

SAE using geospatial data

Nairobi Workshop: Day 4 (geospatial data)

Ann-Kristin Kreutzmann

Josh Merfeld

August 26, 2024

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 SAE

- 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!

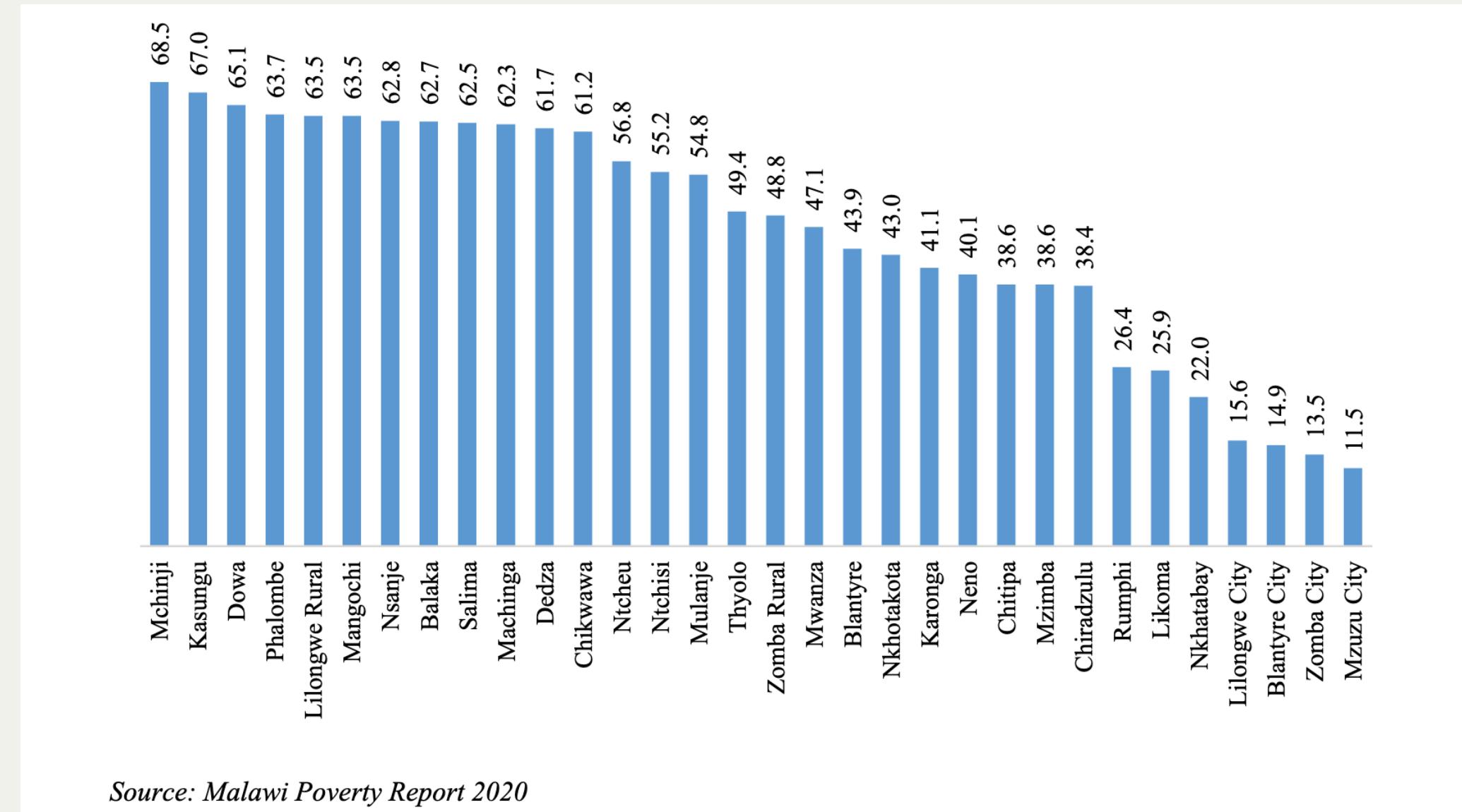
Think of what you need for SAE

- You need a sample, e.g. a household survey
 - This will only cover some of the country
- You need auxiliary data that is:
 - Predictive of the outcome you care about
 - Available throughout the entire country
- Some countries, use administrative data
 - But, importantly, it's often not available or is of low quality!

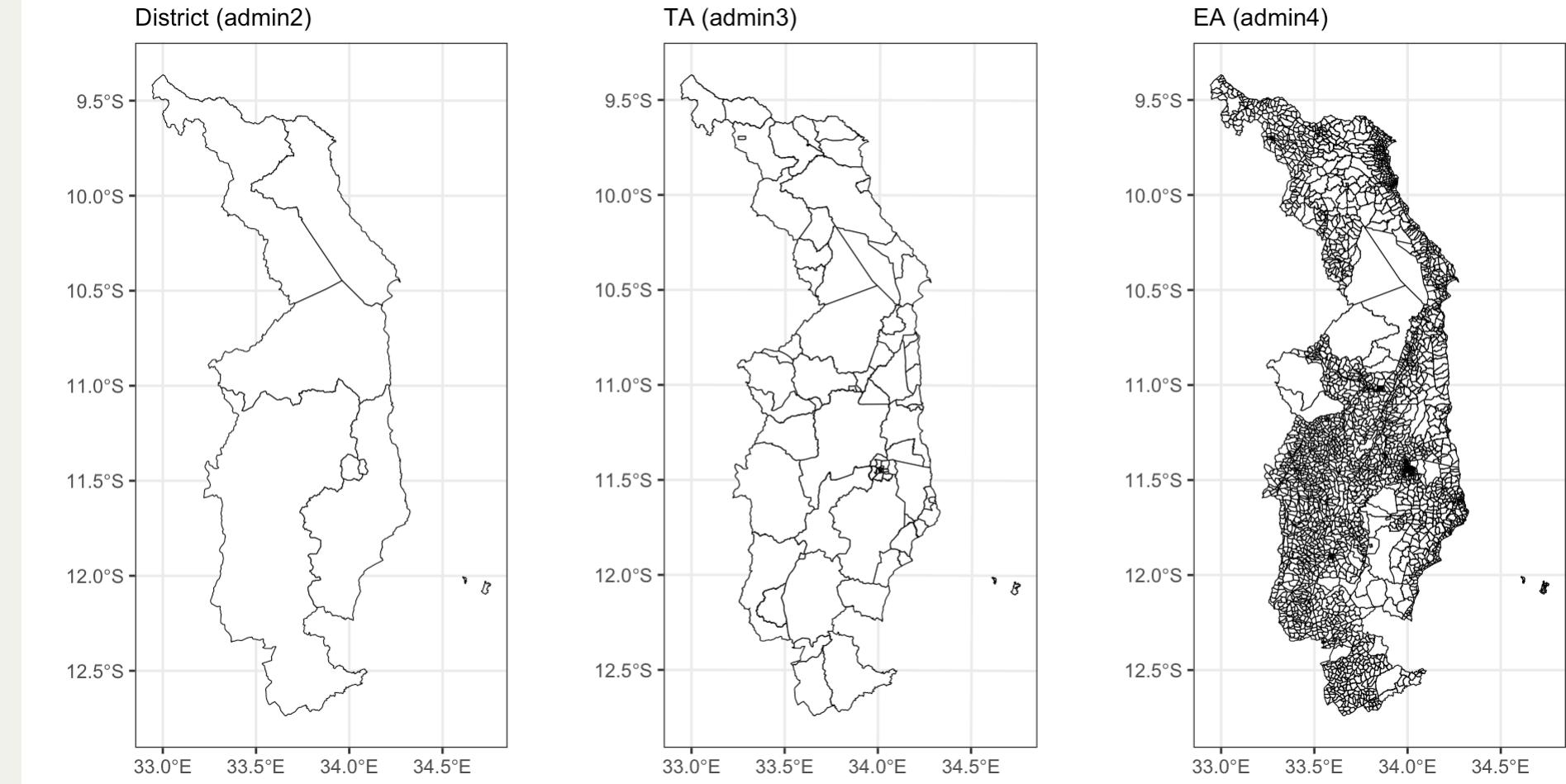
A quick example

- Let's take a look at Malawi
- Why Malawi?
 - I have survey data you can use 😊
 - Only going to use part of Malawi for this example (size of data)
- Consider the 2019/2020 Integrated Household Survey (IHS5)
 - Was used for the Malawi Poverty Report 2020
 - Can say things about poverty at the district level
 - If you want to split by urban/rural, only at the region level

A quick example

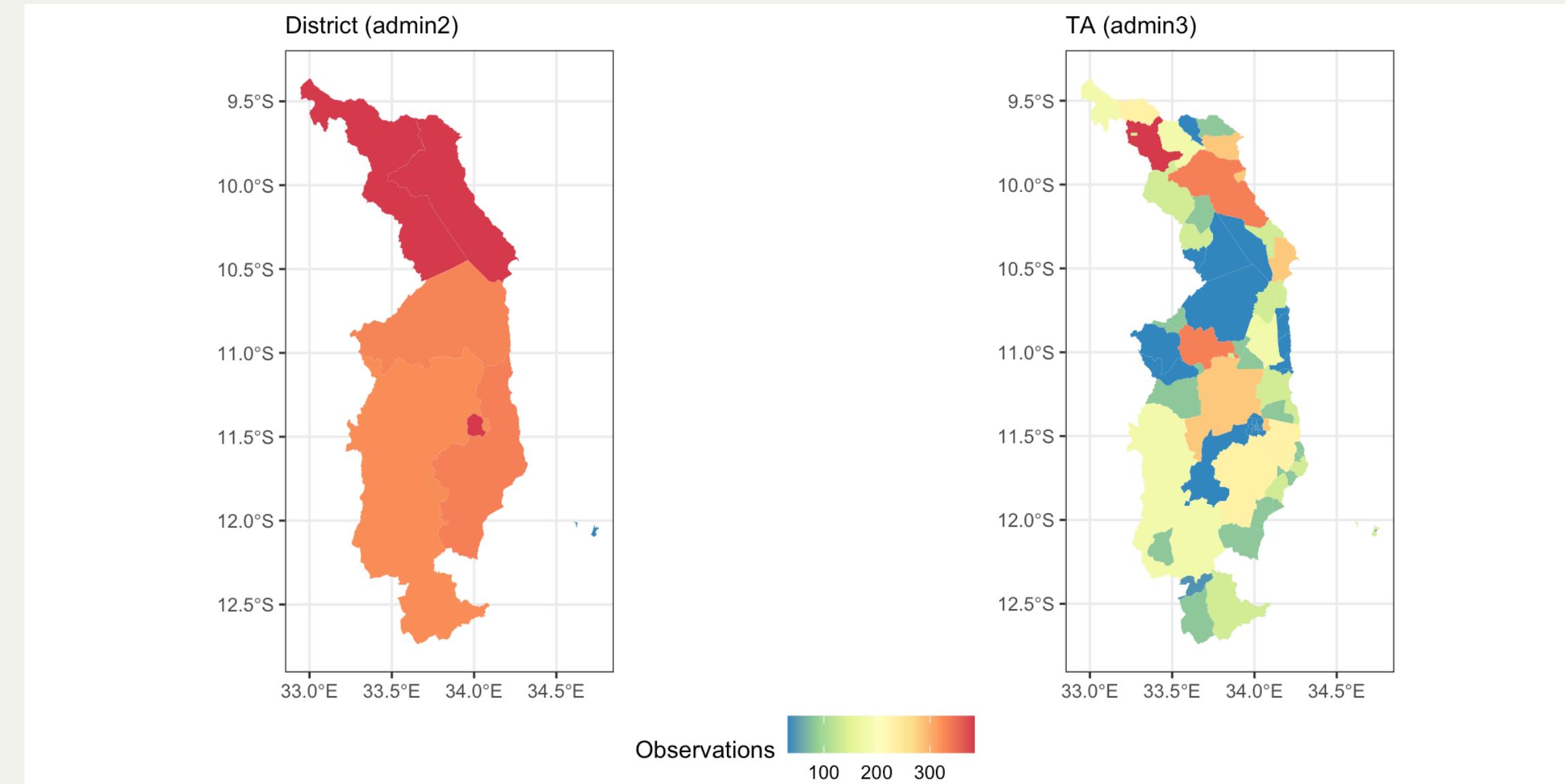


Malawi admin areas - Northern region only



- Survey only lets us say things about the districts!
- What if we want to say something about traditional authorities (TAs)?
- Individual TAs might not have enough observations
- We could use SAE! But what auxiliary data?

Observations at the district and TA level



Sub-area model with sectors

- One option: estimate a sub-area model at the EA level!
- Steps:
 - Collapse survey data to the EA level
 - Extract geospatial data at the EA level
 - Estimate the model

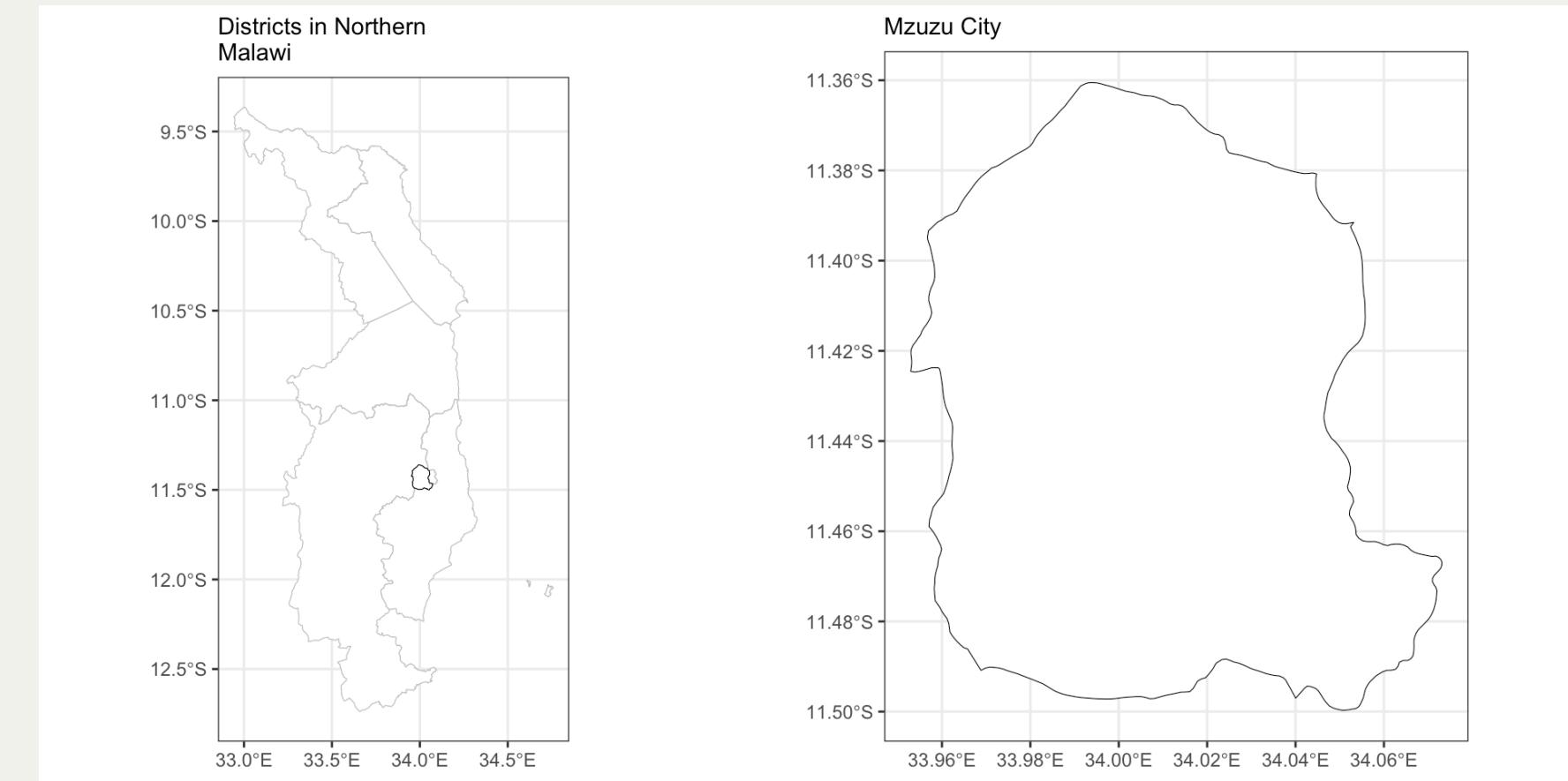
Getting started with geospatial data

- Due to time, this introduction will be necessarily brief
- We are going to learn about the following:
 - Shapefiles
 - Rasters
 - Extracting data

