Political Economy of Development:

Spatial Analysis with R

Eduardo Montero

April 21, 2023

Table of Contents

Table of Contents

Spatial Data in Development: Examples

Spatial Data in Development: Examples

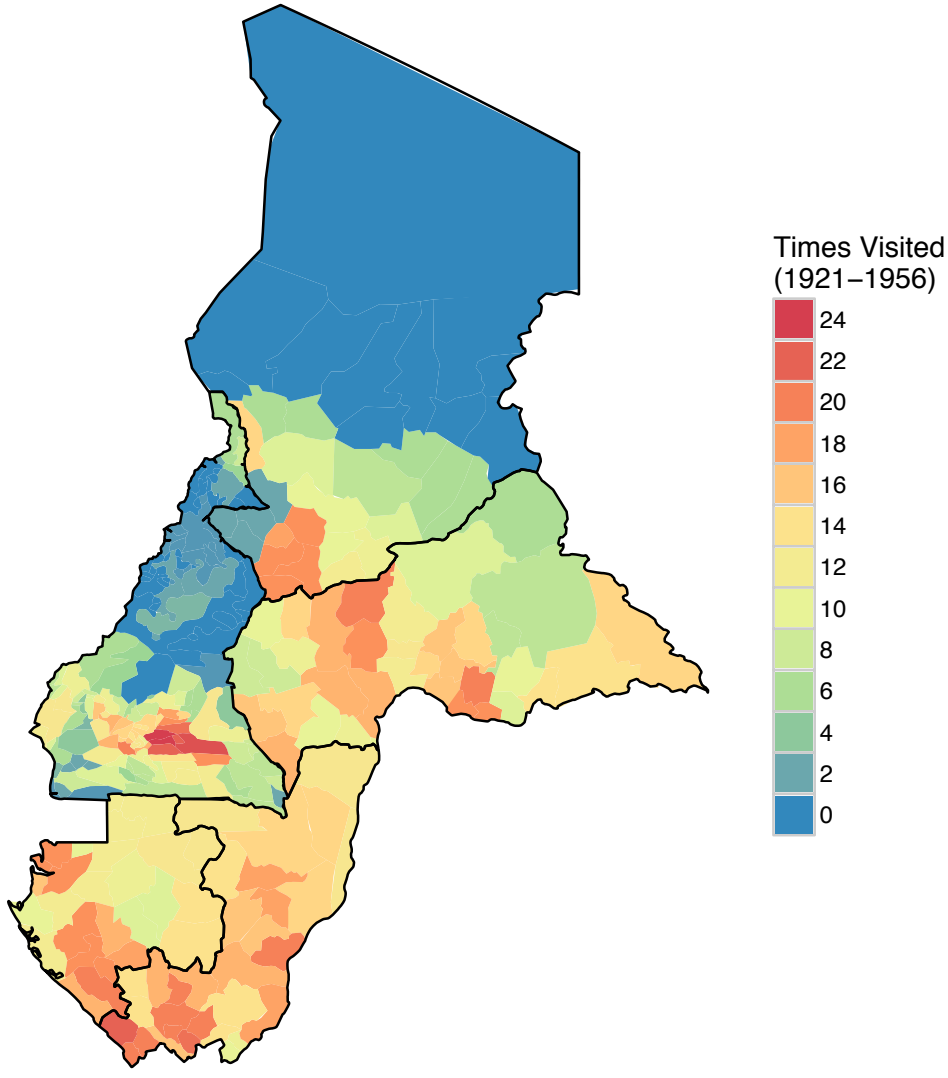
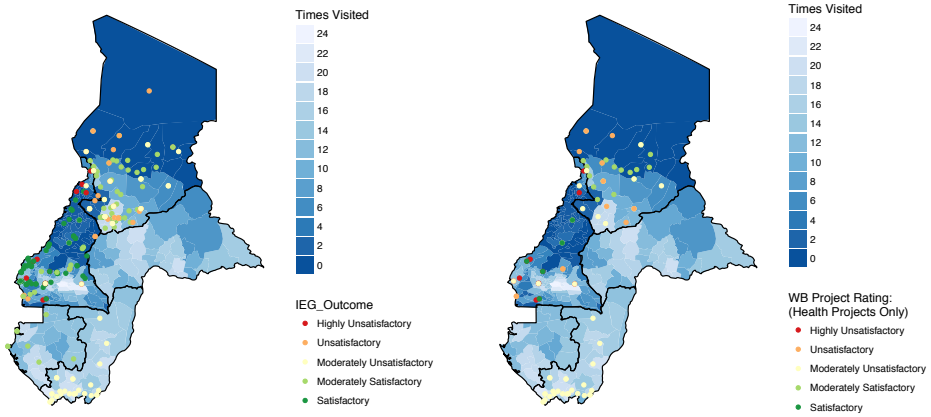


Figure: Colonial Medical Team Visits

Spatial Data in Development: Examples

Figure 15: World Bank Projects and Outcome Rating



(a) All Sectors

(b) Health Sector Only

Figure: World Bank Project Data

Spatial Data in Development: Examples

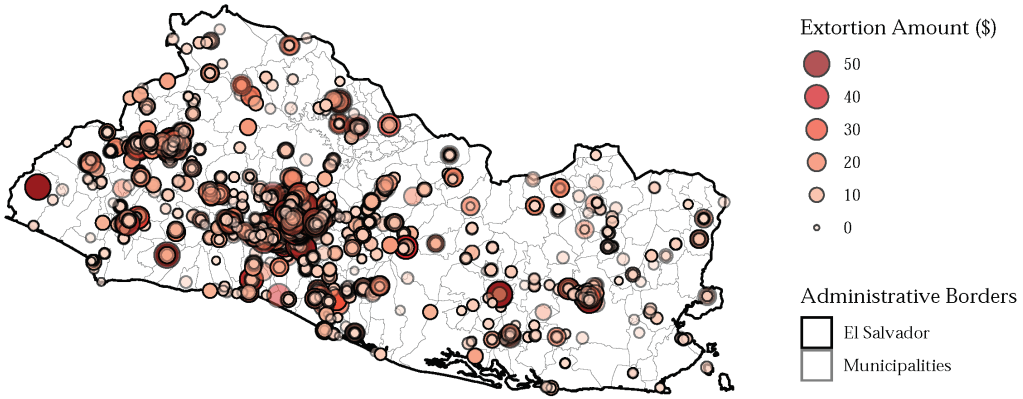


Figure: Extortion in El Salvador

Spatial Data in Development: Examples

Figure: Extortion and Deliveries in El Salvador

Spatial Data in Development: Examples

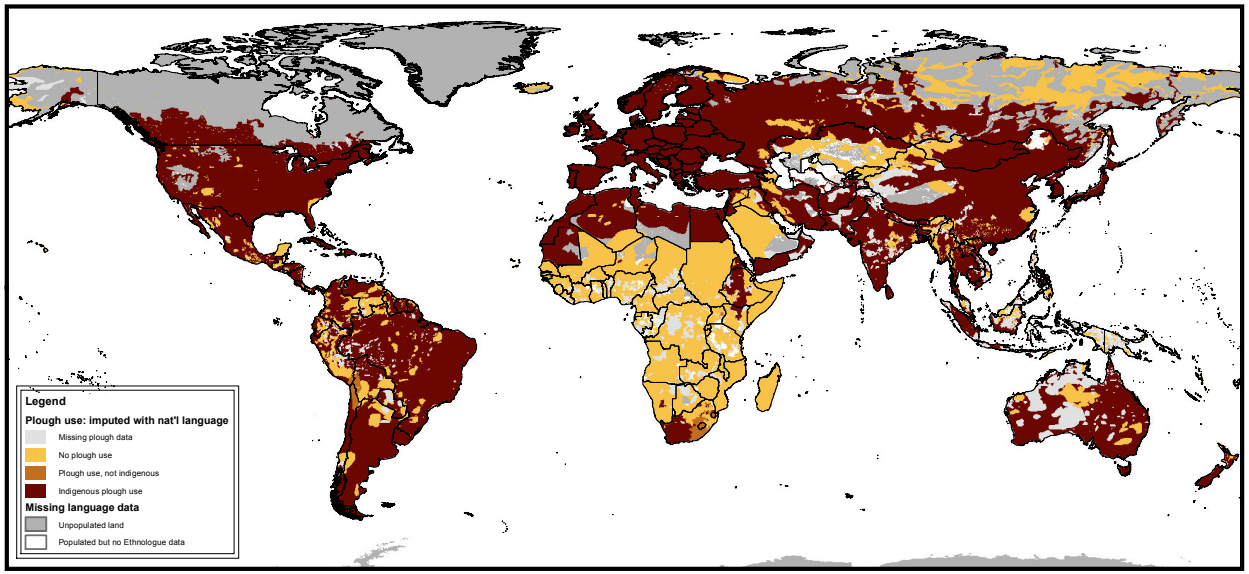


Figure: Use of the Plough

Spatial Data in Development

- We've seen many papers so far using spatial data
- Sources ranging from satellite dataset to official government records, all with a spatial component
- Many projects involve working with spatial data
- This lecture is intended to give you an introduction to spatial data and analysis

Table of Contents

Spatial Data in R

- R has many great packages and functions for using spatial data
- It is a big advantage over using Stata, especially as spatial data becomes more widely available
- Also processes much faster than ArcGIS (made for spatial analysis) and processing/plotting can be used seemlessly with analysis
- We will learn to use the sf package in R
- Highly recommend the online (free) book "Geocomputation with R": Link

Spatial Data in R: sf Package

What's so nice about sf?

- Easy to work with spatial data because the distinction between spatial data and other forms of data is minimized
- Spatial objects are stored as dataframes, with the feature geometries stored in list-columns
- All functions begin with st_ for easy autofill with RStudio tab
- Functions are pipe-friendly (%>%)
- ggplot2 is able to plot sf objects directly

References on sf

- Today's lecture is meant to introduce you to some of the most useful sf behaviors for development economics
- But, there are many other great online resources on sf
- For example, RStudio has a great cheatsheet on sf (we will see it later one)
- And, see the online (free) book "Geocomputation with R": Link

Spatial Data in R

- To start, we need to load a few important packages.
- With **Windows**–easy: install.packages(c("sf", "raster", "rgdal"))
- With **Mac**–might work: install.packages(c("sf", "raster", "rgdal"))
- If not... less easy:
  - With Capitan, see here Instructions. Otherwise, try:
  - Download and install GDAL complete Frameworks
  - Download the rgdal package: rdgal
  - Place the downloaded rgdal_1.0-4.tgz in your Desktop folder
  - Run install.packages("~/Desktop/rgdal1.0-4.tgz", repos=NULL)
  - Install raster and sf by running: install.packages(c("sf","raster"))

Spatial Data in R

- Two main spatial data types: vector data and raster data
- Download and load libraries: sf, raster, rdgal, ggplot2, broom, dplyr, spData

```
#Package preload
library(sf)

## Warning: package 'sf' was built under R version 4.1.1
## Linking to GEOS 3.9.1, GDAL 3.2.3, PROJ 7.2.1

library(raster)

## Loading required package: sp

library(rgdal)

## rgdal: version: 1.5-23, (SVN revision 1121)
## Geospatial Data Abstraction Library extensions to R
successfully loaded
## Loaded GDAL runtime: GDAL 3.2.3, released 2021/04/27
## Path to GDAL shared files:
```

Spatial Data in R

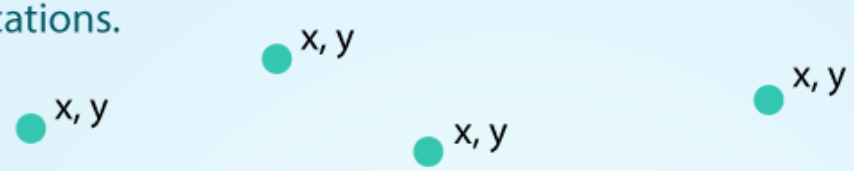
- We will cover the types of spatial data: vector data and raster data
- We will start with vector data

Table of Contents

Vector Data Types: 3 Types

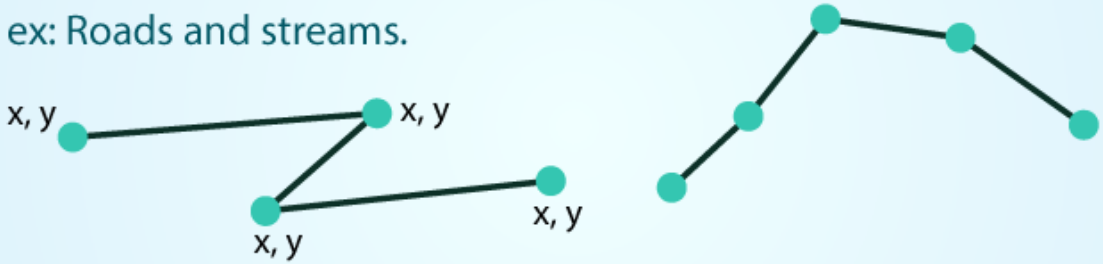
POINTS: Individual **x, y** locations.

ex: Center point of plot locations, tower locations, sampling locations.



