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!

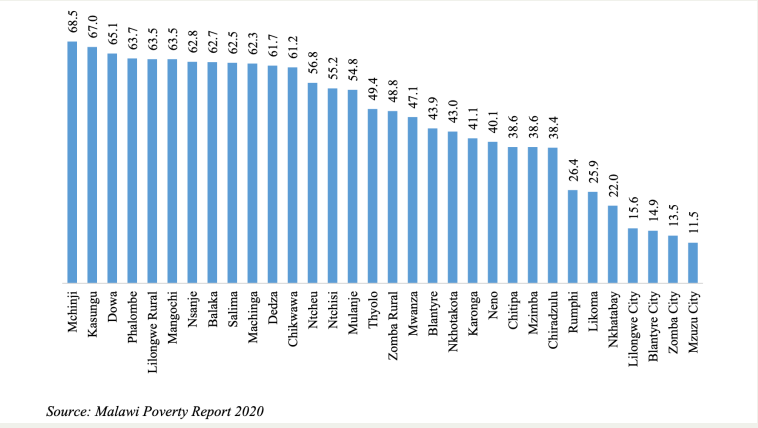
Think of what you need for SAE

- 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

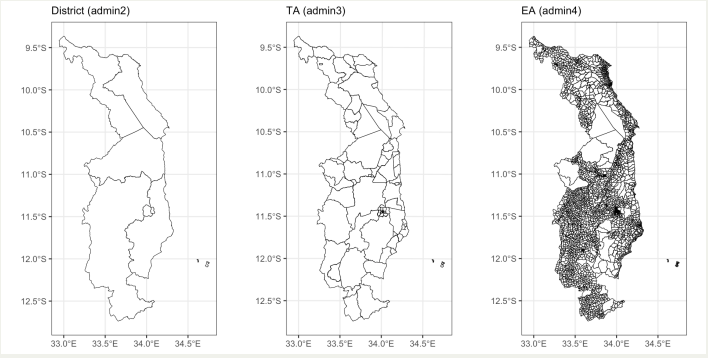
- Let's take a look at Malawi
- Why Malawi?
  - I have survey data you can use 😃
  - Only going to use part of Malawi for this example (size of data)
- Consider the 2019/2020 Integrated Household Survey (IHS5)
  - Was used for the Malawi Poverty Report 2020
  - Can say things about poverty at the district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

A quick example



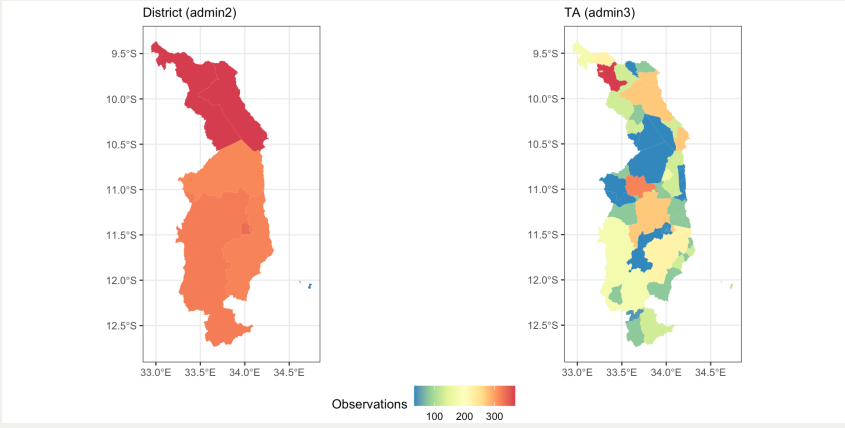
Source: Malawi Poverty Report 2020

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

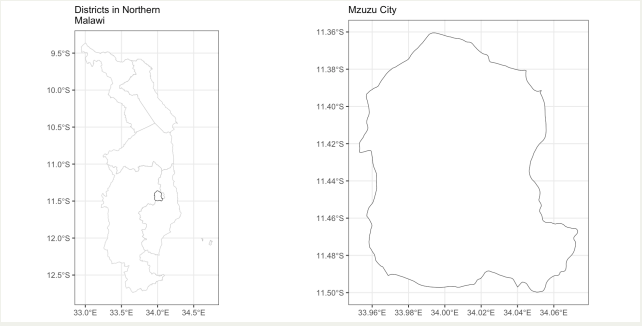
Getting started with geospatial data

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

Shapefiles

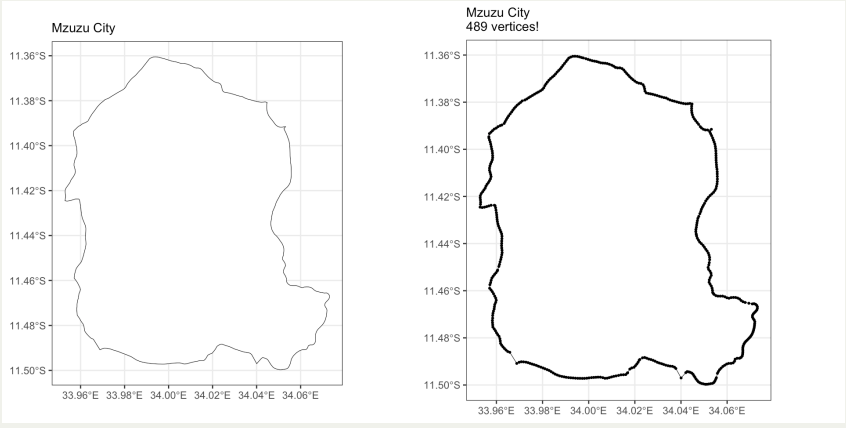
- The maps I just showed you are `shapefiles`
- Shapefiles are a common format for geospatial data
  - They are a form of `vector` data
- Shapefiles are made up of *at least* four files:
  - `.shp` - the shape itself
  - `.shx` - the index
  - `.dbf` - the attributes
  - `.prj` - the projection
  - What 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

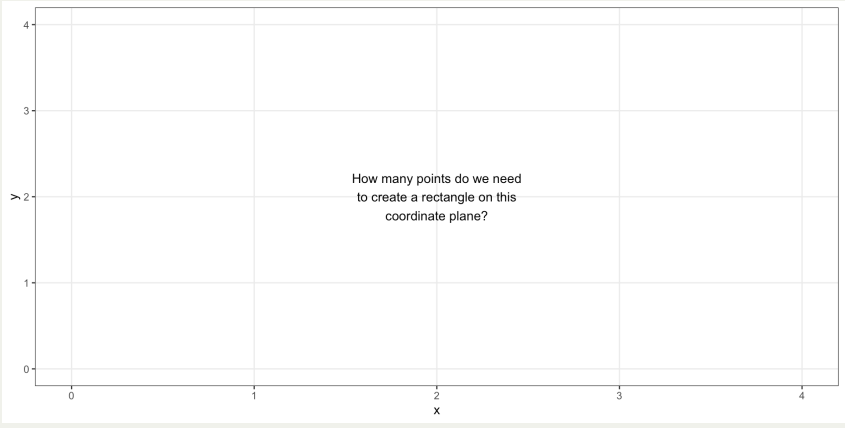


- Left: collection of `features`
- Right: one `feature`

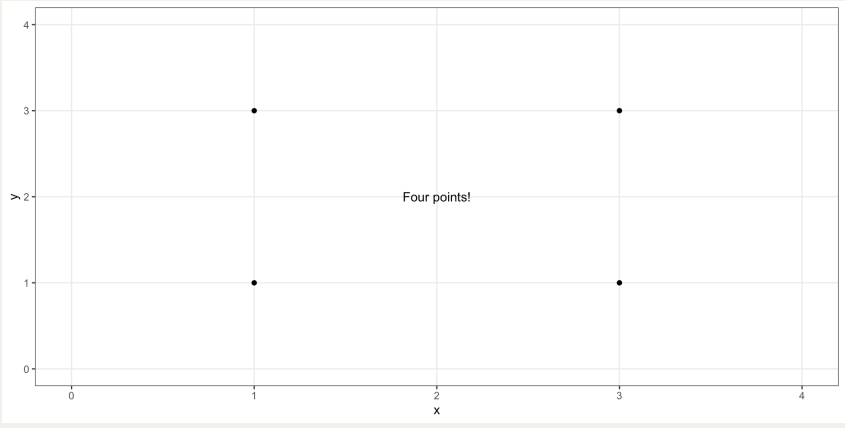
Features are made of vertices, which connect



Imagine a rectangle, on a coordinate plane

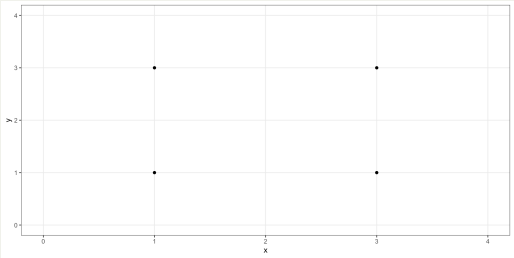


Imagine a rectangle, on a coordinate plane



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!