Shapefiles

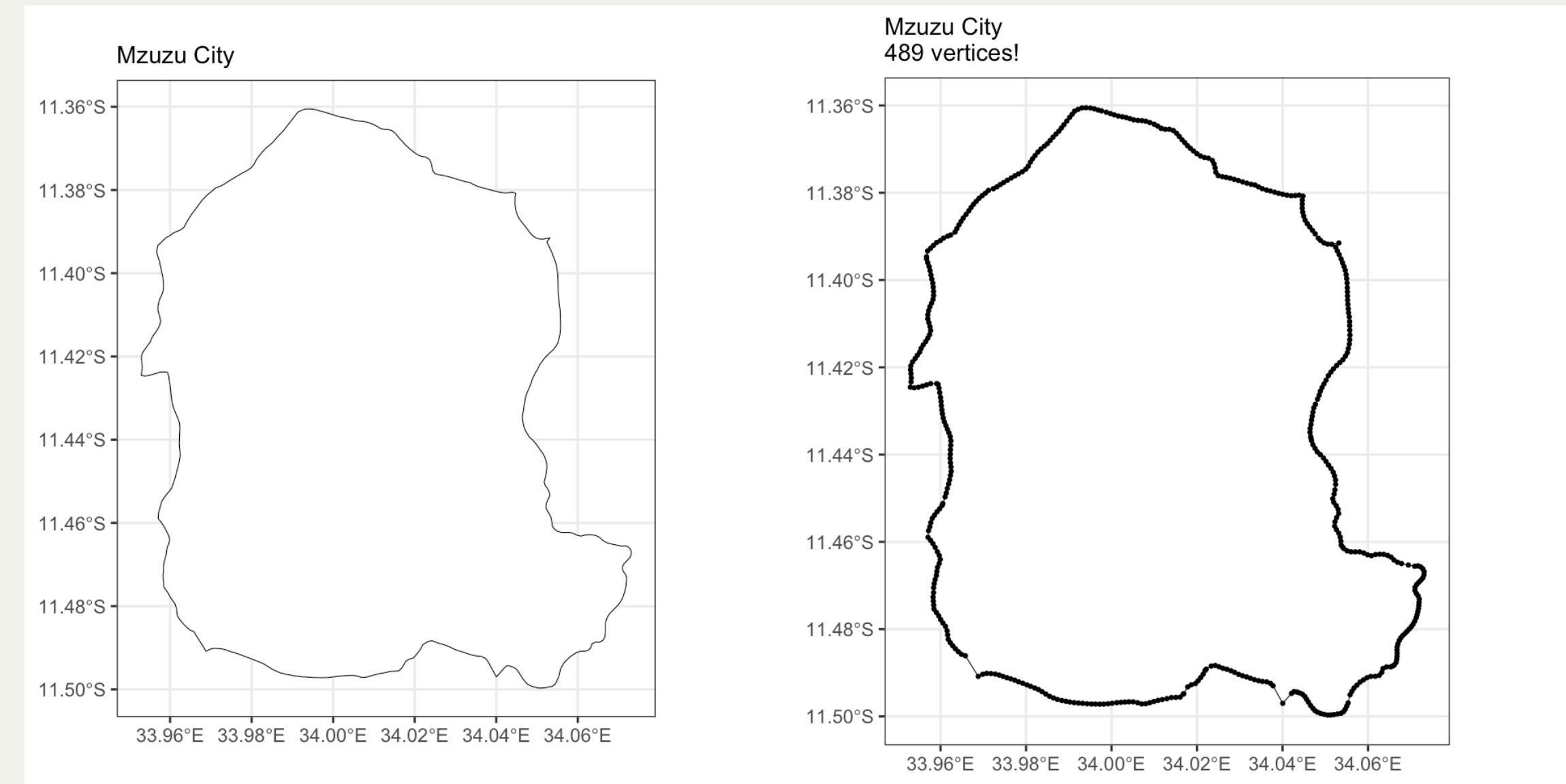
- The maps I just showed you are **shapefiles**
- Shapefiles are a common format for geospatial data
 - They are a form of **vector** data
- Shapefiles are made up of *at least* four files:
 - **.shp** - the shape itself
 - **.shx** - the index
 - **.dbf** - the attributes
 - **.prj** - the projection
 - What these all mean isn't important for now, just make sure they are there! Check the **day4data** folder on github.

Let's look at Northern Malawi again

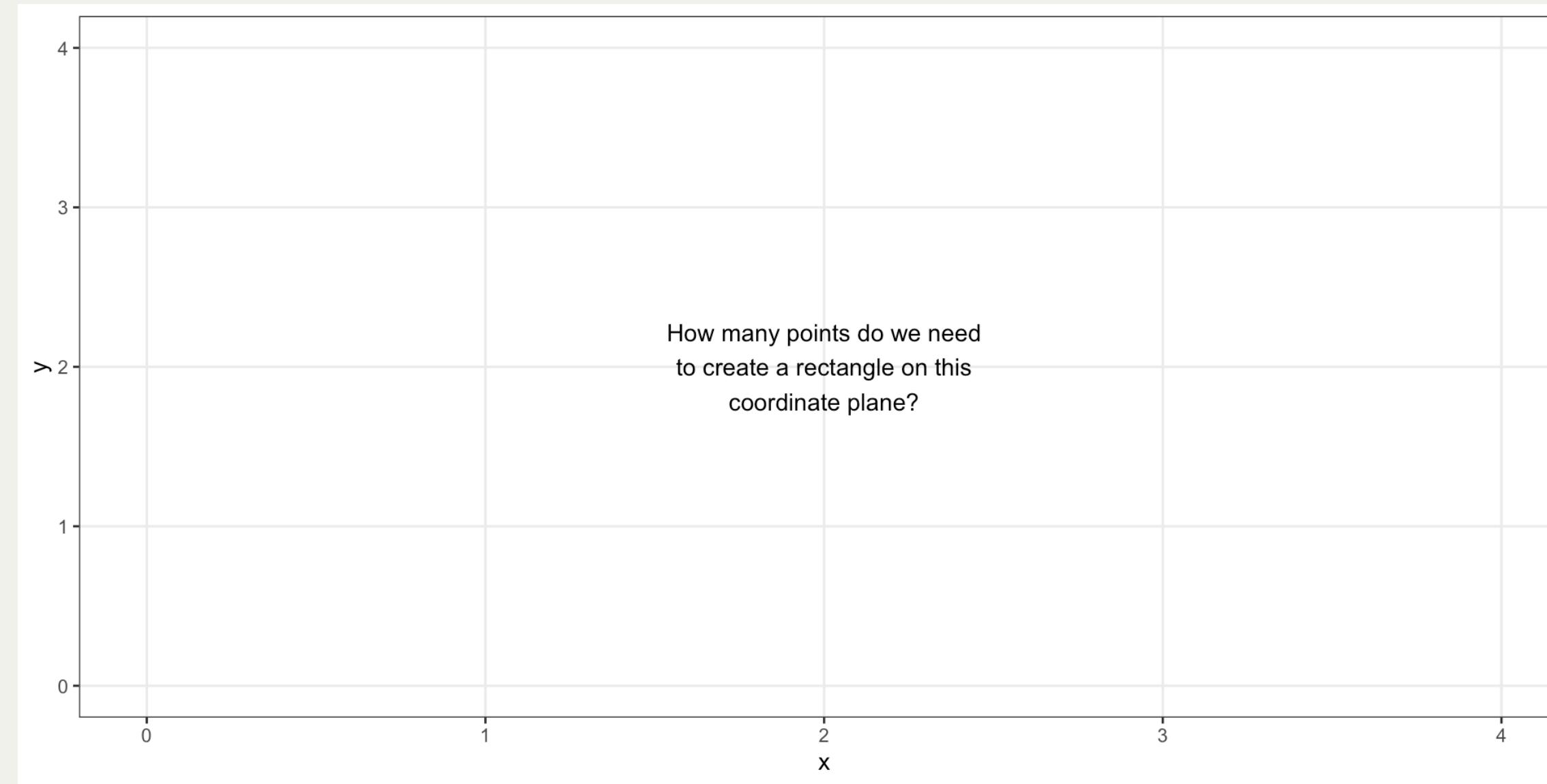


- Left: collection of **features**
- Right: one **feature**

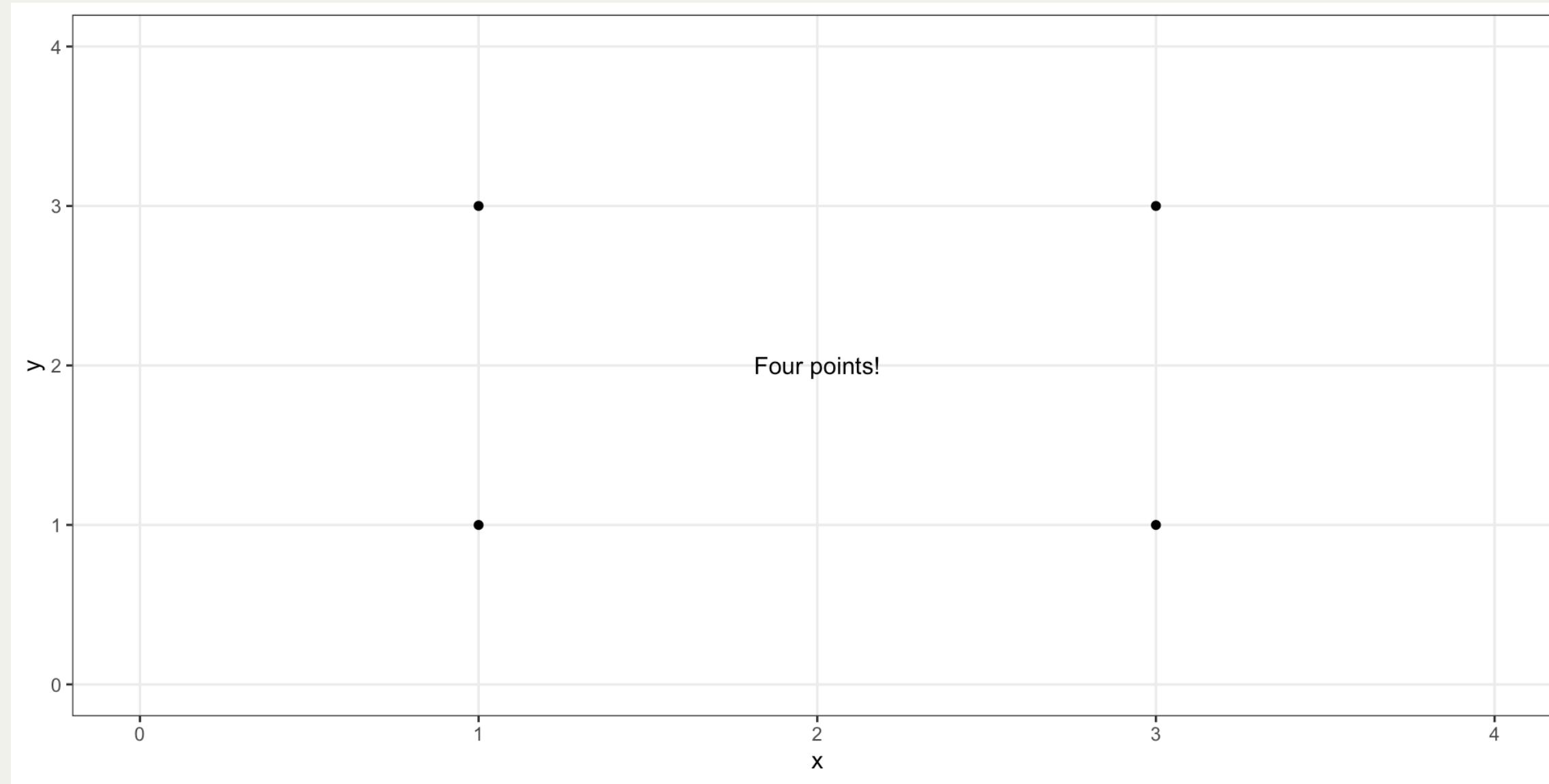
Features are made of vertices, which connect



Imagine a rectangle, on a coordinate plane



Imagine a rectangle, on a coordinate plane

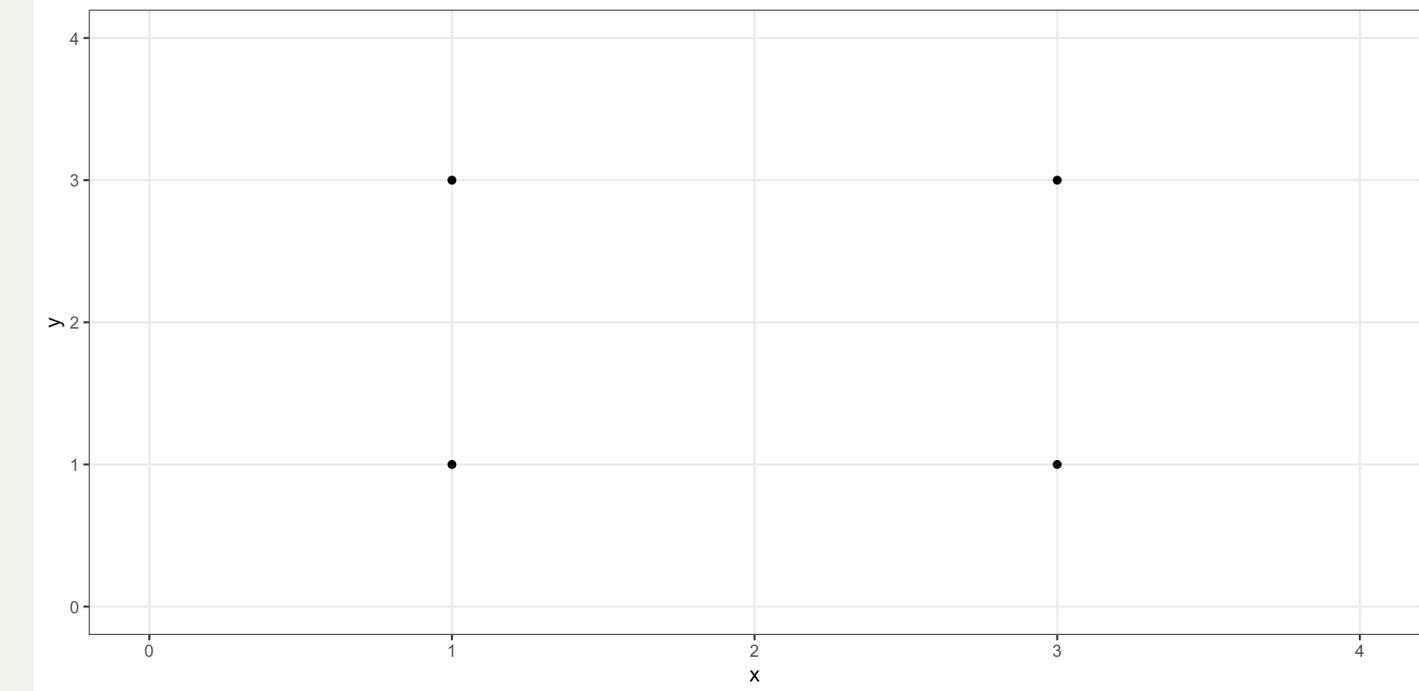


Imagine a rectangle, on a coordinate plane

- We need four points.
- But features in shapefiles are a little different.
 - We have to “close” the feature
- We do this by adding a fifth point: the same as the first point!

