Spatial joins, feature selection, and SAE

Nairobi Workshop: Day 3

Josh Merfeld
University of Queensland

David Newhouse
World Bank

2025-10-20

Intersections and spatial joins

Let’s move on to a new topic

- Here’s our goal:
    - We want to count the number of points in each polygon
    - We want to count points in each ADMIN2 (not admin3)
- We have two files: one is data with lon/lat coordinates and one is admin3 polygons
    - We need to create a shapefile from the points
    - We need to aggregate the admin3 to admin2
    - We need to extract the admin2 identifier into points
    - Then we need to join the two together

What we need

- Points: `points.csv`
  - This is a list of health facilities in Malawi
  - Note that this data is NOT complete (it comes from OpenStreetMap)
- Polygons: `mw3allcountry.shp`
  - Let's use the entire country

Step 1: Load the points

▼ Code

```r
# note that this function is in tidyverse!
points <- read_csv("day3files/points.csv")
```

- Find the name of the columns that represent the coordinates
    - We'll need these to create the spatial object

▼ Code

```r
# Let's turn it into a `terra` object
points <- vect(points, geom = c("x", "y"), crs = "EPSG:4326") # these are lon/lat
points
```

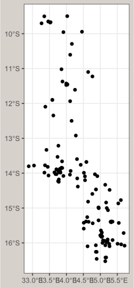
```
class       : SpatVector
geometry    : points
dimensions  : 155, 1  (geometries, attributes)
extent      : 32.88771, 35.71544, -16.52659, -9.496221  (xmin, xmax, ymin, ymax)
coord. ref. : lon/lat WGS 84 (EPSG:4326)
names       :         fclass
```

```
type        :            <chr>
values      : health facility
              health facility
              health facility
```

Let’s just plot the points

▼ Code

```
ggplot() +
  geom_spatvector(data = points) +
  theme_bw()
```



Now let's load the admin3 and aggregate

▼ Code

```r
adm3 <- vect("day3files/mw3allcountry.shp")
```

- Do you remember how to aggregate to admin2?
  - You need to use `aggregate` and the column name!
  - Look at the column names

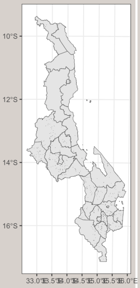
▼ Code

```r
adm2 <- aggregate(adm3, "DIST_NAME")
```

Let’s plot it

▼ Code

```r
ggplot() +
  geom_spatvector(data = adm2) +
  theme_bw()
```



Time to extract

- We are going to use the function `relate` to find which adm2 feature the point lies within
  - This is a spatial join
  - We are also going to use this to learn a bit more about `R`

▼ Code

```
# let's first make sure it's the same CRS!
points <- project(points, crs(adm2))

join <- relate(adm2, points, "contains")
# check the dimensions
dim(join)
```

```
[1]  32 155
```

The `apply` function

Example data

	col1	col2	col3	col4
Point 1	TRUE	FALSE	FALSE	TRUE
Point 2	FALSE	FALSE	TRUE	FALSE
Point 3	FALSE	TRUE	FALSE	TRUE
Point 4	FALSE	TRUE	TRUE	FALSE
Point 5	FALSE	FALSE	FALSE	TRUE
Point 6	TRUE	TRUE	FALSE	TRUE

- Let's look at an example
- An important note:
  - If you take the `mean` of logical statements, it treats `TRUE` as 1 and `FALSE` as 0
  - Same with `sum`

The `apply` function

Example data

	col1	col2	col3	col4
Point 1	TRUE	FALSE	FALSE	TRUE
Point 2	FALSE	FALSE	TRUE	FALSE
Point 3	FALSE	TRUE	FALSE	TRUE
Point 4	FALSE	TRUE	TRUE	FALSE
Point 5	FALSE	FALSE	FALSE	TRUE
Point 6	TRUE	TRUE	FALSE	TRUE

- In this data, what happens if we:
    - Take the `sum` of each row?
    - What values do we get?

