Geospatial data: Vector files and rasters

Nairobi Workshop: Day 1 and 2

Josh Merfeld
University of Queensland

David Newhouse
World Bank

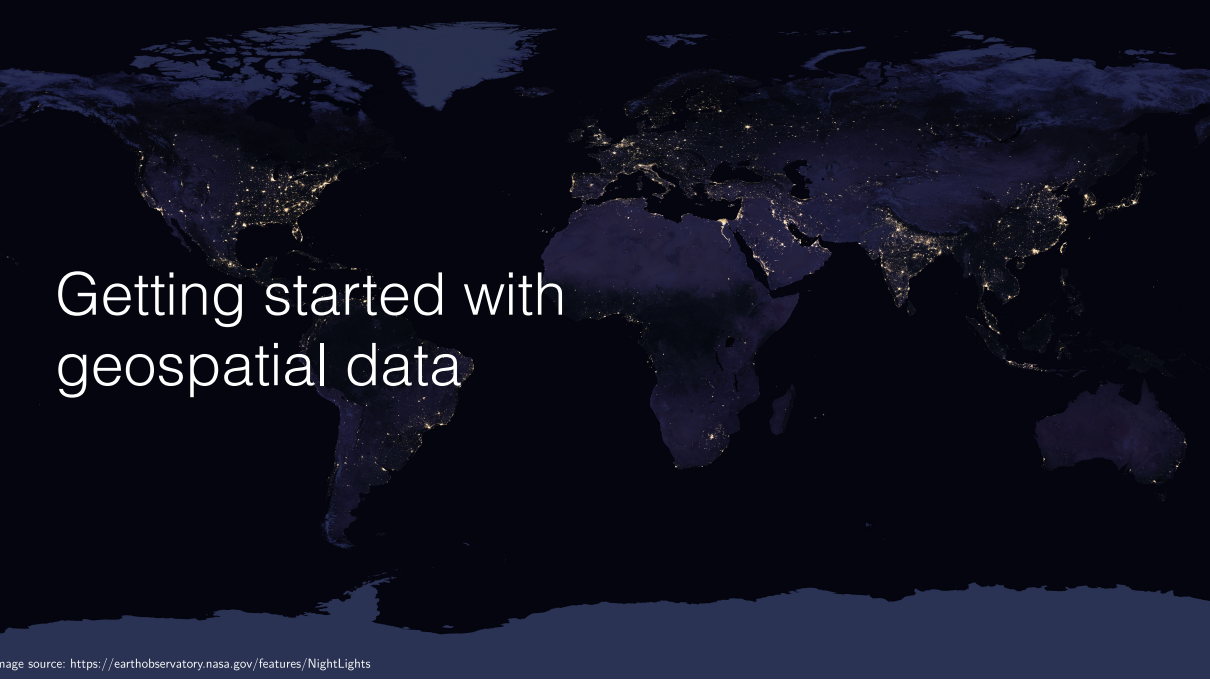
2025-10-20

Introduction to geospatial data

- One estimate says that 100 TB of only weather data are generated every single day
  - This means there is a lot of data to work with!
  - Note that this is also problematic, since it can be difficult to work with such large datasets
- Geospatial data is used in a variety of fields
  - Agriculture
  - Urban planning
  - Environmental science
  - Public health
  - Transportation
  - And many more!

The amount of geospatial data is useful for many applications!

- Geospatial data can be highly predictive of e.g. poverty
  - Urbanity
  - Land class/cover
  - Vegetation indices
  - Population counts
  - etc. etc.
- More importantly: it's available everywhere!



Getting started with geospatial data

mage source: https://earthobservatory.nasa.gov/features/NightLights

Getting started with geospatial data

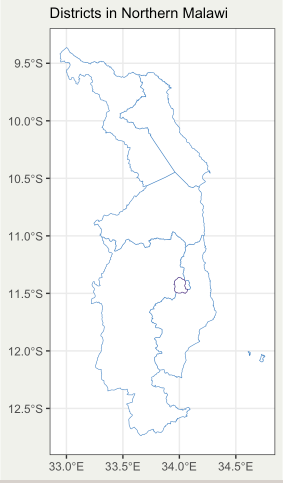
- What are we doing today?
  - Shapefiles
    - Polygons
    - Points
    - Lines
    - Mapping with the package `sf`
  - Coordinate reference systems
    - Latitude/longitude
    - Projections

Shapefiles

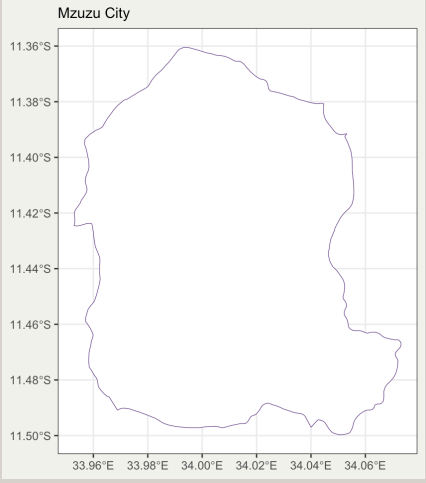
- Shapefiles are a common format for geospatial data
  - They are a form of `vector` data
- Shapefiles are made up of *at least* three files:
  - `.shp` - the shape itself
  - `.shx` - the index
  - `.dbf` - the attributes
  - `.prj` - the projection
    - This one is not technically necessary! But it's common to have.
  - What these all mean isn't important for now, just make sure they are there! Check the `day2files` folder on github.

Let's look at Northern Malawi

- Collection of `features`



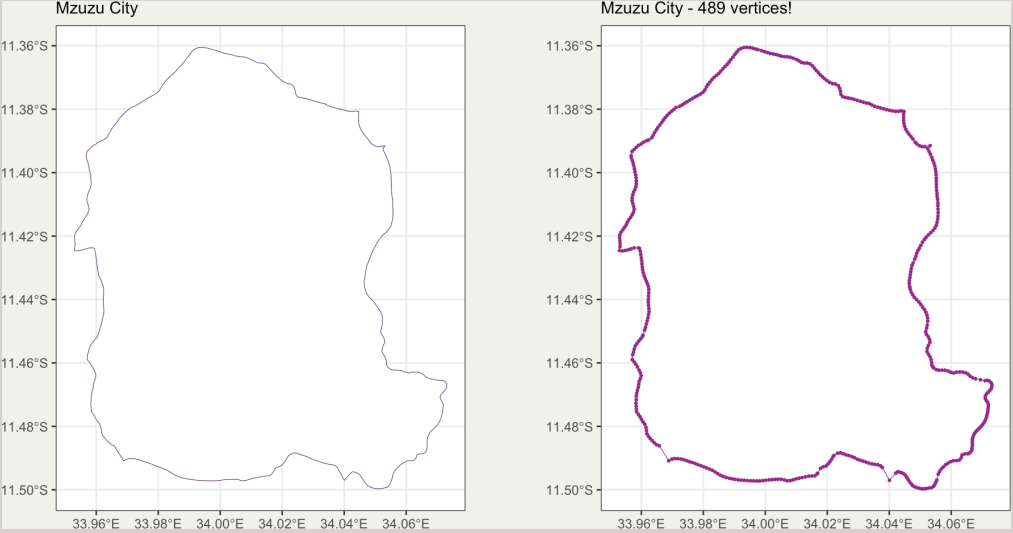
- One `feature`



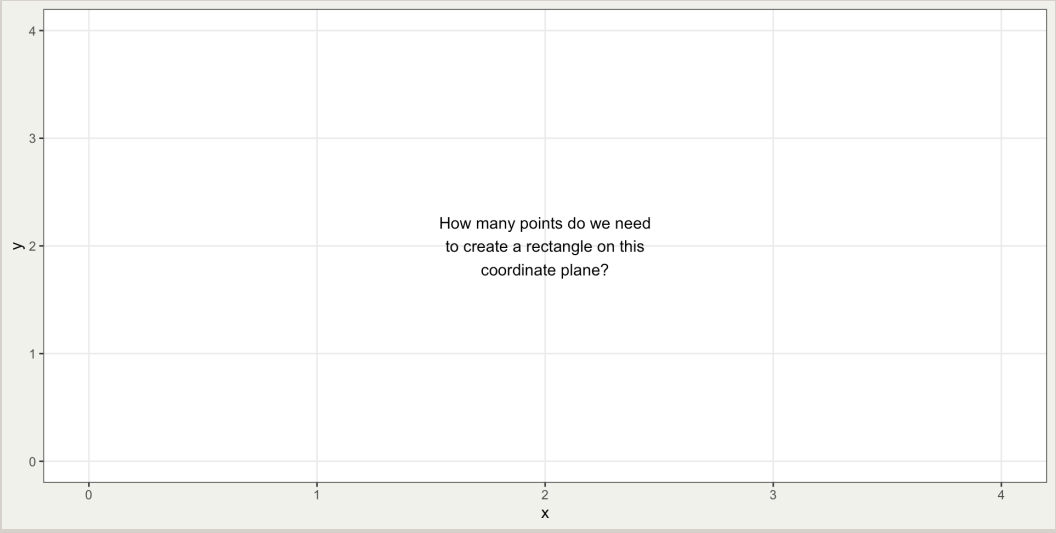
Types of features

- `Polygons`
  - Areas
  - Districts, countries, etc.
- `Lines`
  - Lines
  - Roads, rivers, etc.
- `Points`
  - Points

