Session 6 - Spatial Data

R for Stata Users

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

March 2024

Table of contents

Overview of GIS conceps

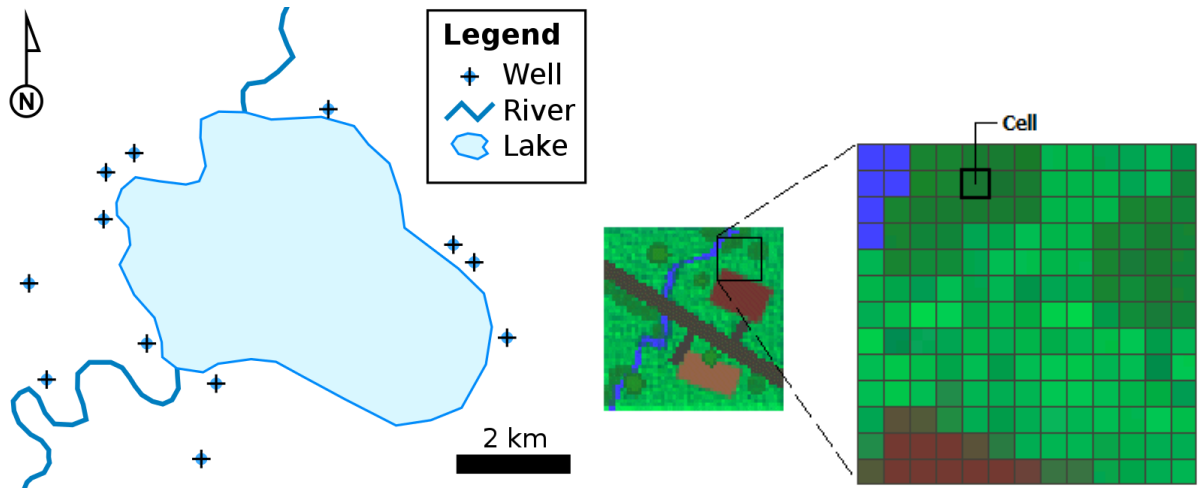
Spatial data: The two main types of spatial data are **vector data** and **raster data**

Vector data

- Points, lines, or polygons
- Common file formats include shapefiles (.shp) and geojsons (.geojson)
- Examples: polygons on countries, polylines of roads, points of schools

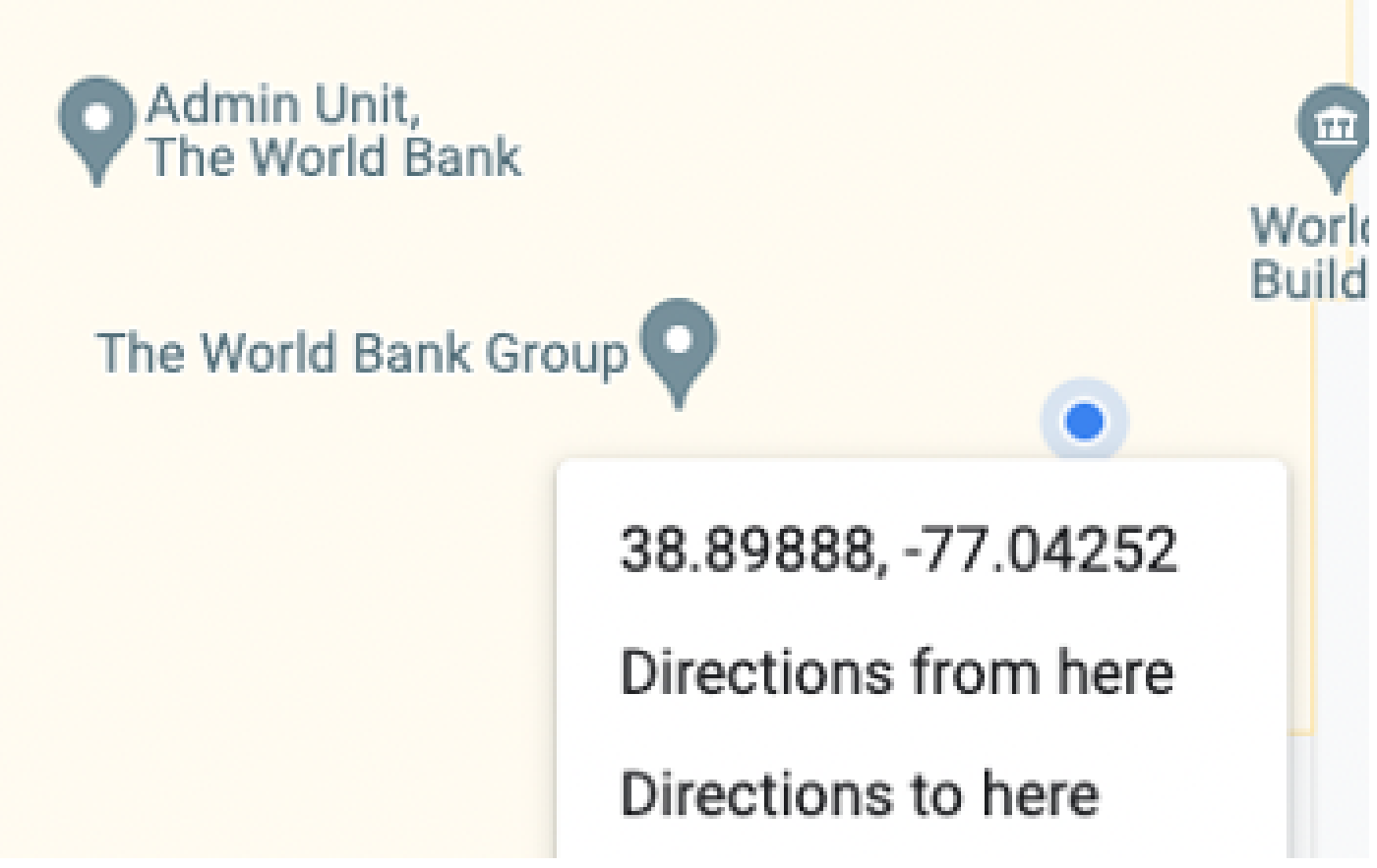
Raster data

- Spatially referenced grid
- Common file format is a geotif (.tif)
- Example: Satellite imagery of nighttime lights



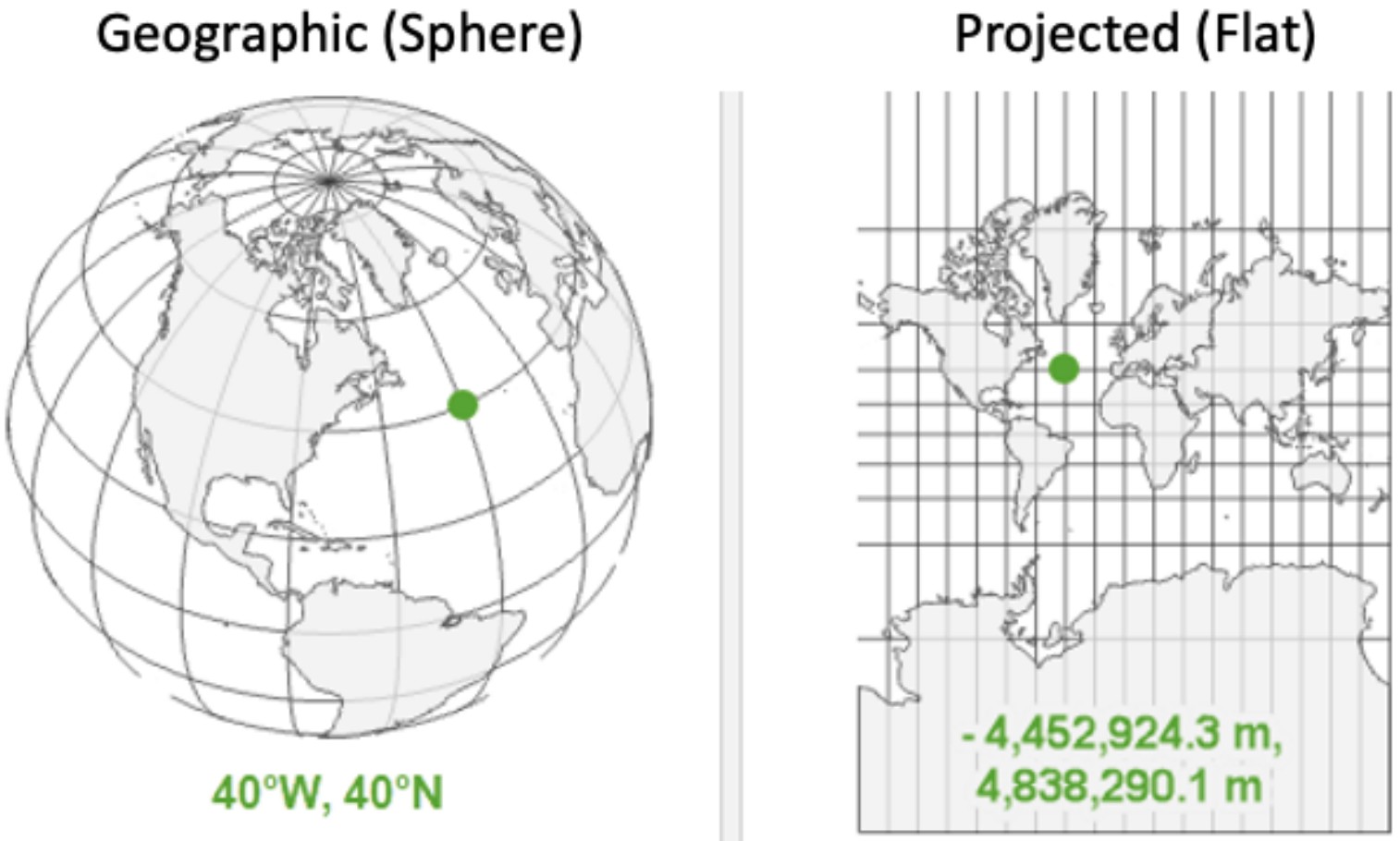
Coordinate Reference Systems (CRS)

- Coordinate reference systems use pairs of numbers to define a location on the earth
- For example, the World Bank is at a latitude of 38.89 and a longitude of -77.04



Coordinate Reference Systems (CRS)

There are many different coordinate reference systems, which can be grouped into **geographic** and **projected** coordinate reference systems. Geographic systems live on a sphere, while projected systems are “projected” onto a flat surface.