The `apply` function

Example data

	col1	col2	col3	col4	sum
Point 1	TRUE	FALSE	FALSE	TRUE	2
Point 2	FALSE	FALSE	TRUE	FALSE	1
Point 3	FALSE	TRUE	FALSE	TRUE	2
Point 4	FALSE	TRUE	TRUE	FALSE	2
Point 5	FALSE	FALSE	FALSE	TRUE	1
Point 6	TRUE	TRUE	FALSE	TRUE	3

- We can use the `apply` function to do this!
  - `data$sum = apply(data, 1, "sum")`
  - This will sum each row and put it in a new column called `sum`
  - What happens if we instead use `"mean"`?
- The `1` denotes doing this by row… how do we apply the function by column?

The `apply` function

Example data

	col1	col2	col3	col4	mean
Point 1	TRUE	FALSE	FALSE	TRUE	0.5
Point 2	FALSE	FALSE	TRUE	FALSE	0.25
Point 3	FALSE	TRUE	FALSE	TRUE	0.5
Point 4	FALSE	TRUE	TRUE	FALSE	0.5
Point 5	FALSE	FALSE	FALSE	TRUE	0.25
Point 6	TRUE	TRUE	FALSE	TRUE	0.75

```
data$mean = apply(data, 1, "mean")
```

Back to our data

- Check the dimensions of the data
    - What do rows and what do columns represent?

▼ Code

```
dim(join)
```

```
[1]  32 155
```

▼ Code

```
join
```

```
      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9] [,10] [,11] [,12]
[1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE  TRUE  TRUE
[4,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

```
 [6,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [7,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [8,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
 [9,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[10,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[11,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[12,] FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[13,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[14,] FALSE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[15,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[16,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[17,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

How do we use `apply` here?

- So the columns are POINTS and the rows are POLYGONS (adm2)
- What do we want to do?
- We want to find the number of points in each polygon
  - It is `TRUE/FALSE` so we want to `sum` BY rows!

How do we use `apply` here?

▼ Code

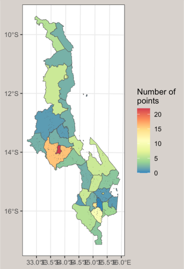
```r
adm2$points = apply(join, 1, "sum")
adm2$points
```

```
 [1]  6  2 14  5  0  5  3  2  2  1  0 16 22  3  6  2  4  1  6  7  6  2  2  3  2
[26]  1  1  2  6 10  1  7
```

Now we can graph it!

▼ Code

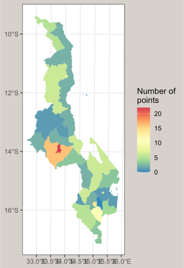
```r
ggplot() +
  geom_spatvector(data = adm2, aes(fill = points)) +
  scale_fill_distiller("Number of\npoints", palette = "Spectral") +
  theme_bw()
```



Make it a little nicer

▼ Code

```r
ggplot() +
  geom_spatvector(data = adm2, aes(fill = points), color = NA) +
  scale_fill_distiller("Number of\npoints", palette = "Spectral") +
  theme_bw()
```



Extract admin information to points

- What if we have our household survey data and we want to figure out within which admin area a point lies?

It looks like this

▼ Code

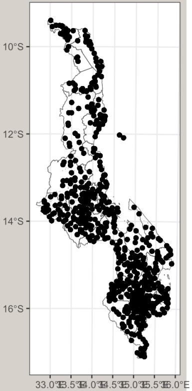
```r
# households are not currently a vector file
households <- read_dta("day3files/households.dta")
households
```

```
# A tibble: 11,290 × 44
   case_id      ea_id dist_road dist_agmrkt dist_auction dist_admarc dist_border
   <chr>        <chr>     <dbl>       <dbl>        <dbl>       <dbl>       <dbl>
 1 101011000014 1010…      3.60        4.10         213.        12.8        3.40
 2 101011000023 1010…      3.60        4.10         213.        12.8        3.40
 3 101011000040 1010…      3.60        4.10         213.        12.8        3.40
 4 101011000071 1010…      3.60        4.10         213.        12.8        3.40
 5 101011000095 1010…      3.60        4.10         213.        12.8        3.40
 6 101011000115 1010…      3.60        4.10         213.        12.8        3.40
 7 101011000126 1010…      3.60        4.10         213.        12.8        3.40
 8 101011000135 1010…      3.60        4.10         213.        12.8        3.40
 9 101011000183 1010…      3.60        4.10         213.        12.8        3.40
10 101011000190 1010…      3.60        4.10         213.        12.8        3.40
# ℹ 11,280 more rows
# ℹ 37 more variables: dist_popcenter <dbl>, dist_boma <dbl>,
#   ssa_aez09 <dbl+lbl>, twi_mwi <dbl>, sq1 <dbl+lbl>, sq2 <dbl+lbl>,
#   sq3 <dbl+lbl>, sq4 <dbl+lbl>, sq5 <dbl+lbl>, sq6 <dbl+lbl>, sq7 <dbl+lbl>,
#   af_bio_1_x <dbl>, af_bio_8_x <dbl>, af_bio_12_x <dbl>, af_bio_13_x <dbl>,
```