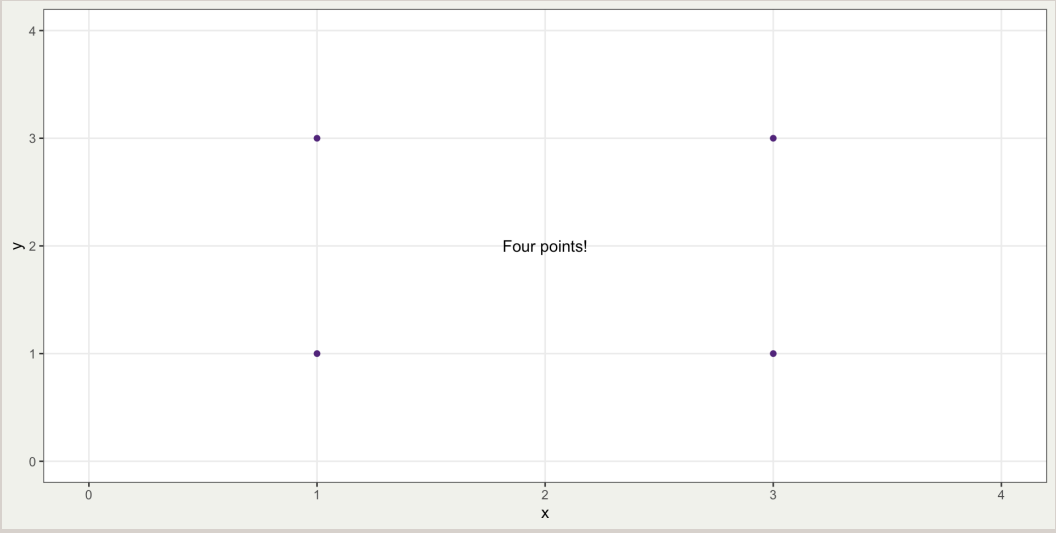
Let's start with polygons



Imagine a rectangle, on a coordinate plane

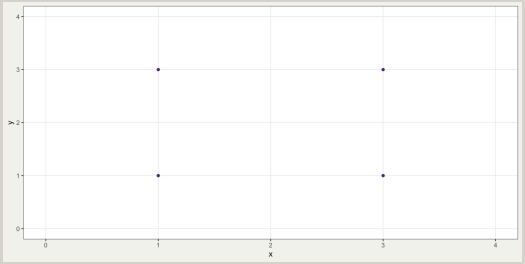


Imagine a rectangle, on a coordinate plane



Imagine a rectangle, on a coordinate plane

- We need four points.
- But polygons in shapefiles are a little different.
  - We have to “close” the feature so it knows it’s a polygon!
- We do this by adding a fifth point: the same as the first point!

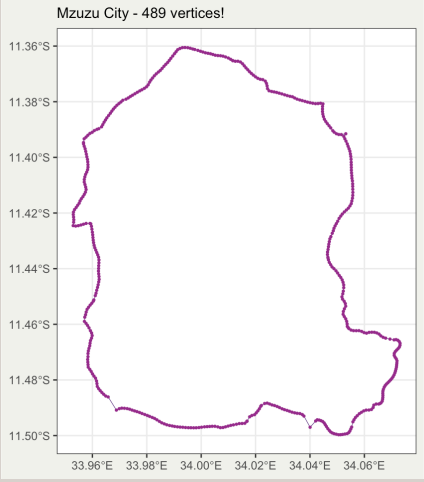


FIVE POINTS (VERTICES) IN OUR FEATURE

	X value	Y value
Point 1	1	1
Point 2	3	1
Point 3	3	3
Point 4	1	3
Point 5	1	1

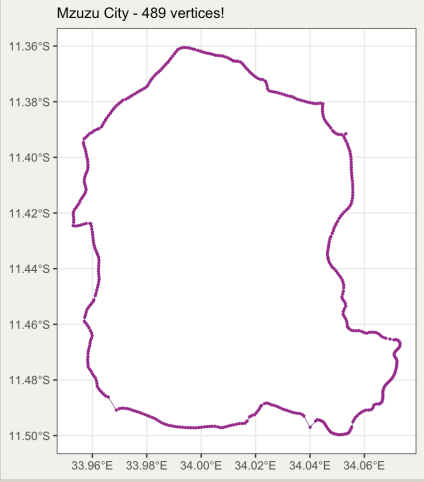
All features are made of vertices

- So we have all our vertices (489 of them!)
- The question:
  - What is the coordinate system here?

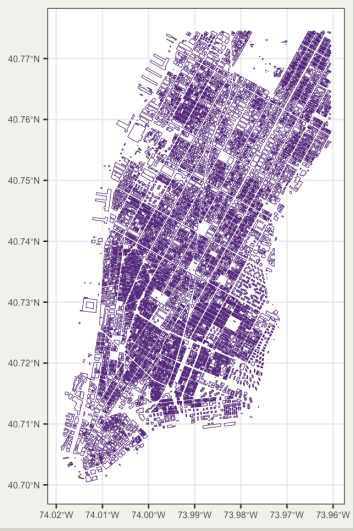


All features are made of vertices

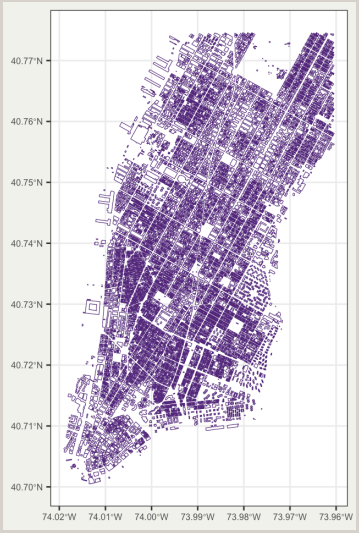
- So we have all our vertices (489 of them!)
- The question:
  - What is the coordinate system here?
- We will return to this in a bit!

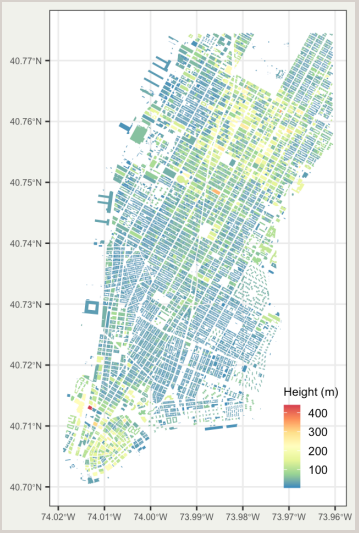


One more example of polygons



One more example of polygons

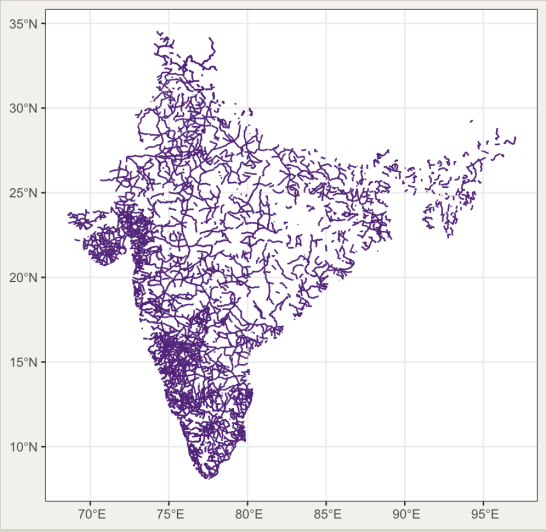




Lines

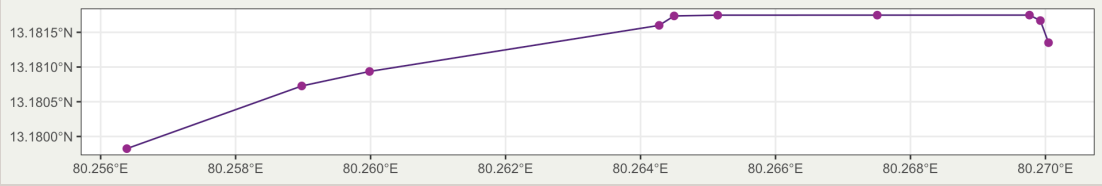
- Lines are also made up of vertices
- But they are not closed

Lines example - “Primary” roads in India (2014)



One road

Length of this line feature: 1619.93 (m)



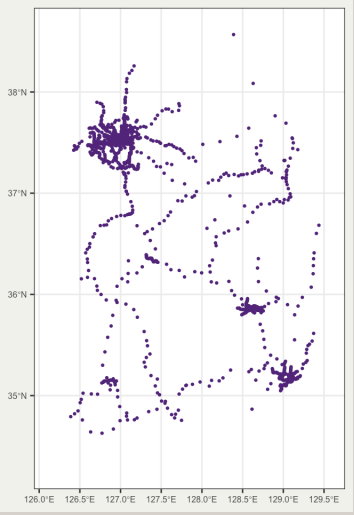
Points

- Points are exactly what they sound like: points!
- What could be a point?

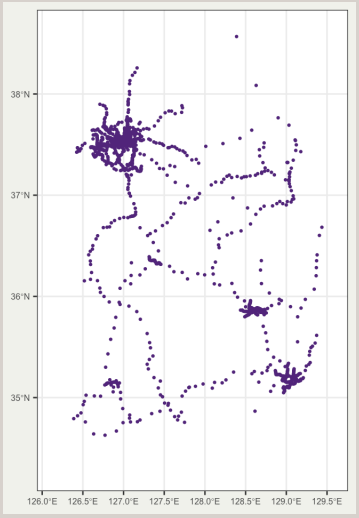
Points

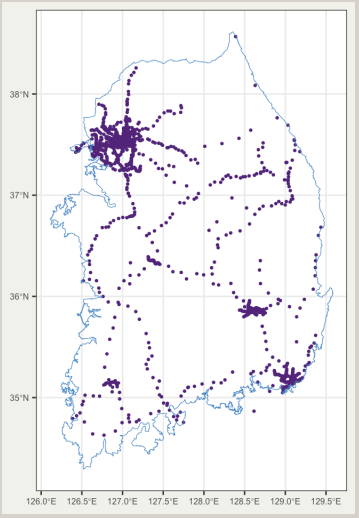
- Points are exactly what they sound like: points!
- What could be a point?
  - A city
  - A weather station
  - A tree
  - A household
  - etc.

What do you think this is?



What do you think this is?