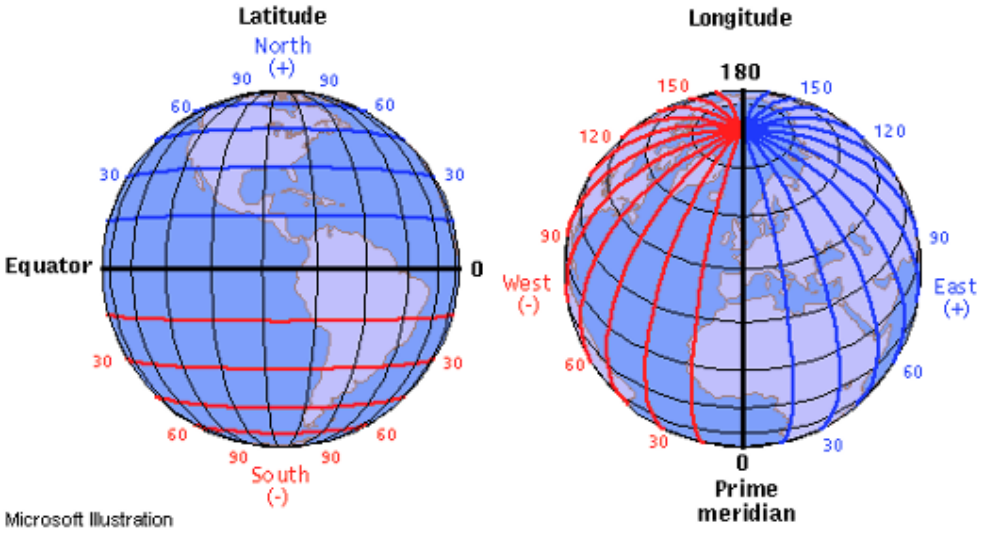


Geographic Coordinate Systems

Units: Defined by latitude and longitude, which measure angles and units are typically in decimal degrees. (Eg, angle is latitude from the equator).

Latitude & Longitude:

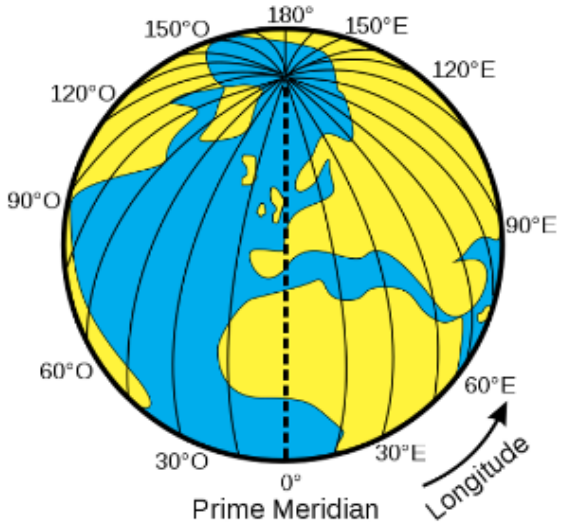
- On a grid X = longitude, Y = latitude; sometimes represented as (longitude, latitude).
- Also has become convention to report them in alphabetical order: (latitude, longitude) — such as in Google Maps.
- Valid range of latitude: -90 to 90
- Valid range of longitude: -180 to 180
- **{Tip}** Latitude sounds (and looks!) like latter.

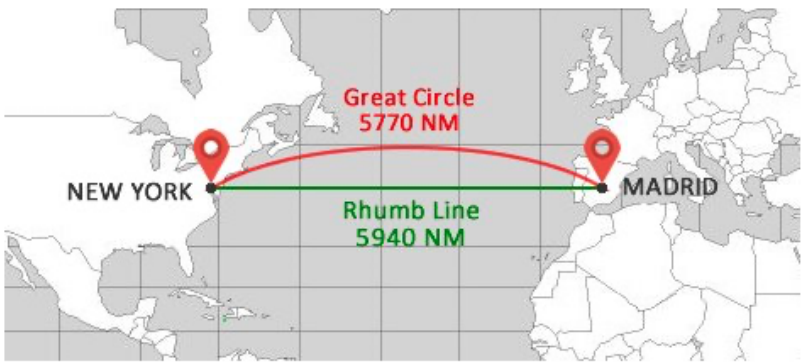


Geographic Coordinate Systems

Distance on a sphere

- At the equator (latitude = 0), a 1 decimal degree longitude distance is about 111km; towards the poles (latitude = -90 or 90), a 1 decimal degree longitude distance converges to 0 km.
- We must be careful (ie, use algorithms that account for a spherical earth) to calculate distances! The distance along a sphere is referred to as a great circle distance.
- Multiple options for spherical distance calculations, with trade-off between accuracy & complexity. (See distance section for details).





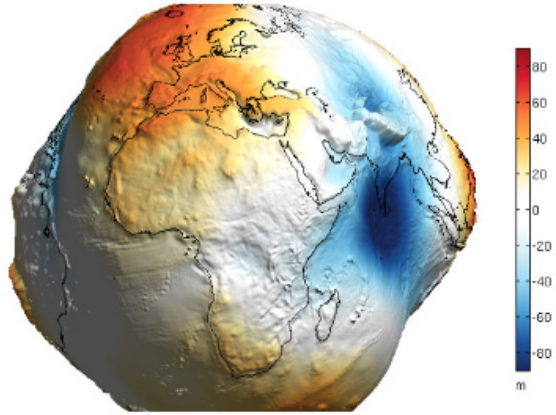
Geographic Coordinate Systems

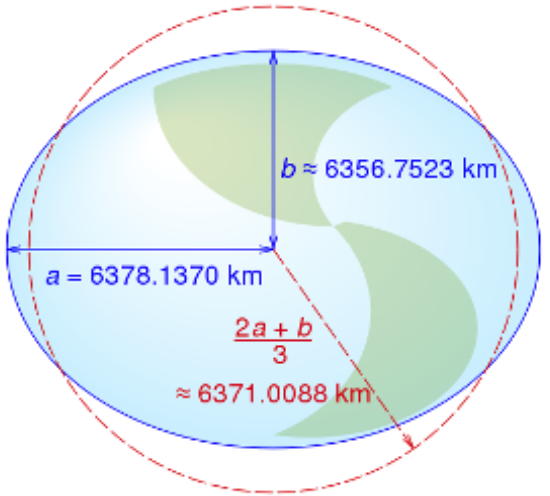
Datums

- **Is the earth flat?** No!
- **Is the earth a sphere?** No!
- **Is the earth a lumpy ellipsoid?** Yes!

The earth is a lumpy ellipsoid, a bit flattened at the poles.

- A datum is a model of the earth that is used in mapping. One of the most common datums is WGS 84, which is used by the Global Positional System (GPS).
- A datum is a reference ellipsoid that approximates the shape of the earth.
- Other datums exist, and the latitude and longitude values for a specific location will be different depending on the datum.





Projected Coordinate Systems

Projected coordinate systems project spatial data from a 3D to 2D surface.

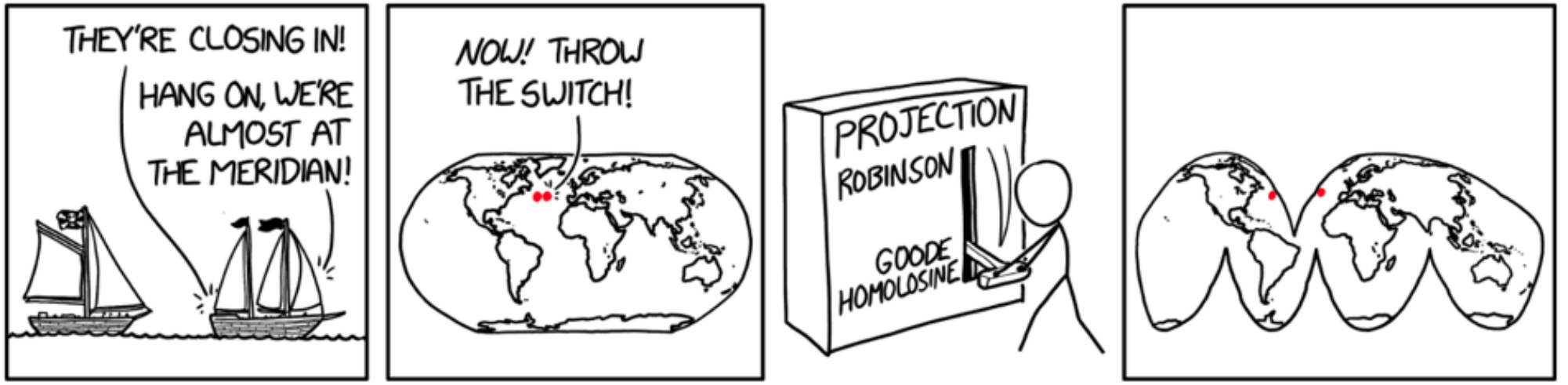
Distortions: Projections will distort some combination of distance, area, shape or direction. Different projections can minimize distorting some aspect at the expense of others.