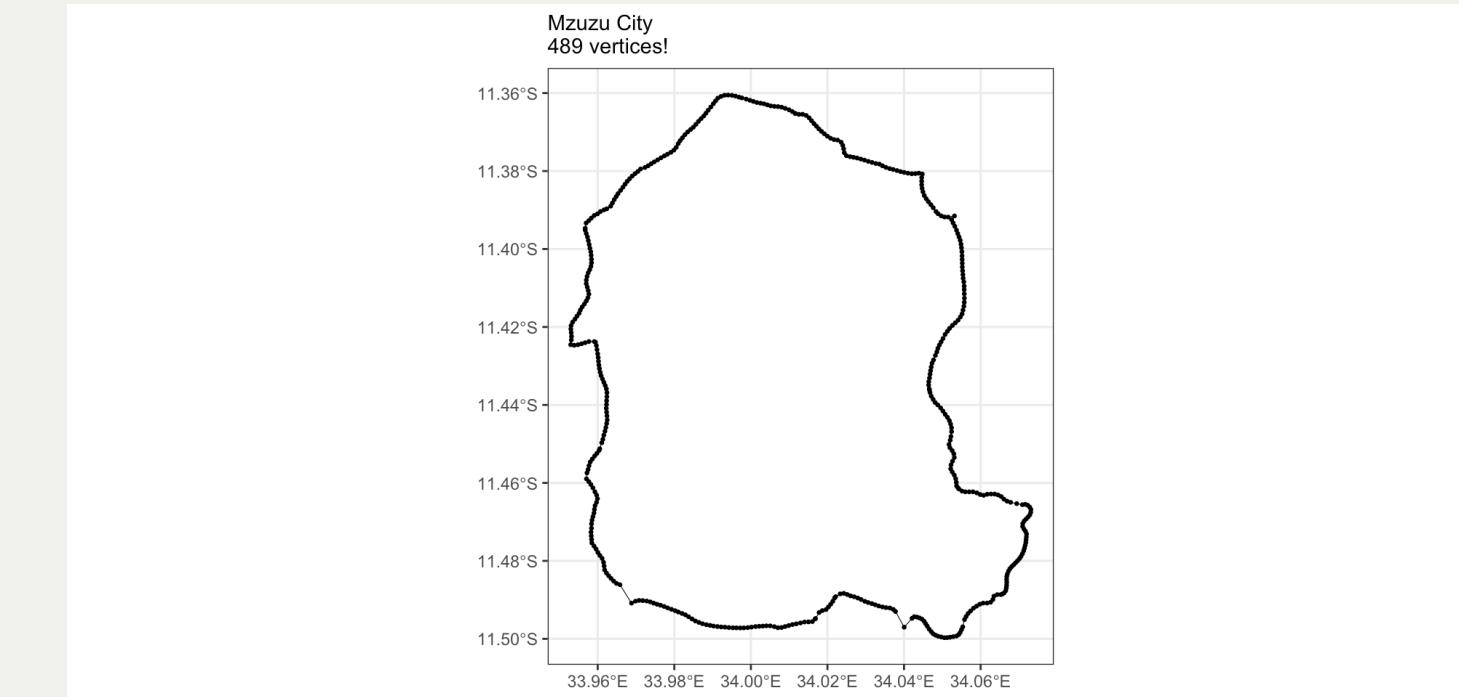
FIVE POINTS (VERTICES) IN OUR FEATURE

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



Features are made of vertices, which connect

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



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



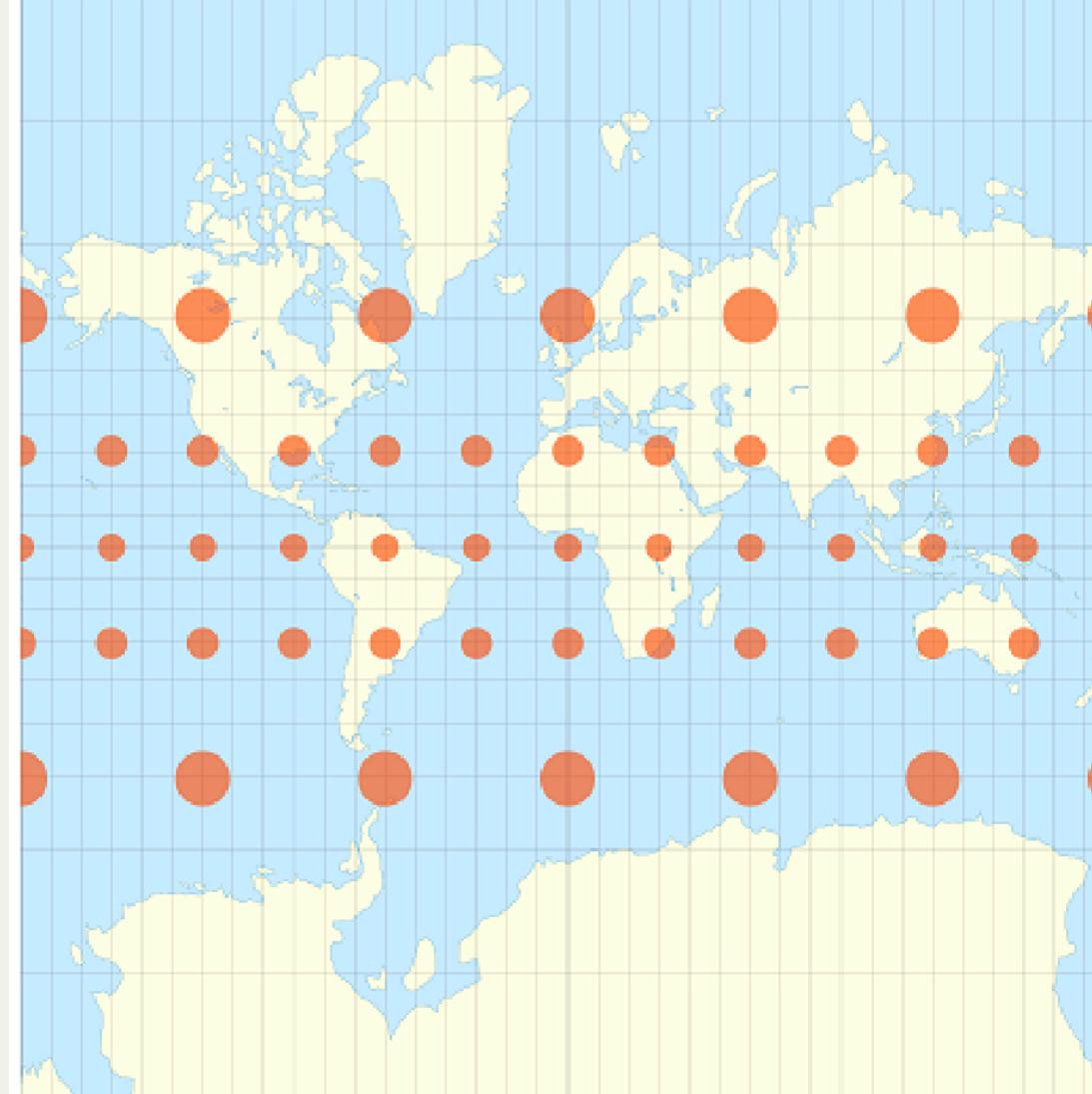
Source: wikipedia

The basic problem

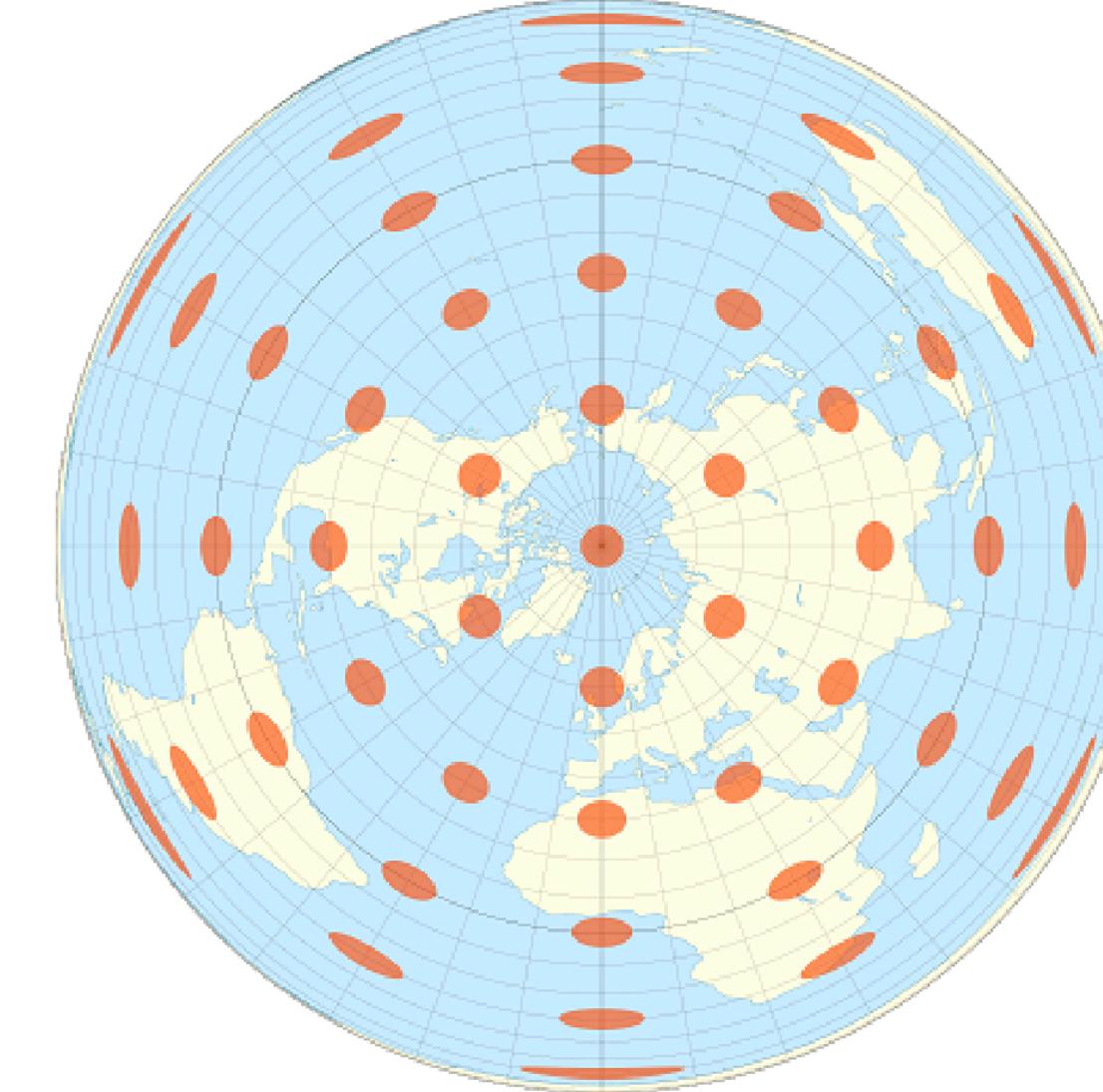
- The basic problem is that one degree of longitude changes at different latitudes!
 - At the equator, one degree of longitude is about 111 km
 - At 15N/S, one degree of longitude is about 107 km
 - At 30N/S, one degree of longitude is about 96 km
 - At 45N/S, one degree of longitude is about 79 km
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



Long story short...

- Using lat/lon is generally fine as long as you don't care about distances
 - But if you do, you need to use a different CRS
- Today we will focus on things that do not require distances
 - So we will generally use lat/lon

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

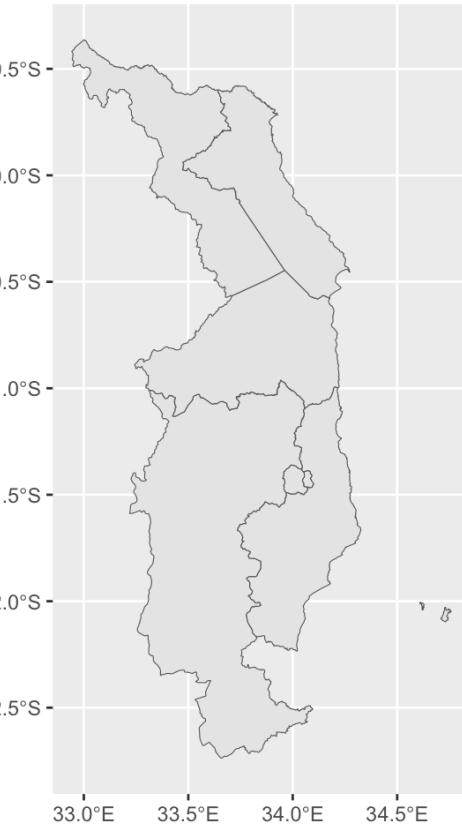
```
1 library(sf)
2 # this is the shapefile for the northern region of Malawi, district level
3 northmw <- read_sf("day4data/mw2.shp")
4 northmw
```

```
Simple feature collection with 7 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 32.9424 ymin: -12.73669 xmax: 34.75834 ymax: -9.363662
Geodetic CRS:  WGS 84
# A tibble: 7 × 2
  DIST_CODE                                geometry
  <chr>                                     <MULTIPOLYGON [°]>
1 101    (((33.00618 -9.367976, 33.00641 -9.368012, 33.00666 -9.367922, 33.00692 -9.367849, 3...
2 102    (((33.73312 -9.581763, 33.73382 -9.582024, 33.73453 -9.581968, 33.73522 -9.581958, 3...
3 106    (((34.73705 -12.04239, 34.7378 -12.04253, 34.73849 -12.04241, 34.73922 -12.04221, 34...
4 105    (((34.079 -11.38773, 34.07986 -11.38818, 34.08067 -11.38875, 34.08152 -11.38936, 34....
5 107    (((34.01161 -11.36523, 34.01255 -11.36546, 34.0135 -11.36544, 34.01444 -11.36573, 34...
```

Plotting is also very easy

▼ Code

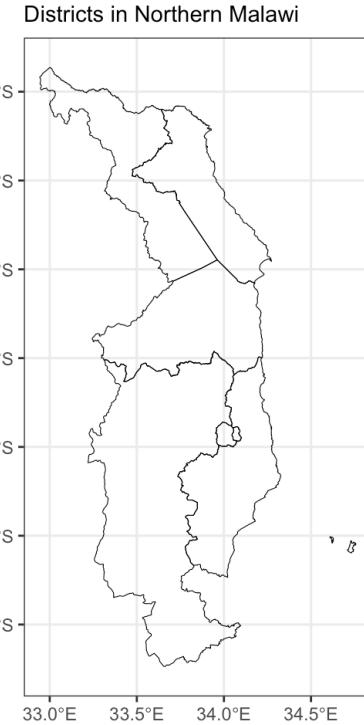
```
1 ggplot() +  
2   geom_sf(data = northmw)
```



My go-to theme

▼ Code

```
1 ggplot() +  
2   geom_sf(data = northmw, fill = NA, color = "black") +  
3   theme_bw() +  
4   labs(subtitle = "Districts in Northern Malawi")
```



Give it a try with TAs (mw3.shp)

▼ Code

```
1 library(sf)
2 # this is the shapefile for the northern region of Malawi, TA level
3 northmw <- read_sf("day4data/mw3.shp")
4 ggplot() +
5   geom_sf(data = northmw)
```