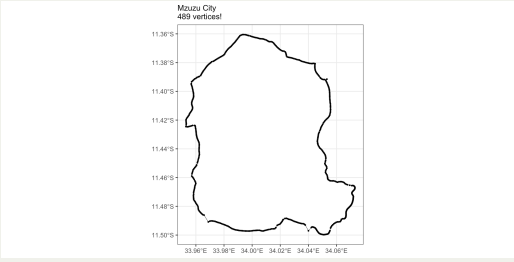


FIVE POINTS (VERTICES) IN OUR FEATURE

	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

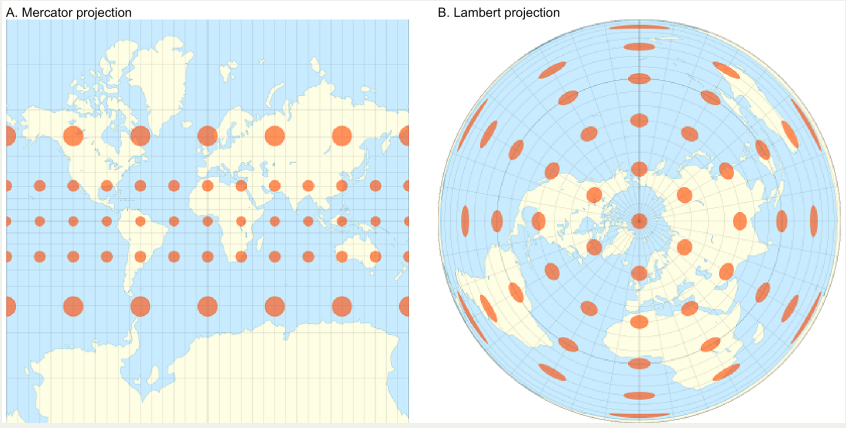

2 million
sq km
30 million
sq km
Source: wikipedia

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



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
```

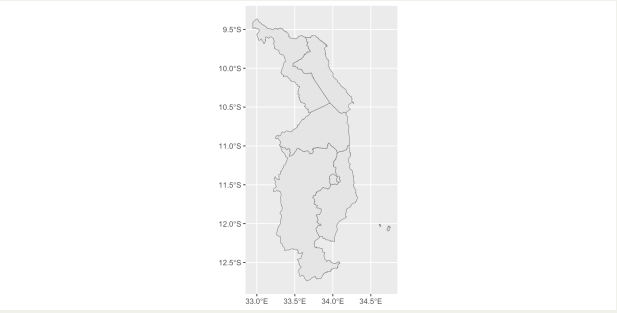
```
Simple feature collection with 7 features and 1 field
Geometry type: MULTIPOLYGON
Dimension:     XY
Bounding box:  xmin: 32.9424 ymin: -12.73669 xmax: 34.75834 ymax: -9.363662
Geodetic CRS:  WGS 84
# A tibble: 7 × 2
  DIST_CODE                                                              geometry
  <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...
6 103       (((34.21913 -11.05184, 34.21931 -11.05268, 34.21969 -11.05357, 34.22008 -11.05455, 3...
7 104       (((34.09466 -10.57433, 34.09563 -10.57449, 34.09763 -10.57337, 34.0983 -10.57367, 34...
```

Plotting is also very easy

▼ Code

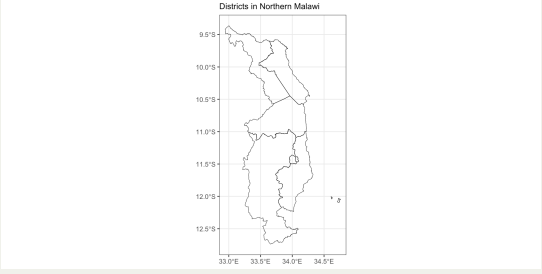
```
1 ggplot() +
2   geom_sf(data = northmw)
```



My go-to theme

▼ Code

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



Give it a try with TAs (mw3.shp)

▼ Code

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

▼ Code

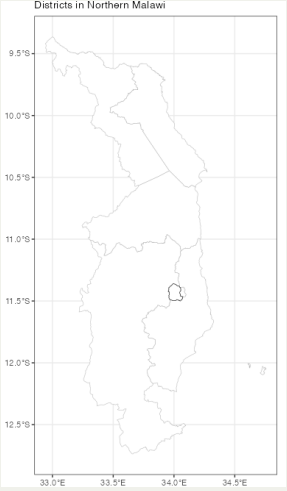
```
ggplot() +
  geom_sf(data = northmw, fill = NA, color = "black") +
  theme_bw() +
  labs(subtitle = "TAs in Northern Malawi")
```

One more example - map from earlier

▼ Code

```
admin2 <- read_sf("day4data/mw2.shp")

ggplot() +
  geom_sf(data = admin2,
    fill = "white", color = "gray") +
  geom_sf(data = admin2 |> filter(DIST_CODE=="10
    fill = "white", color = "black") +
  theme_bw() +
  labs(subtitle = "Districts in Northern Malawi"
```

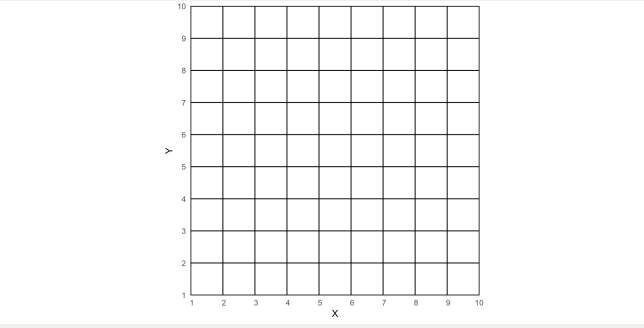


Rasters

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

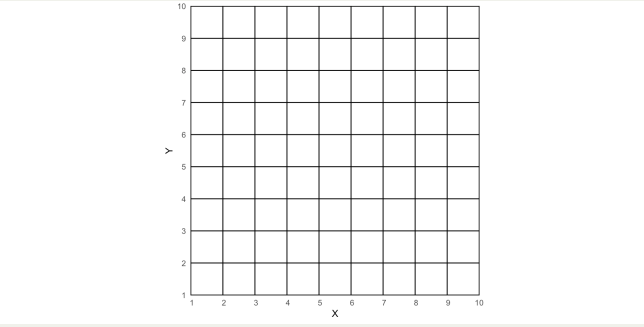
- Rasters are a different type of geospatial data
  - They are made up of a grid of cells
  - Each cell has a value

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?

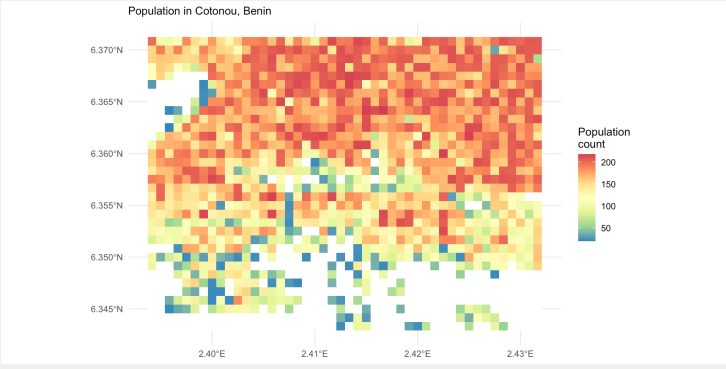


- Need to know location of one grid cell…
  - And the size of each grid!

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
```