▼ Code

```
# turn them into a vector file like this
households <- vect(households, geom = c("ea_lon_mod", "ea_lat_mod"), crs = "EPSG:4326")
```

A map



A map

▼ Code

```
# easy!
households <- project(households, crs(adm2))
join <- intersect(adm2, households)
join
```

```
class       : SpatVector
geometry    : points
dimensions  : 11282, 50  (geometries, attributes)
extent      : 483510.5, 809173.4, 8109036, 8961420  (xmin, xmax, ymin, ymax)
coord. ref. : Arc 1950 / UTM zone 36S
names       : DIST_NAME mean_OBJECTID mean_REG_CODE REG_NAME   TA_CODE
type        :     <chr>         <num>         <num>    <chr> <logical>
values      :    Balaka         261.5             3 Southern      <NA>
                 Balaka         261.5             3 Southern      <NA>
                 Balaka         261.5             3 Southern      <NA>
  TA_NAME agg_n points      case_id    ea_id (and 40 more)
<logical> <int>  <int>        <chr>    <chr>
     <NA>    12      6 312057440010 31205744
     <NA>    12      6 312057440067 31205744
     <NA>    12      6 312057440015 31205744
```

Calculating areas and lengths

- We can also calculate areas and lengths!
- `perim` for length
  - For polygons, it returns the length of the perimeter
  - For lines, it returns the length of the line
- `expanse` for area
  - Returns the area of polygons
  - What will it return for lines or points?
- Believe it or not, using lon/lat gives the most accurate results!
  - This is because of the haversine formula

Area

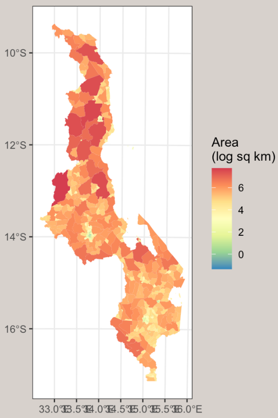
- Let's use the `adm3` shapefile

▼ Code

```
adm3$area <- expanse(adm3, unit="km", transform = TRUE)
```

- Unit: do you want km^2 or m^2?
- Transform: automatically transform to lon/lat?
    - Always do this. `terra` documentation says this will be more accurate

Area


10°S
12°S
14°S
16°S
33.0°E
33.5°E
34.0°E
34.5°E
35.0°E
35.5°E
36.0°E
Area
(log sq km)
6
4
2
0

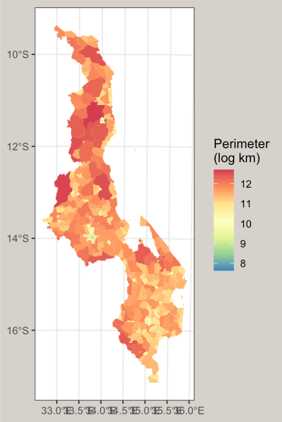
Perimeter

▼ Code

```
adm3$perimeter <- perim(adm3 |> project("EPSG:4326"))
```

- This automatically gives length in meters
- You can add `project` with a pipe operator `|>` to transform to lon/lat

Perimeter


10°S
12°S
14°S
16°S
33.0°E
33.5°E
34.0°E
34.5°E
35.0°E
35.5°E
36.0°E
Perimeter
(log km)
12
11
10
9
8

More distances

- We have points for households
- We have points for health facilities
- We can calculate the distance between each household and each health facility
    - Finding distances between points is a common GIS task!