▼ Code

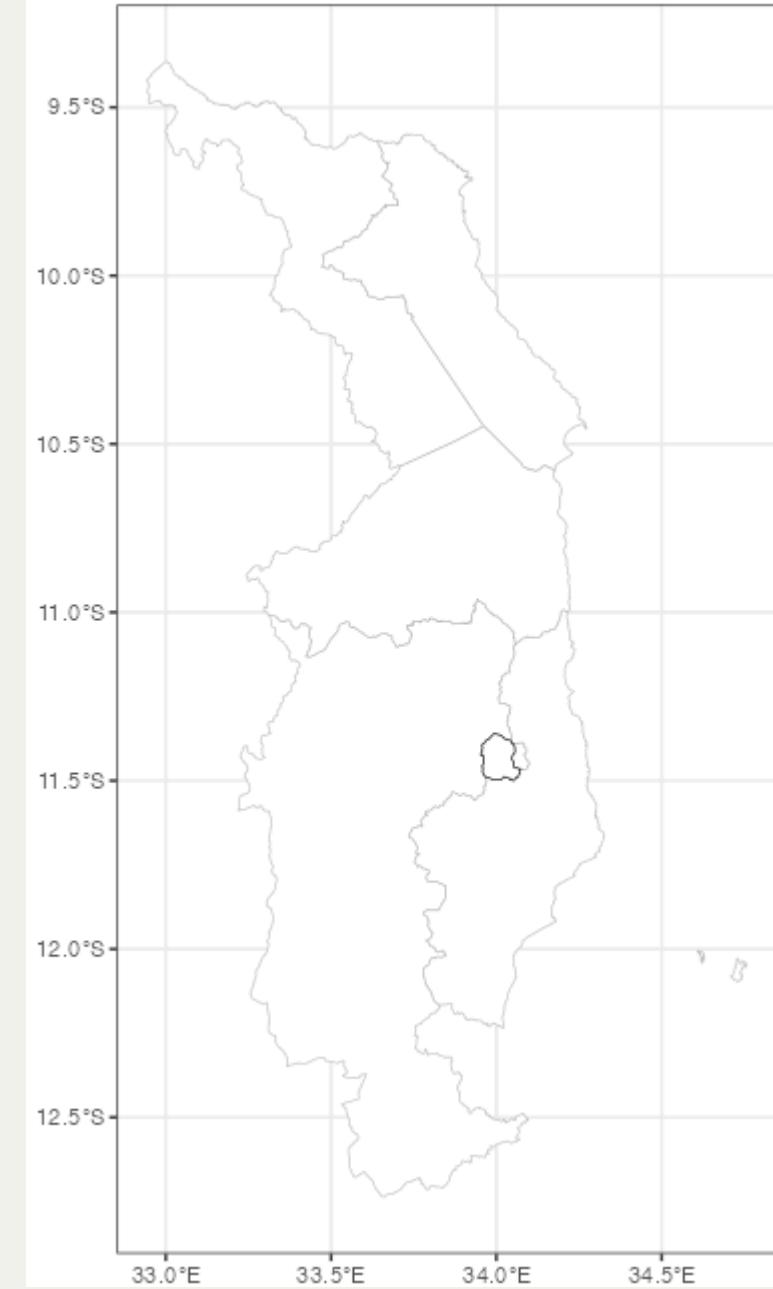
```
1 ggplot() +
2   geom_sf(data = northmw, fill = NA, color = "black") +
3   theme_bw() +
4   labs(subtitle = "TAs in Northern Malawi")
```

One more example - map from earlier

▼ Code

```
1 admin2 <- read_sf("day4data/mw2.shp")  
2  
3 ggplot() +  
4   geom_sf(data = admin2,  
5             fill = "white", color = "gray") +  
6   geom_sf(data = admin2 |> filter(DIST_CODE=="10"),  
7             fill = "white", color = "black") +  
8   theme_bw() +  
9   labs(subtitle = "Districts in Northern Malawi")
```

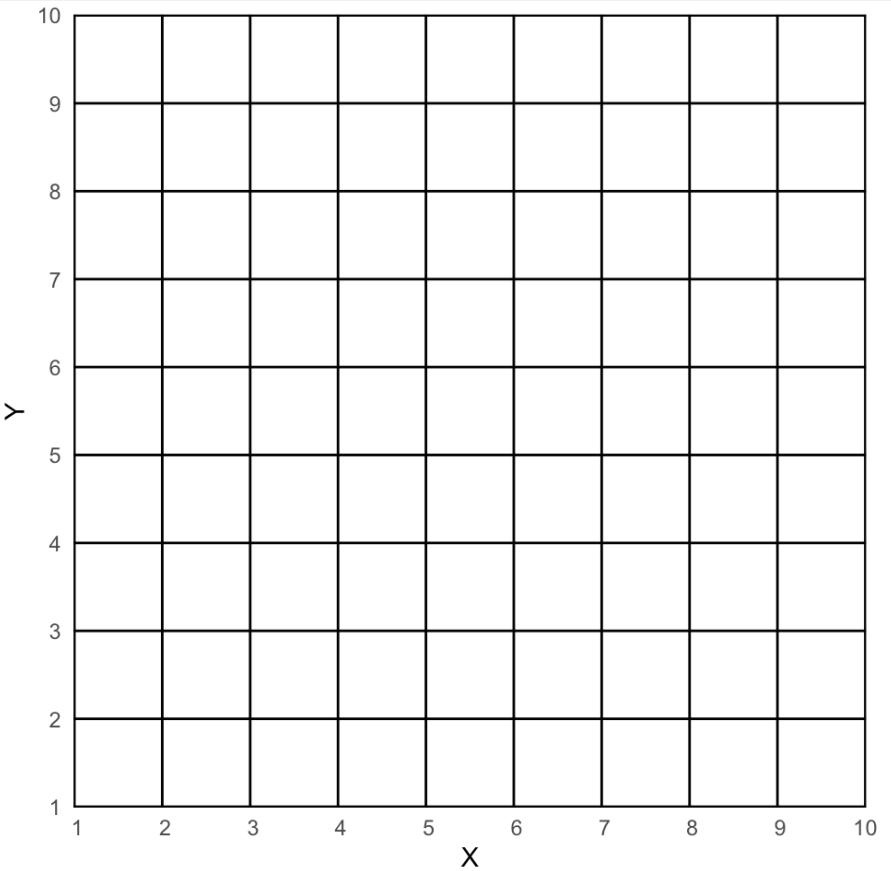
Districts in Northern Malawi



Rasters

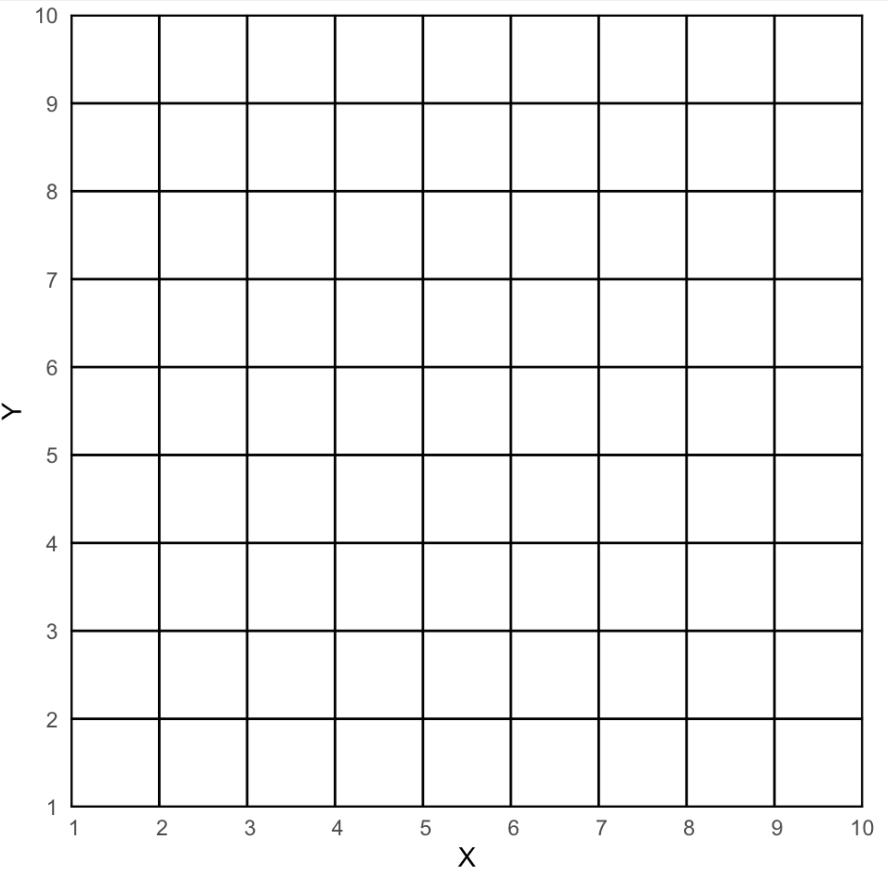
- We've discussed shapefiles
 - Now let's talk about rasters!
- Rasters are a different type of geospatial data
 - They are made up of a grid of cells
 - Each cell has a value

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?

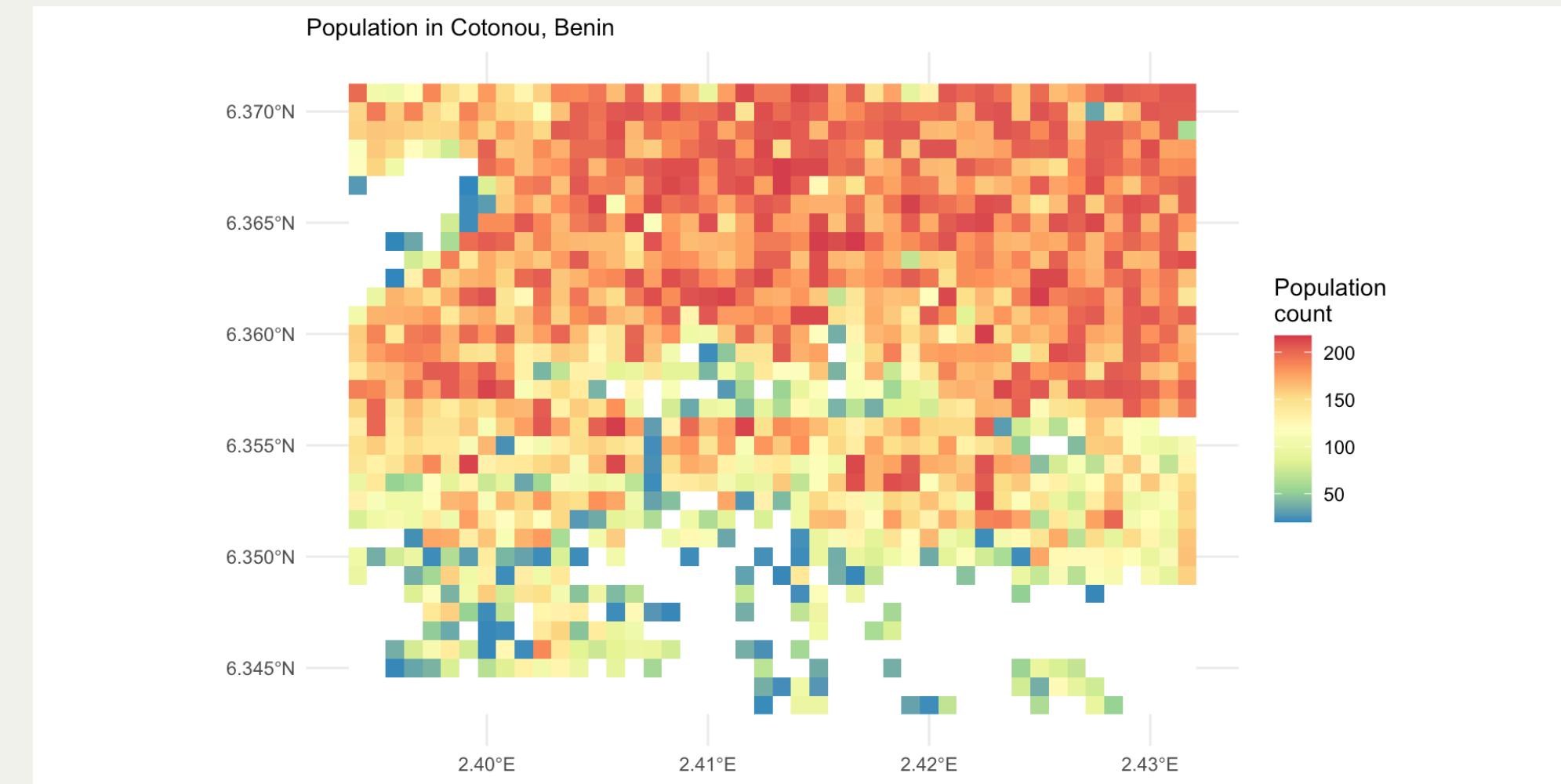


- Need to know location of one grid cell...
 - And the size of each grid!

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
 - `day4data/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("day4data/beninpop.tif")
5 cotonou
```

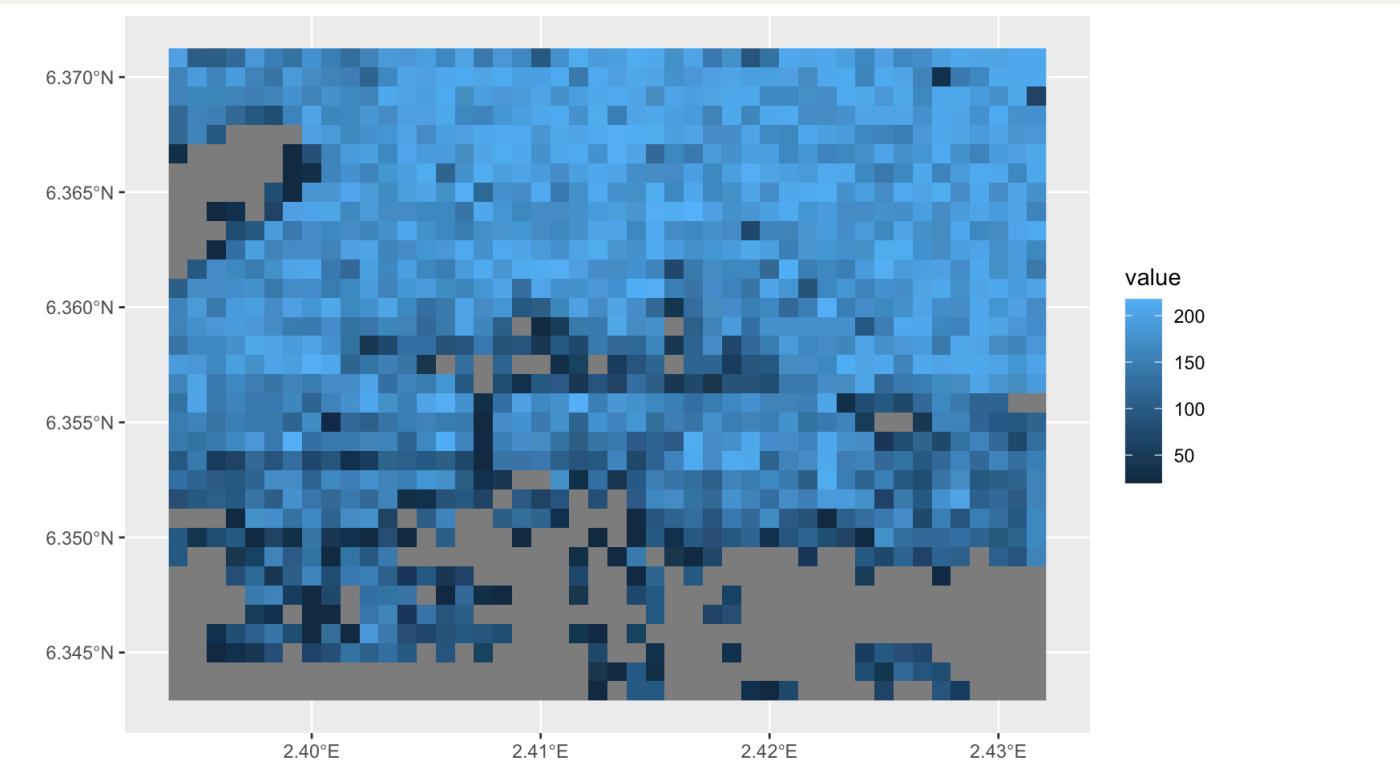
```
class       : SpatRaster
dimensions  : 34, 46, 1  (nrow, ncol, nlyr)
resolution  : 0.0008333333, 0.0008333333  (x, y)
extent      : 2.39375, 2.432083, 6.342917, 6.37125  (xmin, xmax, ymin, ymax)
coord. ref. : lon-lat WGS 84 (EPSG:4326)
```

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

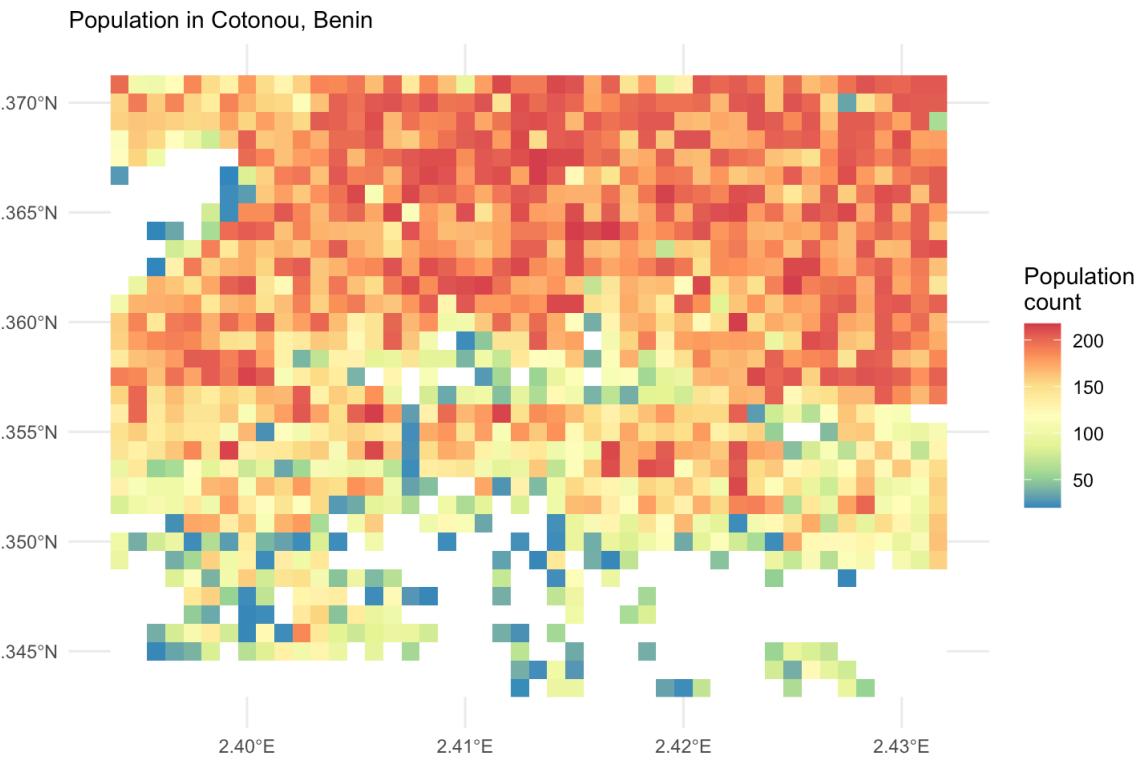
```
1 library(tidyterra)
2
3 ggplot() +
4   geom_spatraster(data = cotonou)
```



Making it nicer

▼ Code