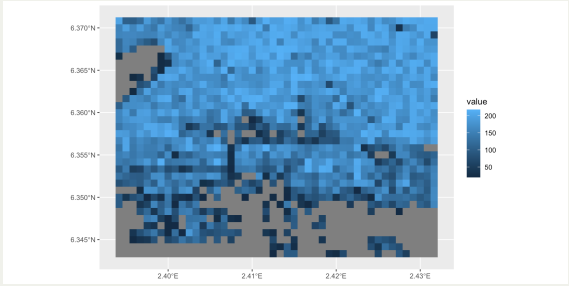
```
class       : SpatRaster
dimensions  : 34, 46, 1  (nrow, ncol, nlyr)
resolution  : 0.0008333333, 0.0008333333  (x, y)
extent      : 2.39375, 2.432083, 6.342917, 6.37125  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (EPSG:4326)
source      : beninpop.tif
```

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)
```

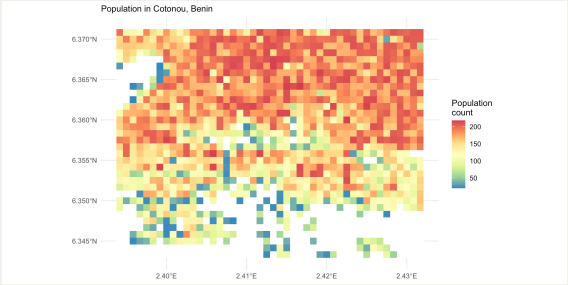


Making it nicer

▼ Code

```
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_minimal() +
  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`

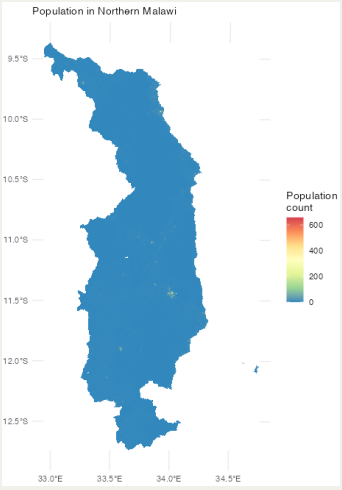
Give it a try

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

▼ Code

```
tif <- rast("day4data/mwpop.tif")

ggplot() +
  geom_spatraster(data = tif) +
  scale_fill_distiller("Population\ncount",
    palette = "Spectral", na.value = "white") +
  theme_minimal() +
  labs(subtitle = "Population in Northern Malawi
```



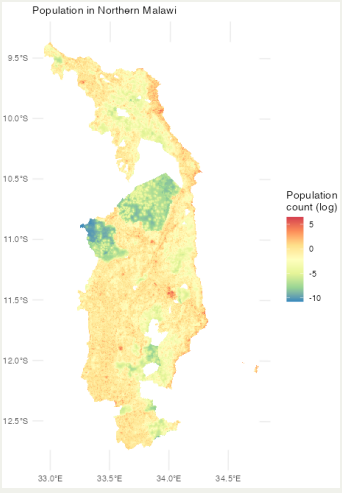
Give it a try

- 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)

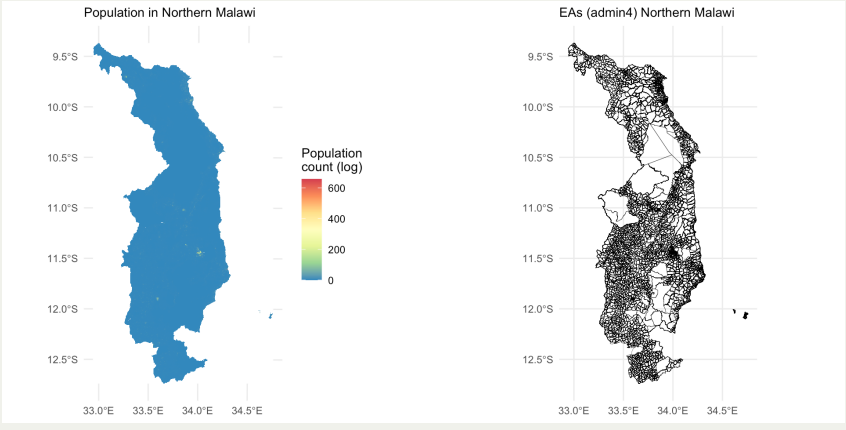
▼ Code

```
tif <- rast("day4data/mwpop.tif")

ggplot() +
  geom_spatraster(data = log(tif)) +
  scale_fill_distiller("Population\ncount (log)"
    palette = "Spectral", na.value = "white") +
  theme_minimal() +
  labs(subtitle = "Population in Northern Malawi
```



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



Let's do it with exactextractr

▼ Code

```r
library(exactextractr)

tif <- rast("day4data/mwpop.tif")
adm4 <- read_sf("day4data/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
```

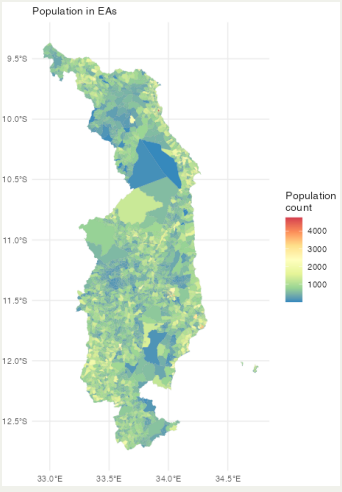
```
4 10507001  749.6960
5 10507009  597.5432
6 10507033  474.3934
```

Now we can join the extracted data to the shapefile

▼ Code

```r
# 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_minimal() +
  labs(subtitle = "Population in EAs")
```

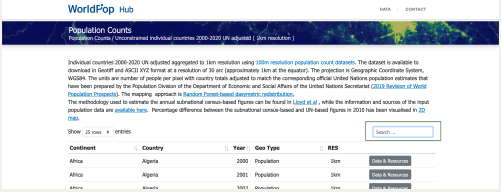


Now it's your turn

- 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?)



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!

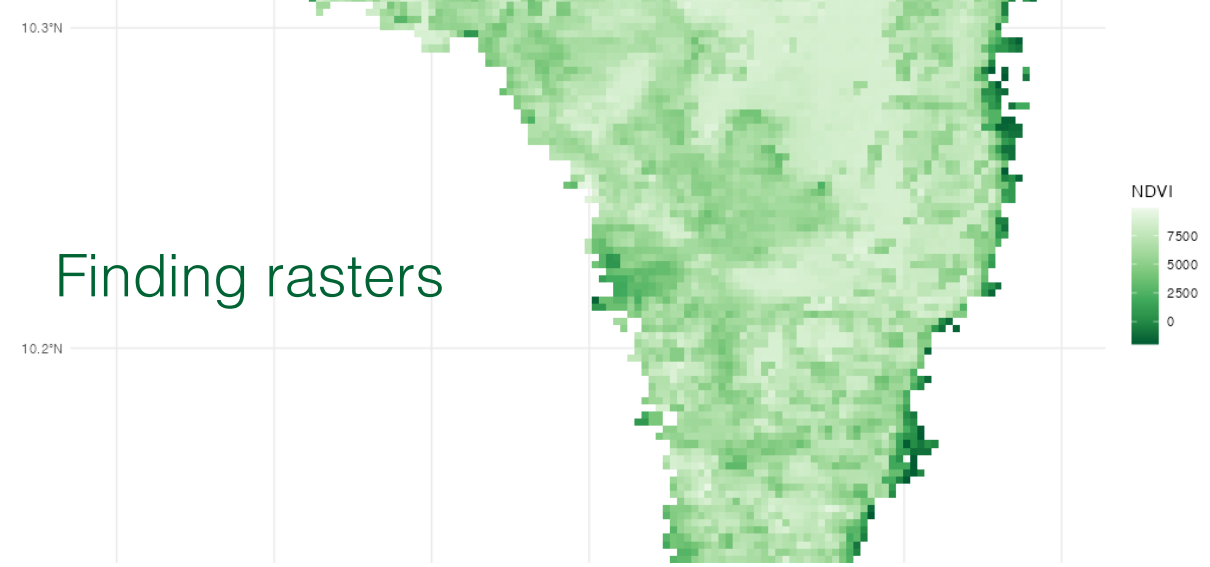
- Now you need to find a shapefile for the same country
- This will be a bit less straightforward
  - Search for "shapefile COUNTRY humdata"
  - You should find a link to the Humanitarian Data Exchange
  - Click on it and see if it has shapefiles for your country of choice
  - If so, download a shapefile (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

Where can you find rasters?