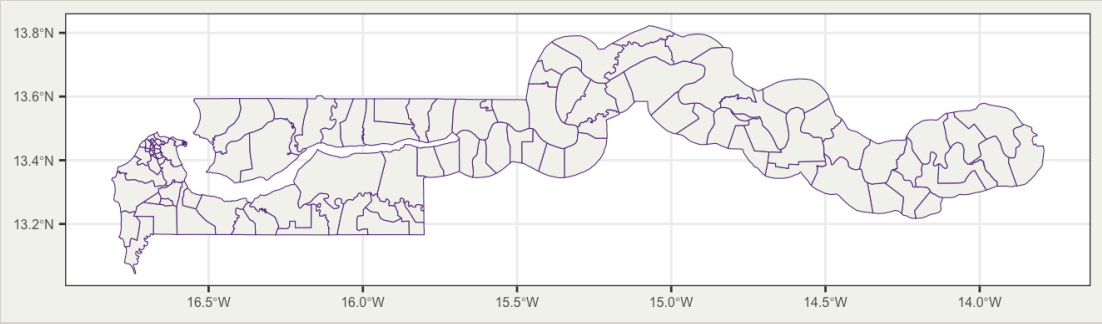
Train stations! First 15

name	name_en	railway	geometry
지평 |J	pyeong |	tation |	OINT (127.6291 37.47681) |
국수 |G	ksu |	tation |	OINT (127.3993 37.51617) |
양평 |Y	ngpyeong |	tation |	OINT (127.4919 37.49285) |
공릉 |G	ngneung |	tation |	OINT (127.073 37.62564) |
남광주 |Na	gwangju | s	ation | P	INT (126.9228 35.13944) |
공항 |A	rport |	tation |	OINT (126.8117 35.1441) |
농성 |N	ngseong |	tation |	OINT (126.884 35.15322) |
금남로5가 |Geu	namro 5-ga | st	tion | PO	NT (126.9101 35.15371) |
금남로4가 |Geu	namno 4(sa)-ga | st	tion | PO	NT (126.9146 35.15071) |
소태 |S	tae |	tation |	OINT (126.9322 35.12361) |
녹동 |N	kdong |	tation |	OINT (126.934 35.10682) |
동구청 |Do	g-gu Office | s	ation | P	INT (128.6322 35.8844) |
동대구 |Do	gdaegu | s	ation | P	INT (128.6275 35.87742) |
북구청 |Bu	-gu Office | s	ation | P	INT (128.5814 35.88381) |
평창 |P	eongchang |	tation |	POINT (128.43 37.56196) |

The train stations are a collection of features

- Just like before, the shapefile is a collection of features!
- The only difference now is that each feature is a point

Reading and plotting shapefiles in R



Reading shapefiles in R

- My go-to package for shapefiles in R is `sf`
- Reading shapefiles is VERY easy! And you can treat them like dataframes.

▼ Code

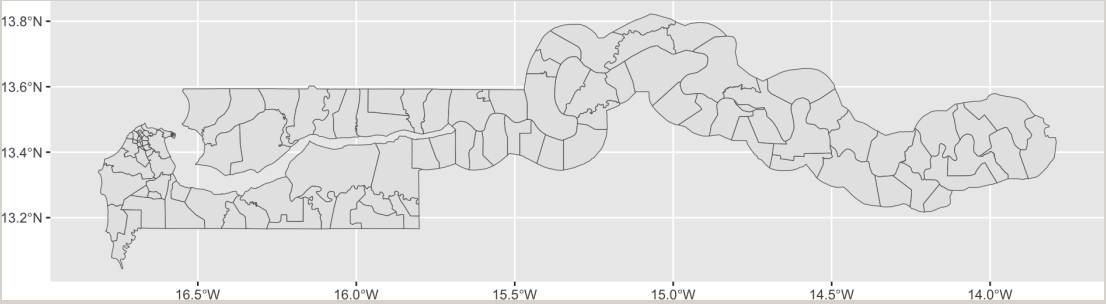
```
library(sf)
# this is the shapefile for Gambia
gambia <- read_sf("day2files/gambia.shp")
gambia
```

```
Simple feature collection with 115 features and 17 fields
Geometry type: POLYGON
Dimension:     XY
Bounding box:  xmin: -16.81426 ymin: 13.04347 xmax: -13.79112 ymax: 13.82274
Geodetic CRS:  WGS 84
# A tibble: 115 × 18
   Shape_Leng Shape_Area ADM3_EN       ADM3_PCODE ADM3_REF ADM3ALT1EN ADM3ALT2EN
        <dbl>      <dbl> <chr>         <chr>      <chr>    <chr>      <chr>
 1     0.0115 0.00000688 New Town East GM010101   <NA>     <NA>       <NA>
 2     0.0206 0.0000161  New Town West GM010102   <NA>     <NA>       <NA>
 3     0.0200 0.0000192  Soldier Town  GM010103   <NA>     <NA>       <NA>
 4     0.0337 0.0000302  Box Bar       GM010201   <NA>     <NA>       <NA>
 5     0.156  0.000591   Campama       GM010202   <NA>     <NA>       <NA>
 6     0.0179 0.0000127  Crab Island   GM010203   <NA>     <NA>       <NA>
 7     0.0322 0.0000533  Half Die      GM010301   <NA>     <NA>       <NA>
 8     0.0216 0.0000236  Jollof Town   GM010302   <NA>     <NA>       <NA>
 9     0.0203 0.0000257  Portugiese T… GM010303   <NA>     <NA>       <NA>
10     0.494  0.00937    Basse         GM020101   <NA>     <NA>       <NA>
```

Plotting is also very easy

▼ Code

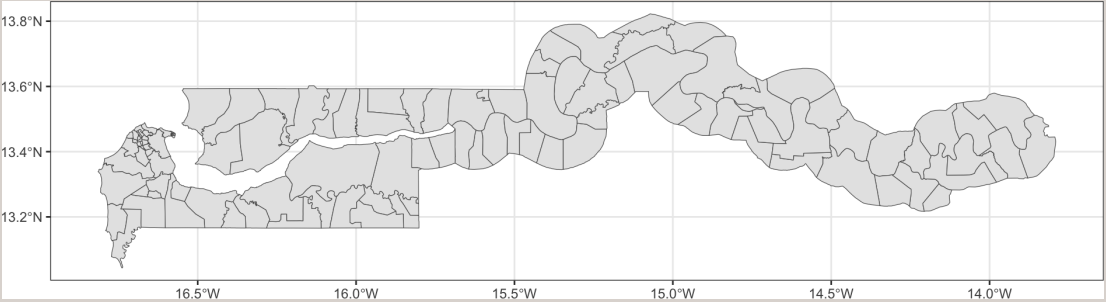
```
ggplot() +
  geom_sf(data = gambia)
```



My go-to theme

▼ Code

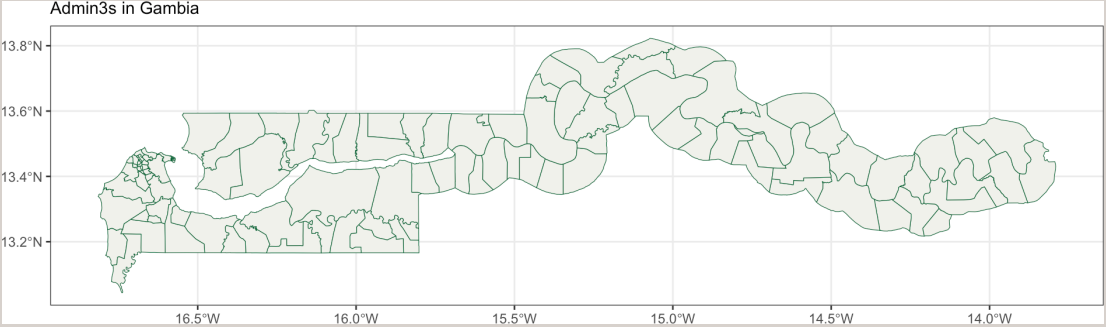
```
ggplot() +
  geom_sf(data = gambia) +
  theme_bw()
```



Other changes you can make

▼ Code

```
ggplot() +
  geom_sf(data = gambia, fill = "#f0f1eb", color = "#006334") +
  theme_bw() +
  labs(subtitle = "Admin3s in Gambia")
```



Give it a try with TAs (admin3) in Malawi (mw3.shp)

▶ Code

```
library(sf)
# this is the shapefile for the northern region of Malawi, TA level
northmw <- read_sf("day2files/mw3.shp")
ggplot() +
  geom_sf(data = northmw)
```

▶ Code

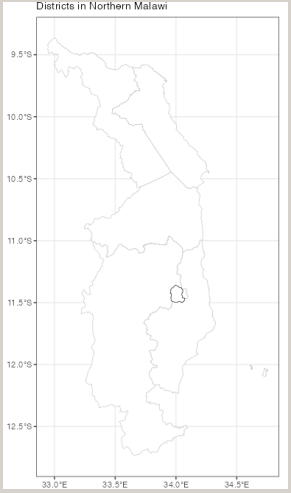
```
ggplot() +
  geom_sf(data = northmw) +
  theme_bw() +
  labs(subtitle = "TAs in Northern Malawi")
```

One more example - map from earlier

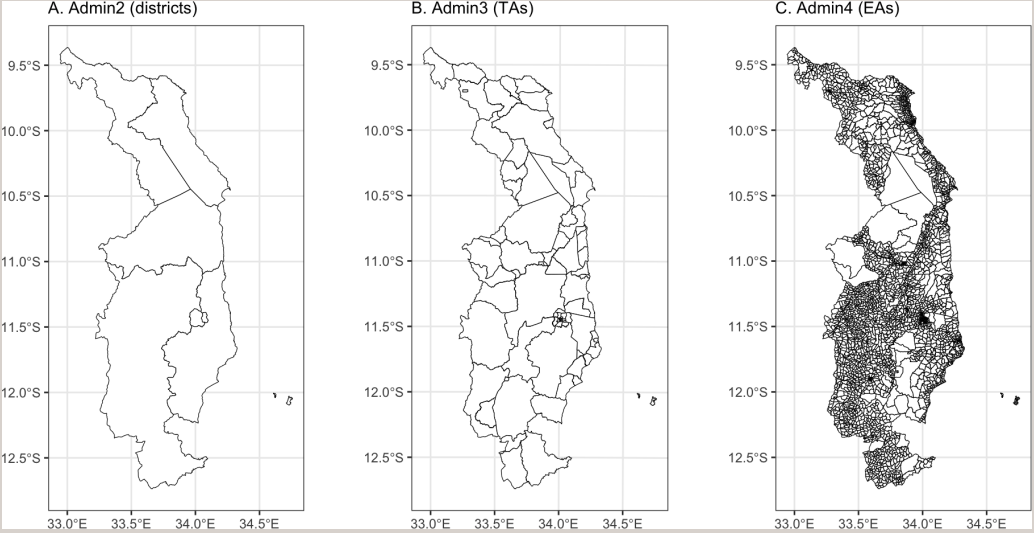
▼ Code

```
admin2 <- read_sf("day2files/mw2.shp")

ggplot() +
  geom_sf(data = admin2,
    fill = "white", color = kdisgray) +
  geom_sf(data = admin2 |> filter(DIST_CODE=="107"),
    fill = "white", color = kdisgreen) +
  theme_bw() +
  labs(subtitle = "Districts in Northern Malawi")
```



What if we want to plot mw2, mw3, and mw4?