Units: When projected, points are represented as "northings" and "eastings." Values are often represented in meters, where northings/eastings are the meter distance from some reference point. Consequently, values can be very large!

Datums still relevant: Projections start from some representation of the earth. Many projections (eg, UTM) use the WGS84 datum as a starting point (ie, reference datum), then project it onto a flat surface.

Click here to see why Toby & CJ are confused (hint: projections!)

Projected Coordinate Systems



Referencing coordinate reference systems

- There are many ways to reference coordinate systems, some of which are verbose.
- **PROJ** (Library for projections) way of referencing WGS84 `+proj=longlat +datum=WGS84 +no_defs +type=crs`
- **EPSG** Assigns numeric code to CRSs to make it easier to reference. Here, WGS84 is `4326`.

Coordinate Reference Systems

Whenever have spatial data, need to know which coordinate reference system (CRS) the data is in.

- You wouldn't say **"I am 5 away"**
- You would say **"I am 5 [miles / kilometers / minutes / hours] away"** (units!)
- Similarly, a "complete" way to describe location would be: I am at **6.51 latitude, 3.52 longitude using the WGS 84 CRS**

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, vector data.
- This is the most common type of spatial data that non-GIS experts will encounter in their work.
- We will use the `sf` package, which is the tidyverse-compatible package for geospatial data in R.
- For visualizing, we'll rely on `ggplot2` for static maps and `leaflet` for interactive maps

Setup

1. Copy/paste the following code into a new RStudio script, **replacing "YOURFOLDERPATHHERE" with the folder within which you'll place this R project**:

```
library(usethis)
use_course(
  url = "https://github.com/worldbank/dime-r-training/archive/main.zip",
  destdir = "YOURFOLDERPATHHERE"
)
```

2. In the console, type in the requisite number to delete the .zip file (we don't need it anymore).

3. A new RStudio environment will open. Use this for the session today.

Setup

Install new packages

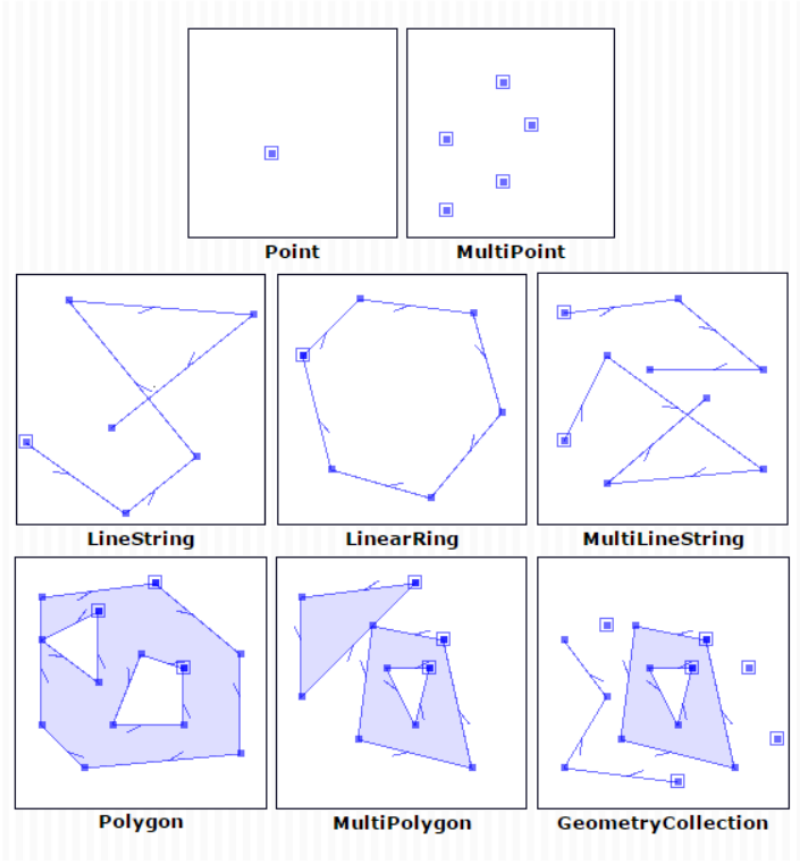
```
install.packages(c("sf",
                   "leaflet",
                   "geosphere"),
                 dependencies = TRUE)
```

And load them

```
library(here)
library(tidyverse)
library(sf)         # Simple features
library(leaflet)    # Interactive map
library(geosphere) # Great circle distances
```

Load and explore polylines, polylines, and points

The main package we'll rely on is the `sf` (simple features) package. With `sf`, spatial data is structured similarly to a **dataframe**; however, each row is associated with a **geometry**. Geometries can be one of the below types.



Load and explore polygon

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

```
nga_sf <-
  st_read(here("DataWork",
               "DataSets",
               "Final",
               "gadm_nga_2.geojson"))
```

```
## Reading layer `gadm_nga_2' from data source
##   `C:\WBG\Repos\dime-r-training\DataWork\DataSets\Final\gadm_nga_2.geojson'
##   using driver `GeoJSON'
## Simple feature collection with 775 features and 12 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2.668431 ymin: 4.270418 xmax: 14.67642 ymax: 13.89201
## Geodetic CRS:  WGS 84
```

Exploring the data

Look at first few observations

```
head(nga_sf)
```

```
## Simple feature collection with 6 features and 12 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 7.291035 ymin: 5.037226 xmax: 7.962045 ymax: 5.837777
## Geodetic CRS:  WGS 84
##   ID_0 COUNTRY NAME_1 NL_NAME_1      ID_2            NAME_2 VARNAME_2 NL_NAME_2
## 1  NGA Nigeria   Abia           NGA.1.1_1         Aba North
## 2  NGA Nigeria   Abia           NGA.1.2_1         Aba South
## 3  NGA Nigeria   Abia           NGA.1.3_1          Arochukw Arochukwu
## 4  NGA Nigeria   Abia           NGA.1.4_1             Bende
## 5  NGA Nigeria   Abia           NGA.1.5_1           Ikwuano
## 6  NGA Nigeria   Abia           NGA.1.6_1 Isiala Ngwa North
##            TYPE_2       ENGTYPE_2 CC_2 HASC_2                        geometry
## 1 Local Authority Local Authority            MULTIPOLYGON (((7.378114 5....
## 2 Local Authority Local Authority            MULTIPOLYGON (((7.314784 5....
## 3 Local Authority Local Authority            MULTIPOLYGON (((7.641509 5....
## 4 Local Authority Local Authority            MULTIPOLYGON (((7.635005 5....
```

Exploring the data

Number of rows

```
nrow(nga_sf)
```

```
## [1] 775
```

Exploring the data

Check coordinate reference system

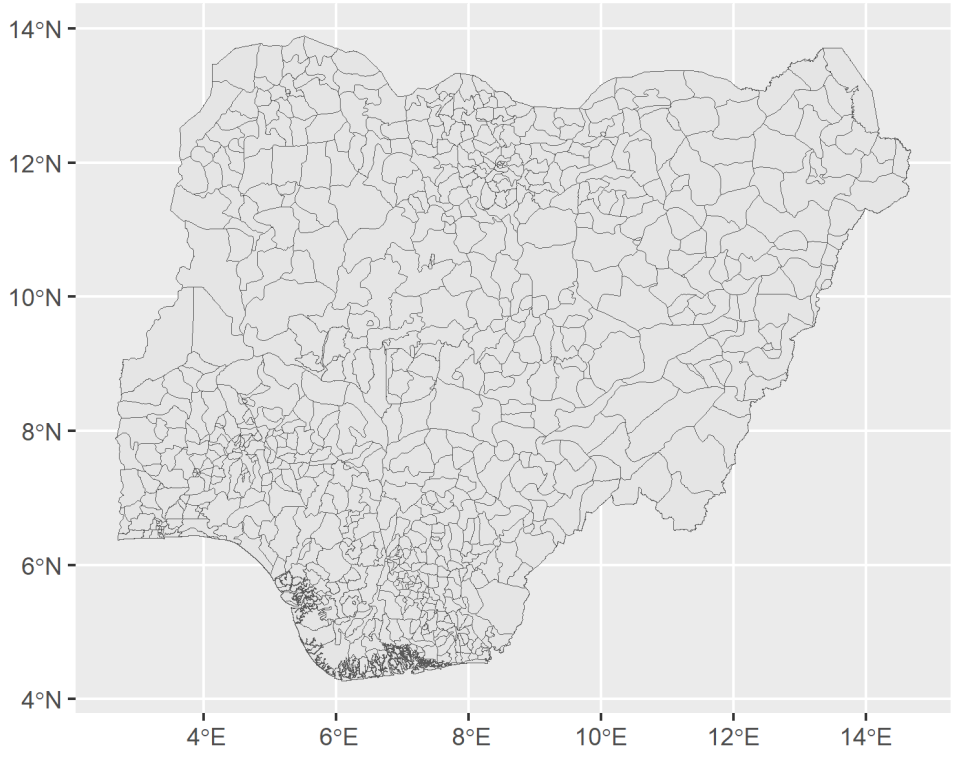
```
st_crs(nga_sf)
```

```
## Coordinate Reference System:
##   User input: WGS 84
##   wkt:
## GEOGCRS["WGS 84",
##     DATUM["World Geodetic System 1984",
##         ELLIPSOID["WGS 84",6378137,298.257223563,
##             LENGTHUNIT["metre",1]]],
##     PRIMEM["Greenwich",0,
##         ANGLEUNIT["degree",0.0174532925199433]],
##     CS[ellipsoidal,2],
##         AXIS["geodetic latitude (Lat)",north,
##             ORDER[1],
##             ANGLEUNIT["degree",0.0174532925199433]],
##         AXIS["geodetic longitude (Lon)",east,
##             ORDER[2],
##             ANGLEUNIT["degree",0.0174532925199433]],
##     ID["EPSG",4326]]
```