Distance matrix

- The name of the households data is `households.dta`
  - This is a Stata dataset
  - You can read it using the package `haven`
- Please go ahead and try loading the dataset and then turning it into a `terra` object
  - You'll have to find the names of the columns that represent the coordinates

Distance matrix

▼ Code

```
# households
households <- read_dta("day3files/households.dta")
households <- vect(households, geom = c("ea_lon_mod", "ea_lat_mod"), crs = "EPSG:4326")
# health facilities
# do it in one line!
health <- vect(read_csv("day3files/points.csv"), geom = c("x", "y"), crs = "EPSG:4326")
```

Distance matrix

▼ Code

```r
distances <- distance(households, health)
dim(households)
```

```
[1] 11290    42
```

▼ Code

```r
dim(health)
```

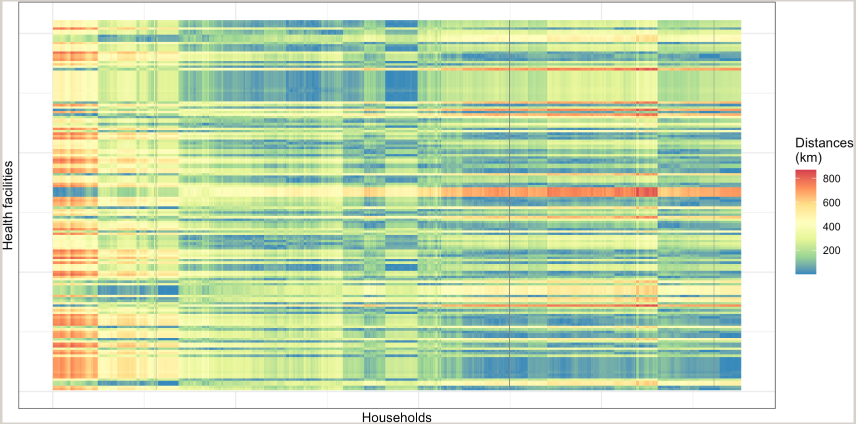
```
[1] 155   1
```

▼ Code

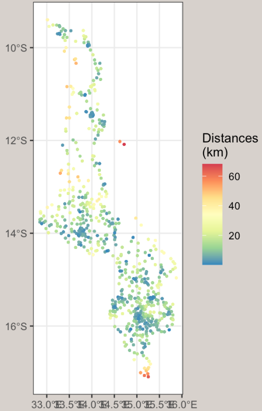
```r
dim(distances)
```

```
[1] 11290   155
```

Distance matrix - “Heat map”



Closest health facility by household



How did I create the map?

- The `distances` object is a matrix
  - What are the rows and what are the columns?
- If we want to find the closest health facility to each household, what do we need to do?
  - We need to find the minimum distance for each row
  - Do you remember?

Closest health facility by household

- We can use the `apply` function!
  - But with `"min"` instead of `"sum"`
  - The rows are in the same order as the households, so…

▼ Code

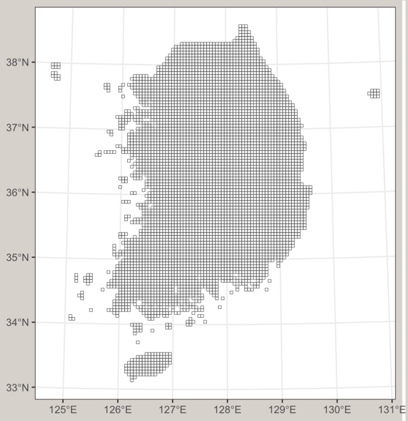
```r
distances <- distance(households, health)
closest <- apply(distances, 1, "min")

# they're in the same order!
households$closest <- closest

ggplot() +
  geom_spatvector(data = households, aes(color = closest/1000), size = 0.5) +
  scale_color_distiller("Distances\n(km)", palette = "Spectral") +
  theme_bw()
```

More advanced operations

Let’s go over some more advanced operations

- This stuff will spillover into next week
- First up:
  - Spatial overlap

A grid in Korea - `kgrid.zip`



A grid in Korea - `kgrid.zip`

- We will discuss how to make a grid after we learn about rasters
- For now, the grid is a shapefile
- A very common operation:
  - We want to know which province/city each grid cell is in
  - This isn't straightforward. Why?