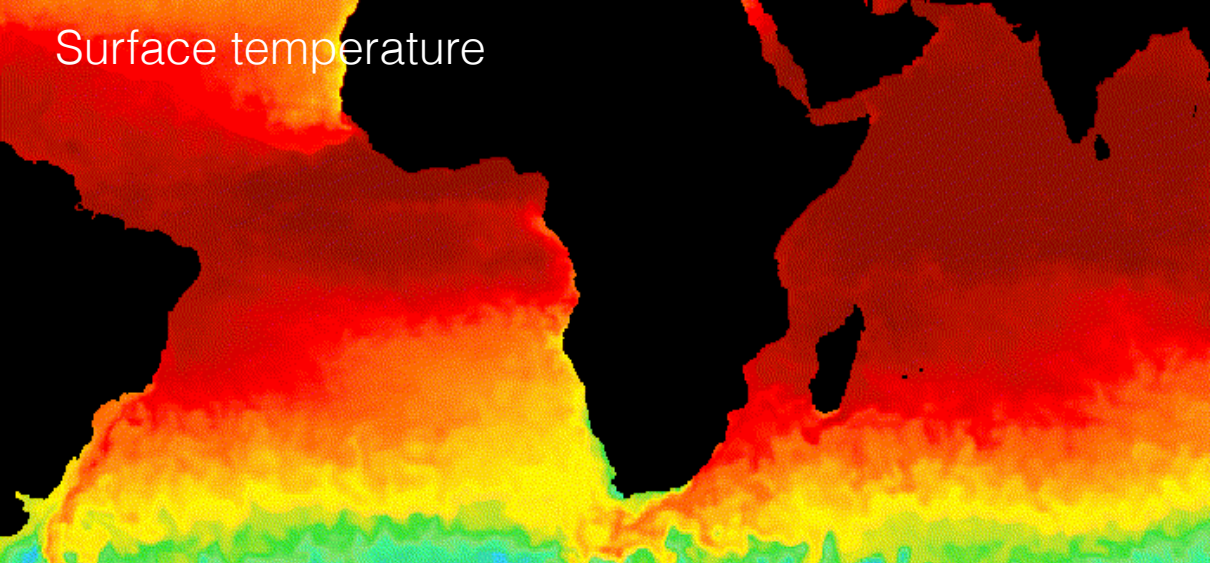
- Depending on the variable, rasters are sometimes quite easy to find!
  - We already saw one example: WorldPop (population counts)

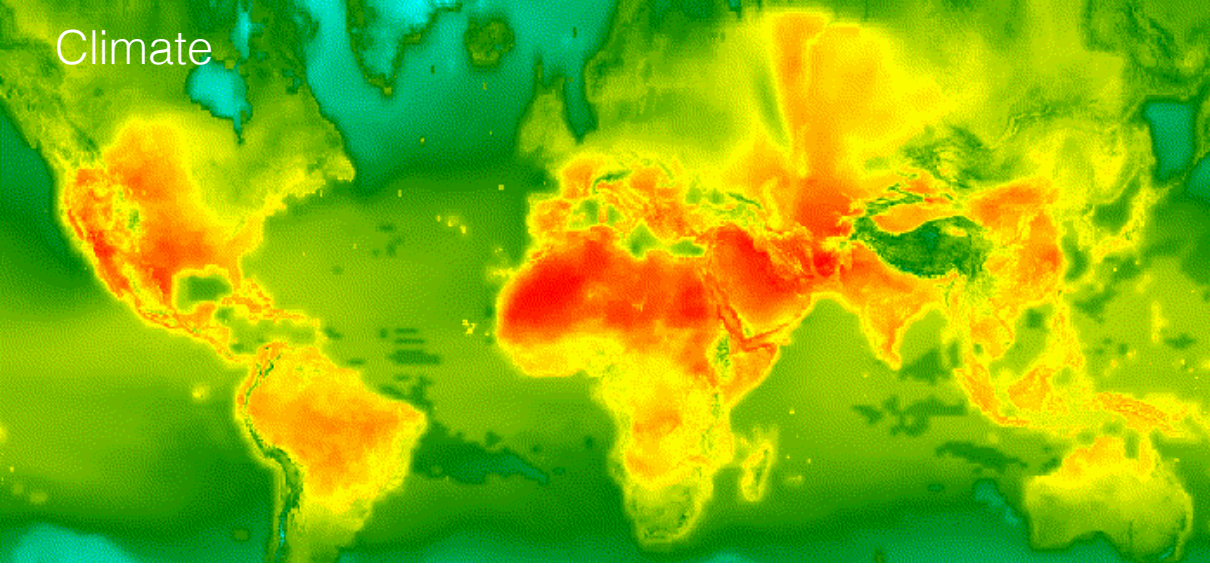
- There are two large online repositories:
  - Google Earth Engine
  - Microsoft Planetary Computer
    - This one is newer and has less data (for now)

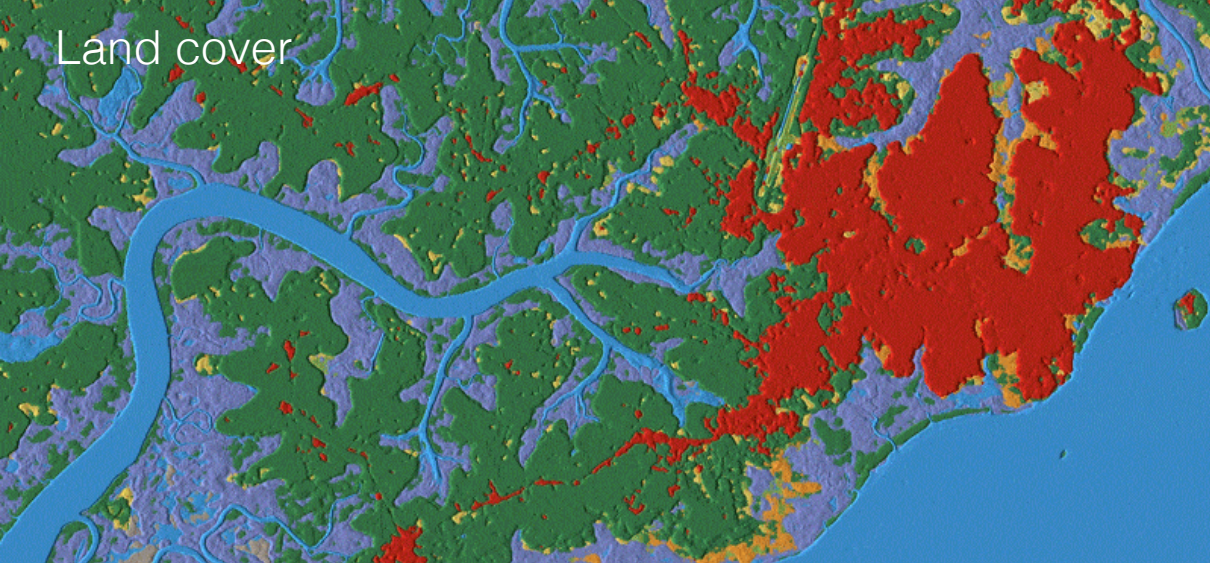
Meet Earth Engine

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

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

The actual code is much easier to use than GEE!