What if we want to plot mw2, mw3, and mw4?

▼ Code

```
admin2 <- read_sf("day2files/mw2.shp")
admin3 <- read_sf("day2files/mw3.shp")
admin4 <- read_sf("day2files/mw4.shp")

g1 <- ggplot() +
  geom_sf(data = admin2, fill = "white", color = "black") +
  theme_bw() + labs(subtitle = "A. Admin2 (districts)")
g2 <- ggplot() +
  geom_sf(data = admin3, fill = "white", color = "black") +
  theme_bw() + labs(subtitle = "B. Admin3 (TAs)")
g3 <- ggplot() +
  geom_sf(data = admin4, fill = "white", color = "black") +
  theme_bw() + labs(subtitle = "C. Admin4 (EAs)")
```

What if we want to plot mw2, mw3, and mw4?

- Enter `cowplot`!

▼ Code

```
1 library(cowplot)
2
3 plot_grid(g1, g2, g3, ncol = 3)
```

A quick note about shapefile sizes

- What do you think is the main determinant of the size of a shapefile?
- The number of vertices!
  - Geographic size doesn't really matter!
- For the three Malawi shapefiles?
  - `mw2.shp`: 448 KB
  - `mw3.shp`: 4.9 MB
  - `mw4.shp`: 40.3 MB
- And this is only for Northern Malawi.
  - The entire country is 123 MB
  - The 2023 shapefile from OSM for Indian roads is 236 MB
  - The shapefile of Indian villages is 614 MB

And that's only shapefile

- Other geospatial data can get even bigger!
- How large do you think the folder on my computer that contains imagery for all of Malawi (at 5m resolution) is?
- About 55 GB!
- This is just a warning…

Let’s talk about coordinates

Latitude and longitude on a globe

- The most common `coordinate reference system (CRS)` is latitude/longitude
  - Latitude: North/South
  - Longitude: East/West
  - The equator is at 0° latitude
  - The prime meridian is at 0° longitude

- But there's a problem with using latitude/longitude
  - The Earth is a sphere (well, more or less; really an oblate spheroid)


Source: wikipedia


2 million
sq km
30 million
sq km
Source: wikipedia

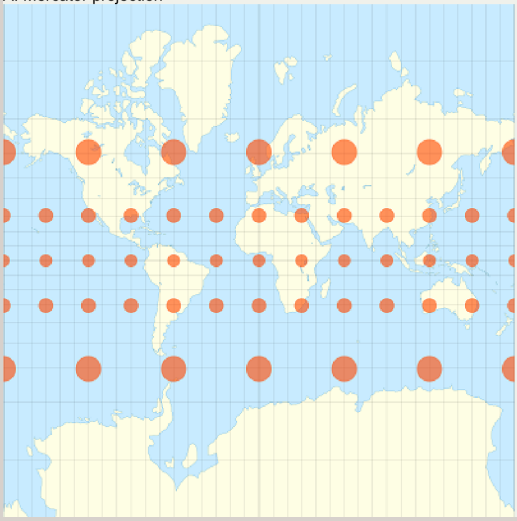
The basic problem

- The basic problem is that one degree of longitude changes at different latitudes!
  - At the equator, one degree of longitude is about 111 km
  - At 15N/S, one degree of longitude is about 107 km
  - At 30N/S, one degree of longitude is about 96 km
  - At 45N/S, one degree of longitude is about 79 km
  - At 60N/S, one degree of longitude is about 56 km
    - This explains Greenland!

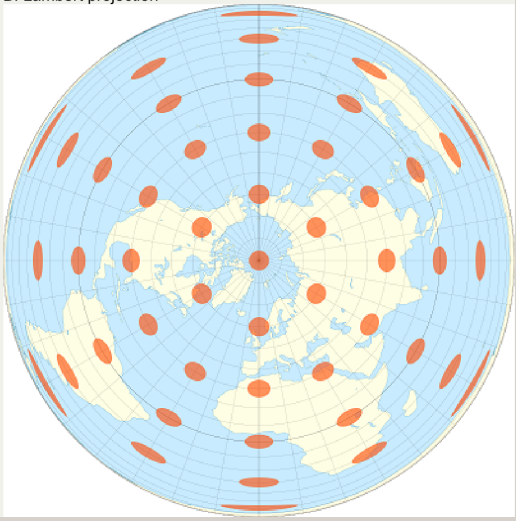
- It's not an easy problem to solve, as all solutions have drawbacks!

Preserve shape, give up area

A. Mercator projection



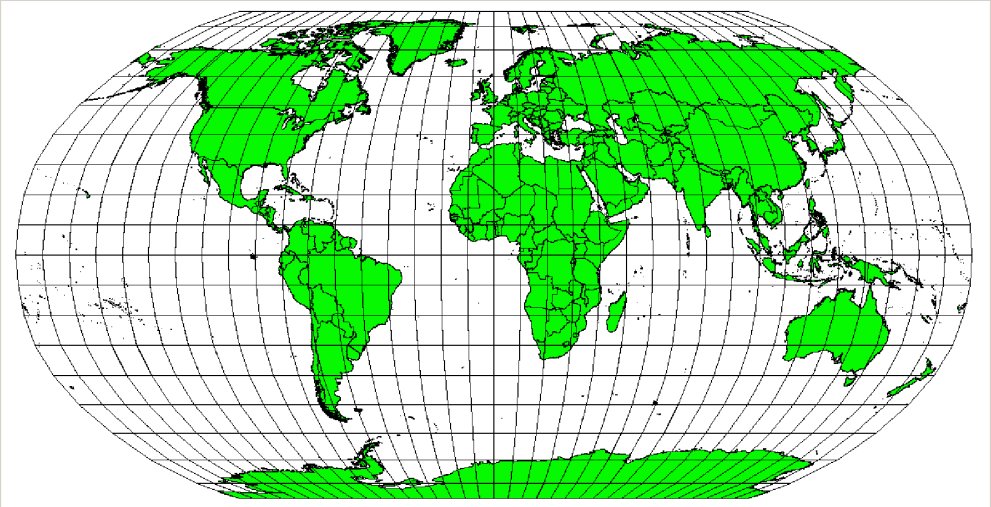
B. Lambert projection



Projections

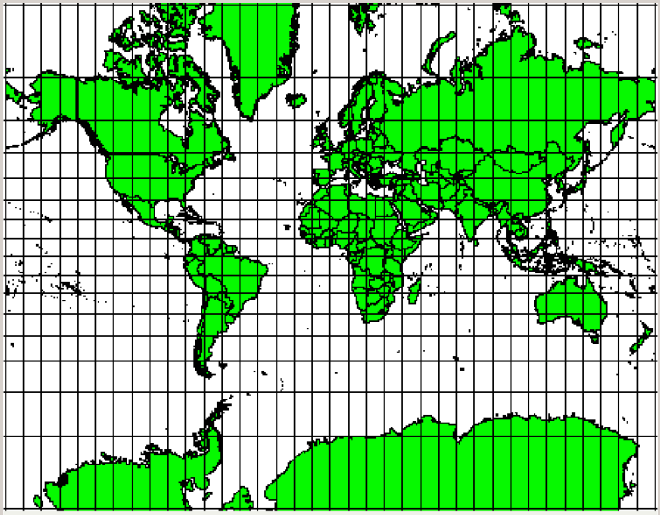
- A `projection` is a way to represent the Earth's surface on a flat plane
  - In other words, it's a way to transform the three-dimensional surface of the earth into two dimensions
  - Think of this as a way to "flatten" the Earth
- "Equal area" projections
  - These projections preserve the area of features
- "Conformal" projections
  - These projections preserve the shape of features
- It is impossible to perfectly preserve both!
  - But they can be close, especially in smaller areas
  - The earth looks quite flat from close up, after all