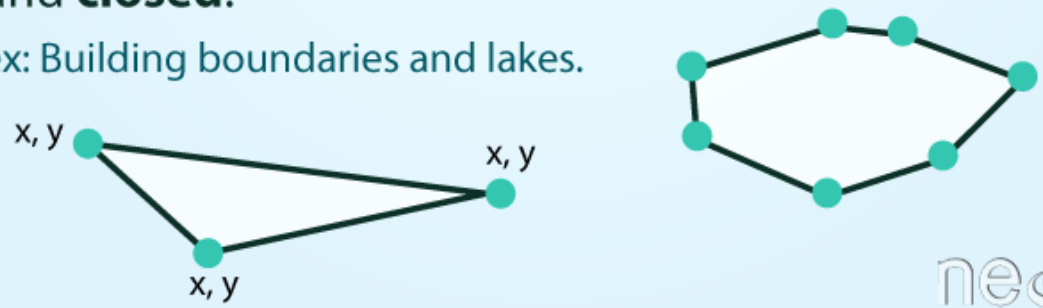
LINES: Composed of many (at least 2) vertices, or points, that are connected.

ex: Roads and streams.



POLYGONS: 3 or more vertices that are connected and **closed**.

ex: Building boundaries and lakes.



neon

Vector Spatial Data in R: Shapefiles

- Geospatial data in vector format are often stored in a shapefile format (.shp files)
- Because the structure of points, lines, and polygons are different, each individual shapefile can only contain one vector type (all points, all lines or all polygons)
- Objects stored in a shapefile often have a set of associated attributes that describe the data.
- For example, a line shapefile that contains the locations of streams, might contain the associated stream name, stream "order" and other information about each stream line object.

Vector Spatial Data in R: Shapefiles

- You use the rgdal package to work with vector data in R
- We will import three different shapefiles: 1) A point shapefile representing the location of field sites, 2) A line shapefile representing roads, and 3) A polygon shapefile representing countries of the world
- st_read() reads in shapefiles in R

Loading Data

```
# points:
sjer_plot_locations <- st_read("data/california/SJER/vector_data/SJER_p
# lines/roads:
sjer_roads <- st_read("data/california/madera-county-roads/tl_2013_0603
# polygon/countries:
library(spData)
worldBound <- world # loaded with spData
```

Viewing the data

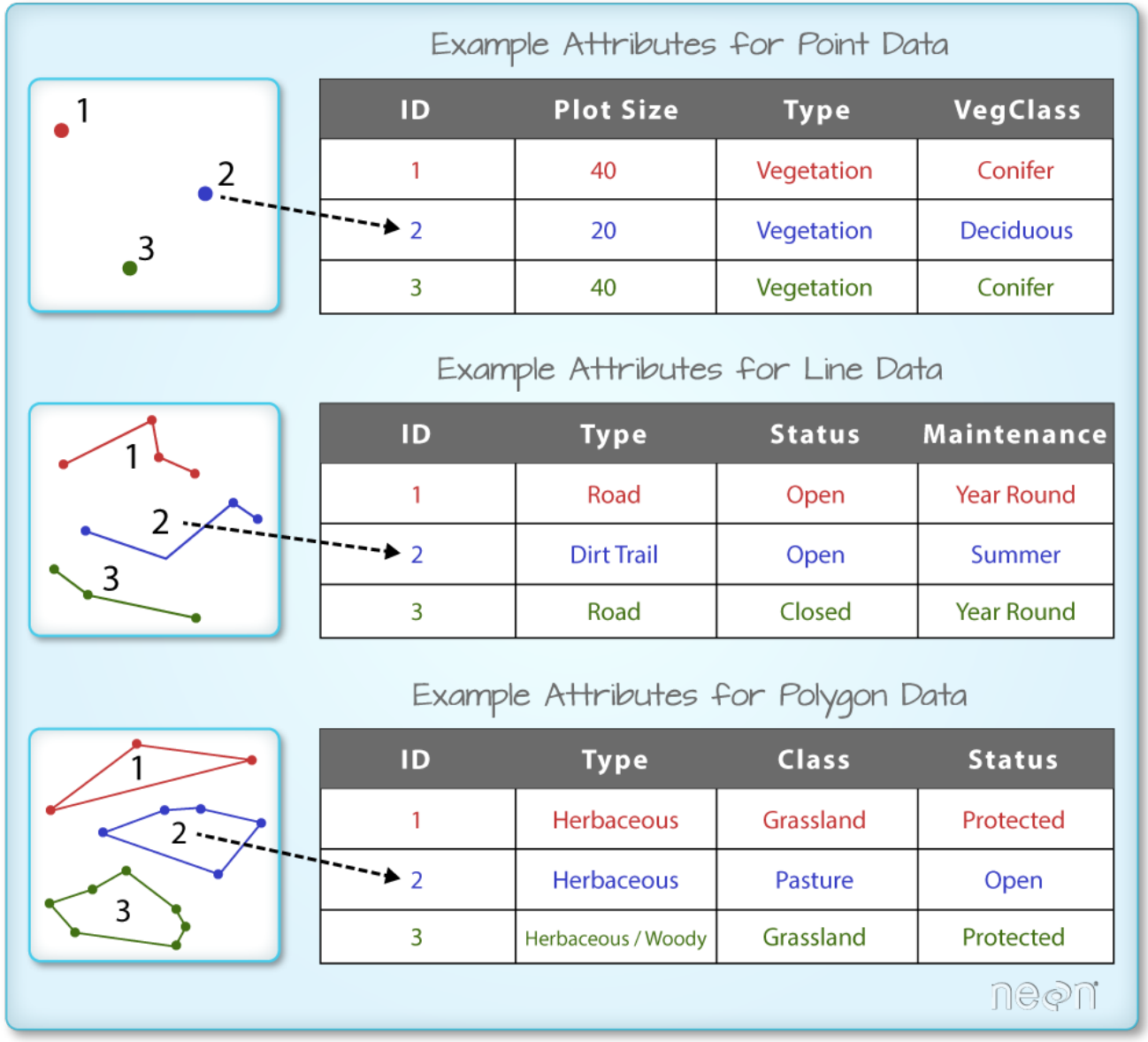
```
# view just the class for the shapefile
class(worldBound)

## [1] "sf"         "tbl_df"     "tbl"        "data.frame"

# view all metadata at same time
glimpse(worldBound)

## Rows: 177
## Columns: 11
## $ iso_a2    <chr> "FJ", "TZ", "EH", "CA", "US", "KZ", "UZ"~
## $ name_long <chr> "Fiji", "Tanzania", "Western Sahara", "C~
## $ continent <chr> "Oceania", "Africa", "Africa", "North Am~
## $ region_un <chr> "Oceania", "Africa", "Africa", "Americas~
## $ subregion <chr> "Melanesia", "Eastern Africa", "Northern~
## $ type      <chr> "Sovereign country", "Sovereign country"~
## $ area_km2  <dbl> 19289.97, 932745.79, 96270.60, 10036042.~
## $ pop       <dbl> 885806, 52234869, NA, 35535348, 31862252~
## $ lifeExp   <dbl> 69.96000, 64.16300, NA, 81.95305, 78.841~
## $ gdpPercap <dbl> 8222.2538, 2402.0994, NA, 43079.1425, 51~
## $ geom      <MULTIPOLYGON [°]> MULTIPOLYGON (((-180 -16.55~
```

Viewing the data

- Each object in a shapefile has one or more attributes associated with it.
- Shapefile attributes are similar to fields or columns in a dataset Each row has a set of columns associated with it that describe the row element.
- In the case of a shapefile, each row represents a spatial object - for example, a road, represented as a line in a line shapefile, will have one "row" associated with it.

```
# view all metadata at same time
worldBound

## Simple feature collection with 177 features and 10 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -89.9 xmax: 180 ymax: 83.64513
## Geodetic CRS:  WGS 84
## # A tibble: 177 x 11
## # iso_a2 name_long  continent  region_un subregion  type
```

Vector Data Types: 3 Types



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

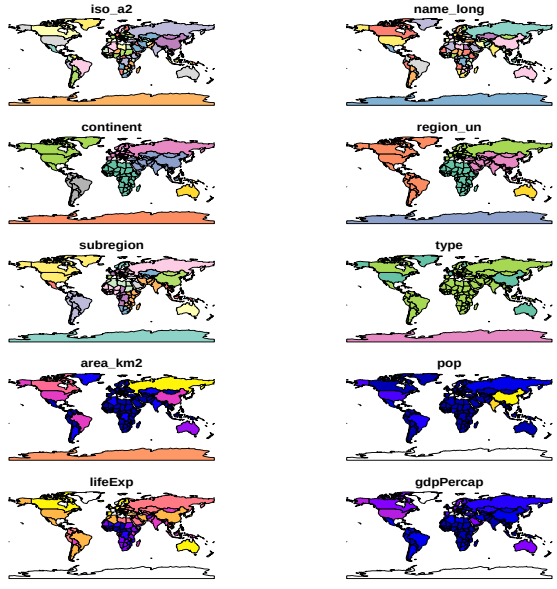
ID	Type	Status	Maintenance
1	Road	Open	Year Round
2	Dirt Trail	Open	Summer
3	Road	Closed	Year Round

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

Viewing the data

- Can plot all the data with base R:

```
plot(worldBound)
```



Viewing the data

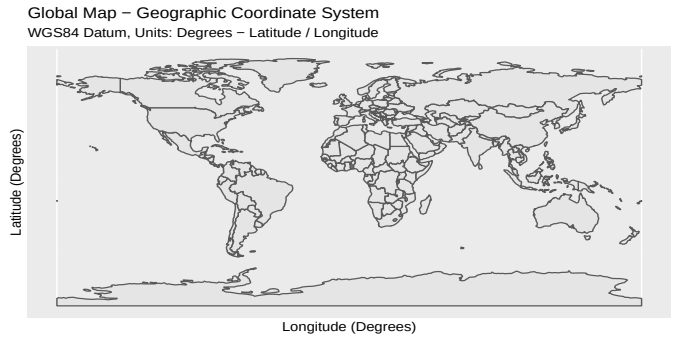
- We can visualize the data using ggplot:

```
## First, Will define a new "Theme" for ggplot map plots
# Here: turn off axis elements in ggplot for better visual comparison
map_theme <- list( theme_bw(),
      theme( line = element_blank(),
       panel.grid.minor=element_blank(), # blank backgroung
      panel.grid.major=element_blank(),
      axis.text.x = element_blank(),
      axis.text.y = element_blank(),
      axis.ticks = element_blank(), # turn off ticks
      axis.title.x = element_blank(), # turn off titles
      axis.title.y = element_blank()))
```