- A grid cell can overlap multiple provinces/cities

The data

- Here is the data:
  - `kshape.shp` is the shapefile of the provinces/cities
  - `kgrid.shp` is the grid
  - I have uploaded .zip files for both

▼ Code

```
kshape <- vect("day3files/kshape.shp")
kgrid <- vect("day3files/kgrid.shp")
kgrid
```

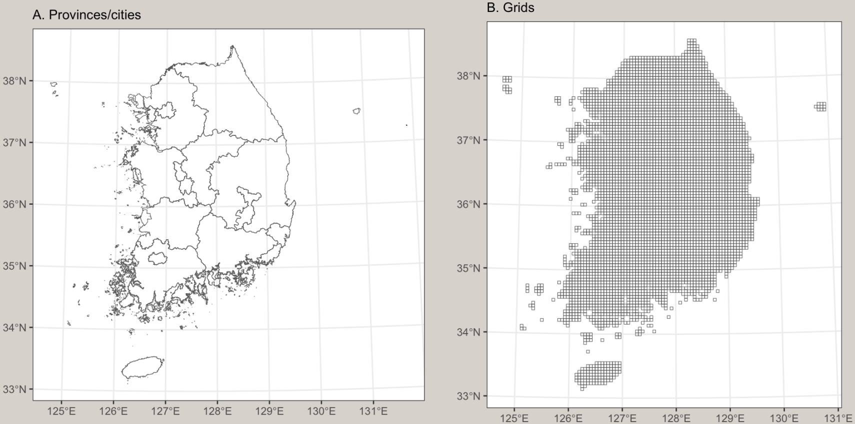
```
class       : SpatVector
geometry    : polygons
dimensions  : 4906, 1  (geometries, attributes)
extent      : 746110.3, 1306110, 1458754, 2068754  (xmin, xmax, ymin, ymax)
```

```
source      : kgrid.shp
coord. ref. : KGD2002_Unified_Coordinate_System (ESRI:102080)
names       :    id
type        : <num>
values      :    65
                 66
                 67
```

The `intersect` function from `terra`

- We are going to use the `intersect` function
- Here's what it will do:
  - It will find the intersection of the grid cell and the province/city
  - Except, it will return a new feature for EACH overlap
- Let's look at some maps

The `intersect` function from `terra`



The `intersect` function from `terra`

▼ Code

```r
intersection <- intersect(kgrid, kshape)
intersection
```

```
class       : SpatVector 
geometry    : polygons 
dimensions  : 5546, 3  (geometries, attributes)
extent      : 746110.3, 1304099, 1458754, 2068444  (xmin, xmax, ymin, ymax)
coord. ref. : KGD2002_Unified_Coordinate_System (ESRI:102080) 
names       :    id CTPRVN_CD CTP_ENG_NM
type        : <num>     <chr>      <chr>
values      :    65        51 Gangwon-do
                 66        51 Gangwon-do
                 67        51 Gangwon-do
```