```
1 library(tidyterra)
2
3 ggplot() +
4   geom_spatraster(data = cotonou) +
5   # distiller is for continuous values
6   # but we can use palettes!
7   # I like spectral a lot
8   scale_fill_distiller("Population\ncount",
9     palette = "Spectral", na.value = "white") +
10  theme_minimal() +
11  labs(subtitle = "Population in Cotonou, Benin")
```



Extracting raster data to shapefiles

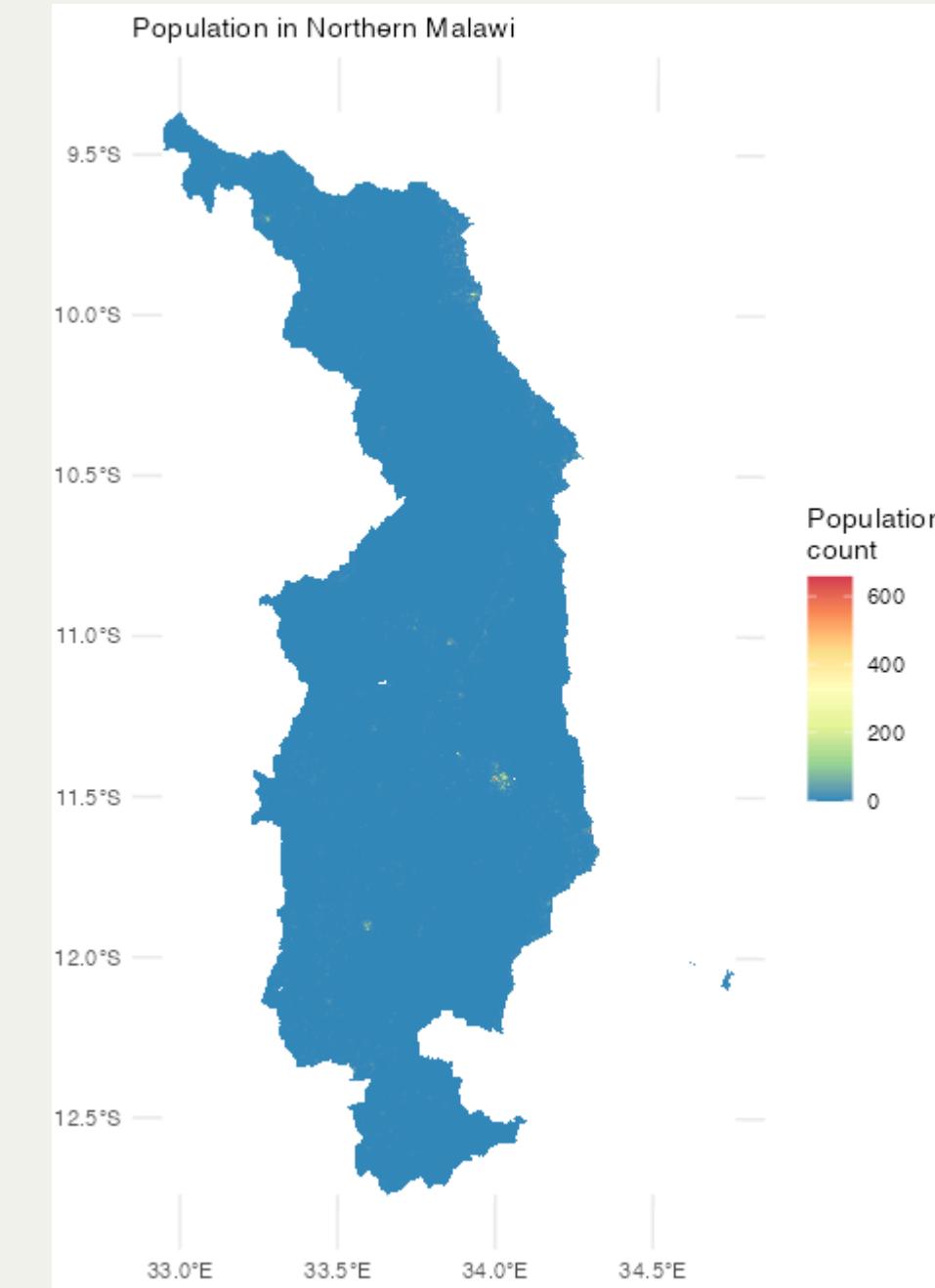
- Let's go back to our use case:
 - We want to estimate a sub-area model at the EA level in Malawi
 - This means we need to extract raster data to the EA level
 - We can do this with `terra`, `sf`, and `exactextractr`
 - `terra` has its own method, but i find `exactextractr` to be MUCH faster
- Let's start by looking at the raster I've uploaded to the day4data: `mwpop.tif`

Give it a try

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

▼ Code

```
1 tif <- rast("day4data/mwpop.tif")
2
3 ggplot() +
4   geom_spatraster(data = tif) +
5   scale_fill_distiller("Population\ncount",
6     palette = "Spectral", na.value = "white") +
7   theme_minimal() +
8   labs(subtitle = "Population in Northern Malawi")
```



Give it a try

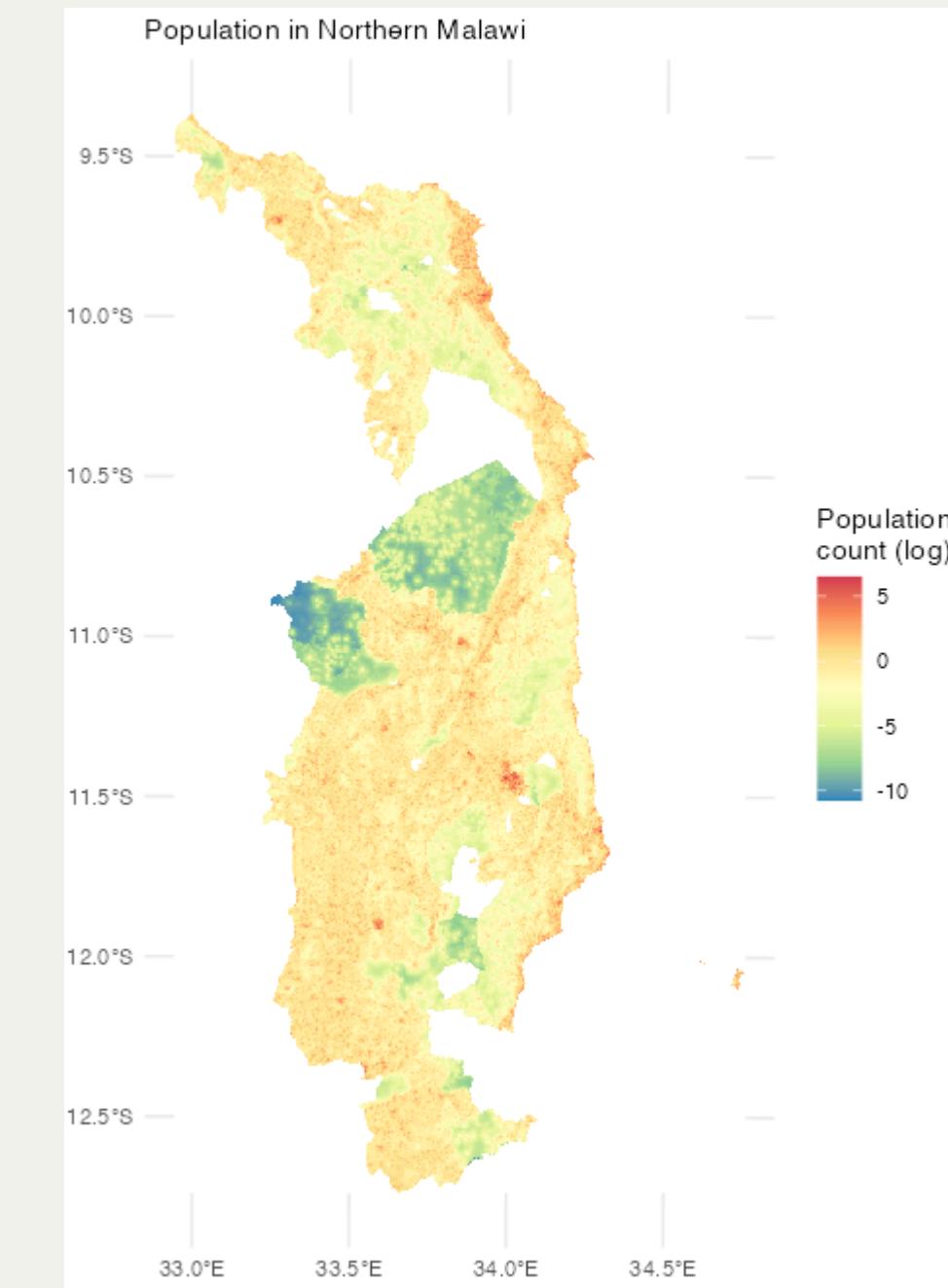
- I actually don't like that map! It's too hard to see because of all the low values.
- So let's take logs, instead!
 - Note that all the zeros become missing (can't log zero)

▼ Code

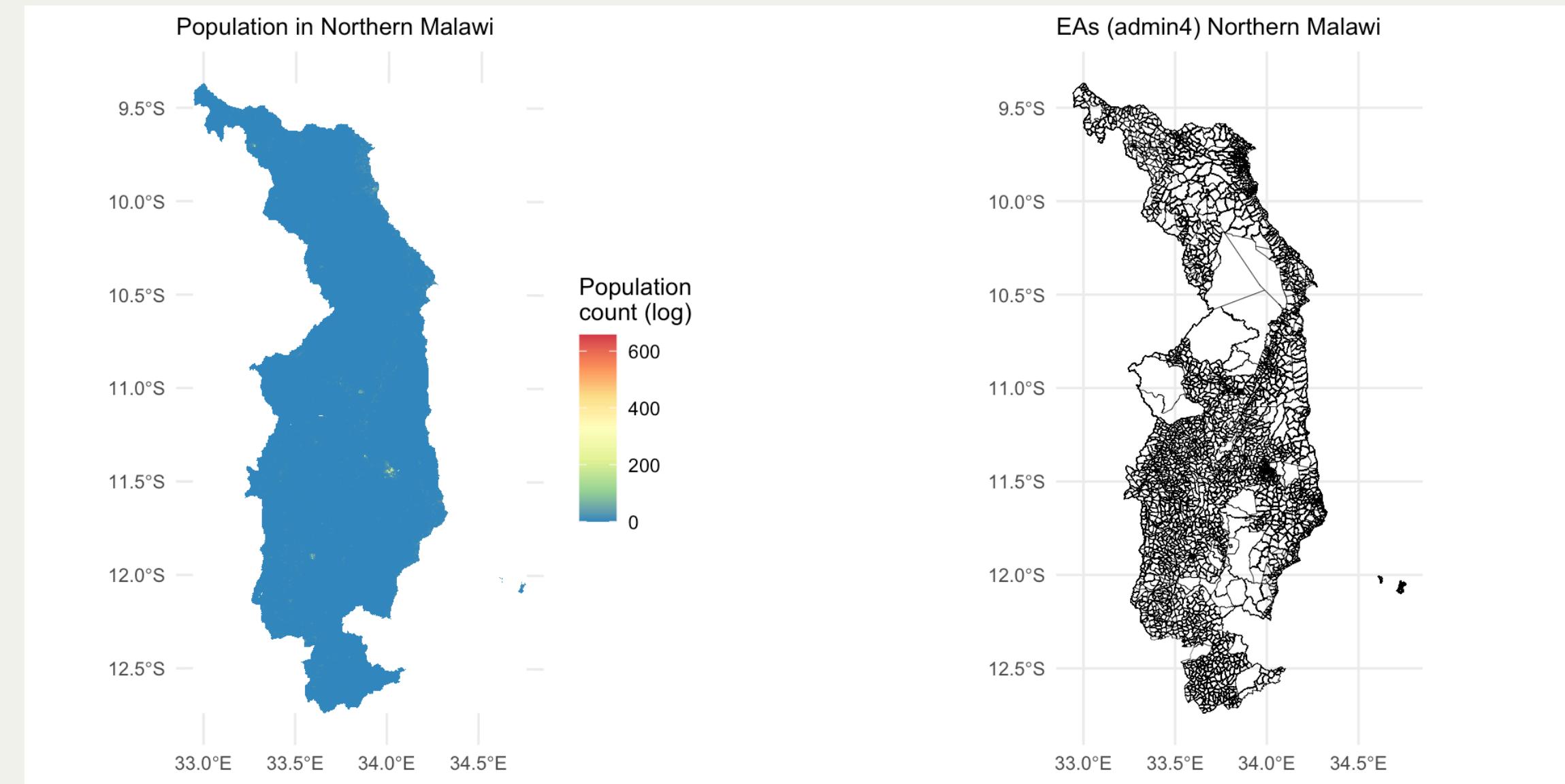
```

1 tif <- rast("day4data/mwpop.tif")
2
3 ggplot() +
4   geom_spatraster(data = log(tif)) +
5   scale_fill_distiller("Population\ncount (log)"
6     palette = "Spectral", na.value = "white") +
7   theme_minimal() +
8   labs(subtitle = "Population in Northern Malawi"

```



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



Let's do it with exactextractr

▼ Code

```
1 library(exactextractr)
2
3 tif <- rast("day4data/mwpop.tif")
4 adm4 <- read_sf("day4data/mw4.shp")
5 # make sure they are in the same CRS! (they already are, but just in case)
6 # st_transform is for the sf object
7 adm4 <- st_transform(adm4, crs = crs(tif))
8
9 # extract the raster values to the shapefile
10 # we are going to SUM, and add the EA_CODE from the shapefile to the result
11 extracted <- exact_extract(tif, adm4, fun = "sum", append_cols = "EA_CODE")
```

▼ Code