Robinson projection: compromise across the board



Source: QGIS

Meractor projection: preserves angular relationships

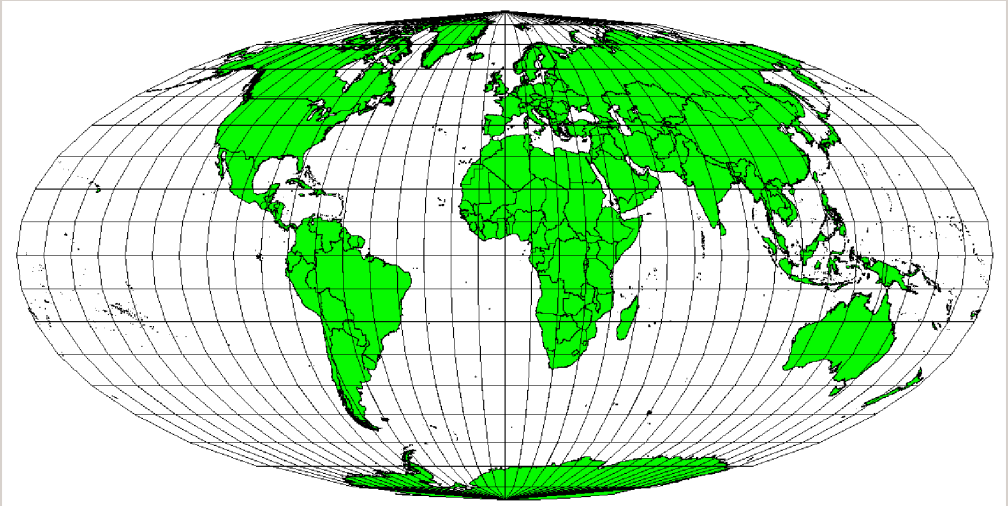


Source: QGIS

Azimuthal Equidistant projection

Source: QGIS/UN

Mollweide Equal Area Cylindrical projection

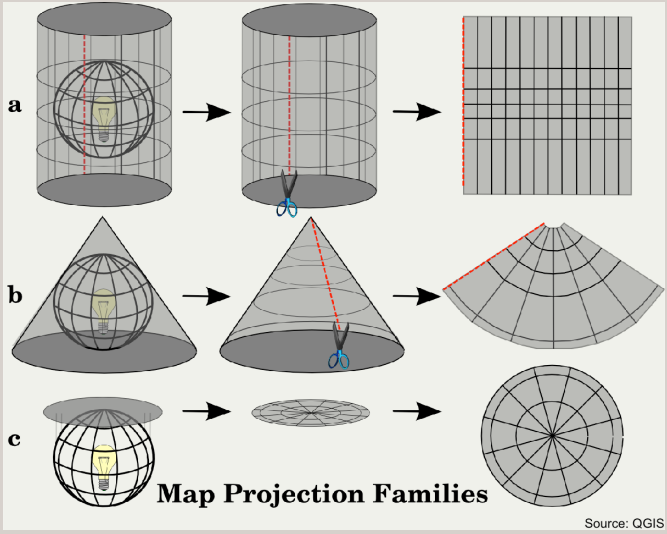


Source: QGIS

Three families of projections

- The QGIS website has a great explanation of the three families of projections
  - Cylindrical
  - Conical
  - Planar

Three families of projections

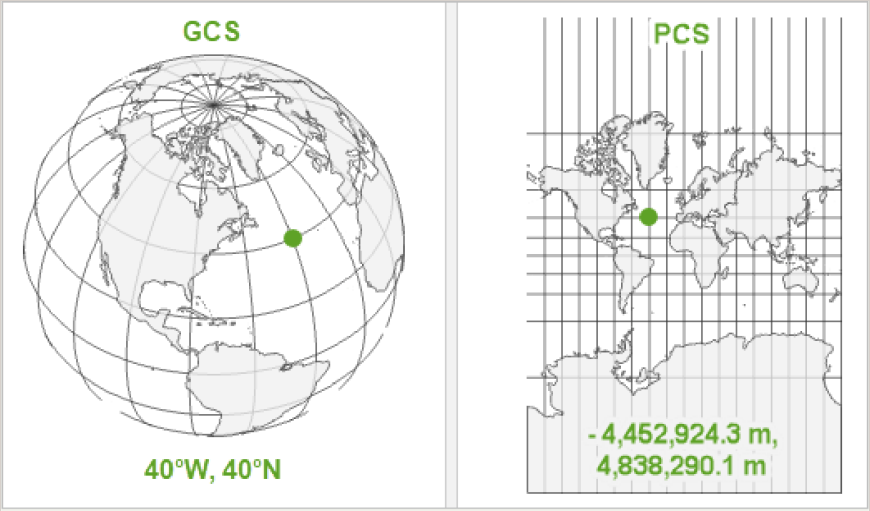


Source: QGIS

Coordinate reference systems

- Coordinate reference systems (CRS) are a way to define how coordinates are represented
  - This includes the projection, but also other things
- The most popular is probably WGS 84 (EPSG:4326), which is a geographic CRS
  - This is latitude/longitude
- One degree of lat/lon:
  - 60 minutes
- One minute:
  - 60 seconds
- Geographic here means it is the location on the earth's surface
  - This is different from a projected CRS, which is more about how to draw the earth on a flat surface

Geographic vs. projected

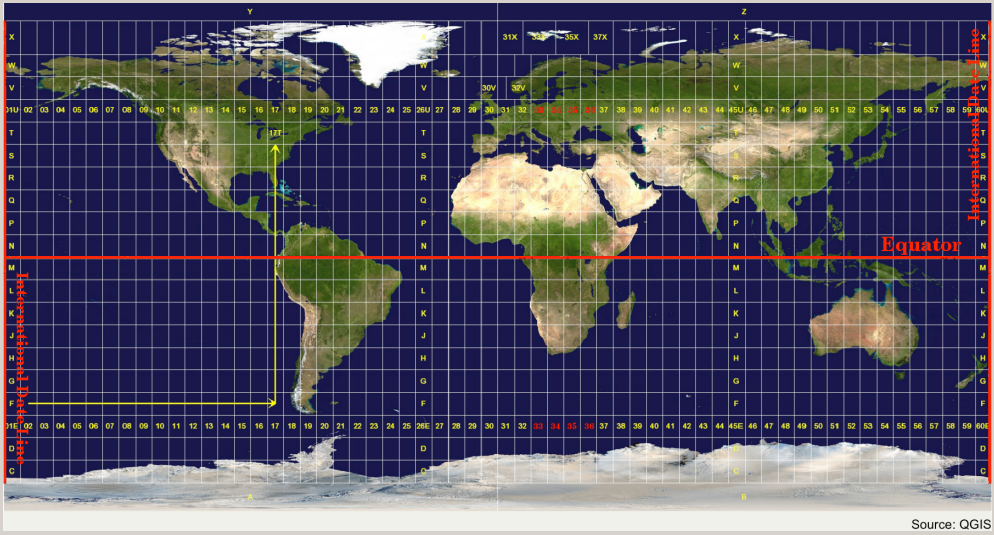


Source: ESRI

A common CRS: Universal Transverse Mercator (UTM)

- Universal Transverse Mercator (UTM) is a projected CRS
  - It divides the world into 60 zones
  - Each zone is 6° wide
  - The equator is the origin of each zone
  - The equator is at 0 m

Universal Transverse Mercator (UTM)

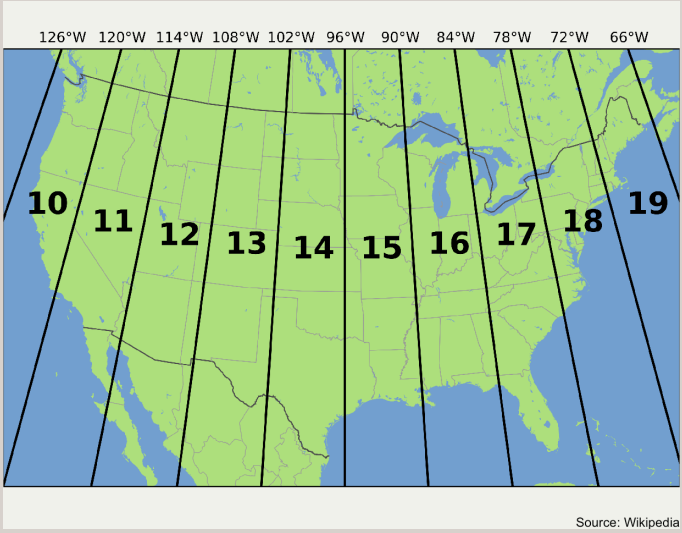


Source: QGIS

A common CRS: Universal Transverse Mercator (UTM)

- UTM defines X values (“longitude”) FROM THE MIDDLE of each zone
  - This middle line is called the central meridian
- UTM defines Y values (“latitude”) from the equater
- There are some other details:
  - UTM values are never negative, so we offset values
    - This is called a “false easting” and “false northing”
    - Details aren’t super important
- Note: it ignores altitude. It assumes a perfect ellipsoid

“Transverse mercator” projection per zone (N and S)



Projections with `sf`

▼ Code

```
admin2 <- read_sf("day2files/mw2.shp")

# Geographic CRS: WGS 84 - lat/lon
crs(admin2)
```

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

What is the appropriate zone for Malawi?

- Go to Google
  - Search “UTM CRS Malawi”
- What did you find?

- UTM zone 36S
- With the `sf` package, we want to find the “EPSG” code to project
  - 20936

What is the appropriate zone for Malawi?