Viewing the data

- Can also plot using ggplot

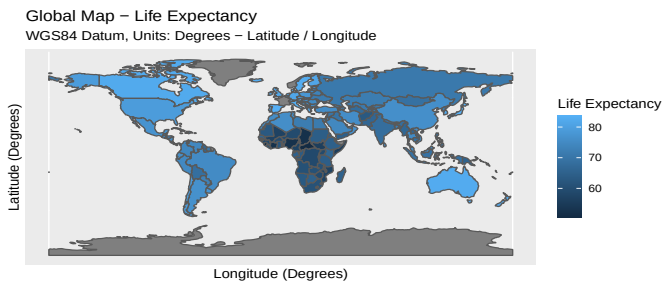
```
# plot map using ggplot
worldMap <- worldBound %>%  ggplot() +
  geom_sf() +
  labs(x = "Longitude (Degrees)",
       y = "Latitude (Degrees)",
      title = "Global Map - Geographic Coordinate System ",
      subtitle = "WGS84 Datum, Units: Degrees - Latitude / Longitude")

worldMap
```



Viewing the data

- Can plot attributes too:

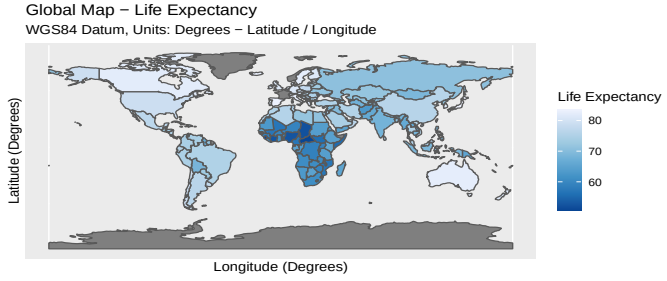
```
# plot map using ggplot and fill with LifeExp information:
worldMap <- worldBound %>%  ggplot() +
  geom_sf(aes(fill=lifeExp)) +
  labs(x = "Longitude (Degrees)",
       y = "Latitude (Degrees)",
       fill="Life Expectancy",
      title = "Global Map - Life Expectancy",
      subtitle = "WGS84 Datum, Units: Degrees - Latitude / Longitude")

worldMap
```



Viewing the data

- Can plot attributes too and use nice ggplot scale packages:

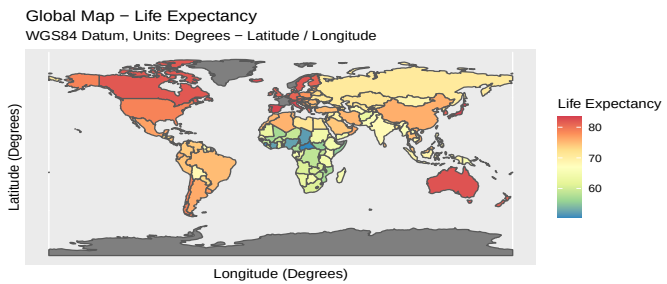
```
# plot map using ggplot and fill with LifeExp information:
worldMap_blues <- worldBound %>%  ggplot() +
  geom_sf(aes(fill=lifeExp)) +
  scale_fill_distiller(palette = "Blues") +
  labs(x = "Longitude (Degrees)",
       y = "Latitude (Degrees)",
       fill="Life Expectancy",
      title = "Global Map - Life Expectancy",
      subtitle = "WGS84 Datum, Units: Degrees - Latitude / Longitude")

worldMap_blues
```



Viewing the data

- For other ggplot scale options, check out options in scale_fill_distiller() (continuous) and scale_fill_brewer()

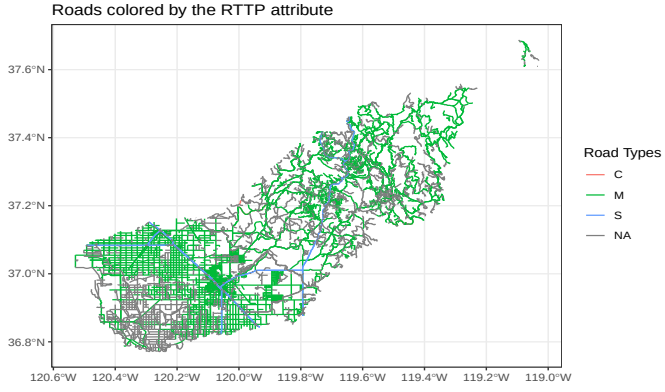
```
# plot map using ggplot and fill with LifeExp information:
worldMap_spectral <- worldBound %>%  ggplot() +
  geom_sf(aes(fill=lifeExp)) +
  scale_fill_distiller(palette = "Spectral", direction=-1) + # flip dir
  labs(x = "Longitude (Degrees)",
       y = "Latitude (Degrees)",
       fill="Life Expectancy",
      title = "Global Map - Life Expectancy",
      subtitle = "WGS84 Datum, Units: Degrees - Latitude / Longitude")

worldMap_spectral
```



Viewing the data

- Can plot attributes too:

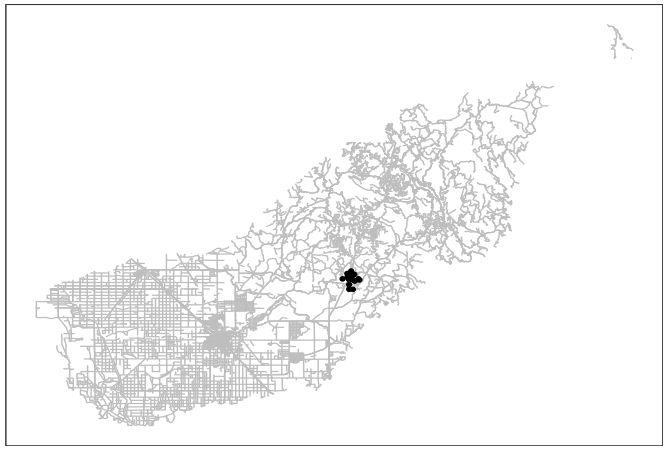
```
# color roads by attribute (type)
roads <- sjer_roads %>%  ggplot() +
  geom_sf(aes(color = factor(RTTYP))) +
labs(color = 'Road Types', # change the legend type
     title = "Roads colored by the RTTP attribute") +
  theme_bw()
roads
```



Viewing the data

- Multiple layers: (can specify xlim or ylim in coord_equal() to limit lat/lon of map)

```
# reproject to lat / long
sjer_plots <- st_transform(sjer_plot_locations, crs(sjer_roads)) # Will
# Plots
roads_and_points <- ggplot() +
  geom_sf(data=sjer_roads,
            color = "grey") +
  geom_sf(data=sjer_plots) +
   map_theme
roads_and_points
```

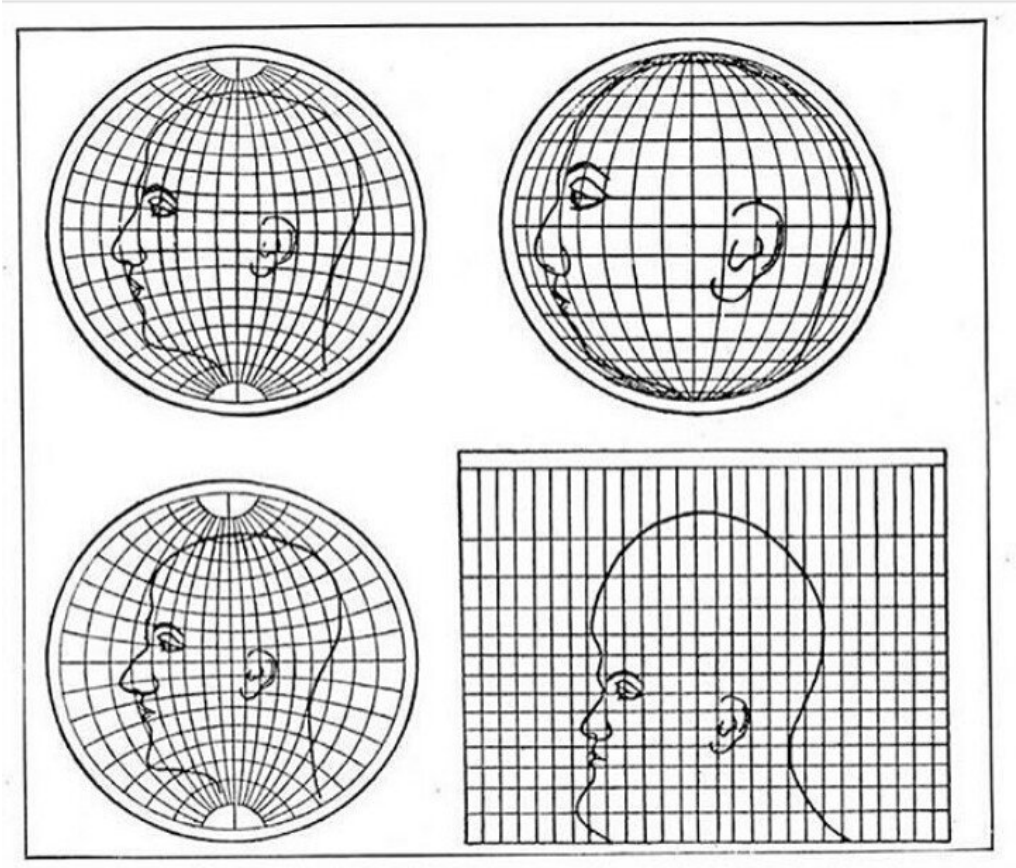


Quick Note on Projections

- When plotting or doing spatial analysis, need to make sure spatial data is using the same projection. What is a projection?

Quick Note on Projections

- When plotting or doing spatial analysis, need to make sure spatial data is using the same projection. What is a projection?



Upper left: Globular. *Upper right:* Orthographic. *Lower left:* Stereographic. *Lower right:* Mercator

What four commonly used projections do, as shown on a human head

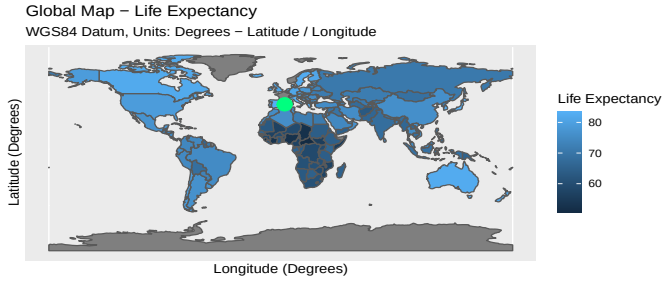
Quick Note on Projections

- Two main types of projections: geographic (lat/lon) projections and projected ("meters")
- These are known as Coordinate Reference Systems (CRS)
- To see how they differ, we will make a layer with the lat/lon location of Mallorca, Spain and then plot it on maps that are in the two different projections

```
# store coordinates in a data.frame
loc_df <- data.frame(lon = c(2.9833),
                     lat = c(39.6167))
```

Quick Note on Projections

```
# add a point to the map
mapLocations <- worldMap +
                geom_point(data = loc_df,
                aes(x = lon, y = lat, group = NULL),
                colour = "springgreen",
                      size = 5)
mapLocations
```



Quick Note on Projections

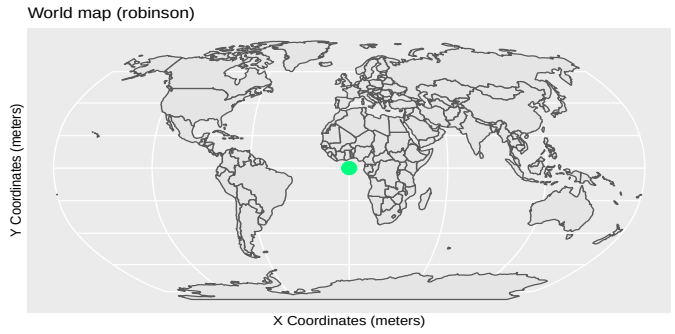
- Can project using st_transform easily in R:

```
# reproject data from longlat to robinson CRS (meters)
worldBound_robin <- st_transform(worldBound,
                                 CRS("+proj=robin"))
# Prep Map:
robMap <- ggplot() + geom_sf(data=worldBound_robin) +
  labs(title = "World map (robinson)",
       x = "X Coordinates (meters)",
       y = "Y Coordinates (meters)")
```