Exploring the data

Plot the data. To plot using `ggplot2`, we use the `geom_sf` geometry.

```
ggplot() +
  geom_sf(data = nga_sf)
```



Attributes of data

We want the area of each location, but we don't have a variable for area

```
names(nga_sf)
```

```
##  [1] "ID_0"      "COUNTRY"   "NAME_1"    "NL_NAME_1" "ID_2"      "NAME_2"
##  [7] "VARNAME_2" "NL_NAME_2" "TYPE_2"    "ENGTYPE_2" "CC_2"      "HASC_2"
## [13] "geometry"
```

Attributes of data

Determine area. Note the CRS is spherical (WGS84), but `st_area` gives area in meters squared. R uses s2 geomety for this.

```
st_area(nga_sf)
```

```
## Units: [m^2]
##   [1]    18115918    51130486   386126742   603925481   237940893   253475003
##   [7]   261346460   336386622   374567649   568416816   192717817   106283762
##  [13]   246502187   293336257   451959706   231141018   131269420  2289574417
##  [19]  4268043310  1884615977  1865638766  1084336031   699838559  2732234473
##  [25]  1725588389  1167047612   585132151   508809998  2175752242   742913573
##  [31]   654421163   209951938   813298273  1605421682  4324634661  4584293228
##  [37]   139996668   370270051   177485439    96405062   189114803   139599636
##  [43]   299011048   193538372   171492891   210962698   175747588   316253840
##  [49]   130882266   269754902   334120006   133544979   473185832   255878306
##  [55]   243461164   346695186   131733298   119971004   204247037   232632892
##  [61]   275252661   151590665    81667384   470353728    53736209   279593007
##  [67]   291550813   127666912   188377399   162798931   380294600   743228061
##  [73]   105322234   389635592   171446836   570044307    58595577   107371646
##  [79]    87655555   111134295   222332919    94111356    64805084   168774033
##  [85]   398905868    28143090    17159027   364759033   231919200   131681287
##  [91]  6133937818  3976854006   653949021   988289702  2498187950   528634061
```

Operations similar to dataframes

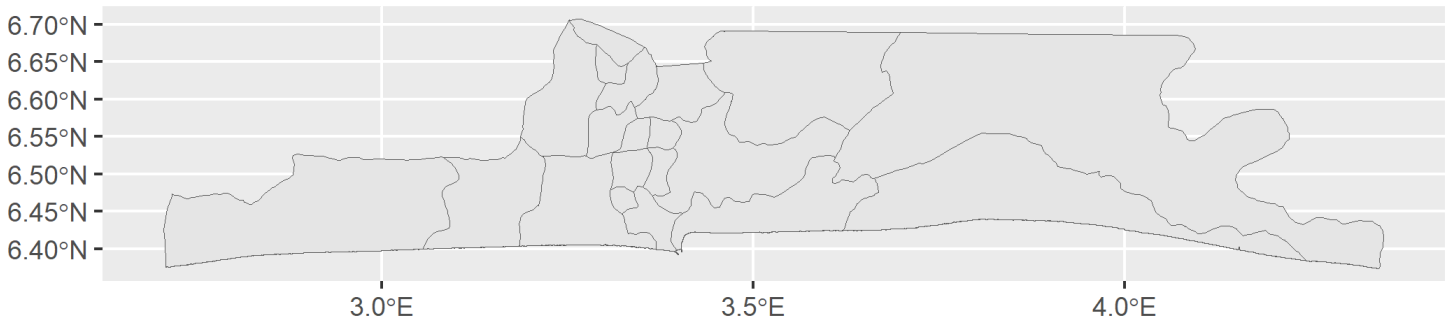
Create new dataset that is just lagos

```
lagos_sf <- nga_sf %>%
  filter(NAME_1 == "Lagos")
```

Operations similar to dataframes

Plot the dataframe

```
ggplot() +
  geom_sf(data = lagos_sf)
```



Load and explore polyline

Exercise:

- Load the roads data `osm_lagos_roads.geojson` and name the object `roads_sf`
- Look at the first few observations
- Check the coordinate reference system
- Map the polyline

Solution:

```
roads_sf <- st_read(here("DataWork", "DataSets", "Final", "osm_lagos_roads.geojson"))

head(roads_sf)

st_crs(roads_sf)

ggplot() +
  geom_sf(data = roads_sf)
```

02:00

Load and explore polyline

```
roads_sf <-
  st_read(here("DataWork",
               "DataSets",
               "Final",
               "osm_lagos_roads.geojson"))
```

```
## Reading layer `osm_lagos_roads' from data source
##   `C:\WBG\Repos\dime-r-training\DataWork\DataSets\Final\osm_lagos_roads.geojson'
##   using driver `GeoJSON'
## Simple feature collection with 3007 features and 34 fields
## Geometry type: LINESTRING
## Dimension:     XY
## Bounding box:  xmin: 2.705985 ymin: 6.379386 xmax: 4.217506 ymax: 6.832435
## Geodetic CRS:  WGS 84
```

```
ggplot() +
  geom_sf(data = roads_sf)
```

Load and explore polyline

Exercise: Determine length of each line (hint: use `st_length`)

Solution:

```
st_length(roads_sf)
```

```
## Units: [m]
##    [1]   794.082050   238.905024 33555.236573  3528.646684    78.305014
##    [6]  1906.248364   101.261589   625.342359   130.178118   335.860392
##   [11]    63.894351    29.269625   336.744791    64.698424  5675.070068
##   [16]   811.315023    60.027144 13110.313140   488.358816  2547.259226
##   [21]    66.730590    56.824559  1372.839939 14361.919350 16510.001084
##   [26]  4298.850486   991.619944   180.578417    66.128111   664.387693
##   [31]   623.150141    39.472571   117.079424    51.801118    46.843432
##   [36]    44.336573    11.152053  1750.478856  2135.870227  3854.862119
##   [41]  3858.886958  2681.089557  2689.057651  1915.872778  5485.677899
##   [46]  1506.045111  1499.337941  1884.624020   192.656690  5491.550596
##   [51]  3553.259716  3546.534051    71.313887  2138.421303   671.348526
##   [56]  1793.092780    32.819448   189.943926   193.794458   312.942975
```

01:00

Load and explore point data

We'll load a dataset of the location of schools in Lagos

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

```
## Rows: 429 Columns: 33
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr  (28): element_type, amenity, min_age, name, opening_hours, wheelchair, ...
## dbl   (4): osmid, addr:postcode, latitude, longitude
## date  (1): survey:date
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Explore data

```
head(schools_df)
```

```
## # A tibble: 6 × 33
##   element_type      osmid amenity min_age name  opening_hours wheelchair website
##   <chr>             <dbl> <chr>   <chr>   <chr> <chr>         <chr>      <chr>
## 1 node           8.53e 9 school  3yrs    St. … Mo-Fr 07:45 … yes        <NA>
## 2 way            3.79e 8 school  <NA>    Alta… <NA>          <NA>       https:…
## 3 way            5.59e 8 school  <NA>    Nean… <NA>          <NA>       <NA>
## 4 node           9.55e 9 school  <NA>    Bich… <NA>          <NA>       http:/…
## 5 node           1.11e10 school  <NA>    Chri… <NA>          <NA>       <NA>
## 6 node           1.11e10 school  <NA>    Cran… <NA>          <NA>       <NA>
## # ℹ 25 more variables: wikidata <chr>, wikipedia <chr>, `addr:city` <chr>,
## #   `addr:postcode` <dbl>, `addr:street` <chr>, email <chr>, phone <chr>,
## #   building <chr>, grades <chr>, `addr:housenumber` <chr>,
## #   official_name <chr>, short_name <chr>, `survey:date` <date>,
## #   description <chr>, source <chr>, `name:en` <chr>, operator <chr>,
## #   religion <chr>, denomination <chr>, `operator:type` <chr>,
## #   `isced:level` <chr>, landuse <chr>, osm_type <chr>, latitude <dbl>, …
```

Explore data

```
names(schools_df)
```

```
##  [1] "element_type"     "osmid"            "amenity"          "min_age"
##  [5] "name"             "opening_hours"    "wheelchair"       "website"
##  [9] "wikidata"         "wikipedia"        "addr:city"        "addr:postcode"
## [13] "addr:street"      "email"            "phone"            "building"
## [17] "grades"           "addr:housenumber" "official_name"    "short_name"
## [21] "survey:date"      "description"      "source"           "name:en"
## [25] "operator"         "religion"         "denomination"     "operator:type"
## [29] "isced:level"      "landuse"          "osm_type"         "latitude"
## [33] "longitude"
```

Convert to spatial object

We define the (1) coordinates (longitude and latitude) and (2) CRS. **Note:** We must determine the CRS from the data metadata. This dataset comes from OpenStreetMaps, which uses WGS:4326.

Assigning the incorrect CRS is one of the most common sources of issues I see with geospatial work. If something looks weird, check the CRS!

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

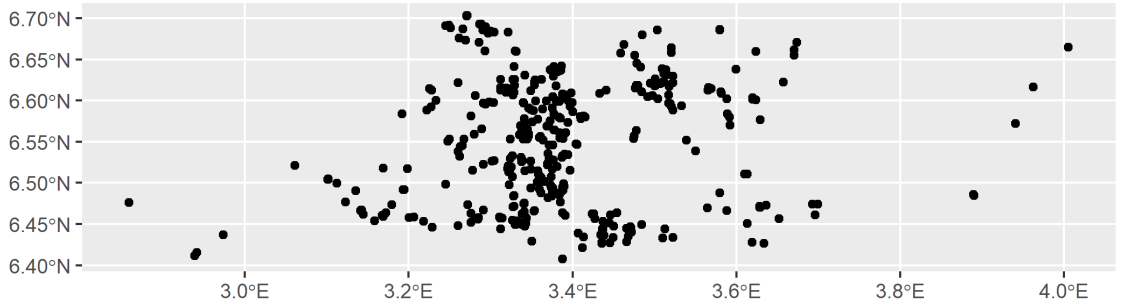
Convert to spatial object