▼ Code

```
library(GeoLink)
library(sf)

adm4 <- read_sf("mw4.shp")
# CHIRPS is rainfall data
adm4_chirps <- geolink_chirps(time_unit = "month",
  start_date = "2019-01-01",
  end_date = "2019-03-01",
  shp_dt = adm4,
  extract_fun = "mean")
summary(adm4_chirps)
```

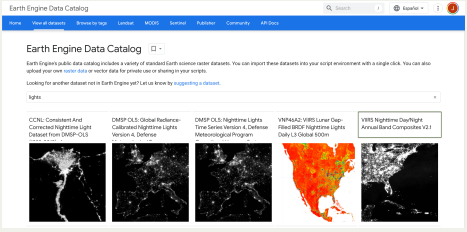
- Keep an eye on this package! It will be very useful when it has more datasets

Google Earth Engine

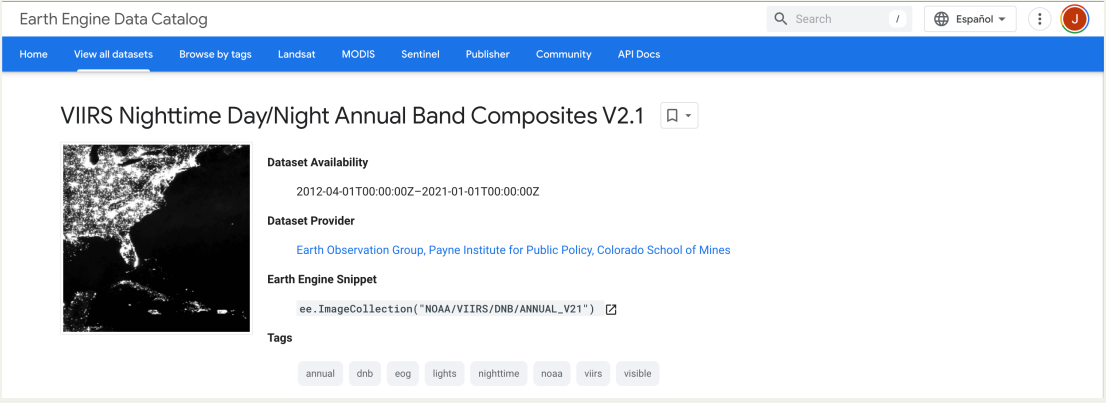
- First things first: you need an account!
- Go to https://earthengine.google.com/ and sign up
  - Top-right corner: `Get Started`
  - Next page: `Create account`
- I'll give you all a couple minutes to get this set up. Let me know if you have problems

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



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_cvg`		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.
`cvg`		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

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`
  - One of the first links should be https://docs.anaconda.com/miniconda/
  - Go down to “Latest Miniconda installer links” and select the correct link for your platform (e.g. Windows)
  - Once it finishes downloading, please go do your downloads folder and double click to install

- Let’s take five minutes to download and install python

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

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

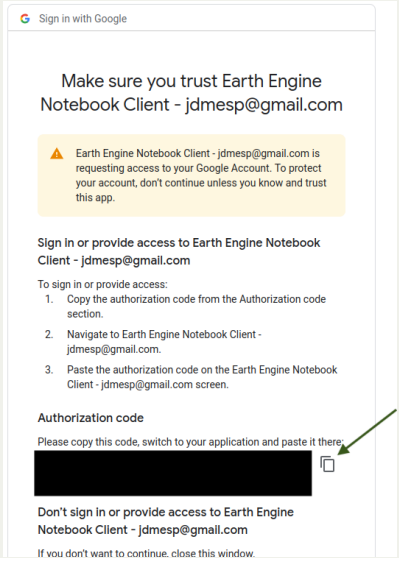
The code

- 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 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

```
# load shapefile
malawi <- read_sf("day4data/mw4.shp")
# this creates the bounding box
bbox <- st_bbox(malawi)
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?

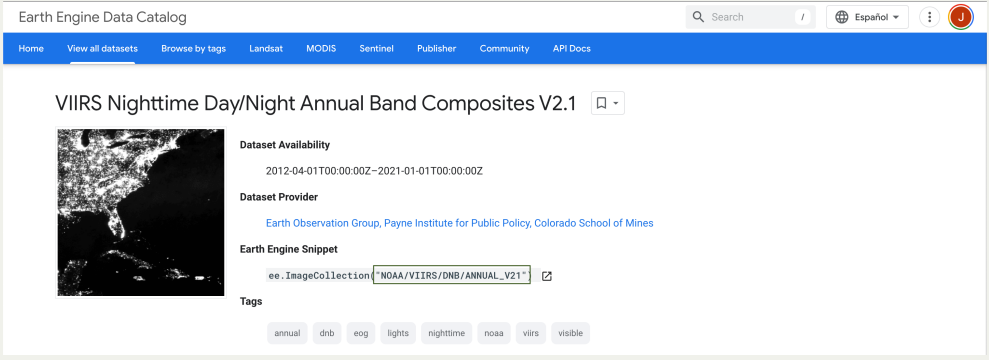


Image collections vs. images

- 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

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

```r
x <- gd_image_from_id("NOAA/VIIRS/DNB/ANNUAL_V21/20190101") |>
  gd_download(
    filename = "temp.tif",
    region = bbox, # region is our bbox
    scale = 500, # resolution of raster is only 500, so no reason to go lower
    crs = 'EPSG:4326', # lat/lon
    overwrite = TRUE, # overwrite if it exists
    silent = FALSE
  )
# we downloaded the raster and called it x
# so let's load it using terra!
x <- rast(x)
# here it is!
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)
source      : temp.tif
names       : average, avera~asked, cf cvg, cvg, maximum, median, ...
```

Quick note: we downloaded many bands!

▼ Code

```
names(x)
```

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

▼ Code

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

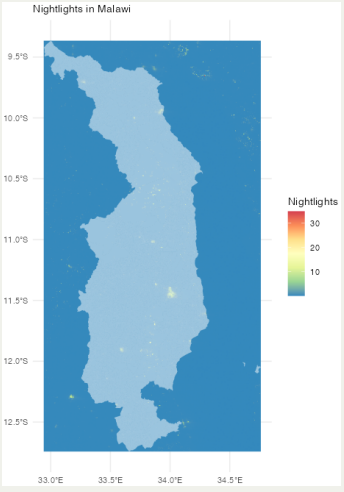
```
# here it is!
x
```

```
class       : SpatRaster
dimensions  : 752, 405, 1  (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)
source      : temp.tif
name        : average
```

What does it look like?

▼ Code

```
adm4 <- read_sf("day4data/mw4.shp")
ggplot() +
  geom_spatraster(data = x) +
  scale_fill_distiller("Nightlights",
    palette = "Spectral") +
  geom_sf(data = adm4,
    color = "white",
    lwd = 0.01,
    alpha = 0.5,
    fill = "transparent") +
  theme_minimal() +
  labs(subtitle = "Nightlights in Malawi")
```



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

We want to extract NTL to the shapefile

▼ Code

```r
adm4 <- read_sf("day4data/mw4.shp")
# exact_extract
library(exactextractr)
ntlextract <- exact_extract(x, adm4, fun = "mean", append_cols = "EA_CODE")
head(ntlextract)
# save it!
write_csv(ntlextract |> rename(ntl = mean), "day4data/mwntlEAs.csv")
```

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

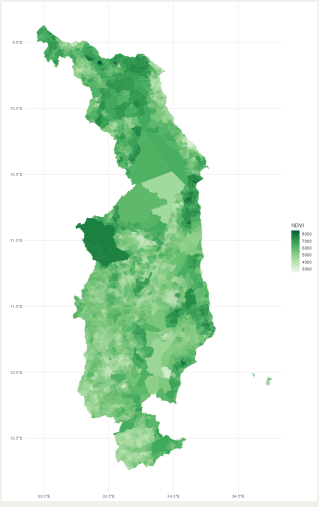
```
# load shapefile
malawi <- read_sf("day4data/mw4.shp")
# this creates the bounding box
bbox <- st_bbox(malawi)

x <- gd_collection_from_name("MODIS/061/MOD13A1") |>
  gd_search(region = bbox)
gd_properties(x)
# the ID for the first image of 2019 is MODIS/061/MOD13A1/2019_01_01
```

Let’s download that id

▼ Code

```r
x <- gd_image_from_id("MODIS/061/MOD13A1/2019_01_01") |>
  gd_download(
    filename = "temp.tif",
    region = bbox, # region is our bbox
    scale = 500, # resolution
    crs = 'EPSG:4326', # lat/lon
    overwrite = TRUE, # overwrite if it exists
    bands = list("NDVI") # only download NDVI
  )
x <- rast(x) # load raster
ndviextract <- exact_extract(x, malawi, fun = "mean", append_co
malawi <- malawi |>
  left_join(ndviextract |> rename(ndvi = mean), by = "EA_CODE")
ggplot() +
  geom_sf(data = malawi, aes(fill = ndvi), lwd = 0.01,) +
  scale_fill_distiller("NDVI",
    palette = "Greens", direction = 1) +
  theme_minimal()
```



This returns a LOT of results - search by date!

▼ Code

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

# so let's filter by date!
x <- gd_collection_from_name("MODIS/061/MOD13A1") |>
  gd_search(region = bbox,
    start_date = "2019-01-01",
    end_date = "2019-12-31")
gd_properties(x)
```

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

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

```
[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
```

- 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

```
# get the dates
dates <- gd_properties(x)$date
# here are the months
months <- month(dates)
ids <- c() # this creates an empty vector
for (m in 1:12){ # this "for loop" loops through each month (1 through 12)
  ids <- c(ids, which(months == m)[1]) # it then takes the LOCATION of the FIRST VALUE equal to m
}
ids <- gd_properties(x)$id[ids] # this gets the image ids at those locations
ids # Now we have all the ids we want to download!
```

```
[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
```

Here's how I would do this

▼ Code

```
adminareas <- malawi |>
  as_tibble() |>
  select(EA_CODE)

for (i in 1:length(ids)){
  x <- gd_image_from_id(ids[i]) |>
  gd_download(
    filename = "temp.tif",
    region = bbox, # region is our bbox
    scale = 500, # resolution
    crs = 'EPSG:4326', # lat/lon
    overwrite = TRUE, # overwrite if it exists
    bands = list("NDVI") # only download NDVI
  )
  x <- rast(x) # load raster
  ndviextract <- exact_extract(x, malawi, fun = "mean", append_cols = "EA_CODE")
  colnames(ndviextract) <- c("EA_CODE", paste0("NDVI_", i))
  adminareas <- adminareas |>
    left_join(ndviextract, by = "EA_CODE")
```

But that's difficult, so I've uploaded the data!