Quick Note on Projections

- Now add points and plot

```
# add a point to the map
newMap <- robMap + geom_point(data = loc_df,
                              aes(x = lon, y = lat, group = NULL),
                              colour = "springgreen",
                              size = 5)

newMap
```



Quick Note on Projections

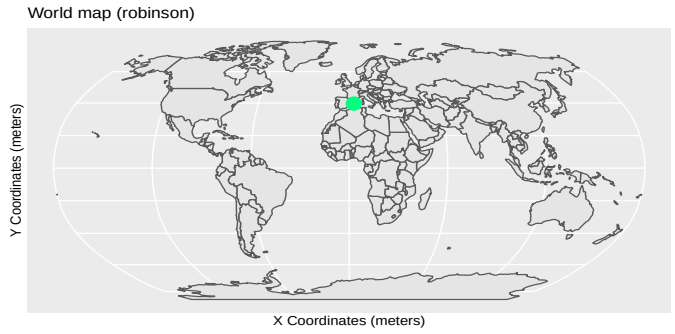
- Need to make coords of Mallora a spatial object and reproject:

```
# convert dataframe to simple features (sf) points
loc_sf <- st_as_sf(loc_df, coords = c("lon", "lat")) # make sf data
loc_sf <- st_set_crs(loc_sf, crs(worldBound)) # add geometry informatio

# reproject data to Robinson
loc_sf_rob <- st_transform(loc_sf, crs("+proj=robin"))
```

Quick Note on Projections

- Plot reporjected data:

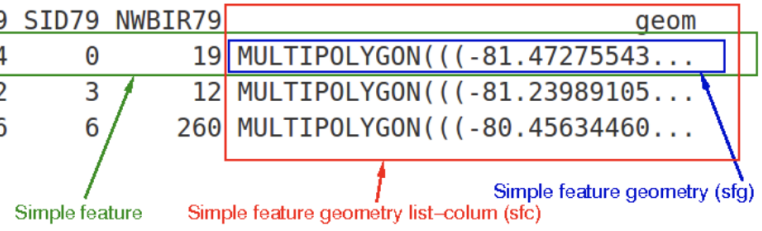
```
# turn off scientific notation
options(scipen = 10000)
# add a point to the map
newMap <- robMap + geom_sf(data = loc_sf_rob,
                           colour = "springgreen",
                           size = 5)
newMap
```



Vector Layers in sf

- The sf class is a hierarchical structure composed of 3 classes:
  - In green, a sf: a Vector layer object data.frame with ≥ 1 attribute column(s) and 1 geometry column
  - In red, sfc: Geometric part of vector layer - geometry column
  - In blue, sfg: Geometry of invidiviual row (one "simple feature")

```
## Simple feature collection with 100 features and 6 fields
## geometry type:  MULTIPOLYGON
## dimension:      XY
## bbox:           xmin: -84.32385 ymin: 33.88199 xmax: -75.45698 ymax: 36.58965
## epsg (SRID):    4267
## proj4string:    +proj=longlat +datum=NAD27 +no_defs
## precision:      double (default; no precision model)
## First 3 features:
##   BIR74 SID74 NWBIR74 BIR79 SID79 NWBIR79                              geom
## 1  1091     1      10  1364     0      19 MULTIPOLYGON(((-81.47275543...
## 2   487     0      10   542     3      12 MULTIPOLYGON(((-81.23989105...
## 3  3188     5     208  3616     6     260 MULTIPOLYGON(((-80.45634460...
```



Vector Layers in sf

- A nice thing about the sf class is that this hierarchical structure lets us also treat sf objects like we would any data.frame

- For example, the commonly used summary() function provides a useful overview of the variables within the world object.

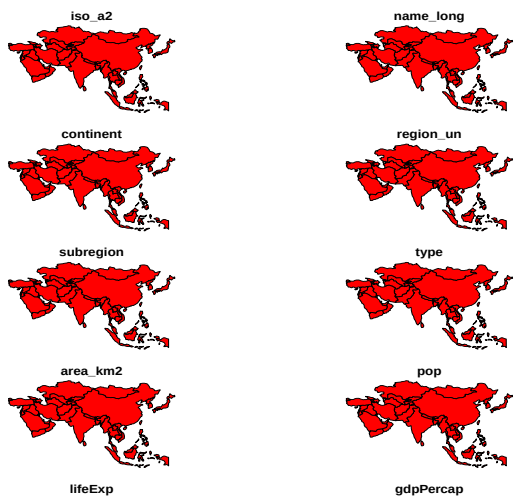
```
summary(world["lifeExp"])
```

```
##     lifeExp                    geom
##  Min.   :50.62   MULTIPOLYGON :177
##  1st Qu.:64.96   epsg:4326    :  0
##  Median :72.87   +proj=long...:  0
##  Mean   :70.85
##  3rd Qu.:76.78
##  Max.   :83.59
##  NA's   :10
```

Vector Layers in sf

- Although we selected one variable for the summary command, it also outputs a report on the geometry on the right
- Known as 'sticky' behavior of the geometry columns of sf objects: the geometry is kept unless the we deliberately removes them (with st_drop_geometry())
- Another nice thing we can do is subset data as with any data frame:

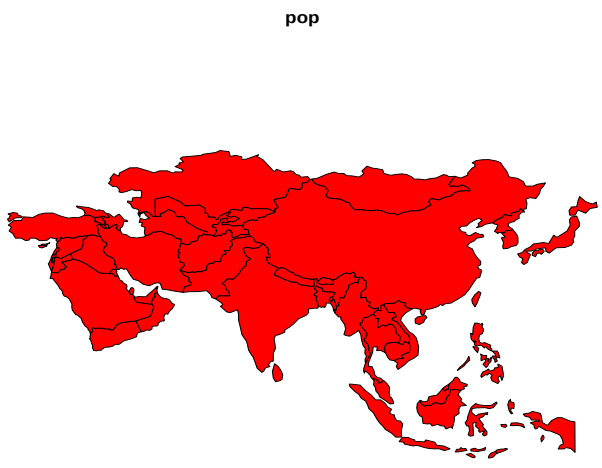
```
world_asia <- world[world$continent == "Asia", ]
plot(world_asia, col = "red")
```



Vector Layers in sf

- We can also use pipes and tidyverse functions:

```
world_asia_filter <- world %>% filter(continent == "Asia") %>% select(p
plot(world_asia_filter, col = "red")
```



Vector Layers in sf

- Other tidyverse functions and piping examples:

```
world_small <- world %>% filter(area_km2 < 10000) # Small Countries
world1 <- world %>%  dplyr::select(name_long, pop) # keep onle some var
world2 <- world %>% dplyr::select( -subregion, -area_km2) # all columns
```

Vector Layers in sf

- Other useful base functions we can use on sf objects:

```
dim(world) # it is a 2 dimensional object, with rows and columns
```

```
## [1] 177  11
```

```
nrow(world) # how many rows?
```

```
## [1] 177
```

```
ncol(world) # how many columns?
```

```
## [1] 11
```

sf Attribute Methods: Aggregations

- sf also has useful methods for spatial vector data
- Aggregation operations summarize datasets by a "grouping variable", typically an attribute column
- An example of attribute aggregation is calculating the number of people per continent based on country-level data (one row per country)
- We can do this with tidyverse:

```
world_agg3 = world %>%
  group_by(continent) %>%
  summarize(pop = sum(pop, na.rm = TRUE), n=n())
```

sf Attribute Methods: Aggregations

- Exercise: chaining together functions, find the world's 3 most populous continents and the number of countries they contain

```
world %>%
  dplyr::select(pop, continent) %>%
  group_by(continent) %>%
  summarize(pop = sum(pop, na.rm = TRUE), n_countries = n()) %>%
  top_n(n = 3, wt = pop) %>%
  st_drop_geometry()
```

```
## # A tibble: 3 x 3
##   continent        pop n_countries
## * <chr>          <dbl>       <int>
## 1 Africa    1154946633          51
## 2 Asia      4311408059          47
## 3 Europe     669036256          39
```

sf Attribute Methods: Joining

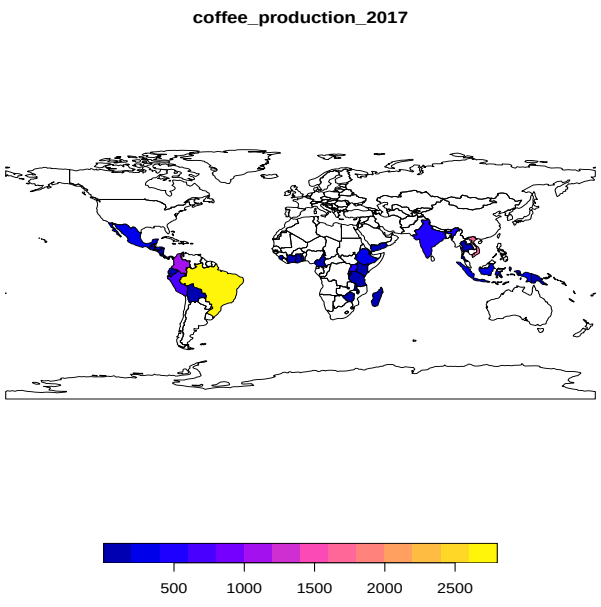
- Combining data from different sources is a common task in data preparation
- Joins do this by combining tables based on a shared 'key' variable
- To demonstrate joins, we will combine data on coffee production with the world dataset:

```
world_coffee = left_join(world, coffee_data)

## Joining, by = "name_long"

class(world_coffee)

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

sf Attribute Methods: Joining

- Because the input datasets share a 'key variable' (name_long) the join worked without using the "by" argument in left_join
- We can plot the result:

```
plot(world_coffee["coffee_production_2017"])
```



sf Attribute Methods: Joining

- If the input datasets share a 'key variable' but it is named differently, we can use the "by" argument in left_join

```
coffee_renamed = rename(coffee_data, nm = name_long)
world_coffee2 = left_join(world, coffee_renamed,
                          by = c(name_long = "nm"))
```

sf Attribute Methods: New Variables

- We can also create variables with mutate:

```
world <- world %>%
  mutate(pop_dens = pop / area_km2)
# Or, with base R: world_new$pop_dens = world_new$pop / world_new$area_
```

sf Attribute Methods: Removing Geometry

- Before, we noted that attribute data operations preserve the geometry of the simple features ("sticky" behavior)
- This behavior ensures that data frame operations do not accidentally remove the geometry column, which is really nice
- But, sometimes, it can be useful to remove the geometry:

```
world_data = world %>% st_drop_geometry()
class(world_data)

## [1] "tbl_df"     "tbl"        "data.frame"
```

sf Spatial Methods:

- Spatial operations are operations that modify a spatial object based on their location and shape
- Many options, and super useful for spatial analysis:
  - Distance from points to a line
  - Area of polygons
  - Length of road networks
- We will cover a few useful functions for vectors from sf