```
1 head(extracted)
```

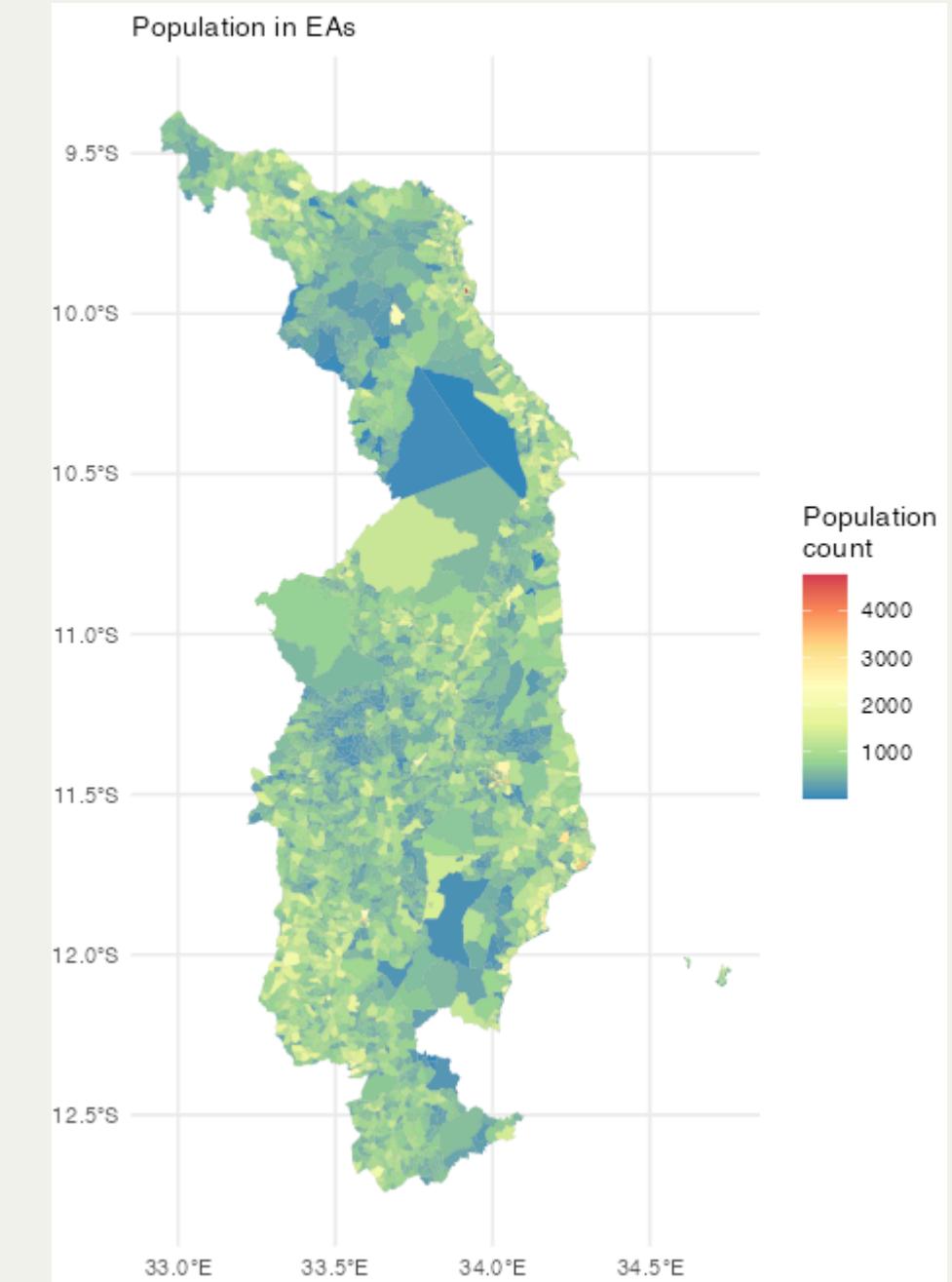
	EA_CODE	sum
1	10507801	1068.7787
2	10507072	695.8013
3	10507010	938.1139

4 10507001 749.6960
5 10507009 597.5432
6 10507033 474.3934

Now we can join the extracted data to the shapefile

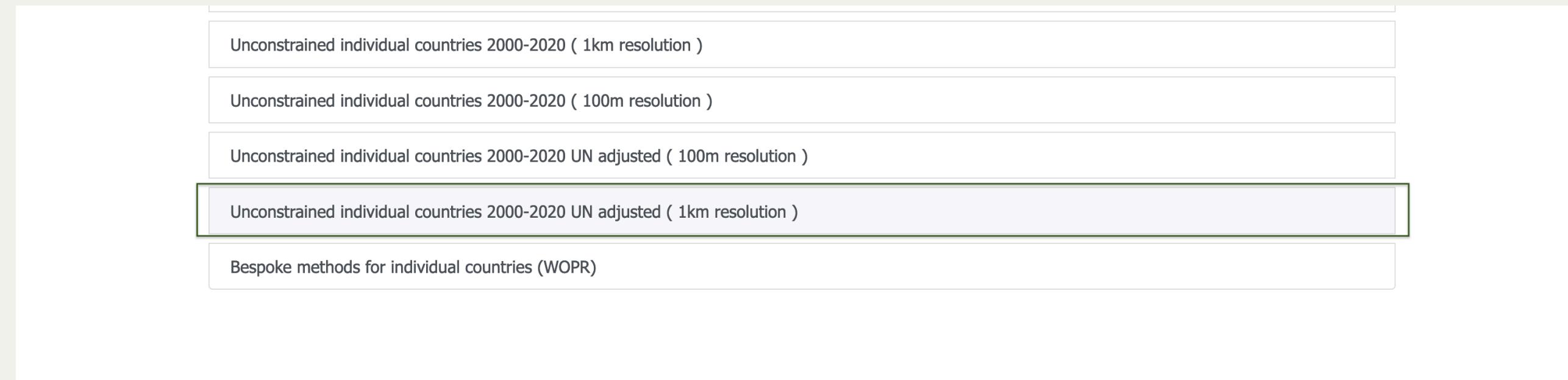
▼ Code

```
1 # join
2 adm4 <- adm4 |>
3   left_join(extracted, by = "EA_CODE")
4
5 # plot it!
6 ggplot() +
7   geom_sf(data = adm4, aes(fill = sum),
8     color = "black", lwd = 0.01) +
9   scale_fill_distiller("Population\ncount",
10    palette = "Spectral", na.value = "white") +
11   theme_minimal() +
12   labs(subtitle = "Population in EAs")
```



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?)

WorldPop Hub DATA | CONTACT

Population Counts

Population Counts / Unconstrained individual countries 2000-2020 UN adjusted (1km resolution)

Individual countries 2000-2020 UN adjusted aggregated to 1km resolution using [100m resolution population count datasets](#). The dataset is available to download in Geotiff and ASCII XYZ format at a resolution of 30 arc (approximately 1km at the equator). The projection is Geographic Coordinate System, WGS84. The units are number of people per pixel with country totals adjusted to match the corresponding official United Nations population estimates that have been prepared by the Population Division of the Department of Economic and Social Affairs of the United Nations Secretariat ([2019 Revision of World Population Prospects](#)). The mapping approach is [Random Forest-based dasymetric redistribution](#). The methodology used to estimate the annual subnational census-based figures can be found in [Lloyd et al](#), while the information and sources of the input population data are [available here](#). Percentage difference between the subnational census-based and UN-based figures in 2010 has been visualised in [2D map](#).

Show 25 rows entries

Search ...

Continent	Country	Year	Geo Type	RES	Data & Resources
Africa	Algeria	2000	Population	1km	Data & Resources
Africa	Algeria	2001	Population	1km	Data & Resources
Africa	Algeria	2002	Population	1km	Data & Resources

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!

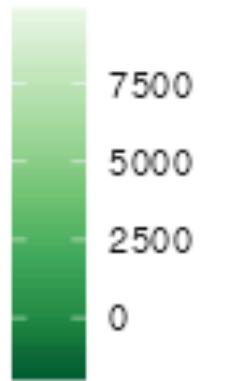
What can you do with that data?

- 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

Finding rasters

10.3°N

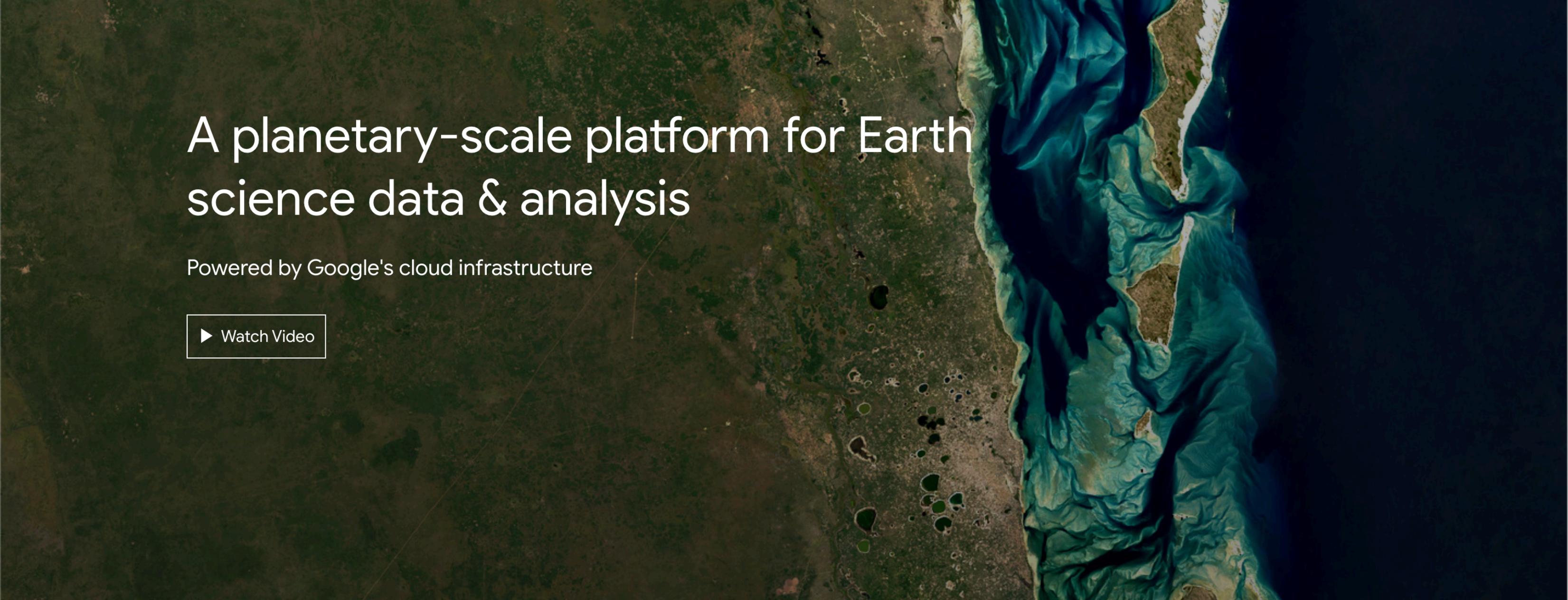
NDVI



10.2°N

Where can you find rasters?

- Depending on the variable, rasters are sometimes quite easy to find!
 - We already saw one example: WorldPop (population counts)
- There are two large online repositories:
 - Google Earth Engine
 - Microsoft Planetary Computer
 - This one is newer and has less data (for now)



A planetary-scale platform for Earth science data & analysis

Powered by Google's cloud infrastructure

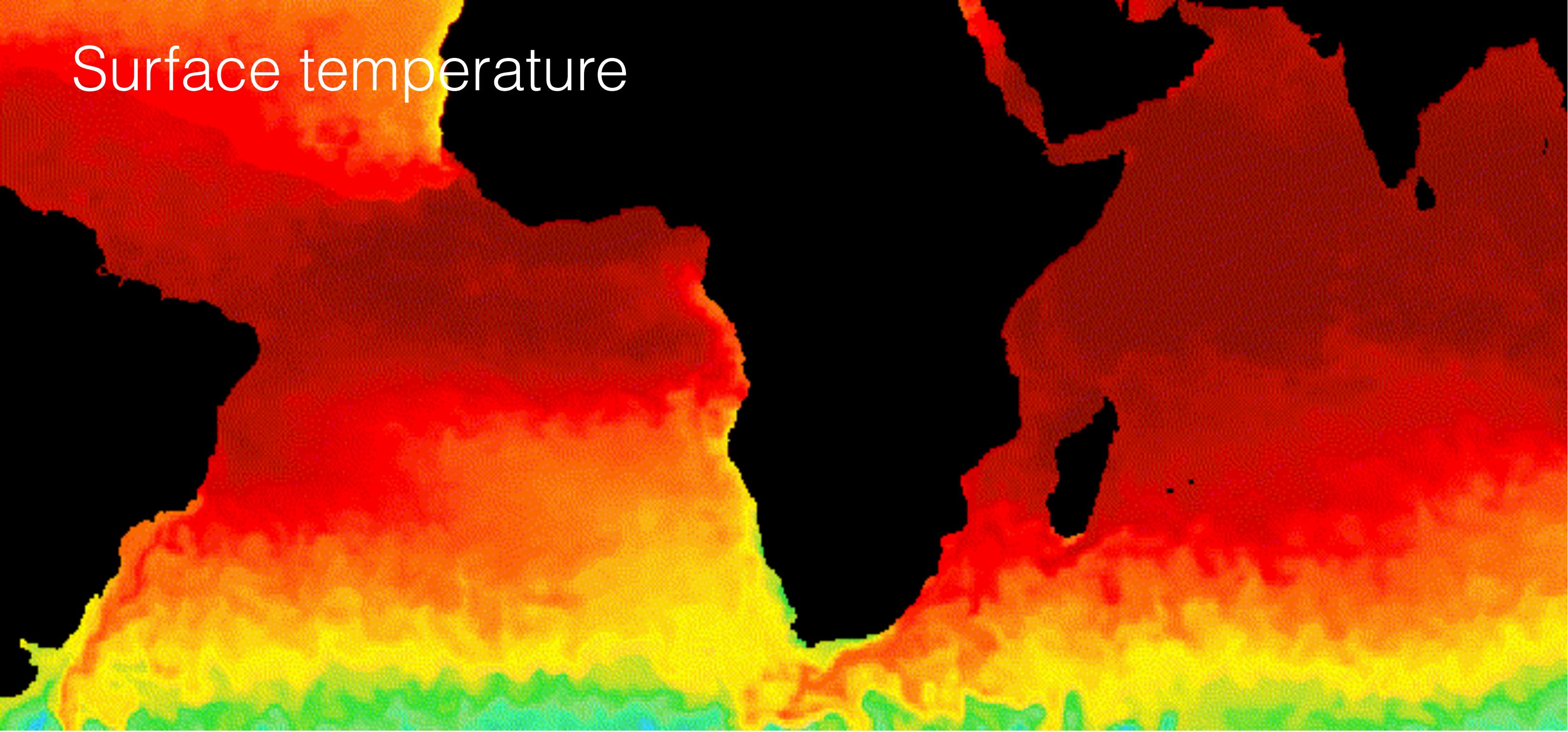
▶ Watch Video

Meet Earth Engine

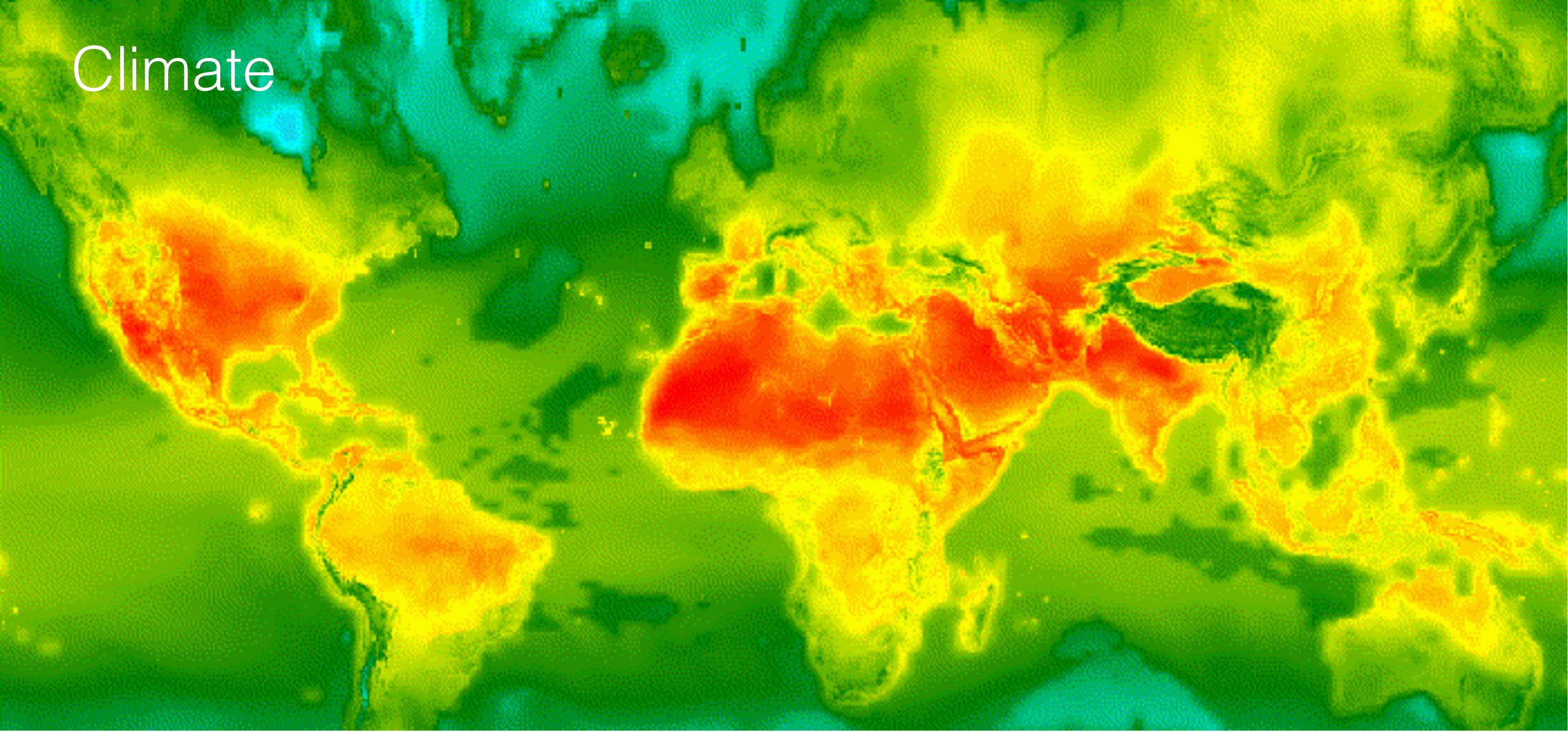
Google Earth Engine

- Google Earth Engine has *a lot* of data.
- Let's see some examples

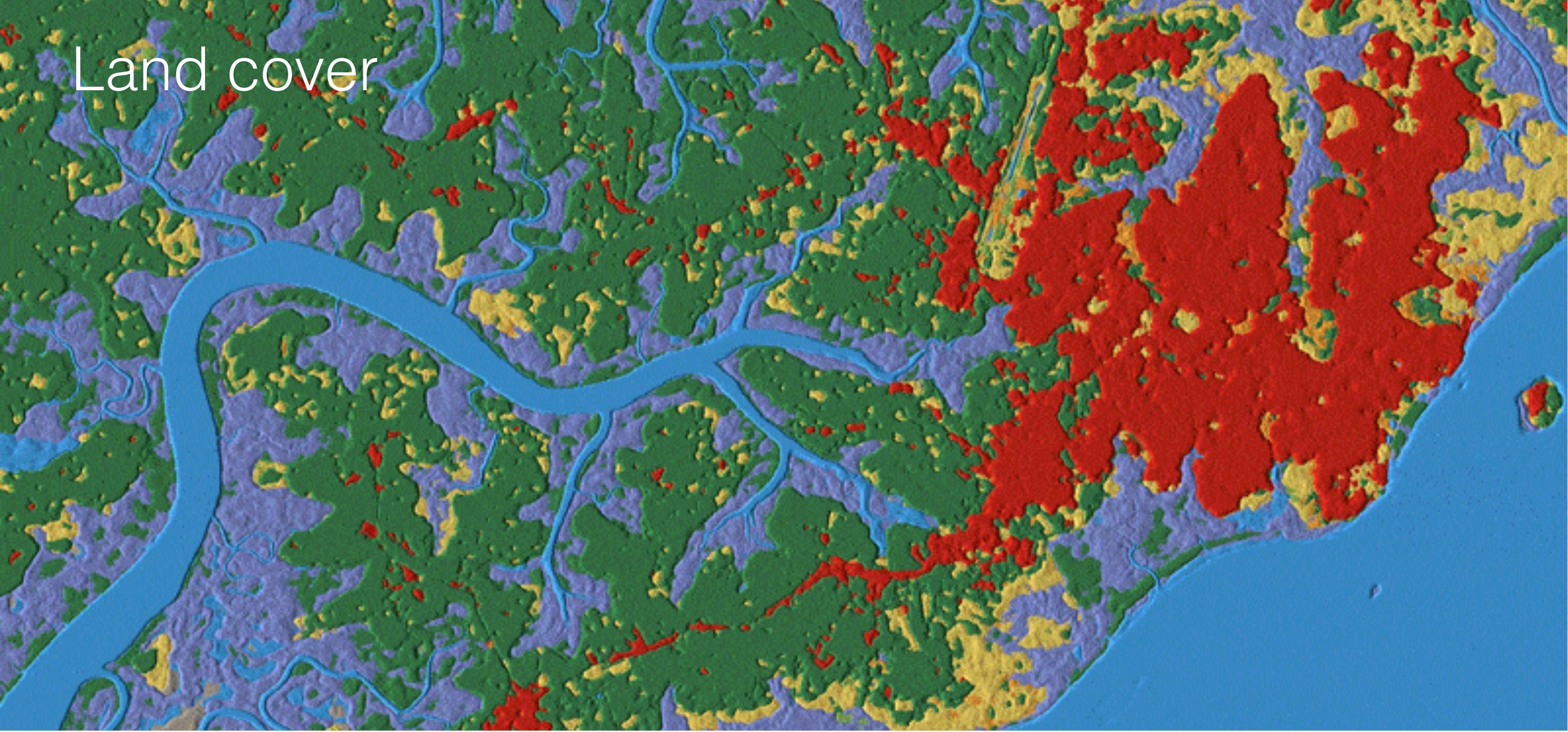
Surface temperature



Climate



Land cover





Imagery

GeoLink

- I'm going to start with some example code for a package called **GeoLink**
 - This package actually uses Microsoft Planetary Computer
 - It is very very easy to use
 - The problem: as of now, it only has a few datasets

▼ Code