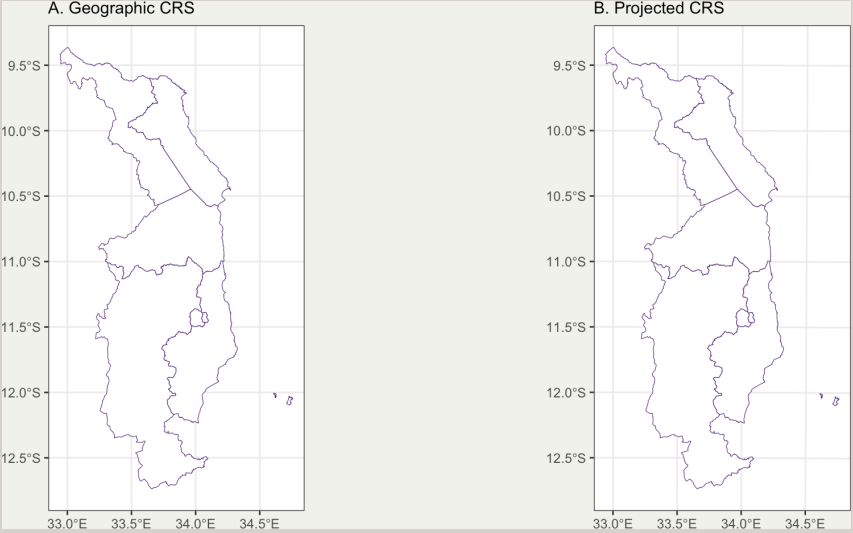
▼ Code

```
admin2proj <- st_transform(admin2, 20936)

# It has changed! Now it's projected
crs(admin2proj)
```

```
[1] "PROJCRS[\"Arc 1950 / UTM zone 36S\",\n    BASEGEOGCRS[\"Arc 1950\",\n        DATUM[\"Arc 1950\",\n            ELLIPSOID[\"Clarke 1880 (Arc)\",6378249.145,293.4663077,\n                LENGTHUNIT[\"metre\",1]]],\n        PRIMEM[\"Greenwich\",0,\n        ANGLEUNIT[\"degree\",0.0174532925199433]],\n        ID[\"EPSG\",4209]],\n    CONVERSION[\"UTM zone 36S\",\n        METHOD[\"Transverse Mercator\",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude of natural origin\",0,\n        ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8801]],\n        PARAMETER[\"Longitude of natural origin\",33,\n        ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n        PARAMETER[\"Scale factor at natural origin\",0.9996,\n        SCALEUNIT[\"unity\",1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",500000,\n            LENGTHUNIT[\"metre\",1],\n            ID[\"EPSG\",8806]],\n        PARAMETER[\"False northing\",10000000,\n            LENGTHUNIT[\"metre\",1],\n        ID[\"EPSG\",8807]]],\n    CS[Cartesian,2],\n        AXIS[\"(E)\",east,\n            ORDER[1],\n            LENGTHUNIT[\"metre\",1]],\n        AXIS[\"(N)\",north,\n            ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    USAGE[\n        SCOPE[\"Engineering survey, topographic mapping.\"],\n        AREA[\"Malawi. Zambia and Zimbabwe - east of 30°E.\"],\n        BBOX[-22.42,30,-8.19,35.93]],\n    ID[\"EPSG\",20936]]"
```

They look quite similar!

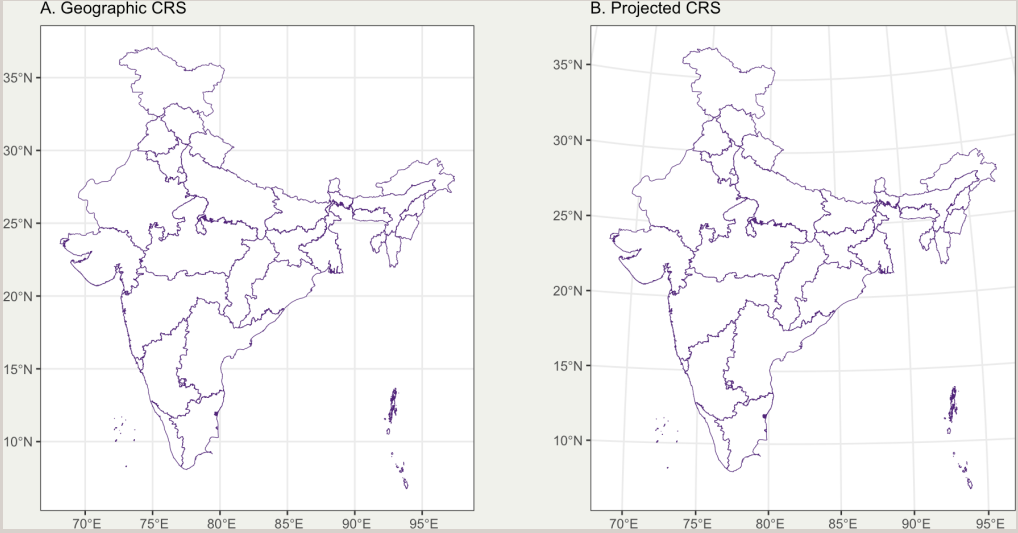


They look quite similar! Different coordinates

▼ Code

```
g1 <- ggplot(admin2) +
  geom_sf(fill = "white", color = kdisgreen) +
  theme_bw() +
  labs(subtitle = "A. Geographic CRS") +
  theme(plot.background = element_rect(fill = "#f0f1eb", color = "#f0f1eb"))
g2 <- ggplot(admin2proj) +
  geom_sf(fill = "white", color = kdisgreen) +
  theme_bw() +
  labs(subtitle = "B. Projected CRS") +
  theme(plot.background = element_rect(fill = "#f0f1eb", color = "#f0f1eb")) +
  coord_sf(crs = st_crs(20936))
```

Malawi is small. What about something larger?



Now it’s your turn!

- Use the Indian roads shapefile (`indiaprimaryroads.shp`)
- Do the following:
  - Find the CRS of the shapefile
  - Transform the shapefile to UTM zone 44N (EPSG: 24344)
  - Graph both side by side using `cowplot`

The solution

▼ Code

```
roads <- read_sf("day2files/indiaprimaryroads.shp")
crs(roads)
```

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

▼ Code

```
roadsproj <- st_transform(roads, 24344)
crs(roadsproj)
```

```
[1] "PROJCRS[\"Kalianpur 1975 / UTM zone 44N\",\n    BASEGEOGCRS[\"Kalianpur 1975\",\n        DATUM[\"Kalianpur 1975\",\n
ELLIPSOID[\"Everest 1830 (1975 Definition)\",6377299.151,300.8017255,\n                LENGTHUNIT[\"metre\",1]],\n
PRIMEM[\"Greenwich\",0,\n            ANGLEUNIT[\"degree\",0.0174532925199433]],\n        ID[\"EPSG\",4146]],\n    CONVERSION[\"UTM zone
44N\",\n        METHOD[\"Transverse Mercator\",\n            ID[\"EPSG\",9807]],\n        PARAMETER[\"Latitude of natural origin\",0,\n
ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8801]],\n        PARAMETER[\"Longitude of natural origin\",81,\n
ANGLEUNIT[\"degree\",0.0174532925199433],\n            ID[\"EPSG\",8802]],\n        PARAMETER[\"Scale factor at natural origin\",0.9996,\n
SCALEUNIT[\"unity\",1],\n            ID[\"EPSG\",8805]],\n        PARAMETER[\"False easting\",500000,\n            LENGTHUNIT[\"metre\",1],
\n            ID[\"EPSG\",8806]],\n        PARAMETER[\"False northing\",0,\n            LENGTHUNIT[\"metre\",1],\n
ID[\"EPSG\",8807]]],\n    CS[Cartesian,2],\n        AXIS[\"(E)\",east,\n            ORDER[1],\n            LENGTHUNIT[\"metre\",1]],\n
AXIS[\"(N)\",north,\n            ORDER[2],\n            LENGTHUNIT[\"metre\",1]],\n    USAGE[\n        SCOPE[\"Engineering survey,
topographic mapping.\"],\n        AREA[\"India - onshore between 78°E and 84°E.\"],\n        BBOX[8.29,78,35.5,84.01]],\n
ID[\"EPSG\",24344]]"
```

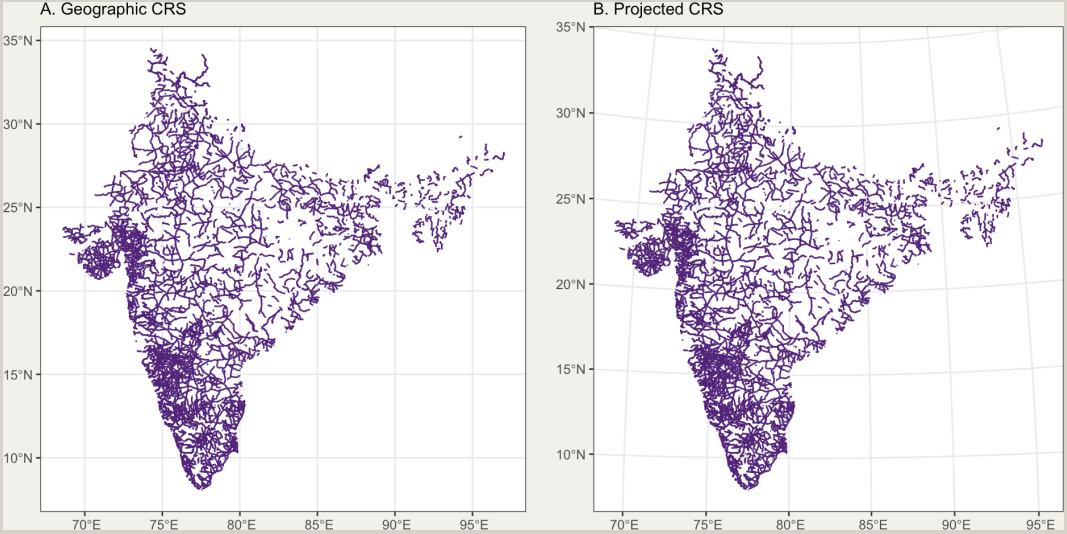
The solution

▼ Code

```
g1 <- ggplot(roads) +
  geom_sf(fill = "white", color = kdisgreen) +
  theme_bw() +
  labs(subtitle = "A. Geographic CRS")
g2 <- ggplot(roadsproj) +
  geom_sf(fill = "white", color = kdisgreen) +
  theme_bw() +
  labs(subtitle = "B. Projected CRS") +
  coord_sf(crs = st_crs(24344))

plot_grid(g1, g2)
```

The solution



More practice

- I'd like you to find a shapefile for a country of your choice
  - Ths isn't always easy
  - I always use Google to find one
  - One place you can often find something on humdata.org (almost always shows up in the search!)
- Do the following:
  - Find the CRS of the shapefile
  - Transform the shapefile to the appropriate UTM zone for the country
    - Note that many countries have multiple UTM zones! You can just choose one
  - Plot them side by side using `cowplot`

Returning to the files

- Shapefiles are made up of *at least* four files:
  - `.shp` - the shape itself
    - The geometry
  - `.shx` - the index
    - The “index”. You can actual recover the .shx file from the .shp file
  - `.dbf` - the attributes
    - The attributes for each features. Could be names, population values, etc.
  - `.prj` - the projection
    - The projection information, which we just discussed
- There are often more files, but these are the main ones
  - You **must** have the first three. The fourth is optional, but common.