```
# Mozambique Forced Cotton Cultivation Shapefile:
cotton <- st_read("./data/Mozambique cotton/Shapefile/Cotton_1953.shp")
# Mozambique 2011 DHS survey locations:
dhs <- st_read("./data/MZ_2011_DHS/mzge61fl/MZGE61FL.shp")
# Filter DHS locations without coords:
dhs <- dhs %>% filter(!(LATNUM==0 & LONGNUM==0) & !(abs(LATNUM) < 0.01)
```

sf Spatial Methods: Spatial Subsets

- Spatial subsetting is the process of selecting features of a spatial object based on whether or not they in some way relate in space to another object
- Example, find which DHS locations are in former cotton areas:

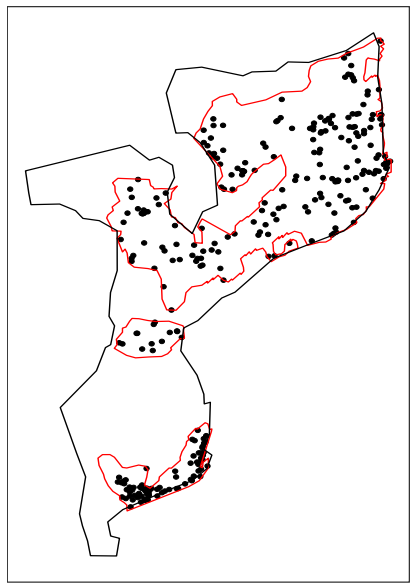
```
# Make sure projections are the same:
mercator <- 3857 # meters/planar projection code
cotton <- st_transform(cotton, mercator)
dhs <- st_transform(dhs, mercator)
# Mozambique 2011 DHS survey locations in cotton areas:
dhs_cotton <- dhs[cotton, ]
```

sf Spatial Methods: Spatial Subsets

- Spatial subsetting is the process of selecting features of a spatial object based on whether or not they in some way relate in space to another object

- Example, find which DHS locations are in former cotton areas:

```
# Plot Result:
mozambique <- world %>% filter(name_long=="Mozambique") %>% st_transfor
ggplot() + geom_sf(data=dhs_cotton) +
  geom_sf(data=cotton, col="red", fill=NA) +
  geom_sf(data=mozambique, col="black", fill=NA) + map_theme
```



sf Spatial Methods: Spatial Subsets

- Another options is to directly use st_intersects
- Example, find which DHS locations are in former cotton areas by creating an indicator

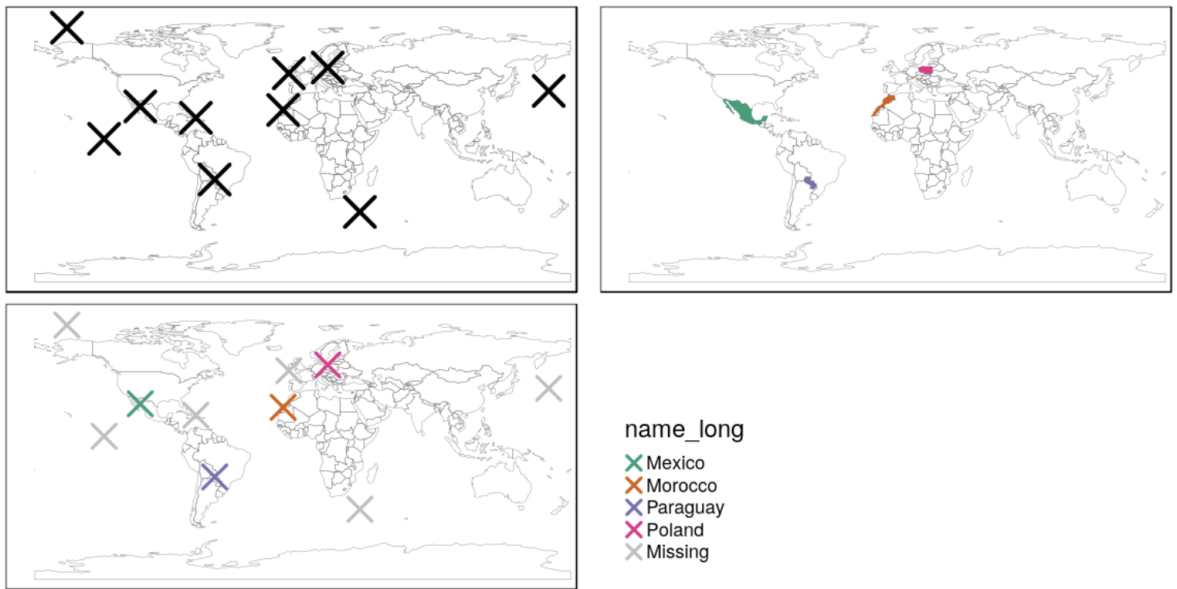
```
intersect_result <- st_intersects(x = dhs, y = cotton,sparse=TRUE) # sp
dhs <- dhs %>% mutate(in_cotton =
                        ifelse(lengths(intersect_result) > 0,
                               1,
                               0)) # One Option, many others
```

sf Spatial Methods: Spatial Subsets

- Many related examples:
- st_disjoint: returns only objects that do not spatially relate in any way to the selecting object
- st_within: returns TRUE only for objects that are completely within the selecting object.
- st_touches: returns TRUE only for objects that are exactly on the border of the selecting object.
- st_is_within_distance: features that do not touch, but almost touch the selection object

sf Spatial Methods: Spatial Joining

- Spatial data joinin relies on shared areas of geographic space
- Tt is also know as spatial overlay
- As with non-spatial joining of data, spatial joining adds a new column to the target object (the argument x in joining functions), from a source object (y)
- Example below: adding country names (y=world) to a set of points (x):



sf Spatial Methods: Spatial Joining

- Example code: adding country names (y=world) to a set of points (x):

```
set.seed(2018) # set seed for reproducibility
(bb_world = st_bbox(world)) # the world's bounds

##       xmin       ymin       xmax       ymax
## -180.00000  -89.90000  179.99999   83.64513

random_df = tibble(
  x = runif(n = 10, min = bb_world[1], max = bb_world[3]),
  y = runif(n = 10, min = bb_world[2], max = bb_world[4])
)
random_points = random_df %>%
  st_as_sf(coords = c("x", "y")) %>% # set coordinates
  st_set_crs(4326) # set geographic CRS
random_joined = st_join(random_points, world["name_long"])
```

sf Spatial Methods: Spatial Joining

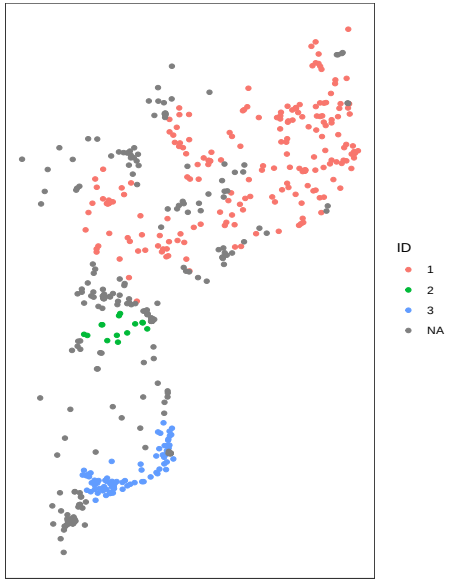
- Let's work through an example: there were 3 cotton areas in Mozambique
- We can assign an ID to each cotton area,
- And then spatially join to the DHS data to have information on which cotton area (ID) it falls in (or not)

```
cotton <- cotton %>% mutate(ID = factor(1:nrow(cotton))) # Add ID
dhs_wcotton <- st_join(dhs, cotton["ID"])
```

sf Spatial Methods: Spatial Joining

- Plot result:

```
ggplot() + geom_sf(data=dhs_wcotton, aes(color=ID)) + map_theme
```



sf Spatial Methods: Distance Relations

- While topological relations are binary — a feature either intersects with another or does not — distance relations are continuous
- The distance between two objects is calculated with the st_distance() function
- Example below: convert cotton to a line (using st_cast()), then calculate distance from DHS points to cotton borders:

```
cotton_border <- st_cast(cotton, "MULTILINESTRING") # See next page on
dist_cotton_border_matrix <- st_distance(dhs, cotton_border) # 3 col ma
 st_distance(dhs, cotton_border) [1,] # tells us the units!!!

## Units: [m]
## [1]    7124.862  732164.708 1101442.150
```

sf Spatial Methods: Distance Relations

However, returns 3 distances (to each area); to keep just the minimum. One option: use rowMins() from the matrixStats library

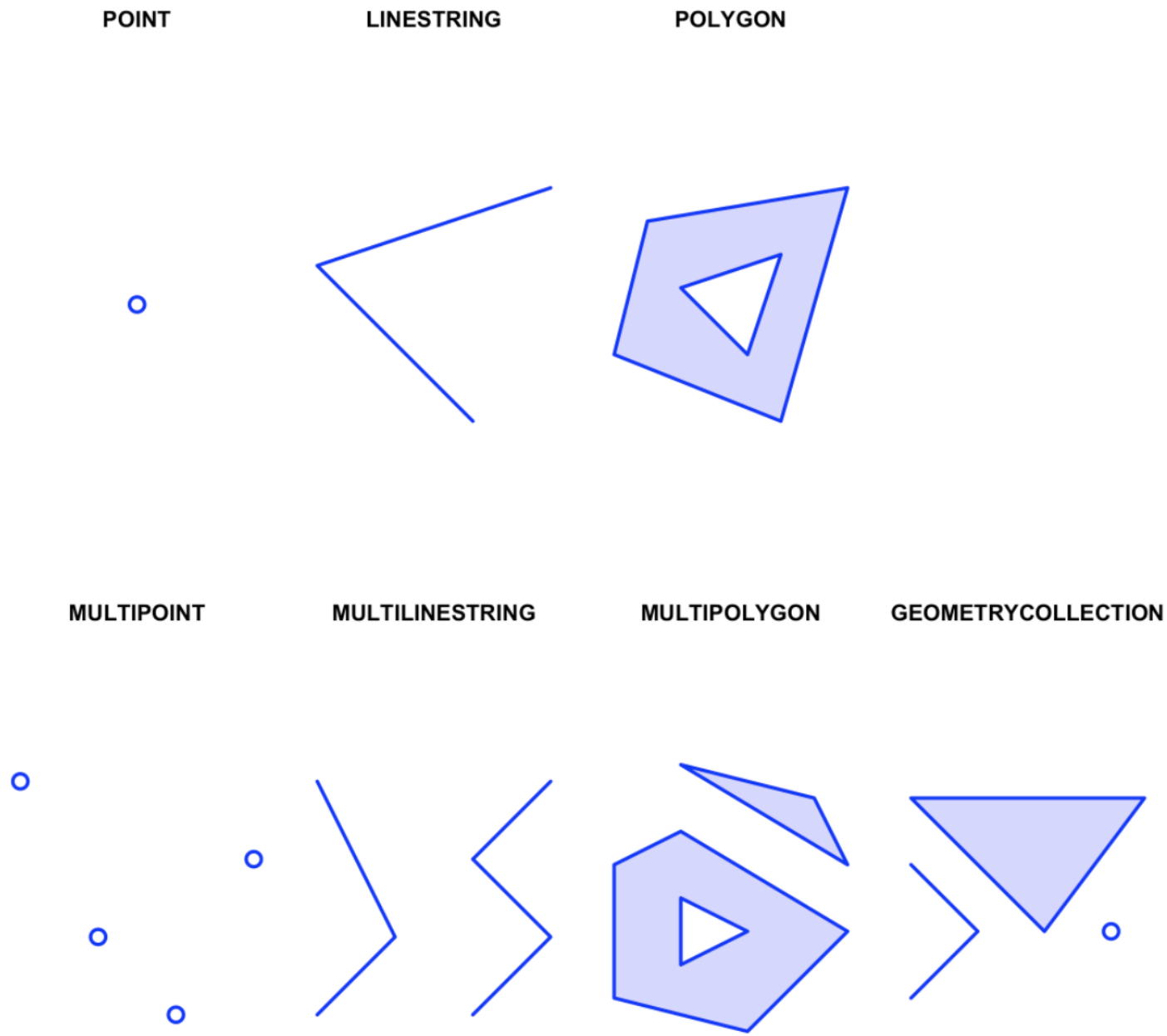
```
 # Load MatrixStats:
 require(matrixStats)

## Loading required package:  matrixStats
##
## Attaching package:  'matrixStats'
## The following object is masked from 'package:dplyr':
##
##     count