- That's a hard thing to wrap your head around if you're new to R
  - If you want to download them, you can just do them one at a
  - I've uploaded the data:
    - `day4data/ndviallmonths.csv`
    - Go ahead and load it and take a look at it

▼ Code

```
ndviall <- read_csv("day4data/ndviallmonths.csv")
ndviall
```

```
# A tibble: 3,212 × 13
   EA_CODE NDVI_1 NDVI_2 NDVI_3 NDVI_4 NDVI_5 NDVI_6 NDVI_7 NDVI_8 NDVI_9 NDVI_10 NDVI_11 NDVI_12
     <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>   <dbl>   <dbl>   <dbl>
 1 10507801  3950.  6635.  6629.  6212.  5028.  3953.  3443.  2956.  2611.   2613.   2546.   3013.
 2 10507072  4812.  7246.  7090.  6210.  5709.  4523.  3966.  3279.  2714.   2758.   2786.   2409.
 3 10507010  4328.  6140.  6316.  6134.  4481.  3909.  3520.  3067.  2735.   2725.   2579.   2770.
```

```
      7   4 10507001  5716.  7187.  7032.  6976.  6409.  5312.  4692.  3989.  3673.   3691.   3606.   3200.
      8   5 10507009  4952.  6510.  6674.  6728.  5407.  4653.  4141.  3532.  3107.   3319.   3112.   3176.
      9   6 10507033  4291.  6216.  6299.  6382.  4868.  4280.  3812.  3443.  2856.   3038.   2902.   3179.
     10   7 10507032  5049.  6668.  6558.  6747.  5073.  4534.  3871.  3384.  2927.   3209.   2931.   2934.
     11   8 10507002  4545.  6682.  6510   6589.  5348.  4460.  3932.  3313.  2946.   2979.   2805.   2702.
     12   9 10507003  4678.  6269.  6066.  5943.  5104.  4139.  3696.  3211.  2940.   2957.   2890.   2831.
     13  10 10507008  4476.  6459.  6445.  6530.  4930.  4276.  3827.  3200.  2892.   2778.   2699.   3462.
     14  # ℹ 3,202 more rows
```

Let's create some new variables

- Let's create three new NDVI variables:
  - Annual minimum
  - Annual maximum
  - Annual average
- New R function: apply!

▼ Code

```r
# create new variable called min. the "1" means across ROWS.
ndviall$ndvimin <- apply(ndviall |> select(starts_with("NDVI")), 1, min, na.rm = TRUE)
# max
ndviall$ndvimax <- apply(ndviall |> select(starts_with("NDVI")), 1, max, na.rm = TRUE)
# mean
ndviall$ndvimean <- apply(ndviall |> select(starts_with("NDVI")), 1, mean, na.rm = TRUE)
# just keep those
ndviall <- ndviall |>
  select(EA_CODE, ndvimin, ndvimax, ndvimean)
```

```
10 # save it!
11 write_csv(ndviall, "day4data/ndviclean.csv")
```

One last step

- We need to get the codes for the survey data!
- I've uploaded the survey data for Malawi
  - `day4data/ihs5_consumption_aggregate.dta`
  - Go ahead and load it (remember to use `haven`)

▼ Code

```
1 library(haven)
2 ihs5 <- read_dta("day4data/ihs5_consumption_aggregate.dta")
3 head(ihs5)
```

```
# A tibble: 6 × 7
  case_id      ea_id       TA adulteq hh_wgt rexpaggpc poor
  <chr>        <chr>    <dbl>   <dbl>  <dbl>     <dbl> <dbl+lbl>
1 101011000014 10101100 10101    3.47   93.7    99918. 1 [Poor]
2 101011000023 10101100 10101    3.82   93.7   169323. 0 [Non-poor]
3 101011000040 10101100 10101    2.67   93.7    93644. 1 [Poor]
4 101011000071 10101100 10101    4.79   93.7   452758. 0 [Non-poor]
5 101011000095 10101100 10101    4.31   93.7   183333. 0 [Non-poor]
6 101011000115 10101100 10101    2      93.7   420656. 0 [Non-poor]
```

One last step

▼ Code

```
1 head(ihs5)
```

```
# A tibble: 6 × 7
  case_id      ea_id       TA adulteq hh_wgt rexpaggpc poor
  <chr>        <chr>    <dbl>   <dbl>  <dbl>     <dbl> <dbl+lbl>
1 101011000014 10101100 10101    3.47   93.7    99918. 1 [Poor]
2 101011000023 10101100 10101    3.82   93.7   169323. 0 [Non-poor]
3 101011000040 10101100 10101    2.67   93.7    93644. 1 [Poor]
4 101011000071 10101100 10101    4.79   93.7   452758. 0 [Non-poor]
5 101011000095 10101100 10101    4.31   93.7   183333. 0 [Non-poor]
6 101011000115 10101100 10101    2      93.7   420656. 0 [Non-poor]
```

- Note that in this case, we already have the ea codes in the survey data
    - So we can just join the two datasets!

Collapse to EA_CODE

▼ Code

```
library(stats) # this is for weighted.mean
ihs5ea <- ihs5 |>
  rename(EA_CODE = ea_id) |>
  group_by(EA_CODE) |>
  # Note that this is a weighted mean!
  summarize(poor = stats::weighted.mean(x = poor, w = hh_wgt*adulteq, na.rm = TRUE), # weighted mean
    total_weights = sum(hh_wgt*adulteq, na.rm = TRUE), # sum total weights
    total_obs = n()) # total observations (households) in the EA
head(ihs5ea)
```

```
# A tibble: 6 × 4
  EA_CODE    poor total_weights total_obs
  <chr>     <dbl>         <dbl>     <int>
1 10101100 0.230          5690.        16
2 10101101 0.444          7614.        16
3 10101102 0.0947         9441.        16
4 10101103 0.376          7486.        16
5 10101104 0.600          9147.        16
6 10101105 0.497          5351.        16
```

But what if you don't have an identifier?

- Sometimes you have GPS coordinates but not matching identifiers
  - Good news! We can use geospatial tools to match the coordinates to the shapefile

▼ Code

```
ihs5coords <- read_dta("day4data/householdgeovariables_ihs5.dta")
# turn it into an sf object
ihs5coords <- ihs5coords |>
  filter(!is.na(ea_lon_mod)) |> # get rid of any missing values (can't use them)
  st_as_sf(coords = c("ea_lon_mod", "ea_lat_mod"), crs = 4326)
head(ihs5coords)
```

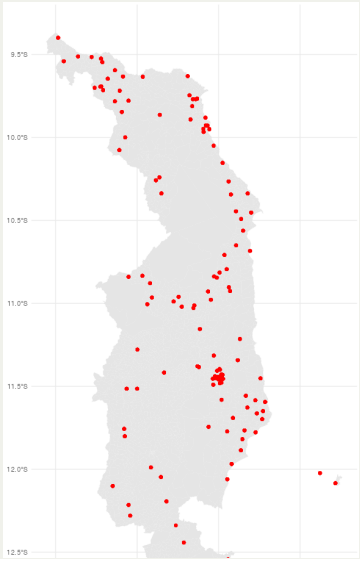
```
Simple feature collection with 6 features and 1 field
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 33.23917 ymin: -9.70057 xmax: 33.23917 ymax: -9.70057
Geodetic CRS:  WGS 84
# A tibble: 6 × 2
  case_id                geometry
  <chr>               <POINT [°]>
1 101011000014 (33.23917 -9.70057)
2 101011000023 (33.23917 -9.70057)
```

```
3 101011000040 (33.23917 -9.70057)
4 101011000071 (33.23917 -9.70057)
5 101011000095 (33.23917 -9.70057)
6 101011000115 (33.23917 -9.70057)
```