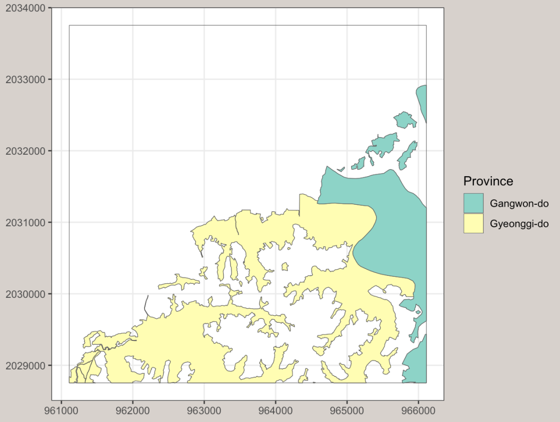
▼ Code

```r
kgrid
```

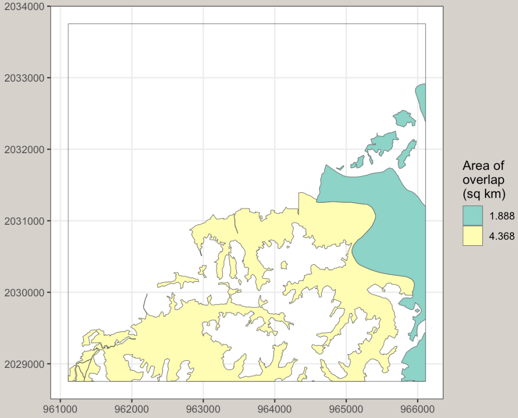
```
class       : SpatVector 
geometry    : polygons 
dimensions  : 4906, 1  (geometries, attributes)
extent      : 746110.3, 1306110, 1458754, 2068754  (xmin, xmax, ymin, ymax)
```

```
source      : kgrid.shp
coord. ref. : KGD2002_Unified_Coordinate_System (ESRI:102080)
names       :    id
type        : <num>
values      :    65
                 66
                 67
```

One grid cell, multiple overlaps



One grid cell, multiple overlaps



Area of overlap

- So what do we want to do?
  - Let's find the area of overlap for each grid cell and each province/city
  - Then let's take the LARGEST overlap and assign that to the grid cell
  - In practice, depending on the context, you could use a weighted mean or something else
    - This won't work with categorical variables, though

▼ Code

```
# Get area of all overlaps
intersection$area <- expanse(intersection)
# turn it into a tibble
intersection <- as_tibble(intersection)
# Get the largest overlap
intersection <- intersection |> group_by(id) |> filter(area==max(area)) |> ungroup()
```

Area of overlap

▼ Code

```
intersection
```

```
# A tibble: 4,906 × 4
      id CTPRVN_CD CTP_ENG_NM      area
   <dbl> <chr>     <chr>          <dbl>
 1    65 51        Gangwon-do   332498.
 2    66 51        Gangwon-do  3885964.
 3    67 51        Gangwon-do  1659454.
 4   193 51        Gangwon-do   477650.
 5   194 51        Gangwon-do 12783973.
 6   195 51        Gangwon-do 14674738.
 7   321 51        Gangwon-do   289674.
 8   322 51        Gangwon-do 14274204.
 9   323 51        Gangwon-do 22582297.
10   324 51        Gangwon-do   604121.
# ℹ 4,896 more rows
```

▼ Code

```
kgrid
```

```
 class       : SpatVector
```

```
geometry    : polygons
dimensions  : 4906, 1  (geometries, attributes)
extent      : 746110.3, 1306110, 1458754, 2068754  (xmin, xmax, ymin, ymax)
source      : kgrid.shp
coord. ref. : KGD2002_Unified_Coordinate_System (ESRI:102080)
names       :    id
type        : <num>
values      :    65
                 66
                 67
```

Intro to SAE

- Let's start with some example data I have
  - This comes from Malawi
    - Northern Malawi only (due to the size of the data)

▼ Code

```
library(tidyverse)
surveycollapsed <- read_csv("day3files/ihs5ea.csv")
predictors <- read_csv("day3files/mosaikvars.csv")
```

A short explanation of SAE

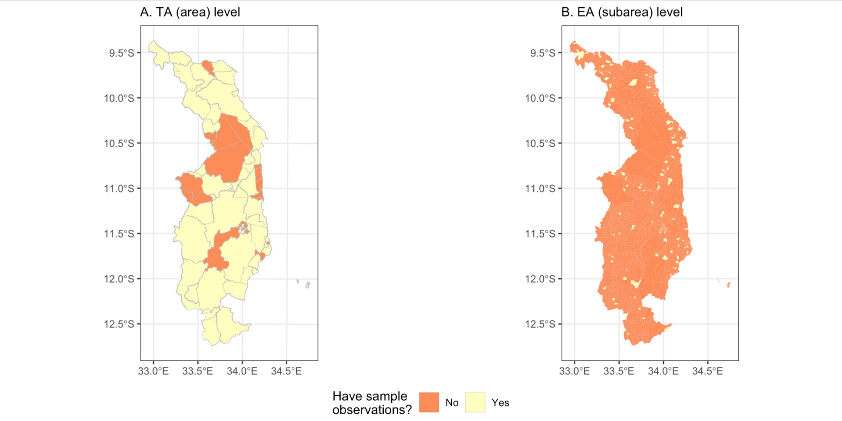
- Small area estimation terminology:
    - In Malawi, we want to estimate poverty at the admin3 (TA) level
- Admin3: “area”
- Admin4: “subarea”

A short explanation

- We have poverty rates for `subareas` (EAs)
  - We pulled geospatial data at the `subarea` level, as well

- So it’s a perfect setup for SAE!
  - We want to estimate poverty at the TA
  - We don’t have any observations in some TAs and we have too few in others
  - We could estimate a subarea model

Observations?