```
head(schools_sf$geometry)
```

```
## Geometry set for 6 features
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: 3.62856 ymin: 6.470489 xmax: 4.0054 ymax: 6.6652
## Geodetic CRS:  WGS 84
## First 5 geometries:

## POINT (4.0054 6.6652)

## POINT (3.96229 6.616529)

## POINT (3.940881 6.572413)

## POINT (3.692594 6.474112)

## POINT (3.629154 6.470489)
```

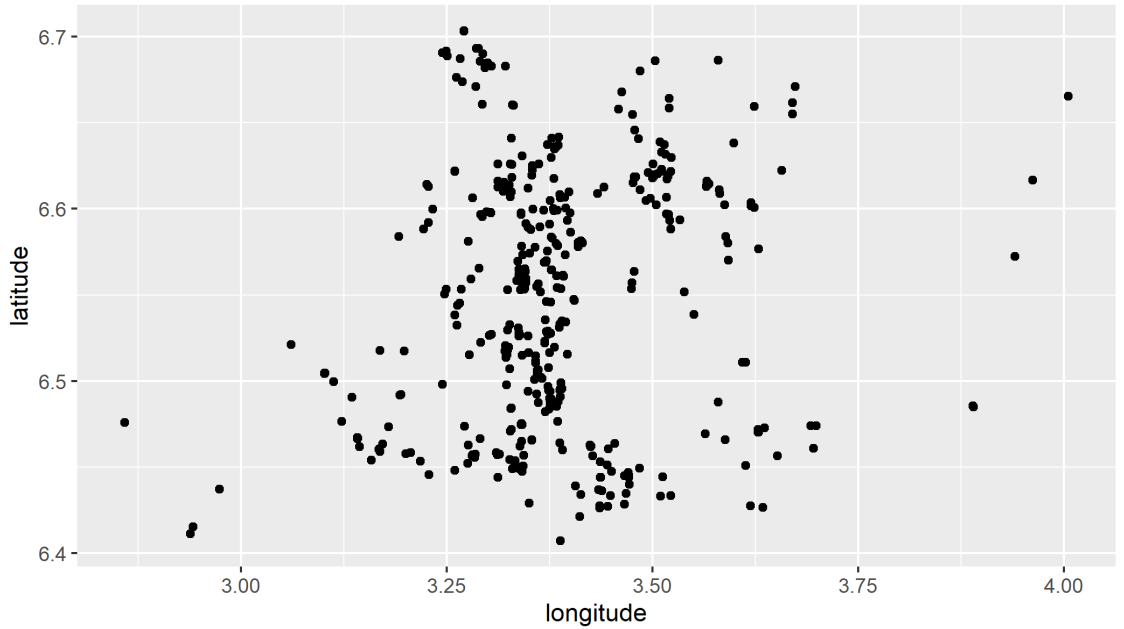
Map points object: Using sf

```
ggplot() +
  geom_sf(data = schools_sf)
```



Map points object: Using dataframe

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



Map points objects

Question: Why do the maps look different? Map using `sf` looks a bit squished!

Solution:

- Units are in decimal degrees.
- The length (eg, meters) between degrees of longitude shrinks as we go towards the equator.
- `geom_point` thinks 1 degree distance is the same for latitudes and longitudes (dumb!).
- `geom_sf` knows that we are in `WGS:4326` (spherical CRS), and adjusts map to minimize distortions (smart!).

00:30

Make better static map

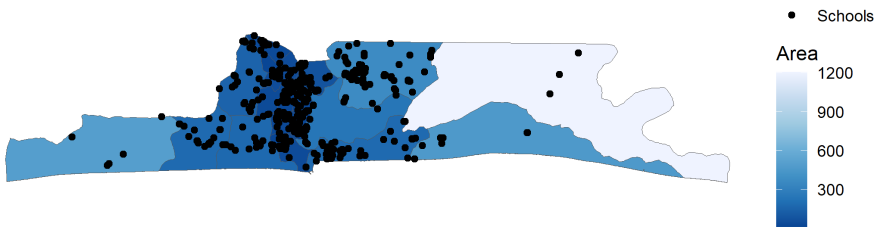
Lets make a better static map.

```
# Adding a variable with squared km
lagos_sf <- lagos_sf %>%
  mutate(area_m = lagos_sf %>% st_area() %>% as.numeric(),
         area_km = area_m / 1000^2)

# Plotting
ggplot() +
  geom_sf(data = lagos_sf,
          aes(fill = area_km)) +
  labs(fill = "Area") +
  scale_fill_distiller(palette = "Blues") +
  theme_void()
```

Make better static map

Lets add another spatial layer

```
ggplot() +
  geom_sf(data = lagos_sf,
          aes(fill = area_km)) +
  geom_sf(data = schools_sf,
          aes(color = "Schools")) +
  labs(fill = "Area",
       color = NULL) +
  scale_fill_distiller(palette = "Blues") +
  scale_color_manual(values = "black") +
  theme_void()
```



Interactive map

We use the `leaflet` package to make interactive maps. Leaflet is a JavaScript library, but the `leaflet` R package allows making interactive maps using R. Use of leaflet somewhat mimics how we use ggplot.

- Start with `leaflet()` (instead of `ggplot()`)
- Add spatial layers, defining type of layer (similar to geometries)

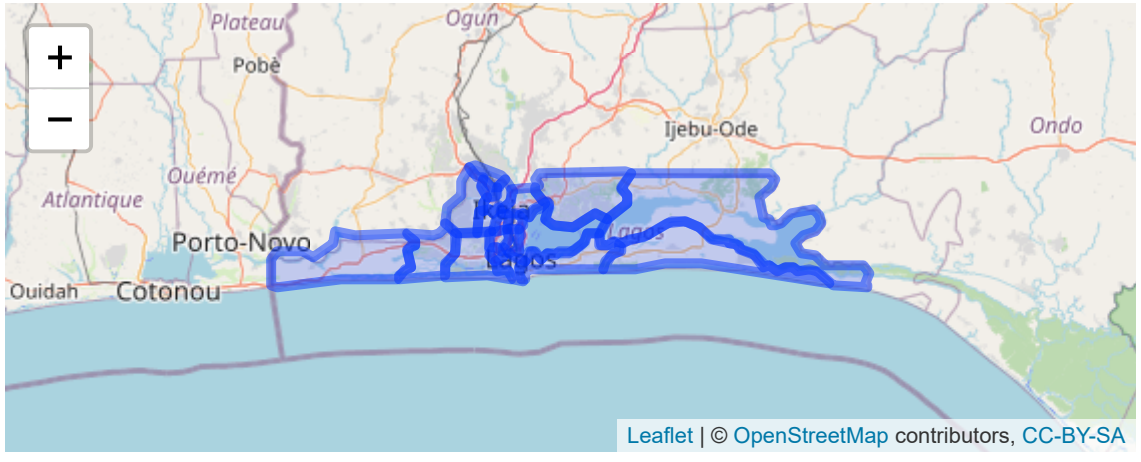
```
leaflet() %>%
  addTiles() # Basemap
```

Interactive map

We use the `leaflet` package to make interactive maps. Leaflet is a JavaScript library, but the `leaflet` R package allows making interactive maps using R. Use of leaflet somewhat mimics how we use ggplot.

- Start with `leaflet()` (instead of `ggplot()`)
- Add spatial layers, defining type of layer (similar to geometries)

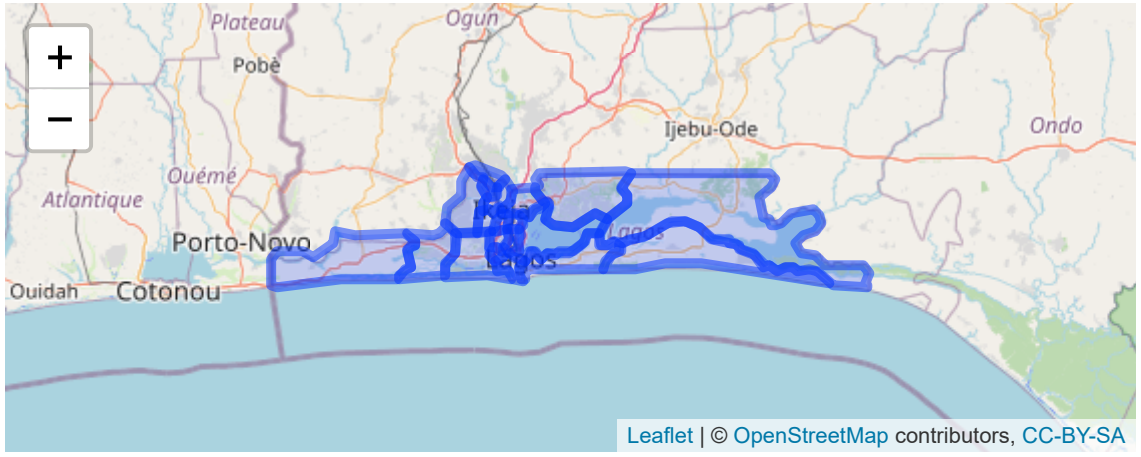
```
leaflet() %>%
  addTiles() %>%
  addPolygons(data = lagos_sf)
```



Interactive map

Add a pop-up

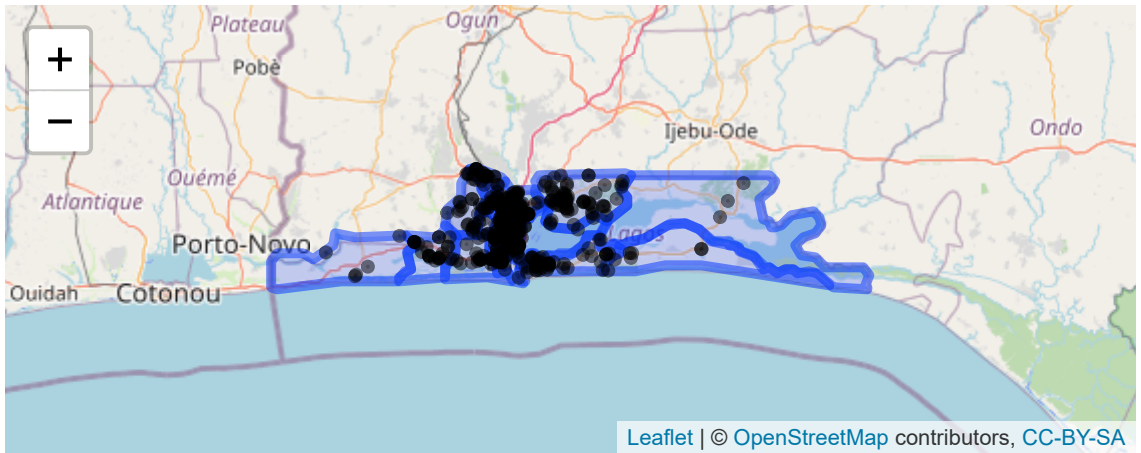
```
leaflet() %>%
  addTiles() %>%
  addPolygons(data = lagos_sf,
              popup = ~NAME_2)
```



Interactive map

Add more than one layer