Here’s how it looks

▼ Code

```r
ihs5coords <- read_dta("day4data/householdgeovariables_ihs5.dta")
# turn it into an sf object
ihs5coords <- ihs5coords |>
  filter(!is.na(ea_lon_mod)) |> # get rid of any missing values (can't use th
  st_as_sf(coords = c("ea_lon_mod", "ea_lat_mod"), crs = 4326)
admin4 <- read_sf("day4data/mw4.shp")
ggplot() +
  geom_sf(data = admin4, color = "transparent", lwd = 0.01) +
  geom_sf(data = ihs5coords, color = "red") +
  theme_minimal()
```



We can join them using `st_join`

▼ Code

```
ihs5coords <- read_dta("day4data/householdgeovariables_ihs5.dta")
# turn it into an sf object
ihs5coords <- ihs5coords |>
  filter(!is.na(ea_lon_mod)) |> # get rid of any missing values (can't use them)
  st_as_sf(coords = c("ea_lon_mod", "ea_lat_mod"), crs = 4326)
admin4 <- read_sf("day4data/mw4.shp")
ihs5coords <- st_join(ihs5coords, admin4)
# and that's it!
head(ihs5coords)
```

```
Simple feature collection with 6 features and 4 fields
Geometry type: POINT
Dimension:     XY
Bounding box:  xmin: 33.23917 ymin: -9.70057 xmax: 33.23917 ymax: -9.70057
Geodetic CRS:  WGS 84
# A tibble: 6 × 5
  case_id                geometry DIST_CODE EA_CODE  TA_CODE
  <chr>               <POINT [°]> <chr>     <chr>    <chr>
1 101011000014 (33.23917 -9.70057) 101       10101006 10101
2 101011000023 (33.23917 -9.70057) 101       10101006 10101
3 101011000040 (33.23917 -9.70057) 101       10101006 10101
4 101011000071 (33.23917 -9.70057) 101       10101006 10101
5 101011000095 (33.23917 -9.70057) 101       10101006 10101
```

Now we need to add the EA_CODE to the poverty data

▼ Code

```r
ihs5 <- read_dta("day4data/ihs5_consumption_aggregate.dta")
ihs5 <- ihs5 |>
  left_join(ihs5coords, by = "case_id")
# collapse to EA_CODE
ihs5ea <- ihs5 |>
  group_by(EA_CODE) |>
  # Note that this is a weighted mean!
  summarize(poor = stats::weighted.mean(x = poor, w = hh_wgt*adulteq, na.rm = TRUE), # weighted mean
    total_weights = sum(hh_wgt*adulteq, na.rm = TRUE), # sum total weights
    total_obs = n()) # total observations (households) in the EA
head(ihs5ea)
```

```
# A tibble: 6 × 4
  EA_CODE    poor total_weights total_obs
  <chr>     <dbl>         <dbl>     <int>
1 10101006 0.230          5690.        16
2 10101011 0.444          7614.        16
3 10101027 0.0947         9441.        16
```

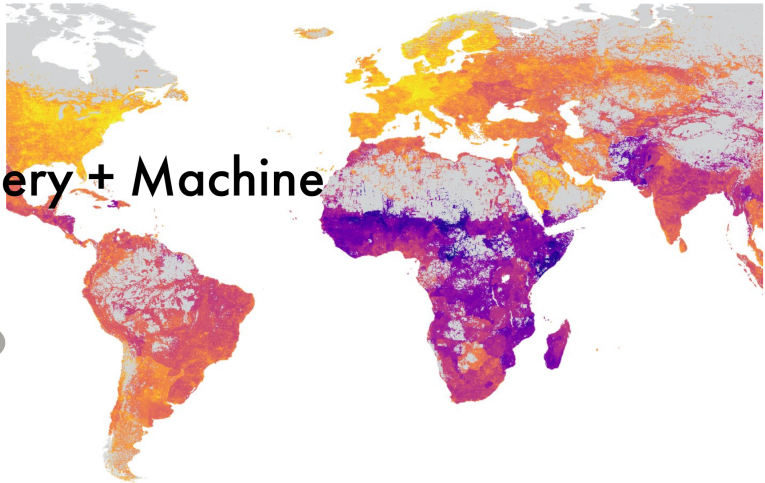
4	10101033	0.376	7486.	16
5	10101039	0.600	9147.	16
6	10101054	0.497	5351.	16

▼ Code

```
1 # save it!
2 write_csv(ihs5ea, "day4data/ihs5ea.csv")
```

One last set of data: MOSAIKS

Satellite Imagery + Machine Learning for Everyone



One last set of data: MOSAIKS

- MOSAIKS is a dataset that has a lot of different variables
  - These variables were constructed by the authors from satellite imagery
  - You can download all of these features using their website
  - https://api.mosaiks.org - you'll have to register
  - The data is quite large, so I've downloaded and uploaded it for you
    - It's a random selection of 500 variables for EAs in Norther Malawi
    - `day4data/mosaikvars.csv`
    - Note that I used `st_join` to match it to the shapefile!

Rolf et al. (2021)

Putting it all together

What have we downloaded?

- We have:
  - Population data from WorldPop
  - Nightlights data from GEE
  - NDVI data from GEE
  - MOSAIKS data
  - Survey data from Malawi

- We have everything we need to estimate a simple SAE model using geospatial data!
  - Note: in practice, I would use even more predictors, but this is a good start

First, let's load all the data

▼ Code

```r
# Population
pop <- read_csv("day4data/mwpopEAs.csv")
# Nightlights
ntl <- read_csv("day4data/mwntlEAs.csv")
# NDVI
ndvi <- read_csv("day4data/ndviclean.csv")
# mosaiks
mosaik <- read_csv("day4data/mosaikvars.csv")
# survey data
ihs5ea <- read_csv("day4data/ihs5ea.csv")
# all EAs
adm4 <- read_sf("day4data/mw4.shp")
# no geometry, just EA_CODE
adm4 <- as_tibble(adm4) |>
  dplyr::select(EA_CODE, TA_CODE)
```

Now, we join it all!

▼ Code

```
adm4 <- adm4 |>
  left_join(ihs5ea, by = "EA_CODE") |> # add survey data
  left_join(pop, by = "EA_CODE") |> # add pop
  left_join(ntl, by = "EA_CODE") |> # add nightlights
  left_join(ndvi, by = "EA_CODE") |> # add ndvi
  left_join(mosaik, by = "EA_CODE") |> # add mosaik

head(adm4)
```

- Oh no! This throws an error!
  - `x$EA_CODE` is a `<character>`
  - `y$EA_CODE` is a `<double>`
  - What's going on?

Now, we join it all!