```
1 # we have to download it differently
2 devtools:::install_github("SSA-Statistical-Team-Projects/GeoLink")
```

The actual code is much easier to use than GEE!

▼ Code

```
1 library(GeoLink)
2 library(sf)
3
4 adm4 <- read_sf("mw4.shp")
5 # CHIRPS is rainfall data
6 adm4_chirps <- geolink_chirps(time_unit = "month",
7   start_date = "2019-01-01",
8   end_date = "2019-03-01",
9   shp_dt = adm4,
10  extract_fun = "mean")
11 summary(adm4_chirps)
```

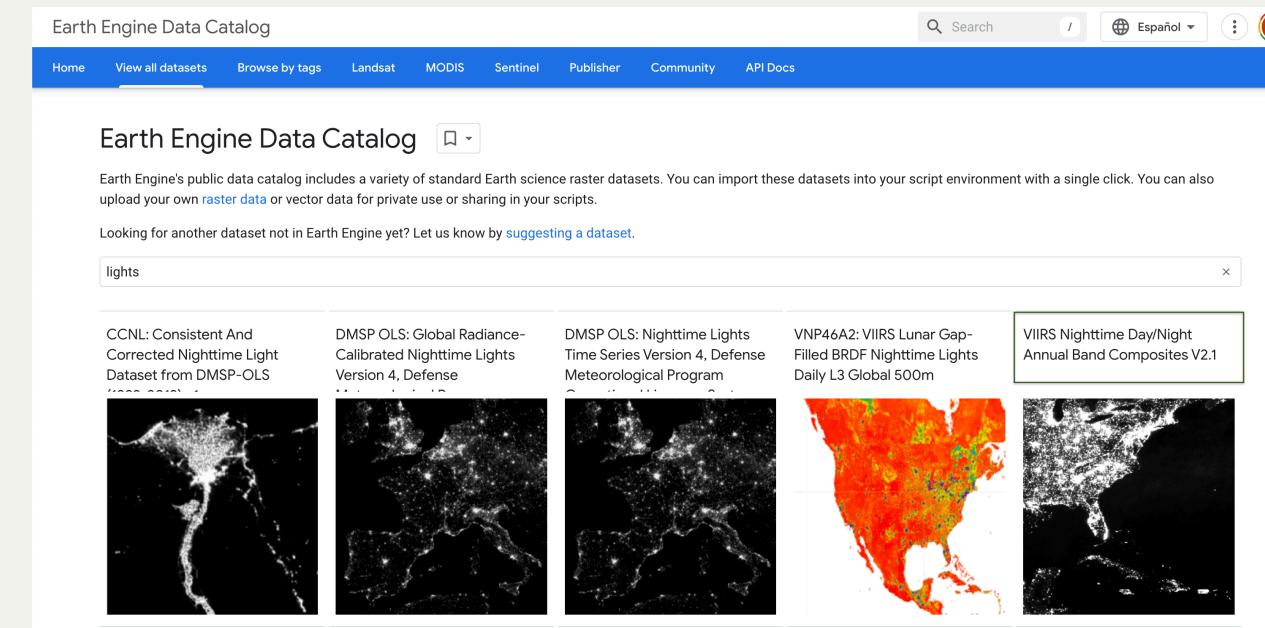
- Keep an eye on this package! It will be very useful when it has more datasets

Google Earth Engine

- First things first: you need an account!
- Go to <https://earthengine.google.com/> and sign up
 - Top-right corner: Get Started
 - Next page: Create account
- I'll give you all a couple minutes to get this set up. Let me know if you have problems

Let's look at a dataset

- On the <https://earthengine.google.com/> page:
 - Click **View all datasets** (near the top)
 - Search for **lights**
 - We want **VIIRS Nighttime Day/Night Annual Band Composites V2.1**



Basic information about the dataset

Earth Engine Data Catalog

Search / Espanol : J

Home View all datasets Browse by tags Landsat MODIS Sentinel Publisher Community API Docs

VIIRS Nighttime Day/Night Annual Band Composites V2.1



Dataset Availability
2012-04-01T00:00:00Z–2021-01-01T00:00:00Z

Dataset Provider
[Earth Observation Group, Payne Institute for Public Policy, Colorado School of Mines](#)

Earth Engine Snippet
`ee.ImageCollection("NOAA/VIIRS/DNB/ANNUAL_V21")`

Tags

annual dnb eog lights nighttime noaa viirs visible

Raster bands - They can have more than one!

Description	Bands	Terms of Use	Citations	DOIs
Resolution				
463.83 meters				
Bands				
Name	Units	Description		
average	nanoWatts/sr/ cm ²	Average DNB radiance values.		
average_ masked	nanoWatts/sr/ cm ²	Average Masked DNB radiance values		
cf_cvг		Cloud-free coverages; the total number of observations that went into each pixel. This band can be used to identify areas with low numbers of observations where the quality is reduced.		
cvг		Total number of observations free of sunlight and moonlight.		
maximum	nanoWatts/sr/ cm ²	Maximum DNB radiance values.		
median	nanoWatts/sr/ cm ²	Median DNB radiance values		
median_masked	nanoWatts/sr/ cm ²	Median masked DNB radiance values.		
minimum	nanoWatts/sr/ cm ²	Minimum DNB radiance values		

Downloading the data is... a pain

- Actually getting the data is a bit of a pain
 - Unless you know Javascript!
- A lot of people use libraries in **R** (or python) to download data, instead.
 - All of them are a bit cumbersome.
 - Especially true in **R**, because we need to use python!
 - We are going to use **rgeedim**
 - Go ahead and install it using **install.packages("rgeedim")**
 - Load it using **library(rgeedim)**
 - Type **Yes** when asked to create a default Python environment

We need to download python... sorry!

- The first time you use `rgeedim`, you will need to install python
 - The easiest way is to search `download miniconda`
- Let's take five minutes to download and install python

The code

▼ Code

```
1 library(rgeedim)
2 gd_install()
3 gd_authenticate(auth_mode = "notebook")
```



- After `gd_authenticate()`, your browser should open.
 - You'll need to sign in to your Google account
 - Continue through the prompts and make sure you select all access

The code

▼ Code

```
1 library(rgeedim)
2 gd_install()
3 gd_authenticate(auth_mode = "notebook")
```



- After `gd_authenticate()`, your browser should open.
 - You'll need to sign in to your Google account
 - Continue through the prompts and make sure you select all access

The code

- You'll arrive at this page.
- Click the **Copy** button

▼ Code

```
1 library(rgeedim)
2 gd_install()
3 gd_authenticate(auth_mode = "notebook")
```

- Then go back to RStudio, and paste (ctrl + v) the code into the console

The screenshot shows a Google sign-in dialog box. At the top left is the Google logo and the text "Sign in with Google". Below that, it says "Make sure you trust Earth Engine Notebook Client - jdmesp@gmail.com". A yellow warning box contains the text: "⚠️ Earth Engine Notebook Client - jdmesp@gmail.com is requesting access to your Google Account. To protect your account, don't continue unless you know and trust this app." Below the warning, there's a section titled "Sign in or provide access to Earth Engine Notebook Client - jdmesp@gmail.com" with the sub-instruction "To sign in or provide access:". Three numbered steps are listed: 1. Copy the authorization code from the Authorization code section. 2. Navigate to Earth Engine Notebook Client - jdmesp@gmail.com. 3. Paste the authorization code on the Earth Engine Notebook Client - jdmesp@gmail.com screen. A green arrow points from the "Authorization code" section down to a large black redacted area where the authorization code would be displayed. This redacted area has a small "copy" icon in the bottom right corner. Below the redacted area, the text "Don't sign in or provide access to Earth Engine Notebook Client - jdmesp@gmail.com" is shown, along with the instruction "If you don't want to continue, close this window."

Sign in with Google

Make sure you trust Earth Engine Notebook Client - jdmesp@gmail.com

⚠️ Earth Engine Notebook Client - jdmesp@gmail.com is requesting access to your Google Account. To protect your account, don't continue unless you know and trust this app.

Sign in or provide access to Earth Engine Notebook Client - jdmesp@gmail.com

To sign in or provide access:

1. Copy the authorization code from the Authorization code section.
2. Navigate to Earth Engine Notebook Client - jdmesp@gmail.com.
3. Paste the authorization code on the Earth Engine Notebook Client - jdmesp@gmail.com screen.

Authorization code

Please copy this code, switch to your application and paste it there:

[REDACTED]

Don't sign in or provide access to Earth Engine Notebook Client - jdmesp@gmail.com

If you don't want to continue, close this window.

The code

▼ Code

```
1 library(rgeedim)
2 #gd_install() # You SHOULD NOT need to do this on each new session
3 gd_authenticate(auth_mode = "notebook") # need to do this
4 gd_initialize() # and you need to do this
```



- After you do `gd_install()` once, you should be good
 - You will need to do `gd_authenticate()` and `gd_initialize()` each time you start a new session

Downloading the data - still not straightforward!

- First, we need to create a “bounding box”
 - This is the area of the globe we want to search
 - We will use the Malawi shapefile for this
 - The “bounding box” is a rectangle that completely contains the shapefile

▼ Code

```
1 # load shapefile
2 malawi <- read_sf("day4data/mw4.shp")
3 # this creates the bounding box
4 bbox <- st_bbox(malawi)
5 bbox
```

xmin	ymin	xmax	ymax
32.942417	-12.740578	34.758878	-9.367346

Basic information about the dataset

- Remember I said we'd need this again?

Earth Engine Data Catalog

Search / [Español](#) [⋮](#) 

Home View all datasets Browse by tags Landsat MODIS Sentinel Publisher Community API Docs

VIIRS Nighttime Day/Night Annual Band Composites V2.1 [🔗](#)



Dataset Availability
2012-04-01T00:00:00Z–2021-01-01T00:00:00Z

Dataset Provider
[Earth Observation Group, Payne Institute for Public Policy, Colorado School of Mines](#)

Earth Engine Snippet

```
ee.ImageCollection("NOAA/VIIRS/DNB/ANNUAL_V21")
```

Tags

annual dnb eog lights nighttime noaa viirs visible

Image collections vs. images

- One key thing to understand about GEE is the difference between image collections and images
- An image collection is what it sounds like: a collection of images
 - The key is that we won't download image collections
 - We'll download individual images
 - So we need to find the images!

Get images from the collection

▼ Code