# Minimum of the three columns:
min_dist <- dist_cotton_border_matrix %>%
 rowMins()
# Add to DHS:
dhs$dist_cotton_border <- min_dist
```

sf Spatial Methods: Distance Relations

Alternative:

```
# Minimum of the three columns:
min_dist <- dist_cotton_border_matrix %>%
  t() %>%  # transpose to be 3 rows instead of 3 cols
  data.frame() %>% # make a data frame
  mutate_all(min) %>% # find min() of each column
  unique() %>% # call unique in case there are ties
  t() # transpose back to be 1 col
# Add to DHS:
dhs$dist_cotton_border <- min_dist
```

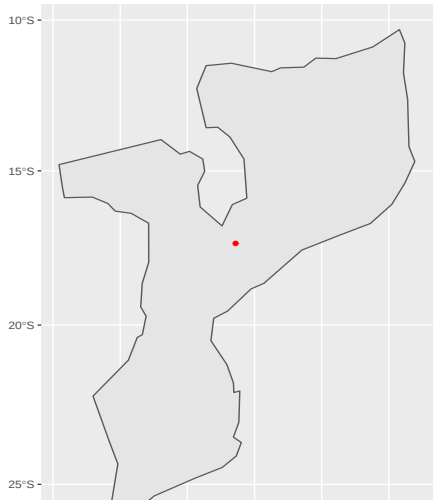
sf Spatial Methods: sf Types



sf Geometric Methods: Centroids

- Centroid operations identify the center of geographic objects
- Many options or ways to define this. The most commonly used is the geographic centroid ("center of mass in a spatial object")
- Example: We can add the centroid of Mozamique to the plot using st_centroid:

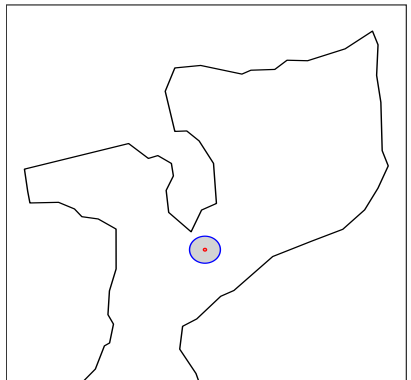
```
mozambique_centroid <- st_centroid(mozambique)
ggplot() + geom_sf(data = mozambique) +
  geom_sf(data = mozambique_centroid, col="red")
```



sf Geometric Methods: Buffers

- Buffers are polygons representing the area within a given distance of a geometric feature
- We can make these using st_buffer() and specifying different distance thresholds
- Example: Different "Buffers" around the center of Mozambique:

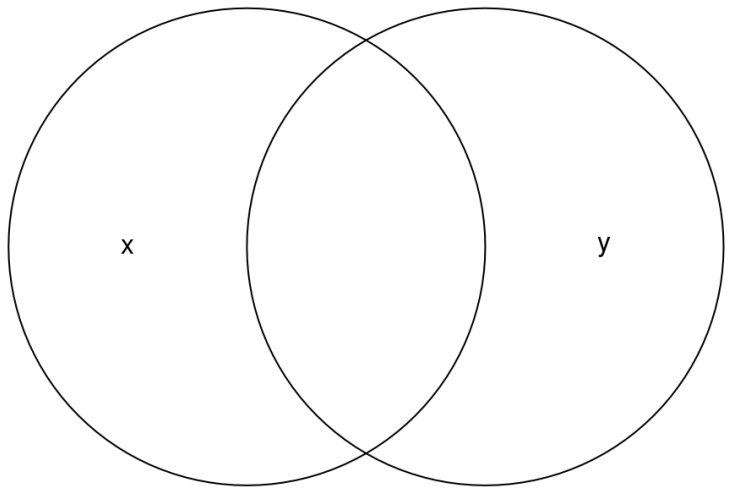
```
mozambique_buffer_5km <- st_buffer(mozambique_centroid, dist = 5000)
mozambique_buffer_50km <- st_buffer(mozambique_centroid, dist = 50000)
ggplot() + geom_sf(data=mozambique,fill=NA,col="black") +
  geom_sf(data=mozambique_buffer_50km, col="blue", fill="light gray") +
  geom_sf(data=mozambique_buffer_5km, col="red", fill="gray") + map_the
```



sf Geometric Methods: Clipping

- Spatial clipping is a form of spatial subsetting that involves changes to the geometry columns of at least some of the affected features

- Easy to visualize the many sf options. Consider the two circles below:



sf Geometric Methods: Clipping

- sf Options:

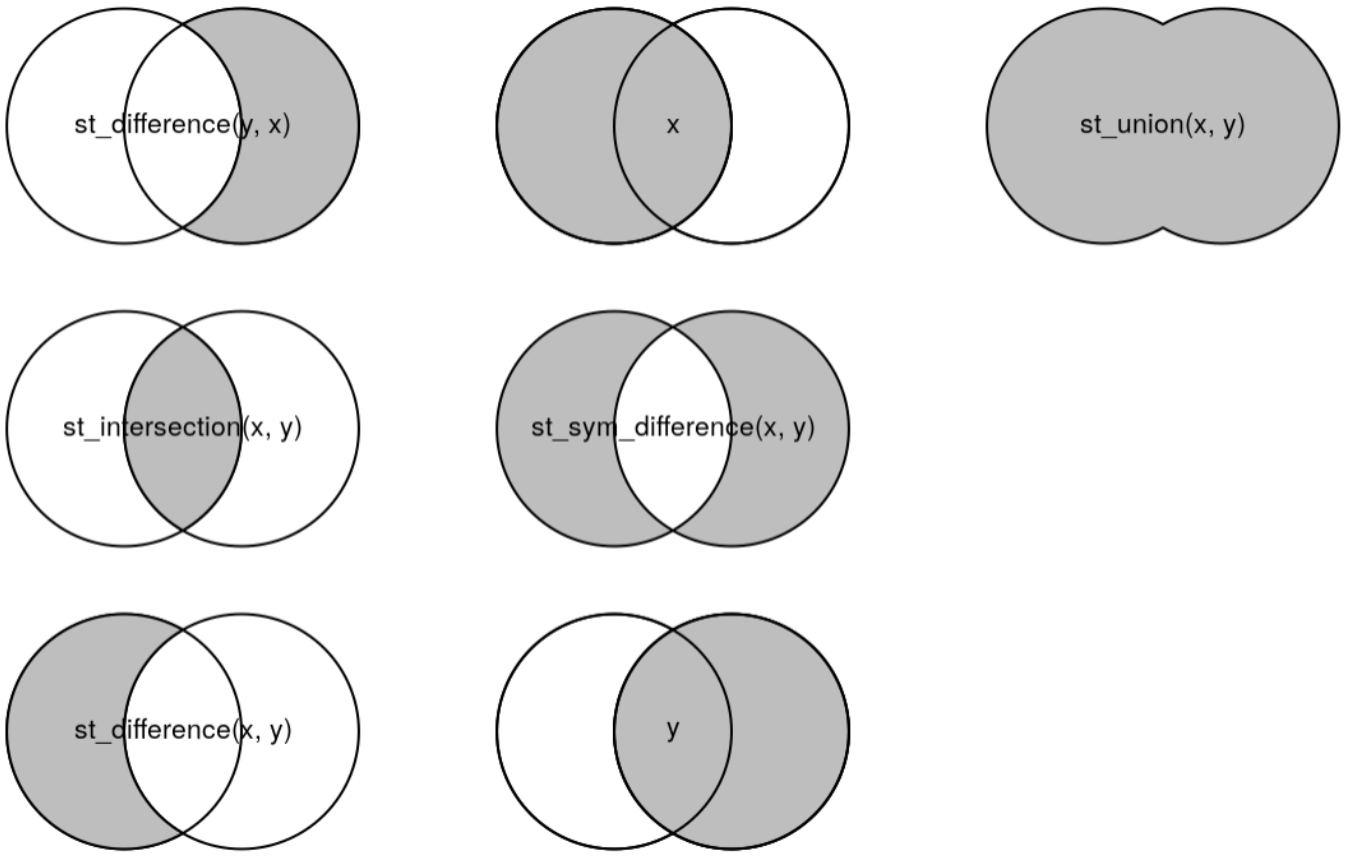
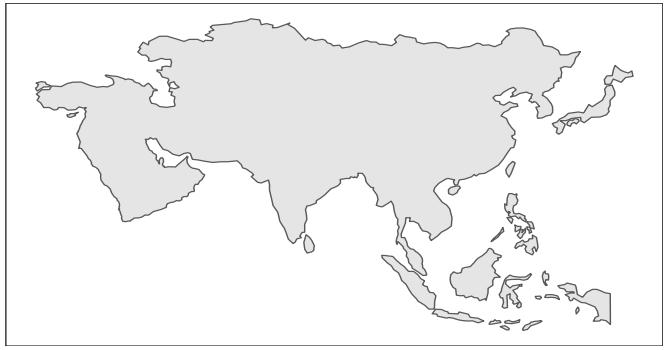


Figure 5.8: Spatial equivalents of logical operators.

sf Geometric Methods: Unions

- We saw examples of aggregating data (using group_by() and summarize())
- We can also aggregate only the geometry information using st_union