▼ Code

```
adm4 <- adm4 |>
  mutate(EA_CODE = as.numeric(EA_CODE)) |>
  left_join(ihs5ea |> mutate(EA_CODE = as.numeric(EA_CODE)), by = "EA_CODE") |> # add survey data
  left_join(pop |> mutate(EA_CODE = as.numeric(EA_CODE)), by = "EA_CODE") |> # add pop
  left_join(ntl |> mutate(EA_CODE = as.numeric(EA_CODE)), by = "EA_CODE") |> # add nightlights
  left_join(ndvi |> mutate(EA_CODE = as.numeric(EA_CODE)), by = "EA_CODE") |> # add ndvi
  left_join(mosaik |> mutate(EA_CODE = as.numeric(EA_CODE)), by = "EA_CODE") # add mosaik

head(adm4)
```

```
# A tibble: 6 × 510
   EA_CODE TA_CODE  poor total_weights total_obs   pop   ntl ndvimin ndvimax ndvimean  mosaik1  mosaik2 mosaik3 mosaik4 mosaik5 m
     <dbl> <chr>   <dbl>         <dbl>     <dbl> <dbl> <dbl>   <dbl>   <dbl>    <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>
1 1 10507801 10507      NA            NA        NA 1069. 0.205   2546.   6635.    4198. 0.00334  0.00344    1.23   0.0964  0.220
2 2 10507072 10507      NA            NA        NA  696. 0.129   2409.   7246.    4511. 0.000329 0.000155   0.804  0.0299  0.0611
3 3 10507010 10507      NA            NA        NA  938. 0.135   2579.   6316.    4114. 0.000196 0.000145   0.875  0.0308  0.0531
4 4 10507001 10507      NA            NA        NA  750. 0.129   3200.   7187.    5134. 0.000932 0.00121    0.699  0.0437  0.0940
5 5 10507009 10507      NA            NA        NA  598. 0.146   3107.   6728.    4653. 0.000321 0.000359   0.763  0.0306  0.0660
6 6 10507033 10507      NA            NA        NA  474. 0.139   2856.   6382.    4343. 0.000430 0.000241   0.667  0.0293  0.0702
# ℹ 348 more variables: mosaik153 <dbl>, mosaik154 <dbl>, mosaik155 <dbl>, mosaik156 <dbl>, mosaik157 <dbl>, mosaik158 <dbl>, mos
# ℹ mosaik239 <dbl>, mosaik240 <dbl>, mosaik241 <dbl>, mosaik242 <dbl>, mosaik243 <dbl>, mosaik244 <dbl>, mosaik245 <dbl>, mosaik
```

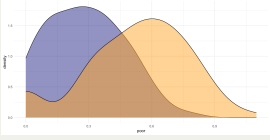
What do poverty rates look like?

▼ Code

```
# levels vs. arcsin squareroot transformation
ggplot() +
  geom_density(data = adm4, aes(x = poor), fill = "navy", alpha = 0.5) +
  geom_density(data = adm4, aes(x = asin(sqrt(poor))), fill = "orange", alpha = 0.5) +
  theme_minimal()
```

▼ Code

```
# Not perfect but neither is horribly non-normal!
```



Before SAE, need to do some cleaning

▼ Code

```
1 # check for missings
2 sum(is.na(adm4$pop))
```

```
1 [1] 0
```

▼ Code

```
1 sum(is.na(adm4$ntl))
```

```
1 [1] 0
```

▼ Code

```
1 sum(is.na(adm4$ndvimin))
```

```
1 [1] 0
```

▼ Code

```
sum(is.na(adm4$mosaik1))
```

```
[1] 301
```

▼ Code

```
# we have missings for the mosaik features!
```

Missing mosaiks, what to do?

▼ Code

```r
# I am going to replace missing mosaiks values with the TA mean
adm4 <- adm4 |>
  group_by(TA_CODE) |>
  mutate(across(starts_with("mosaik"), ~replace_na(., mean(., na.rm = TRUE)))) |>
  ungroup()

sum(is.na(adm4$mosaik1)) # fixed!
```

```
[1] 0
```

Now let's select features

▼ Code

```r
library(glmnet)

# let's also log population!
adm4 <- adm4 |>
  mutate(pop = log(pop))

surveyeas <- adm4 |>
  filter(!is.na(poor))
samplefeatures <- surveyeas |>
  select(-c(EA_CODE, TA_CODE, poor, total_weights, total_obs)) # remove these variables
samplelabels <- surveyeas$poor
sampleweights <- surveyeas$total_weights

# now lasso
set.seed(24826)
lasso <- cv.glmnet(x = as.matrix(samplefeatures), y = samplelabels,
  weights = sampleweights,
  alpha = 1)
```

What did lasso do?

▼ Code

```
coef(lasso, s = "lambda.min")
```

```
506 x 1 sparse Matrix of class "dgCMatrix"
                        s1
(Intercept)   0.44719559
pop          -0.02712011
ntl          -0.04331957
ndvimin       .
ndvimax       .
ndvimean      .
mosaik1       .
mosaik2       .
mosaik3       .
mosaik4       .
mosaik5       .
mosaik6       .
mosaik7       .
mosaik8       .
mosaik9       .
mosaik10      .
```

We want to get the non-zero variables

▼ Code

```
coefs <- coef(lasso, s = "lambda.min")
nonzerocoefs <- row.names(coefs)[which(coefs!=0)]
nonzerocoefs
```

```
[1] "(Intercept)" "pop"         "ntl"         "mosaik263"   "mosaik277"   "mosaik280"   "mosaik293"   "mosaik459"
```

▼ Code

```
# but we don't want the intercept here (you'll see why)
nonzerocoefs <- nonzerocoefs[-1]
nonzerocoefs
```

```
[1] "pop"       "ntl"       "mosaik263" "mosaik277" "mosaik280" "mosaik293" "mosaik459"
```

Now we want to turn that into a formula!

▼ Code

```
formula <- as.formula(paste("poor ~", paste(nonzerocoefs, collapse = " + ")))
formula
```

```
poor ~ pop + ntl + mosaik263 + mosaik277 + mosaik280 + mosaik293 +
    mosaik459
```

▼ Code

```
# you can see it works with a simple linear regression!
lm(formula, data = surveyeas)
```

```

Call:
lm(formula = formula, data = surveyeas)

Coefficients:
(Intercept)          pop          ntl    mosaik263    mosaik277    mosaik280    mosaik293    mosaik459
    0.35065     -0.01400     -0.04580      0.22365     -0.04598    -21.35272     -0.09561     92.93638
```

Finally, we can do SAE

▼ Code

```r
library(povmap)

saeresults <- povmap::ebp(fixed = formula, # the formula
  pop_data = adm4,
  pop_domains = "TA_CODE",
  smp_data = surveyeas,
  smp_domains = "TA_CODE",
  transformation = "arcsin",
  weights = "total_weights",
  pop_weights = "pop",
  weights_type = "nlme",
  MSE = TRUE,
  rescale_weights = TRUE,
  seed = 1234,
  L = 0)
```

```
Time difference of 0.67 secs
```

The results

▼ Code

```
# What is R2?
summary(saeresults)$coeff_determ
```

```
  Marginal_R2 Conditional_R2 Marginal_Area_R2 Conditional_Area_R2
    0.3135187      0.4101433        0.5493752           0.7391761
```

▼ Code

```
# Relatively normal?
summary(saeresults)$normality
```

```
                Skewness Kurtosis Shapiro_W    Shapiro_p
Error         -0.4285633 3.271975 0.9844836 1.529020e-01
Random_effect -0.5277791 6.586239 0.9033095 2.636191e-05
```

Now let's get the predictions

▼ Code

```r
# get the predictions
hat <- as_tibble(saeresults$ind)
hat <- hat |>
  select(TA_CODE = Domain, poor = Mean)
# We also want variance estimates
mse <- as_tibble(saeresults$MSE) |>
  select(TA_CODE = Domain, mse = Mean)
# note to get the standard error we have to take the square root!
mse <- mse |>
  mutate(se = sqrt(mse))

# finally, join them!
final <- left_join(hat, mse, by = "TA_CODE")

head(final)
```

```
# A tibble: 6 × 4
  TA_CODE  poor     mse     se
  <fct>    <dbl>   <dbl>  <dbl>
1 10101    0.366 0.00211 0.0460
```

```
5  2 10102   0.351 0.00443 0.0665
6  3 10103   0.263 0.00362 0.0602
7  4 10104   0.273 0.0106  0.103
8  5 10105   0.377 0.00444 0.0666
9  6 10106   0.383 0.00704 0.0839
```

Let's map it!

▼ Code

```r
# Admin3 (TA)
adm3 <- read_sf("day4data/mw3.shp")
adm3 <- adm3 |>
  left_join(final, by = "TA_CODE")

# let's output two things: poverty rate and CV (se/mean)
g1 <- ggplot() +
  geom_sf(data = adm3, aes(fill = poor), color = "white") +
  scale_fill_distiller("Poverty Rate", palette = "Spectral", direction = -1) +
  theme_minimal() +
  theme(legend.position = "bottom")
g2 <- ggplot() +
  geom_sf(data = adm3, aes(fill = se/poor), color = "white") +
  scale_fill_distiller("CV", palette = "Spectral", direction = -1) +
  theme_minimal() +
  theme(legend.position = "bottom")
g3 <- ggplot() +
  geom_sf(data = adm3, aes(fill = as.factor((se/poor)<=0.4)), color = "white") +
  scale_fill_brewer("CV below 0.4?", palette = "Reds", direction = -1) +
```

The final result!

▼ Code

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