```
leaflet() %>%
  addTiles() %>%
  addPolygons(data = lagos_sf,
              popup = ~NAME_2) %>%
  addCircles(data = schools_sf,
             popup = ~name,
             color = "black")
```



Interactive map of roads

Exercise: Create a leaflet map with roads, using the `roads_sf` dataset. (**Hint:** Use `addPolylines()`)

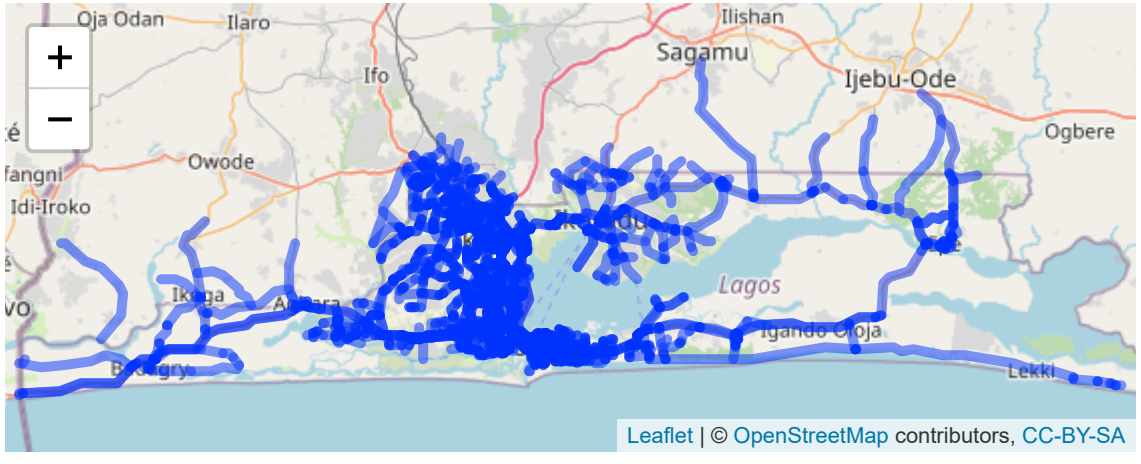
Solution:

```
leaflet() %>%
  addTiles() %>%
  addPolylines(data = roads_sf)
```

02:00

Interactive map of roads

```
leaflet() %>%
  addTiles() %>%
  addPolylines(data = roads_sf)
```



Interactive maps

We can spent lots of time going over what we can done with leaflet - but that would take up too much time. This resource provides helpful tutorials for things like:

- Changing the basemap
- Adding colors
- Adding a legend
- And much more!

Spatial operations applied on single dataset

- `st_transform`: Transform CRS
- `st_buffer`: Buffer point/line/polygon
- `st_combine`: Dissolve by attribute
- `st_convex_hull`: Create convex hull
- `st_centroid`: Create new sf object that uses the centroid
- `st_drop_geometry`: Drop geometry; convert from sf to dataframe
- `st_coordinates`: Get matrix of coordinates
- `st_bbox`: Get bounding box

Transform CRS

We want to compute the length of roads in Lagos using the roads dataset. The roads dataset is currently in a geographic CRS (WGS84), where the units are in decimal degrees. We'll tranform the CRS to a projected CRS that is suitable for Nigeria (EPSG:32632), and where the units will be in meters.

Note that coordinate values are large! Values are large because units are in meters. Large coordinate values suggest projected CRS; latitude is between -90 and 90 and longitude is between -180 and 180.

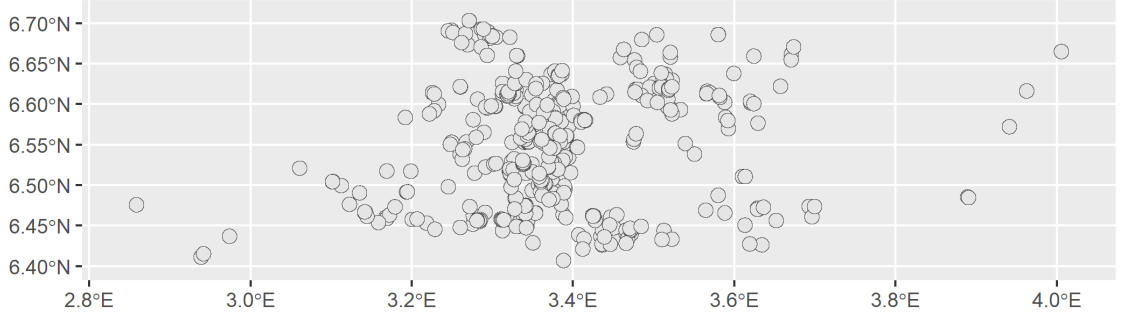
```
schools_utm_sf <- st_transform(schools_sf, 32632)

schools_utm_sf$geometry %>% head(2) %>% print()
```

```
## Geometry set for 2 features
## Geometry type: POINT
## Dimension:     XY
## Bounding box:  xmin: -57566.51 ymin: 734187.7 xmax: -52728.69 ymax: 739539.7
## Projected CRS: WGS 84 / UTM zone 32N

## POINT (-52728.69 739539.7)

## POINT (-57566.51 734187.7)
```

Buffer

We have the points of schools. Now we create a 1km buffer around schools.

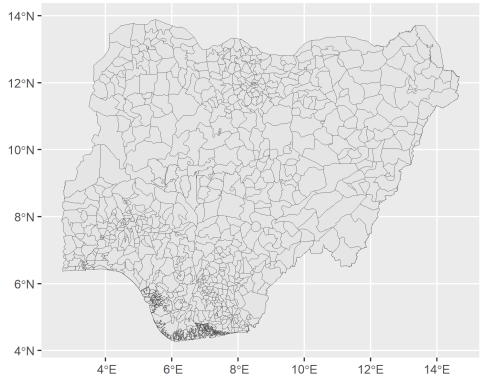
```
schools_1km_sf <- schools_sf %>%
  st_buffer(dist = 1000) # Units are in meters. Thanks s2!

ggplot() +
  geom_sf(data = schools_1km_sf)
```



Dissolve by an attribute

Below we have the second administrative regions of Nigeria. Using this dataset, let's create a new object at the first administrative region level.

```
nga1_sf <- nga_sf %>%
  group_by(NAME_1) %>%
  summarise(geometry = st_combine(geometry)) %>%
  ungroup()

ggplot() +
  geom_sf(data = nga1_sf)
```

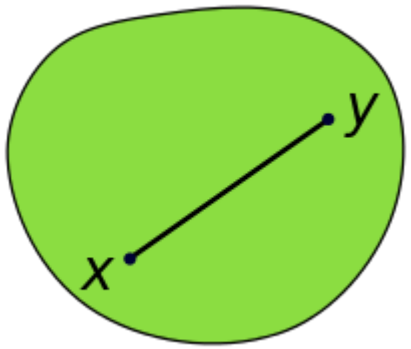


Convex Hull

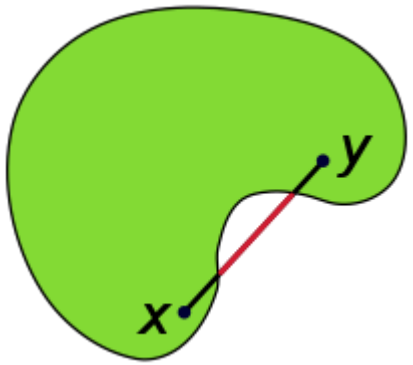
Simple definition: Get the outer-most coordinates of a shape and connect-the-dots.

Formal definition: A convex hull of a shape the smallest "convex set" that contains it. (A convex set is where a straight line can be drawn anywhere in the space and the space fully contains the line).

Convex



Not convex



Source: Wikipedia

Convex hull

In the below example, we create a conex hull around schools; creating a polygon that includes all schools.

Incorrect attempt

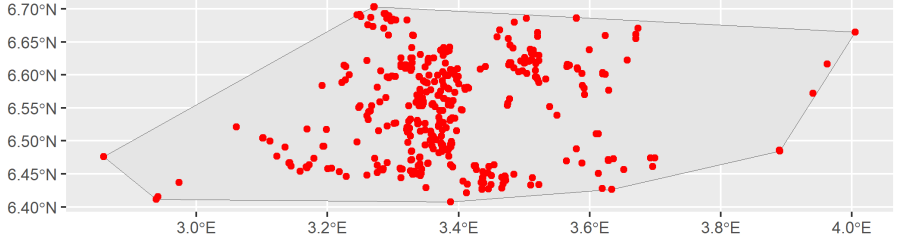
```
schools_chull1_sf <- schools_sf %>%
  st_convex_hull()

nrow(schools_chull1_sf)
```

```
## [1] 429
```

Convex hull

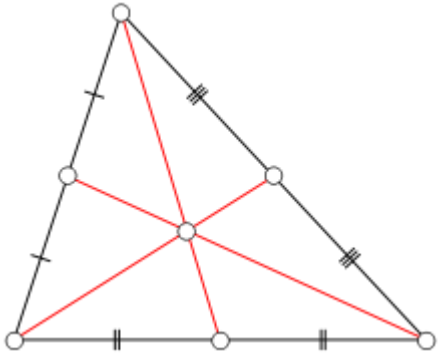
Correct