```
asia_countries <- world[world$continent == "Asia", ]
asia <- st_union(asia_countries)
asia %>% ggplot() + geom_sf() + map_theme
```



sf Methods: Many Options

- Highly recommend the sf RStudio cheatsheet (page 1/2):

Spatial manipulation with sf: : CHEAT SHEET

The sf package provides a set of tools for working with geospatial vectors, i.e. points, lines, polygons, etc.

Geometric confirmation

st_contains(x, y, ...) Identifies if x is within y (i.e. point within polygon)

st_covered_by(x, y, ...) Identifies if x is completely within y (i.e. polygon completely within polygon)

st_covers(x, y, ...) Identifies if any point from x is outside of y (i.e. polygon outside polygon)

st_crosses(x, y, ...) Identifies if any geometry of x have commonalities with y

st_disjoint(x, y, ...) Identifies when geometries from x do not share space with y

st_equals(x, y, ...) Identifies if x and y share the same geometry

st_intersects(x, y, ...) Identifies if x and y geometry share any space

st_overlaps(x, y, ...) Identifies if geometries of x and y share space, are of the same dimension, but are not completely contained by each other

st_touches(x, y, ...) Identifies if geometries of x and y share a common point but their interiors do not intersect

st_within(x, y, ...) Identifies if x is in a specified distance to y

Geometric operations

st_boundary(x) Creates a polygon that encompasses the full extent of the geometry

st_buffer(x, dist, nQuadSegs) Creates a polygon covering all points of the geometry within a given distance

st_centroid(x, ..., of_largest_polygon) Creates a point at the geometric centre of the geometry

st_convex_hull(x) Creates geometry that represents the minimum convex geometry of x

st_line_merge(x) Creates linestring geometry from sewing multi linestring geometry together

st_node(x) Creates nodes on overlapping geometry where nodes do not exist

st_point_on_surface(x) Creates a point that is guarenteed to fall on the surface of the geometry

st_polygonize(x) Creates polygon geometry from linestring geometry

st_segmentize(x, dfMaxLength, ...) Creates linesting geometry from x based on a specified length

st_simplify(x, preserveTopology, dTolerance) Creates a simplified version of the geometry based on a specified tolerance

Geometry creation

st_triangulate(x, dTolerance, bOnlyEdges) Creates polygon geometry as triangles from point geometry

st_voronoi(x, envelope, dTolerance, bOnlyEdges) Creates polygon geometry covering the envolope of x, with x at the centre of the geometry

st_point(x, c(numeric vector), dim = "XYZ") Creating point geometry from numeric values

st_multipoint(x = matrix(numeric values in rows), dim = "XYZ") Creating multi point geometry from numeric values

st_linestring(x = matrix(numeric values in rows), dim = "XYZ") Creating linestring geometry from numeric values

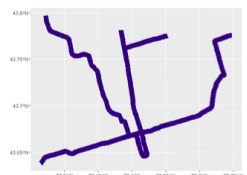
st_multilinestring(x = list(numeric matricesin rows), dim = "XYZ") Creating multi linestring geometry from numeric values

st_polygon(x = list(numeric matrices in rows), dim = "XYZ") Creating polygon geometry from numeric values

st_multipolygon(x = list(numeric matrices in rows), dim = "XYZ") Creating multi polygon geometry from numeric values

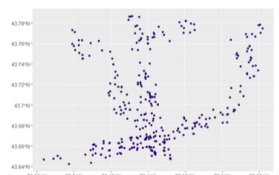
ggplot() +
geom_sf(data = schools)

+



ggplot() +
geom_sf(data = subway)

=>



ggplot() +
geom_sf(data = st_intersection(schools, st_buffer(subway, 1000)))

This cheatsheet presents the sf package [Edzer Pebesma 2018] in version 0.6.3. See https://github.com/r-spatial/sf for more details.

sf Methods: Many Options

- Highly recommend the sf RStudio cheatsheet (page 2/2):

Spatial manipulation with sf: : CHEAT SHEET

The sf package provides a set of tools for working with geospatial vectors, i.e. points, lines, polygons, etc.

Geometry operations

st_contains(x, y, ...) Identifies if x is within y (i.e. point within polygon)

st_crop(x, y, ..., xmin, ymin, xmax, ymax) Creates geometry of x that intersects a specified rectangle

st_difference(x, y) Creates geometry from x that does not intersect with y

st_intersection(x, y) Creates geometry of the shared portion of x and y

st_sym_difference(x, y) Creates geometry representing portions of x and y that do not intersect

st_snap(x, y, tolerance) Snap nodes from geometry x to geometry y

st_union(x, y, ..., by_feature) Creates multiple geometries into a a single geometry, consisiting of all geometry elements

Geometric measurement

st_area(x) Calculate the surface area of a polygon geometry based on the current coordinate reference system

st_distance(x, y, ..., dist_fun, by_element, which) Calculates the 2D distance between x and y based on the current coordinate system

st_length(x) Calculates the 2D length of a geometry based on the current coordinate system

Misc operations

st_as_sf(x, ...) Create a sf object from a non-geospatial tabular data frame

st_cast(x, to, ...) Change x geometry to a different geometry type

st_coordinates(x, ...) Creates a matrix of coordinate values from x

st_crs(x, ...) Identifies the coordinate reference system of x

st_join(x, y, join, FUN, suffix, ...) Performs a spatial left or inner join between x and y

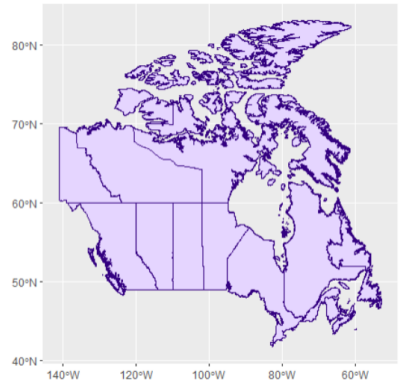
st_make_grid(x, cellsize, offset, n, crs, what) Creates rectangular grid geometry over the bounding box of x

st_nearest_feature(x, y) Creates an index of the closest feature between x and y

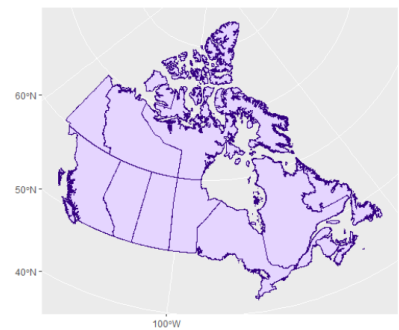
st_nearest_points(x, y, ...) Returns the closest point between x and y

st_read(dsn, layer, ...) Read file or database vector dataset as a sf object

st_transform(x, crs, ...) Convert coordinates of x to a different coordinate reference system




```
ggplot() +
  geom_sf(data = cdn) +
  coord_sf(crs = st_crs(4326))
```




```
ggplot() +
  geom_sf(data = cdn) +
  coord_sf(crs = st_crs(3347))
```

This cheatsheet presents the sf package [Edzer Pebesma 2018] in version 0.6.3. See https://github.com/r-spatial/sf for more details.

Table of Contents

Spatial Data in R

- Rasters are pixel/cell level spatial data sets. Example: Elevation, or Nightlights
- Rasters are much more compact that vectors
- We will first explore raster data and then see how it can be combined with vector data

Spatial Data in R – Rasters

A raster dataset has three primary components:

- A grid, which consists of:
  - dimensions (number of rows and columns),
  - resolution (size of sides of each cell),
  - and extent (where the edges of the grid "are")
- A set of values associated with each cell in the grid
- Projection data about how the grid relates to the physical world

Rasters

- Rasters process very quickly due to matrix storage:
  - Starting from the origin, we can easily access and modify each single cell by either using the ID of a cell (see below) or by explicitly specifying the rows and columns
  - Means that there is no need to store coordinates of each grid-cell

A. Cell IDs

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

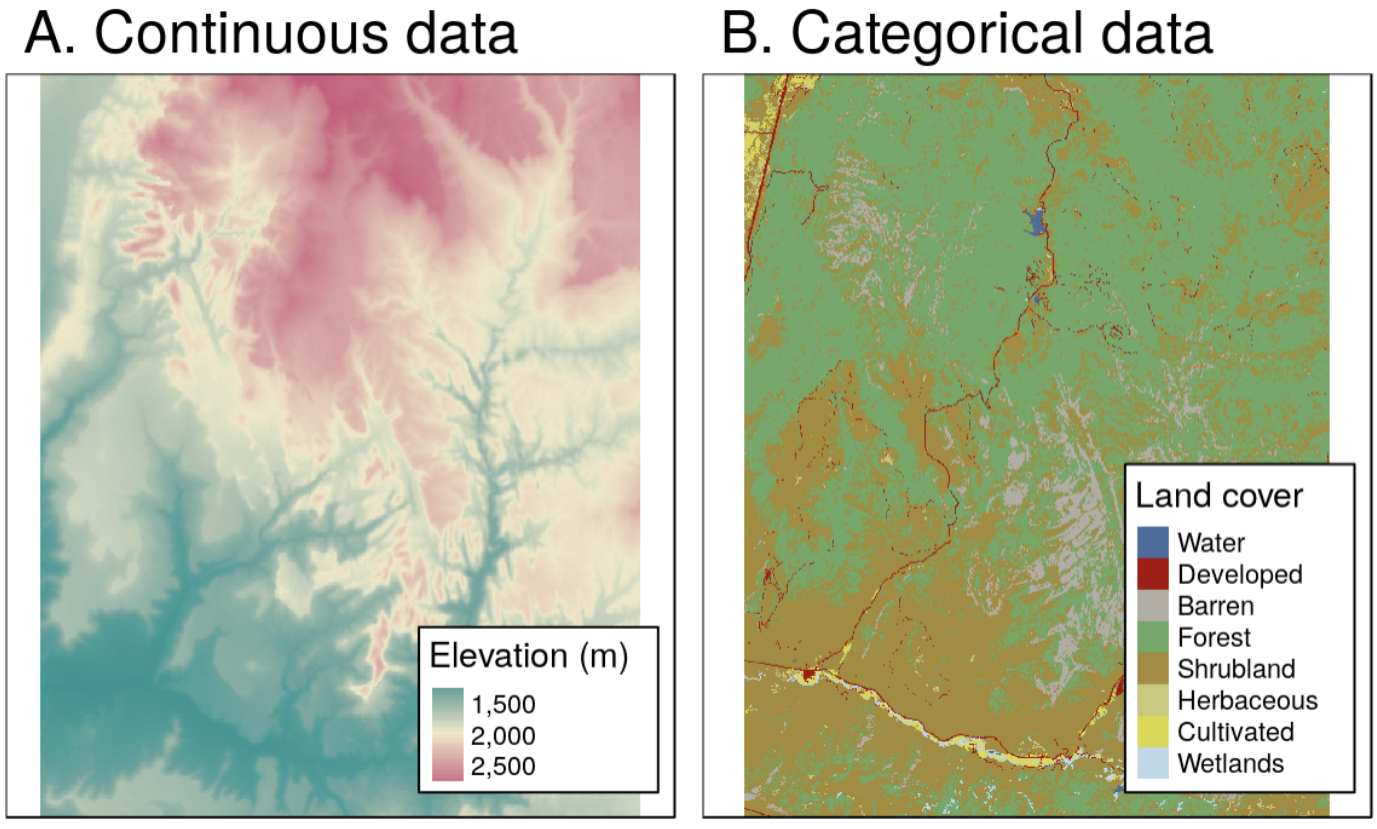
B. Cell values

22	74	28	91
72	84	NA	85
NA	92	24	53
31	62	56	5

C. Colored values

Rasters

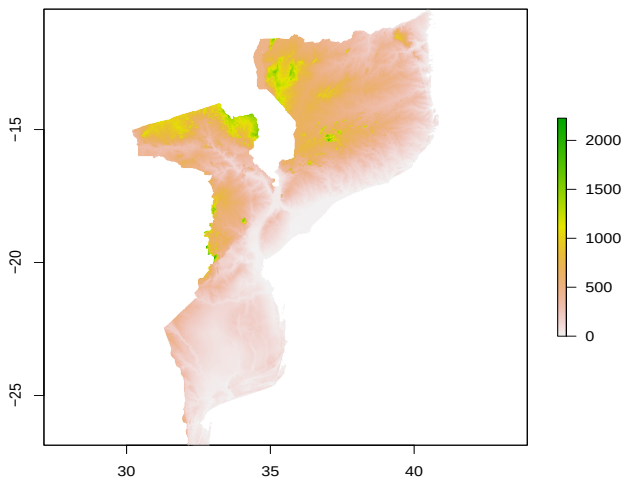
- However, in contrast to vector data, the cell of one raster layer can only hold a single value
- Value can be either continuous or categorical:



Raster Data in R

- The raster library can read many file types on it's own. For example, let's load nightlights data and elevation for Mozambique:

```
mozambique_nighlights <- raster("./data/mozambique_nightlights_10000m.t
mozambique_elev <- raster("./data/mozambique_elev_10000m.tif") # Source
plot(mozambique_elev)
```



Raster Data in R

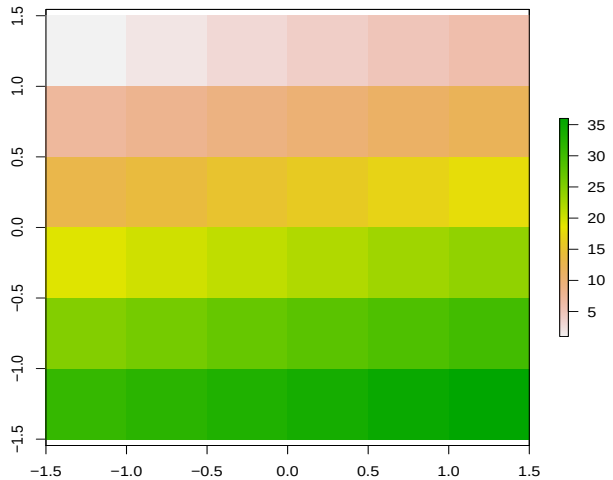
- rasters also have projections and can be re-projected
- Can use projectRaster() to re-project rasters

```
projection(mozambique_elev)

## [1] "+proj=longlat +datum=WGS84 +no_defs"

# If you want to re-project: projectRaster(mozambique_nighlights,crs =
```

Raster Data in R

- You can also subset specific cells from raster files. Since the elevation and nightlights rasters are very large, we can create a sample raster