```
1 x <- gd_collection_from_name("NOAA/VIIRS/DNB/ANNUAL_V21") |>  
2   gd_search(region = bbox)  
3 gd_properties(x)
```

	id	date
1	NOAA/VIIRS/DNB/ANNUAL_V21/20130101	2013-01-01 09:00:00
2	NOAA/VIIRS/DNB/ANNUAL_V21/20140101	2014-01-01 09:00:00
3	NOAA/VIIRS/DNB/ANNUAL_V21/20150101	2015-01-01 09:00:00
4	NOAA/VIIRS/DNB/ANNUAL_V21/20160101	2016-01-01 09:00:00
5	NOAA/VIIRS/DNB/ANNUAL_V21/20170101	2017-01-01 09:00:00
6	NOAA/VIIRS/DNB/ANNUAL_V21/20180101	2018-01-01 09:00:00
7	NOAA/VIIRS/DNB/ANNUAL_V21/20190101	2019-01-01 09:00:00
8	NOAA/VIIRS/DNB/ANNUAL_V21/20200101	2020-01-01 09:00:00
9	NOAA/VIIRS/DNB/ANNUAL_V21/20210101	2021-01-01 09:00:00

- The survey data I have from Malawi is 2019/2020, so let's download the 2019-01-01 data
 - We want to use the id: **NOAA/VIIRS/DNB/ANNUAL_V21/20190101**

We can FINALLY download the raster!

▼ Code

```
1 x <- gd_image_from_id("NOAA/VIIRS/DNB/ANNUAL_V21/20190101") |>
2   gd_download(
3     filename = "temp.tif",
4     region = bbox, # region is our bbox
5     scale = 500, # resolution of raster is only 500, so no reason to go lower
6     crs = 'EPSG:4326', # lat/lon
7     overwrite = TRUE, # overwrite if it exists
8     silent = FALSE
9   )
10 # we downloaded the raster and called it x
11 # so let's load it using terra!
12 x <- rast(x)
13 # here it is!
14 x
```

```
class      : SpatRaster
dimensions : 752, 405, 9  (nrow, ncol, nlyr)
resolution : 0.004491576, 0.004491576 (x, y)
extent     : 32.94122, 34.76031, -12.7426, -9.364937 (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (EPSG:4326)
```

Quick note: we downloaded many bands!

▼ Code

```
1 names(x)
```



```
1 [1] "average"           "average_masked" "cf_cvg"          "cvg"           "maximum"        "median"         "median_masked" "minim
```



▼ Code

```
1 # but we really only want average nightlights
2 # so here's how you can download just the average
3 x <- gd_image_from_id("NOAA/VIIRS/DNB/ANNUAL_V21/20190101") |>
4   gd_download(
5     filename = "temp.tif",
6     region = bbox, # region is our bbox
7     scale = 500, # resolution of raster is only 500, so no reason to go lower
8     crs = 'EPSG:4326', # lat/lon
9     overwrite = TRUE, # overwrite if it exists
10    silent = FALSE,
11    bands = list("average"),
12  )
13 # we downloaded the raster and called it x
14 # so let's load it using terra!
15 x <- rast(x)
```



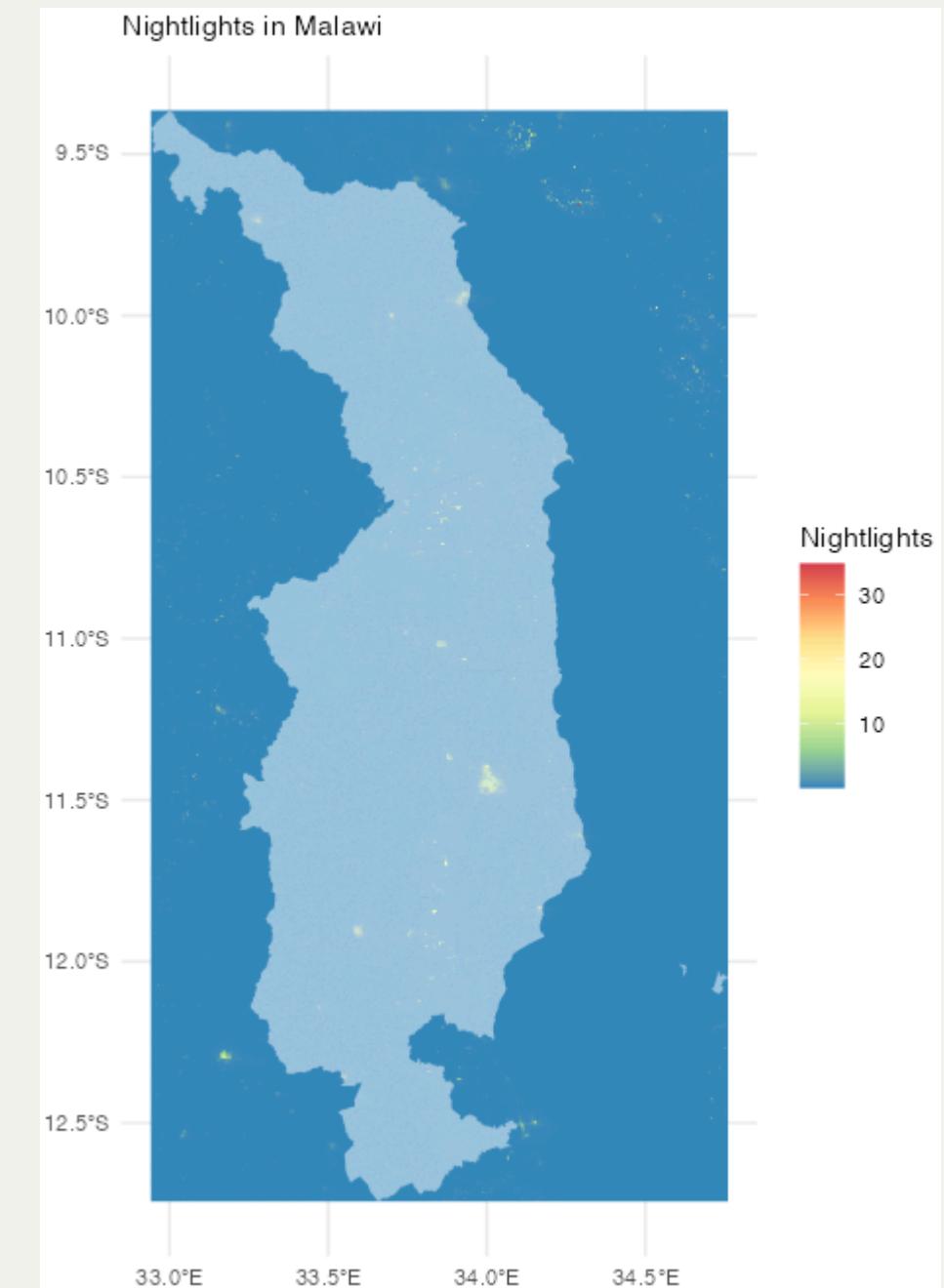
```
16 # here it is!  
17 x
```

```
1 class      : SpatRaster  
2 dimensions : 752, 405, 1 (nrow, ncol, nlyr)  
3 resolution : 0.004491576, 0.004491576 (x, y)  
4 extent     : 32.94122, 34.76031, -12.7426, -9.364937 (xmin, xmax, ymin, ymax)  
5 coord. ref. : lon/lat WGS 84 (EPSG:4326)  
6 source     : temp.tif  
7 name       : average
```

What does it look like?

▼ Code

```
1 adm4 <- read_sf("day4data/mw4.shp")
2 ggplot() +
3   geom_spatraster(data = x) +
4   scale_fill_distiller("Nightlights",
5     palette = "Spectral") +
6   geom_sf(data = adm4,
7     color = "white",
8     lwd = 0.01,
9     alpha = 0.5,
10    fill = "transparent") +
11   theme_minimal() +
12   labs(subtitle = "Nightlights in Malawi")
```



- Note what the “bounding box” does!
 - `st_bbox` as a reminder

Now it's your turn!

- We want to download NDVI data for Malawi
 - We want the [Terra Vegetation Indices 16-Day Global 500m](#) data (you can just search this)
 - Download the first observation from 2019
 - Extract it to the mw4 shapefile!

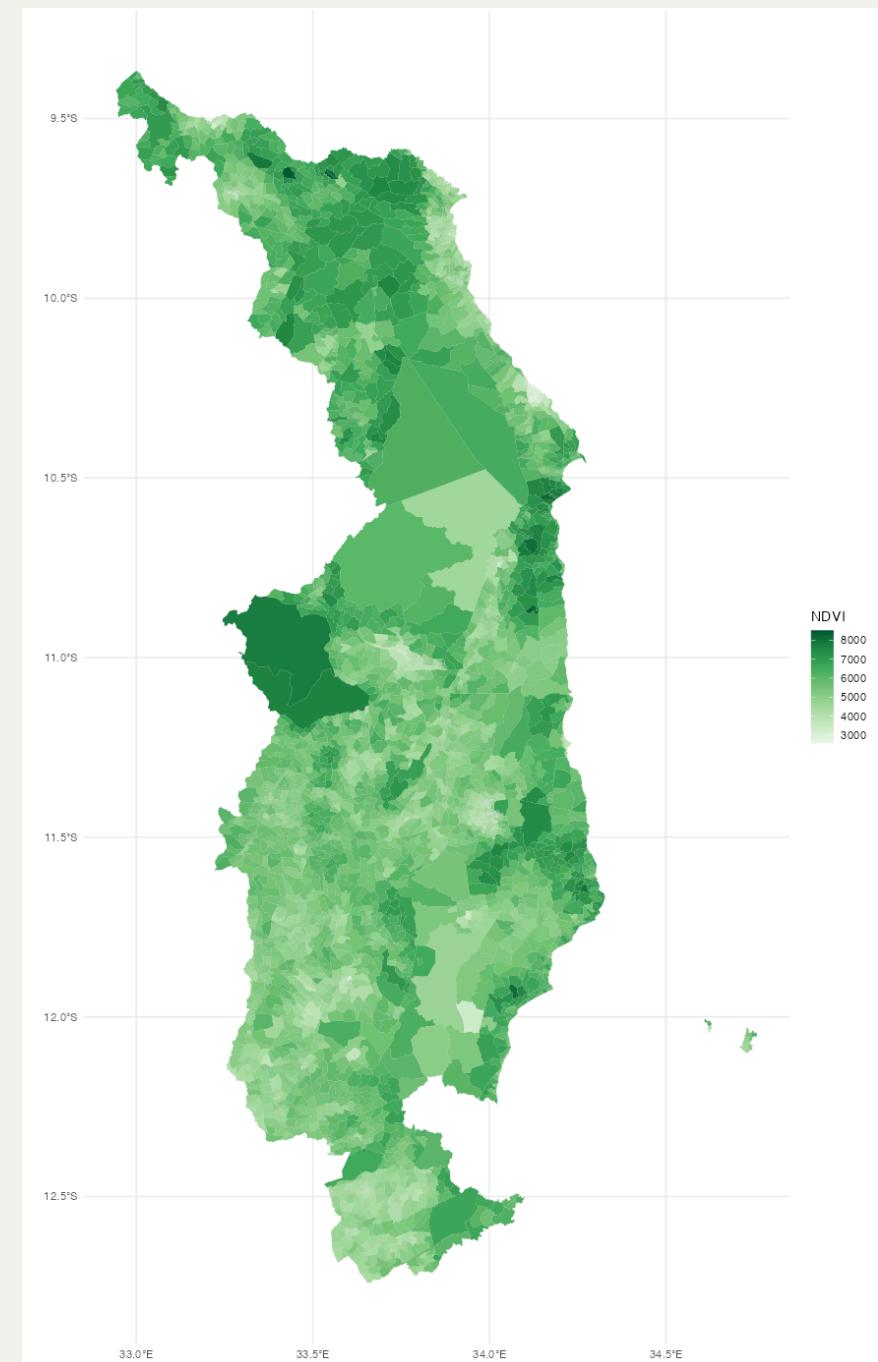
▼ Code

```
1 # load shapefile
2 malawi <- read_sf("day4data/mw4.shp")
3 # this creates the bounding box
4 bbox <- st_bbox(malawi)
5
6 x <- gd_collection_from_name("MODIS/061/MOD13A1") |>
7   gd_search(region = bbox)
8 gd_properties(x)
9 # the ID for the first image of 2019 is MODIS/061/MOD13A1/2019_01_01
```

Let's download that id

▼ Code

```
1 x <- gd_image_from_id("MODIS/061/MOD13A1/2019_01_01") |>
2   gd_download(
3     filename = "temp.tif",
4     region = bbox, # region is our bbox
5     scale = 500, # resolution
6     crs = 'EPSG:4326', # lat/lon
7     overwrite = TRUE, # overwrite if it exists
8     bands = list("NDVI") # only download NDVI
9   )
10 x <- rast(x) # load raster
11 ndviextract <- exact_extract(x, malawi, fun = "mean", append_col = TRUE)
12 malawi <- malawi |>
13   left_join(ndviextract |> rename(ndvi = mean), by = "EA_CODE")
14 ggplot() +
15   geom_sf(data = malawi, aes(fill = ndvi), lwd = 0.01,) +
16   scale_fill_distiller("NDVI",
17     palette = "Greens", direction = 1) +
18   theme_minimal()
```



This returns a LOT of results - search by date!

▼ Code

```
1 # load shapefile
2 malawi <- read_sf("day4data/mw4.shp")
3 # this creates the bounding box
4 bbox <- st_bbox(malawi)
5
6 # so let's filter by date!
7 x <- gd_collection_from_name("MODIS/061/MOD13A1") |>
8   gd_search(region = bbox,
9     start_date = "2019-01-01",
10    end_date = "2019-12-31")
11 gd_properties(x)
```

		id		date
1				
2	1	MODIS/061/MOD13A1/2019_01_01	2019-01-01	09:00:00
3	2	MODIS/061/MOD13A1/2019_01_17	2019-01-17	09:00:00
4	3	MODIS/061/MOD13A1/2019_02_02	2019-02-02	09:00:00
5	4	MODIS/061/MOD13A1/2019_02_18	2019-02-18	09:00:00
6	5	MODIS/061/MOD13A1/2019_03_06	2019-03-06	09:00:00
7	6	MODIS/061/MOD13A1/2019_03_22	2019-03-22	09:00:00
8	7	MODIS/061/MOD13A1/2019_04_07	2019-04-07	09:00:00

9 8 MODIS/061/MOD13A1/2019_04_23 2019-04-23 09:00:00
10 9 MODIS/061/MOD13A1/2019_05_09 2019-05-09 09:00:00
11 10 MODIS/061/MOD13A1/2019_05_25 2019-05-25 09:00:00
12 11 MODIS/061/MOD13A1/2019_06_10 2019-06-10 09:00:00
13 12 MODIS/061/MOD13A1/2019_06_26 2019-06-26 09:00:00
14 13 MODIS/061/MOD13A1/2019_07_12 2019-07-12 09:00:00
15 14 MODIS/061/MOD13A1/2019_07_28 2019-07-28 09:00:00
16 15 MODIS/061/MOD13A1/2019_08_13 2019-08-13 09:00:00
17 16 MODIS/061/MOD13A1/2019_08_29 2019-08-29 09:00:00
18 17 MODIS/061/MOD13A1/2019_09_14 2019-09-14 09:00:00
19 18 MODIS/061/MOD13A1/2019_09_30 2019-09-30 09:00:00

Let's look at the dates

▼ Code

```
1 gd_properties(x)$date
```



```
1 [1] "2019-01-01 09:00:00 KST" "2019-01-17 09:00:00 KST" "2019-02-02 09:00:00 KST" "2019-02-18 09:00:00 KST" "2019-03-06 09:00:00 KST"
```



- What should we download?
 - NDVI is a vegetation index, which means it varies quite a bit throughout the year
 - We could take average NDVI throughout the year
 - Or we could take NDVI at a specific time of year
 - Or we could take the max... or the min... or all of the above!
- Let's download one raster PER MONTH
 - How?

Let's look at the dates

- There are 23 dates
- This code is a bit complicated, so let me explain

▼ Code

```
1 # get the dates
2 dates <- gd_properties(x)$date
3 # here are the months
4 months <- month(dates)
5 ids <- c() # this creates an empty vector
6 for (m in 1:12){ # this "for loop" loops through each month (1 through 12)
7   ids <- c(ids, which(months == m)[1]) # it then takes the LOCATION of the FIRST VALUE equal to m
8 }
9 ids <- gd_properties(x)$id[ids] # this gets the image ids at those locations
10 ids #
```

```
1 [1] "MODIS/061/MOD13A1/2019_01_01" "MODIS/061/MOD13A1/2019_02_02" "MODIS/061/MOD13A1/2019_03_06" "MODIS/061/MOD13A1/2019_04_07"
```