Goal

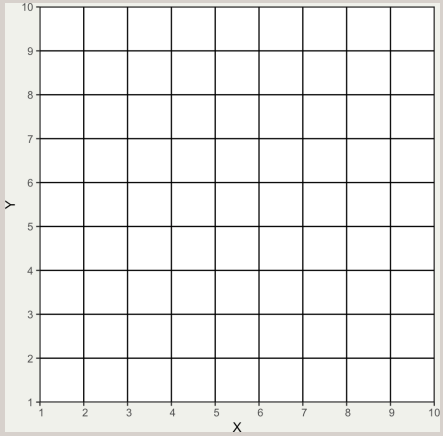
- By the end of the day, we want to be able to:
  - Use geospatial data to estimate a sub-area model
  - We won't actually do that today, we'll do that tomorrow
  - Today focus is getting ALL of the data we need

Rasters

- We've discussed shapefiles
  - Now let's talk about rasters!

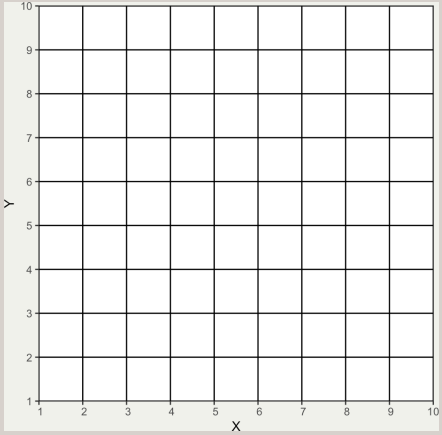
- Rasters are a different type of geospatial data
  - They are made up of a grid of cells
  - Each cell has a value

Example raster grid - how much info do we need?



- Here's a grid.
  - How many points do we need?

Example raster grid - how much info do we need?



- Need to know location of one grid cell…
  - And the size of each grid!

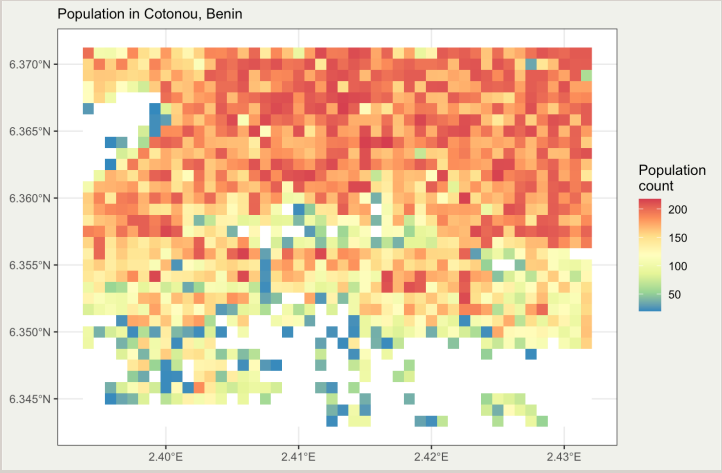
How much info do we need?

- In other words, we do not need a point for every raster cell

- We just need to know:
  - The location of one cell
  - The size of each cell
    - This is called the `resolution` of the raster

- Example:
  - I know the first grid cell in bottom left is at (0, 0)
  - I know each grid cell is 1 meter by 1 meter (the resolution)
  - Then I know the exact location of every single grid cell

Population in Cotonou, Benin



- What are the white values?

Population in Cotonou, Benin

- Here's the information for this raster
  - What's the resolution? What are the units?

```
class       : SpatRaster
dimensions  : 34, 46, 1  (nrow, ncol, nlyr)
resolution  : 0.0008333333, 0.0008333333  (x, y)
extent      : 2.39375, 2.432083, 6.342917, 6.37125  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (EPSG:4326)
source      : beninpop.tif
name        : beninpop
```

Rasters

- Rasters are defined by the grid layout and the resolution
  - Grid cells are sometimes called pixels (just like images, which are often rasters!)

- There are many different file types for rasters
  - `.tif` or `.tiff` (one of the most common)
  - `.nc` (NetCDF, common for very large raster data)
  - Image files, e.g. `png`, `jpg`, etc.

Reading rasters in R

- Reading rasters is also quite easy!
  - Going to use the `terra` package for it
    - Note: can use `terra` for shapefiles, too
  - `day2files/beninpop.tif` is a raster of population counts in Benin

▼ Code

```
1 library(terra)
2
3 # this is the raster for Cotonou, Benin
4 cotonou <- rast("day2files/beninpop.tif")
5 cotonou
```

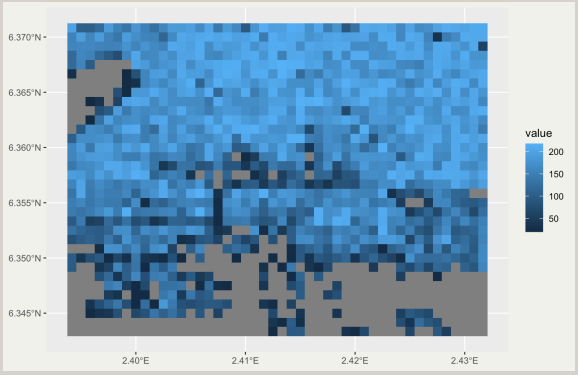
```
class       : SpatRaster
dimensions  : 34, 46, 1  (nrow, ncol, nlyr)
resolution  : 0.0008333333, 0.0008333333  (x, y)
extent      : 2.39375, 2.432083, 6.342917, 6.37125  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (EPSG:4326)
source      : beninpop.tif
name        : beninpop
```

Plotting rasters

- Plotting rasters only with `terra` is a bit of a pain
  - Can't use `ggplot`
  - So, I load another package that lets me use `ggplot` with rasters
    - `tidyterra`

▼ Code

```
1  library(tidyterra)
2
3  ggplot() +
4    geom_spatraster(data = cotonou) +
5    theme_bw() +
6    theme(plot.background = element_rect(fill = "#f0f1eb",
7    theme(legend.background = element_rect(fill = "#f0f1eb"
```

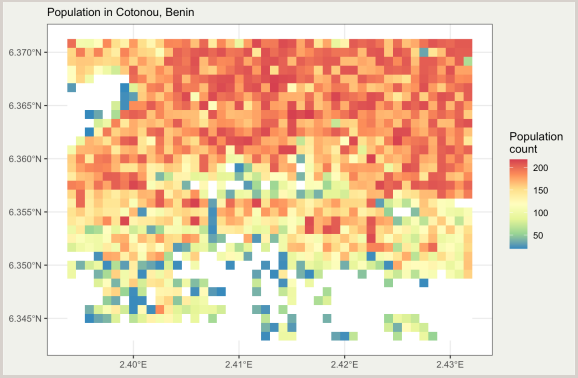


Making it nicer

▼ Code

```
library(tidyterra)

ggplot() +
  geom_spatraster(data = cotonou) +
  # distiller is for continuous values
  # but we can use palettes!
  # I like spectral a lot
  scale_fill_distiller("Population\ncount",
    palette = "Spectral", na.value = "white") +
  theme_bw() +
  labs(subtitle = "Population in Cotonou, Benin") +
  theme(plot.background = element_rect(fill = "#f0f1eb",
  theme(legend.background = element_rect(fill = "#f0f1eb"
```



Extracting raster data to shapefiles

- Let's go back to our use case:
  - We want to estimate a sub-area model at the EA level in Malawi
  - This means we need to extract raster data to the EA level
  - We can do this with `terra`, `sf`, and `exactextractr`
    - `terra` has its own method, but i find `exactextractr` to be MUCH faster

- Let's start by looking at the raster I've uploaded to the `day2data`: `mwpop.tif`

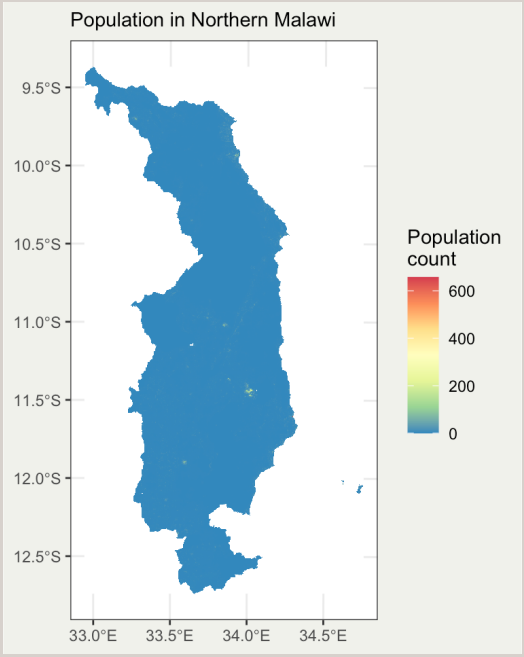
Give it a try

- Try to load it into R using terra, then plot it with tidyterra and ggplot

▼ Code

```
tif <- rast("day2files/mwpop.tif")

ggplot() +
  geom_spatraster(data = tif) +
  scale_fill_distiller("Population\ncount",
    palette = "Spectral", na.value = "white") +
  theme_bw() +
  labs(subtitle = "Population in Northern Malawi") +
  theme(plot.background = element_rect(fill = "#f0f1eb", color =
  theme(legend.background = element_rect(fill = "#f0f1eb", color
```



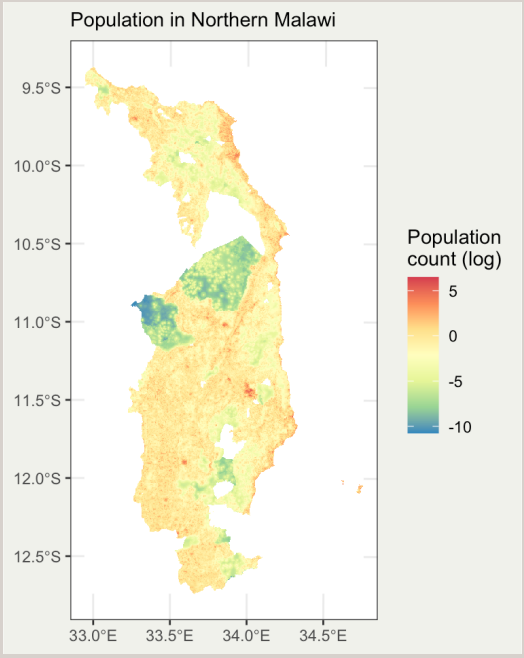
Give it a try

- I actually don't like that map! It's too hard to see because of all the low values.
- So let's take logs, instead!
  - Note that all the zeros become missing (can't log zero)