```
elev = raster(nrows = 6, ncols = 6, res = 0.5,
              xmn = -1.5, xmx = 1.5, ymn = -1.5, ymx = 1.5,
              vals = 1:36)
plot(elev)
```



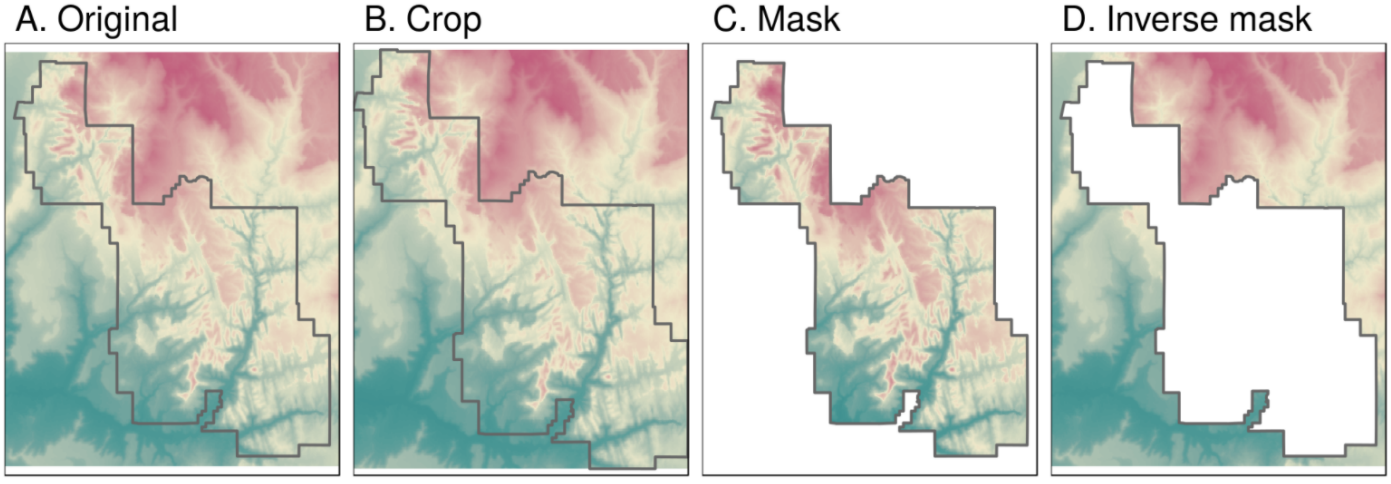
Raster Data in R

- You can also subset specific cells from raster files
- Since the elevation and nightlights rasters are very large, we can create a sample raster
- Referencing cells:

```
# row 1, column 1
elev[1, 1]

## [1] 1

# cell ID 1
elev[1]

## [1] 1
```

Rasters: Spatial Operations

- One of the most useful operations for rasters is to combine them with vectors
- Many possibilities, but we will cover some of the most useful ones
- A few quick options regarding re-sizing rasters, using crop(), mask(), and mask(,inverse=TRUE)



Rasters: Extraction

- But, *the* most useful operations for rasters when combined with vectors is extract
- = identifying and returning values associated with a 'target' raster at specific locations, based on a vector geographic 'selector' object
- Results depend on the type of selector used (points, lines or polygons) and arguments passed (functions)

Rasters: Extraction with Point Vectors

- Example: calculate the mean elevation at each DHS cluster
- The following command extracts elevation values from the raster and assigns the resulting vector to a new column (elevation) in the dhs points dataset

```
dhs$elevation <- raster::extract(mozambique_elev, dhs) # exact_extract(
```

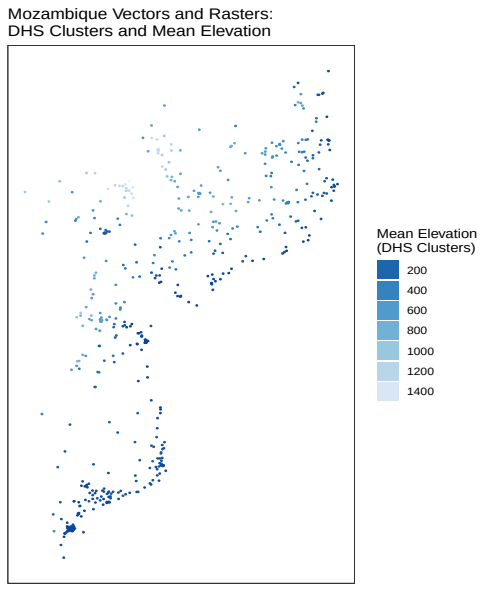
Rasters: Extraction with Polygon Vectors

- Also useful with polygons
- But, here we need a function for how to assign values
- extract from raster works well but it is a bit slow. Really fast option: exact_extract from exactextractr library
  - Handles re-projection too
- Example: calculate the mean elevation at a polygon buffer of 5km from each DHS cluster

```
library(exactextractr) # faster extract
dhs_buffer_5km <- st_buffer(dhs, dist=5000)
dhs_buffer_5km$mean_elevation <-
  exact_extract(mozambique_elev,
                dhs_buffer_5km,
                'mean')

##
  |
  |                                                      |   0%
```

Rasters: Extraction with Polygons - Map

```
library(scales) # for pretty_breaks
dhs_buffer_5km %>% ggplot() + geom_sf(aes(fill=mean_elevation),col=NA)
  scale_fill_distiller(palette="Blues",
                       breaks=pretty_breaks(n=6)) +
  map_theme +
  guides(fill = guide_legend(title="Mean Elevation\n(DHS Clusters)",
                             order=1)) + # \n adds a line break
  ggtitle("Mozambique Vectors and Rasters:\nDHS Clusters and Mean Eleva
```



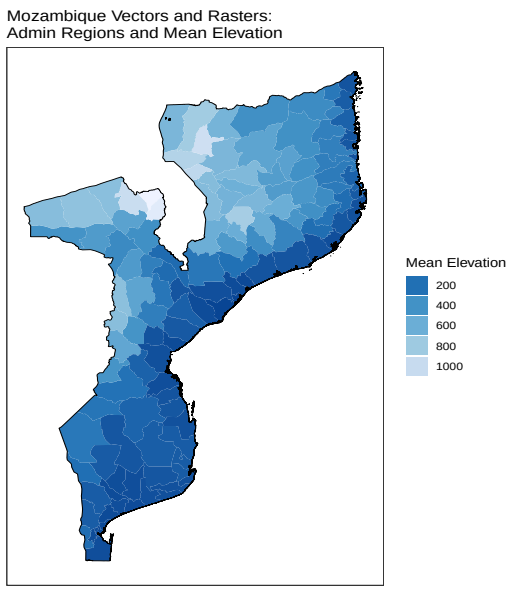
Rasters: Extraction with Polygon Vectors

- Exercise: Load Mozambique 2nd-level administrative areas (polygons), calculate average elevation in each area, and make a map:

```
mozambique_admin2 <- st_read("./data/gadm36_MOZ_shp/gadm36_MOZ_2.shp")

## Reading layer `gadm36_MOZ_2' from data source
##   `/Users/emontero/Dropbox/Teaching/PEoDev_spring2023/R_Assignment/R
##   using driver `ESRI Shapefile'
## Simple feature collection with 129 features and 13 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 30.21741 ymin: -26.86869 xmax: 40.83931 ymax: -
## Geodetic CRS:  WGS 84

mozambique_admin2$mozambique_elev <-
  exact_extract(mozambique_elev,
                mozambique_admin2,
                'mean')

##
```

Rasters: Extraction with Polygons - Map

```
library(scales)
mozambique_admin2 %>% ggplot() + geom_sf(aes(fill=mozambique_elev),col=
  scale_fill_distiller(palette="Blues",
                       breaks=pretty_breaks(n=6),
                       direction=-1) + # flip direction in scales
  map_theme +
  geom_sf(data=mozambique_admin0, fill=NA,col="black", size=0.25) +
  guides(fill = guide_legend(title="Mean Elevation",
                             order=1)) + # \n adds a line break
  ggtitle("Mozambique Vectors and Rasters:\nAdmin Regions and Mean Elev
```



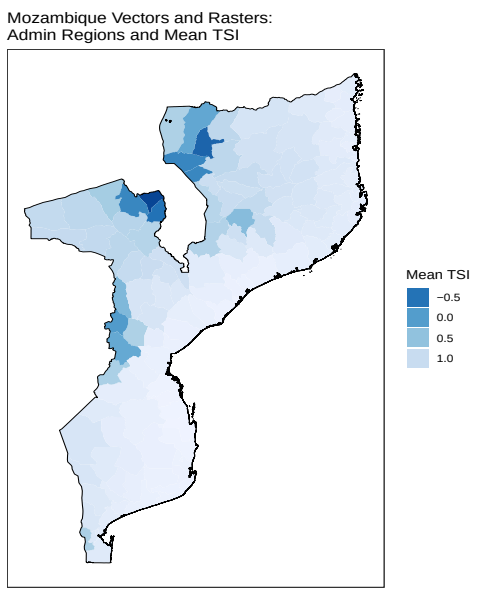
Rasters: Extraction with Polygon Vectors

- Exercise: Load the TseTse Suitability (TSI) raster file and calculate average TSI in each 2nd level admin unit:

```
tsi <- raster("./data/tsi.tif") # read data
mozambique_admin2$tsi <-
  exact_extract(tsi,
                mozambique_admin2,
                'mean')

##
  |
  |                                                      |   0%
  |
  |                                                      |   1%
  |
  |=                                                     |   2%
  |
  |==                                                    |   3%
  |
  |==                                                    |   4%
  |
  |==                                                    |   5%
```

Rasters: Extraction with Polygons - Map

```
# Make a map of the TSI across Mozambique:
mozambique_admin2 %>% ggplot() + geom_sf(aes(fill=tsi),col=NA) + # set
  scale_fill_distiller(palette="Blues",
                       breaks=pretty_breaks(n=6),
                       direction=-1) + # flip direction in scales
  map_theme +
  geom_sf(data=mozambique_admin0, fill=NA,col="black", size=0.25) +
  guides(fill = guide_legend(title="Mean TSI",
                             order=1)) + # \n adds a line break
  ggtitle("Mozambique Vectors and Rasters:\nAdmin Regions and Mean TSI"
```



Rasters: Extraction with Polygon Vectors

- Exercise: Load the nightlights raster file and calculate average nightlights in each 2nd level admin unit:

```
nl <- raster("./data/mozambique_nightlights_10000m.tif") # read data
mozambique_admin2$nl <-
  exact_extract(nl,
                mozambique_admin2,
                'mean')

##
  |
  |                                                      |   0%
  |
  |                                                      |   1%
  |
  |=                                                     |   2%
  |
  |==                                                    |   3%
  |
  |==                                                    |   4%
  |
  |==                                                    |   5%
```

Rasters: Extraction with Polygon Vectors

- Exercise: Explore the relationship between TseTse Suitability (TSI) and Nightlight Intensity across 2nd level admin units in Mozambique:
  - Due to large outliers in the nightlights data, use the log of nightlights: $\log(\text{nl} + 1)$
  - Control for mean elevation

```
reg <- lm(log(nl+1) ~ tsi + mozambique_elev,
          data=mozambique_admin2)
library(sandwich); library(lmtest)
coeftest(reg, vcov = vcovHC(reg, type="HC1"))

##
## t test of coefficients:
##
##                    Estimate  Std. Error t value Pr(>|t|)
## (Intercept)      0.32340284  0.14442193  2.2393  0.02689 *
## tsi             -0.14516946  0.07052737 -2.0583  0.04162 *
## mozambique_elev -0.00032387  0.00016325 -1.9839  0.04944 *
## ---
## Signif. codes:
```

Table of Contents

Conclusion & References

- This is meant only to introduce you to R's spatial capabilities, and highlight the most useful ones for development economics
- Again, I highly recommend the online (free) book "Geocomputation with R": Link
- It discussed many other things you can do with rasters: rasterize, rasterBricks, etc.
- It also discusses how to map suing tmap() library instead of ggplot(), and how to make interactive maps for websites

Thank you!