```
schools_chull2_sf <- schools_sf %>%
  summarise(geometry = st_combine(geometry)) %>%
  st_convex_hull()

ggplot() +
  geom_sf(data = schools_chull2_sf) +
  geom_sf(data = schools_sf, color = "red")
```



Determine centroid

Sometimes we want to represent a polygon or polyline as a single point. For this, we can compute the centroid (ie, geographic center) of a polygon/polyline.



Source: Wikipedia

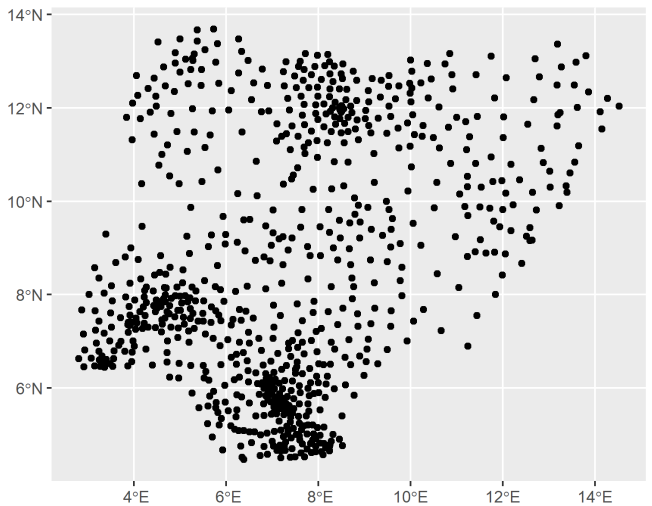
Determine centroid

Determine centroid of Nigeria ADM2s

```
nga_c_sf <- st_centroid(nga_sf)
```

```
## Warning: st_centroid assumes attributes are constant over geometries
```

```
ggplot() +
  geom_sf(data = nga_c_sf)
```



Remove geometry

Incorrect approach

```
lagos_sf %>%
  select(-geometry) %>%
  head()
```

```
## Simple feature collection with 6 features and 14 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2.70628 ymin: 6.375416 xmax: 3.404933 ymax: 6.705109
## Geodetic CRS:  WGS 84
##   ID_0 COUNTRY NAME_1 NL_NAME_1       ID_2           NAME_2 VARNAME_2 NL_NAME_2
## 1  NGA Nigeria  Lagos           NGA.25.1_1            Agege
## 2  NGA Nigeria  Lagos           NGA.25.2_1 Ajeromi/Ifelodun
## 3  NGA Nigeria  Lagos           NGA.25.3_1         Alimosho
## 4  NGA Nigeria  Lagos           NGA.25.4_1     Amuwo Odofin
## 5  NGA Nigeria  Lagos           NGA.25.5_1            Apapa
## 6  NGA Nigeria  Lagos           NGA.25.6_1         Badagary   Badagry
##            TYPE_2       ENGTYPE_2 CC_2 HASC_2    area_m   area_km
## 1 Local Authority Local Authority             16575387  16.57539
## 2 Local Authority Local Authority             11643619  11.64362
```

Remove geometry

Correct

```
lagos_sf %>%
  st_drop_geometry() %>%
  head()
```

```
##   ID_0 COUNTRY NAME_1 NL_NAME_1       ID_2           NAME_2 VARNAME_2 NL_NAME_2
## 1  NGA Nigeria  Lagos           NGA.25.1_1            Agege
## 2  NGA Nigeria  Lagos           NGA.25.2_1 Ajeromi/Ifelodun
## 3  NGA Nigeria  Lagos           NGA.25.3_1         Alimosho
## 4  NGA Nigeria  Lagos           NGA.25.4_1     Amuwo Odofin
## 5  NGA Nigeria  Lagos           NGA.25.5_1            Apapa
## 6  NGA Nigeria  Lagos           NGA.25.6_1         Badagary   Badagry
##            TYPE_2       ENGTYPE_2 CC_2 HASC_2    area_m   area_km
## 1 Local Authority Local Authority              16575387  16.57539
## 2 Local Authority Local Authority              11643619  11.64362
## 3 Local Authority Local Authority             148062074 148.06207
## 4 Local Authority Local Authority             174091716 174.09172
## 5 Local Authority Local Authority              41688102  41.68810
## 6 Local Authority Local Authority             494706196 494.70620
```

Grab coordinates

Create a matrix of coordinates

```
schools_sf %>%
  st_coordinates() %>%
  head()
```

```
##              X        Y
## [1,] 4.005400 6.665200
## [2,] 3.962290 6.616529
## [3,] 3.940881 6.572413
## [4,] 3.692594 6.474112
## [5,] 3.629154 6.470489
## [6,] 3.628560 6.471747
```

Get bounding box

```
schools_sf %>%
  st_bbox()
```

```
##      xmin      ymin      xmax      ymax
## 2.858534 6.407246 4.005400 6.703460
```

Exercise

Exercise: Create a polyline of all trunk roads in Lagos, and buffer the poyline by 10 meters. In `roads_sf`, the `highway` variable notes road types.

Solution:

```
roads_sf %>%
  filter(highway == "trunk") %>%
  summarise(geometry = st_combine(geometry)) %>%
  st_buffer(dist = 10)
```

```
## Simple feature collection with 1 feature and 0 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: 2.705843 ymin: 6.37871 xmax: 3.520564 ymax: 6.716115
## Geodetic CRS:  WGS 84
##                        geometry
## 1 MULTIPOLYGON (((3.255848 6....
```

02:00

Spatial operations using multiple datasets

- `st_distance`: Calculate distances.
- `st_intersects`: Indicates whether simple features intersect.
- `st_intersection`: Cut one spatial object based on another.
- `st_difference`: Remove part of spatial object based on another.
- `st_join`: Spatial join (ie, add attributes of one dataframe to another based on location).

Distances

For this example, we'll compute the distance between each school to a motorway.

```
motor_sf <- roads_sf %>%
  filter(highway == "motorway")

# Matrix: distance of each school to each motorway
dist_mat <- st_distance(schools_sf, motor_sf)

# Take minimun distance for each school
dist_mat %>% apply(1, min) %>% head()
```

```
## [1] 67892.77 62462.18 59452.39 31450.97 24430.84 24376.20
```

Distances

There are multiple ways to calculate distances!

- **Great circle:** sf, by default, uses s2 to computer distance (in meters) when data has a geographic CRS
- **Great circle:** Other formulas beyond s2, such as Haversine, Vincenty, and Karney's method. See the geosphere and geodist packages. Vincenty is more precise than Haversine, and Karney's method is more precise than Vincenty's method. Greater precision comes with heavy computation. For more information, see here.
- **Projected:** We can use a projected CRS, where units are in meters already.

Distances

```
# s2
st_distance(schools_sf[1,], schools_sf[2,]) %>%
  as.numeric()
```

```
## [1] 7208.358
```

```
# Nigeria-specific CRS
schools_utm_sf <- st_transform(schools_sf, 32632)
st_distance(schools_utm_sf[1,], schools_utm_sf[2,]) %>
  as.numeric()
```

```
## [1] 7214.479
```

```
# World mercator
schools_merc_sf <- st_transform(schools_sf, 3395)
st_distance(schools_merc_sf[1,], schools_merc_sf[2,])
  as.numeric()
```

```
## [1] 7238.156
```

```
# Haversine
distHaversine(
  p1 = schools_sf[1,] %>% st_coordinates,
  p2 = schools_sf[2,] %>% st_coordinates)
```

```
## [1] 7216.422
```

```
# Vincenty's method
distVincentySphere(
  p1 = schools_sf[1,] %>% st_coordinates,
  p2 = schools_sf[2,] %>% st_coordinates)
```

```
## [1] 7216.422
```

```
# Karney's method
distGeo(p1 = schools_sf[1,] %>% st_coordinates,
        p2 = schools_sf[2,] %>% st_coordinates)
```

```
## [1] 7189.913
```

Intersects

For this example we'll determine which of Lagos's second administrative divisions intersects with a motorway.

