     high = "yellow") +  
  theme_void()
```


Combining plots with the same projection

Basic geometry operations

Basic geometry operations

Here are some of the most common shapefile operations and their corresponding `sf` commands:



Final challenge

Exercise

Create a map of the World Bank projects in Mozambique

Here's some pseudo code:

```
# 1 Create a polygon of Mozambique by subsetting the worldmap sf  
# 2 Make sure the Moz polygon and the wb_projects shapefile have the same projection  
# 3 Create a shapefile containing only Moz projects using one of the sf functions in the previous image  
# 4 Create a map with the resulting data and customize as you like
```

Final challenge

Useful Resources

- [Rspatial](#) provides tutorials for many topics in GIS.
- [Nick Eubank Tutorials](#) -- another great set of tutorials.
- [This](#) provides useful links to a bunch of other resources.
- [Visualizing geospatial data](#)
- [Geocomputation with R](#)

Thank you!