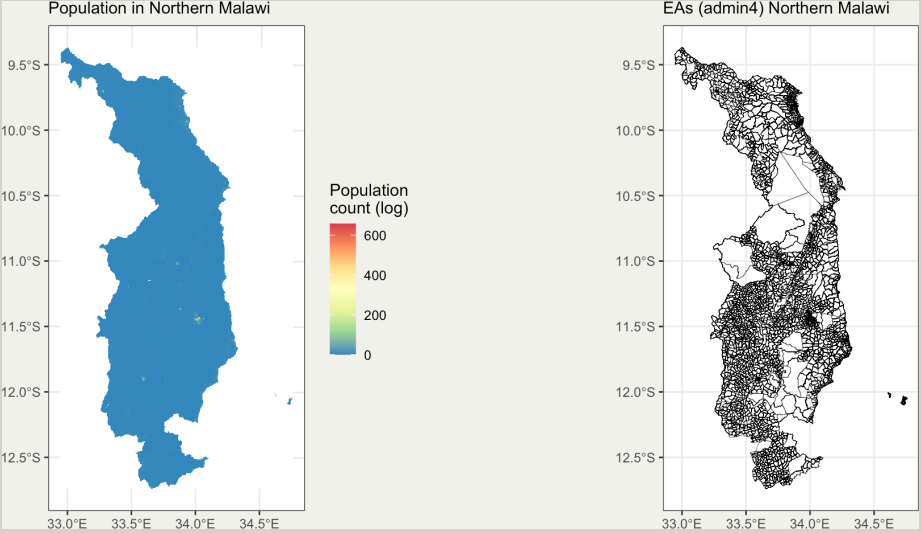
▼ Code

```
tif <- rast("day2files/mwpop.tif")

ggplot() +
  geom_spatraster(data = log(tif)) +
  scale_fill_distiller("Population\ncount (log)",
    palette = "Spectral", na.value = "white") +
  theme_bw() +
  labs(subtitle = "Population in Northern Malawi") +
  theme(plot.background = element_rect(fill = "#f0f1eb", color =
  theme(legend.background = element_rect(fill = "#f0f1eb", color
```



We want to extract the .tif values to the .shp



Let's do it with exactextractr

▼ Code

```r
library(exactextractr)

tif <- rast("day2files/mwpop.tif")
adm4 <- read_sf("day2files/mw4.shp")
# make sure they are in the same CRS! (they already are, but just in case)
# st_transform is for the sf object
adm4 <- st_transform(adm4, crs = crs(tif))

# extract the raster values to the shapefile
# we are going to SUM, and add the EA_CODE from the shapefile to the result
extracted <- exact_extract(tif, adm4, fun = "sum", append_cols = "EA_CODE")
```

▼ Code

```r
head(extracted)
```

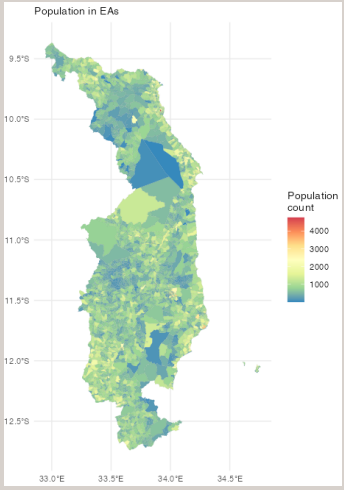
```
   EA_CODE       sum
1 10507801 1068.7787
2 10507072  695.8013
3 10507010  938.1139
4 10507001  749.6960
5 10507009  597.5432
6 10507033  474.3934
```

Now we can join the extracted data to the shapefile

▼ Code

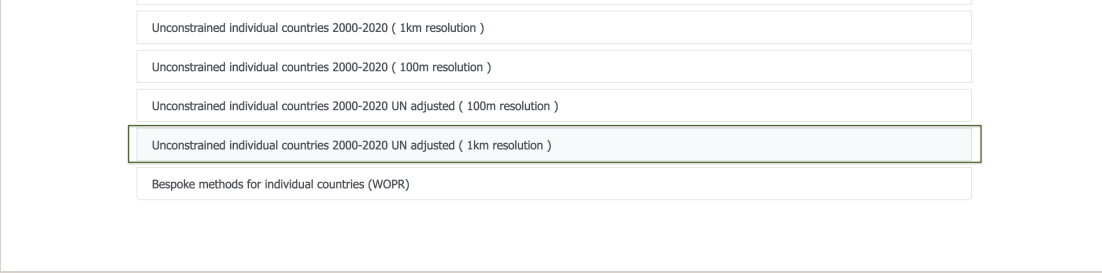
```
# join
adm4 <- adm4 |>
  left_join(extracted, by = "EA_CODE")

# plot it!
ggplot() +
  geom_sf(data = adm4, aes(fill = sum),
    color = "black", lwd = 0.01) +
  scale_fill_distiller("Population\ncount",
    palette = "Spectral", na.value = "white") +
  theme_bw() +
  labs(subtitle = "Population in EAs") +
  theme(plot.background = element_rect(fill = "#f0f1eb",
  theme(legend.background = element_rect(fill = "#f0f1eb"
```



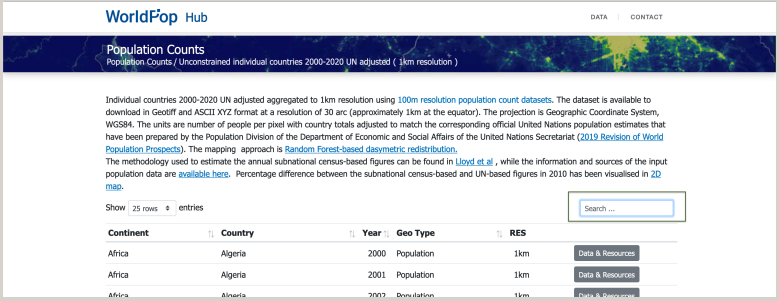
Now it's your turn

- Here's your task:
  - Search for "worldpop population counts"
    - Should be the first result (link: https://hub.worldpop.org/project/categories?id=3)
  - Scroll down the page, click on "unconstrained individual countries 2000-2020 UN adjusted (1km resolution)



Now it's your turn

- Here's your task:
  - Search for "worldpop population counts"
    - Should be the first result (link: https://hub.worldpop.org/project/categories?id=3)
  - Scroll down the page, click on "unconstrained individual countries 2000-2020 UN adjusted (1km resolution)
  - Then, search for a country (maybe yours?)



Now it's your turn

- Here's your task:
  - Search for "worldpop population counts"
    - Should be the first result (link: https://hub.worldpop.org/project/categories?id=3)
  - Scroll down the page, click on "unconstrained individual countries 2000-2020 UN adjusted (1km resolution)
  - Then, search for a country (maybe yours?)
  - Click on "Data & Resources" for 2020
  - Scroll down to the bottom of the page and download the .tif

Now it’s your turn

- Load the .tif into R using `terra`
- Plot the raster using `tidyterra` and `ggplot`
  - Make it look nice!

Let's keep going!

- Now you need to find a shapefile for the same country
- This will be a bit less straightforward
  - Search for "shapefile COUNTRY humdata"
  - You should find a link to the Humanitarian Data Exchange
  - Click on it and see if it has shapefiles for your country of choice
  - If so, download a shapefile (it can be at a higher admin level)
    - If not, raise your hand and I'll come help you find a shapefile
  - Load it into R and plot it!

One last thing

- You have the population tif and the shapefile
- Extract the population data (using sum, don't forget!) to the shapefile
  - Use `append_cols` and make sure you choose the correct identifier!
- Join the data to the shapefile
- Plot the shapefile with the population data
  - Make it look nice!

What can you do with that data?

- Now you have a shapefile with population data
- You can save it as a `.csv` and use it in your analysis!
  - We'll get to this point eventually.
  - We will also discuss adding the survey data and then estimating a sub-area model

Creating a grid

- Yesterday, we used a grid in Korea
  - kgrid.shp
- By now, you can probably see that a grid is very similar to a raster!

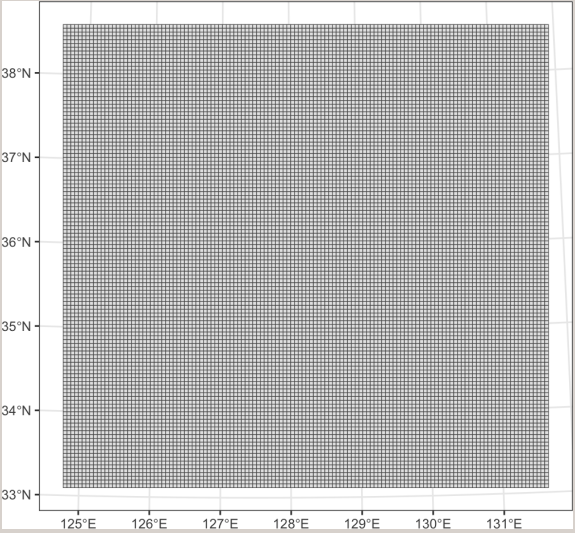
Load the shapefile

- Let's load kshape.shp

▼ Code

```
kshape <- vect("day2files/kshape.shp")
kgrid <- rast(kshape, res = 5000)
kgrid <- as.polygons(kgrid)
kgrid$id <- 1:nrow(kgrid)
```

The grid



Not quite done

- We aren't quite done. What do we want to do now?

▼ Code

```
intersection <- intersect(kshape, kgrid)
kgrid <- kgrid |>
  filter(id %in% intersection$id)
```

Not quite done