```
# Sparse matrix
st_intersects(lagos_sf, motor_sf) %>% print()
```

```
## Sparse geometry binary predicate list of length 20, where the predicate
## was `intersects'
## first 10 elements:
##  1: (empty)
##  2: 51, 54, 55, 56, 57, 60
##  3: (empty)
##  4: 57, 60
##  5: 21, 22, 28, 29, 32, 35, 36, 57, 58, 59, ...
##  6: (empty)
##  7: (empty)
##  8: 9, 19, 22, 69, 72
##  9: (empty)
##  10: (empty)
```

Intersects

Take `max` (`FALSE` corresponds to 0 and `TRUE` corresponds to 1). So taking max will yeild if unit intersects with *any* motorway

```
# Matrix
st_intersects(lagos_sf, motor_sf, sparse = F) %>%
  apply(1, max) %>%
  head()
```

```
## [1] 0 1 0 1 1 0
```

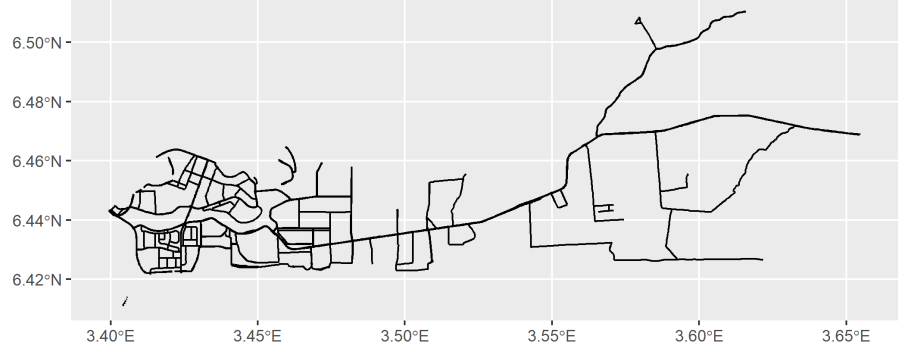
Intersection

We have roads for all of Lagos. Here, we want to create new roads object that **only includes** roads in the Eti-Osa unit.

```
etis_sf <- lagos_sf %>%
  filter(NAME_2 == "Eti-Osa")

roads_etis_sf <- st_intersection(roads_sf, etis_sf)
```

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

```
ggplot() +
  geom_sf(data = roads_etis_sf)
```



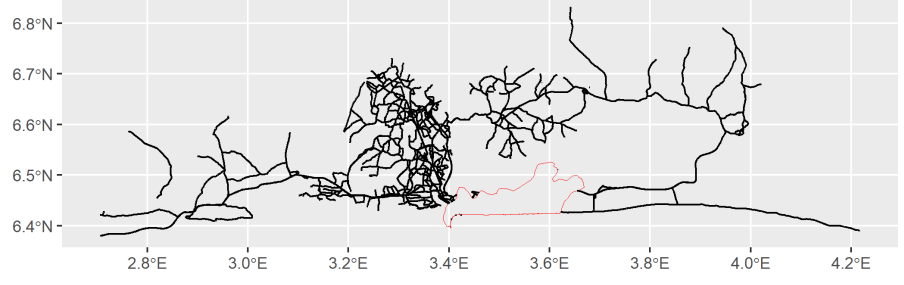
Difference

We have roads for all of Lagos. Here, we want to create new roads object that **excludes** roads in the Eti-Osa unit.

```
roads_notetis_sf <- st_difference(roads_sf, etis_sf)
```

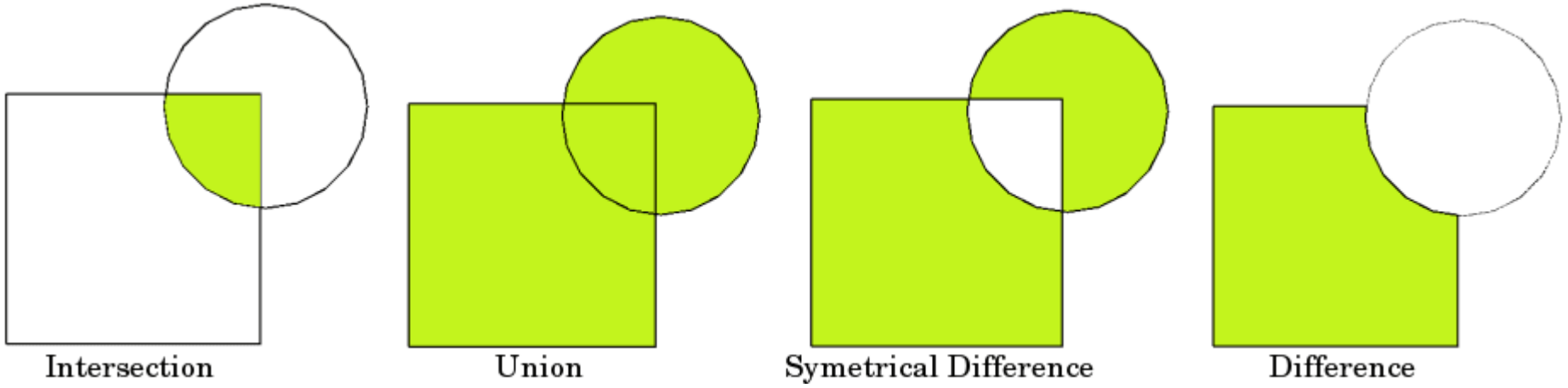
```
## Warning: attribute variables are assumed to be spatially constant throughout
## all geometries
```

```
ggplot() +
  geom_sf(data = etis_sf, fill = NA, color = "red") +
  geom_sf(data = roads_notetis_sf)
```



Overlay

Intersections and differencing are **overlay** functions



Spatial join

We have a dataset of schools. The school dataframe contains information such as the school name, but not on the administrative region it's in. To add data on the administrative region that the school is in, we'll perform a spatial join.

Check the variable names. No names of second administrative divison :(

```
names(schools_sf)
```

```
##  [1] "element_type"     "osmid"            "amenity"          "min_age"
##  [5] "name"             "opening_hours"    "wheelchair"       "website"
##  [9] "wikidata"         "wikipedia"        "addr:city"        "addr:postcode"
## [13] "addr:street"      "email"            "phone"            "building"
## [17] "grades"           "addr:housenumber" "official_name"    "short_name"
## [21] "survey:date"      "description"      "source"           "name:en"
## [25] "operator"         "religion"         "denomination"     "operator:type"
## [29] "isced:level"      "landuse"          "osm_type"         "geometry"
```

Spatial join

Use `st_join` to add attributes from `lagos_sf` to `schools_sf`. `st_join` is similar to other join methods (eg, `left_join`); instead of joining on a varible, we join based on location.

```
schools_lagos_sf <- st_join(schools_sf, lagos_sf)

schools_lagos_sf %>%
  names() %>%
  print() %>%
  tail(10)
```

```
##  [1] "element_type"     "osmid"            "amenity"          "min_age"
##  [5] "name"             "opening_hours"    "wheelchair"       "website"
##  [9] "wikidata"         "wikipedia"        "addr:city"        "addr:postcode"
## [13] "addr:street"      "email"            "phone"            "building"
## [17] "grades"           "addr:housenumber" "official_name"    "short_name"
## [21] "survey:date"      "description"      "source"           "name:en"
## [25] "operator"         "religion"         "denomination"     "operator:type"
## [29] "isced:level"      "landuse"          "osm_type"         "geometry"
## [33] "ID_0"             "COUNTRY"          "NAME_1"           "NL_NAME_1"
## [37] "ID_2"             "NAME_2"           "VARNAME_2"        "NL_NAME_2"
## [41] "TYPE_2"           "ENGTYPE_2"        "CC_2"             "HASC_2"
```

Spatial join

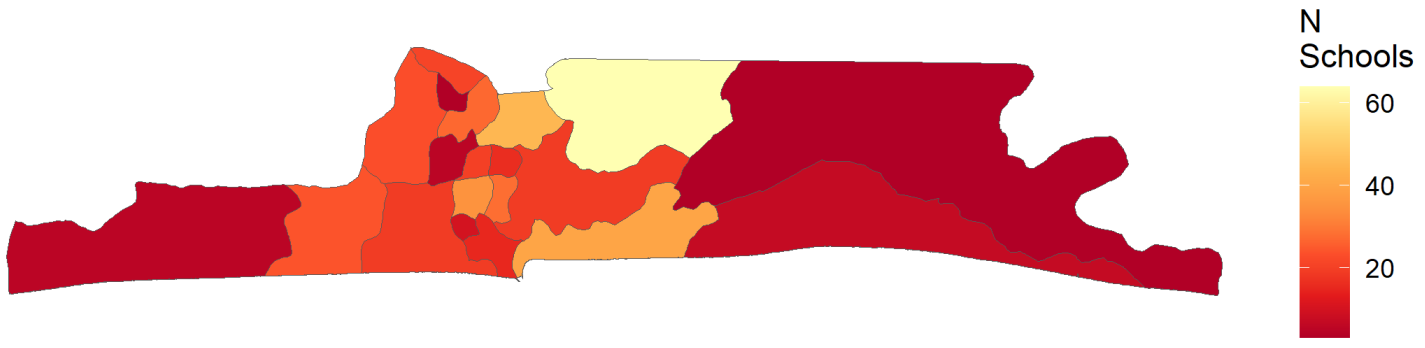
Exercise: Make a static map using of Lagos administrative areas, where each administrative area polygon displays the number of schools.

Solution:

```
## Dataframe of number of schools per NAME_2
n_school_df <- schools_lagos_sf %>%
  st_drop_geometry() %>%
  group_by(NAME_2) %>%
  summarise(n_school = n()) %>%
  ungroup()

## Merge info with lagos_sf
lagos_sch_sf <- lagos_sf %>% left_join(n_school_df, by = "NAME_2")

## Map
p <- ggplot() +
  geom_sf(data = lagos_sch_sf,
          aes(fill = n_school))
```

Spatial join

```
ggplot() +
  geom_sf(data = lagos_sch_sf,
          aes(fill = n_school)) +
  labs(fill = "N\nSchools") +
  scale_fill_distiller(palette = "YlOrRd") +
  theme_void()
```



Resources

- sf package cheatsheet
- Spatial Data Science with Applications in R
- Geocomputation with R

Thank you!