Predictive features

- I also have a bunch of predictive features!
  - The data come from something called MOSAIKS, that we'll discuss briefly tomorrow
  - In short, they are variables derived from satellite imagery
  - Take a look at this

▾ Code

```
predictors
```

```
# A tibble: 2,911 × 501
    EA_CODE  mosaik1  mosaik2 mosaik3 mosaik4 mosaik5 mosaik6 mosaik7 mosaik8
      <dbl>    <dbl>    <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
 1 10101001 0.00143  0.00242    0.632  0.0334  0.0684   0.223 0.00641  0.0753
 2 10101002 0.000659 0.000250   0.911  0.0468  0.0978   0.357 0.00359  0.127
 3 10101003 0.000657 0.000403   0.811  0.0373  0.0794   0.326 0.00285  0.100
 4 10101004 0.00102  0.000769   0.975  0.0578  0.111    0.369 0.00584  0.140
 5 10101005 0.000472 0.000351   0.815  0.0344  0.0668   0.381 0.00287  0.0947
 6 10101006 0.00107  0.000835   0.861  0.0496  0.122    0.315 0.00536  0.137
```

```
   7 10101007 0.00132  0.000842   1.13   0.0549  0.0999   0.594 0.00649  0.154
   8 10101008 0.00202  0.00182    1.05   0.0796  0.166    0.415 0.00953  0.179
   9 10101009 0.000445 0.000417   0.834  0.0332  0.0663   0.375 0.00278  0.0950
  10 10101010 0.000720 0.000438   0.794  0.0367  0.0849   0.328 0.00377  0.109
# ℹ 2,901 more rows
# ℹ 492 more variables: mosaik9 <dbl>, mosaik10 <dbl>, mosaik11 <dbl>,
#   mosaik12 <dbl>, mosaik13 <dbl>, mosaik14 <dbl>, mosaik15 <dbl>,
#   mosaik16 <dbl>, mosaik17 <dbl>, mosaik18 <dbl>, mosaik19 <dbl>,
#   mosaik20 <dbl>, mosaik21 <dbl>, mosaik22 <dbl>, mosaik23 <dbl>,
#   mosaik24 <dbl>, mosaik25 <dbl>, mosaik26 <dbl>, mosaik27 <dbl>,
```

We have a problem

▼ Code

```
# this is how many subarea observations we have
nrow(surveycollapsed)
```

```
[1] 107
```

▼ Code

```
# this is how many predictors we have
ncol(predictors)
```

```
[1] 501
```

- What's the problem?

- It's actually impossible to estimate a model with more predictors than observations!

Another problem: overfitting

- There's another problem, too

- If we have too many predictors, we can "overfit" the model
    - This means the model is too complex
    - It fits the data we have *too* well
    - This means it doesn't generalize well to new data

- So we need to select the best predictors
    - What does "best" mean here?

Generalizing out-of-sample

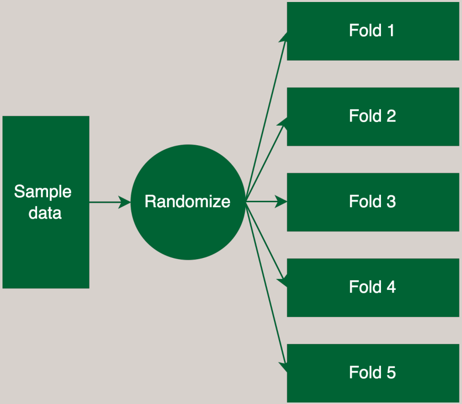
- We want to know what best predicts OUT of sample

- So we are going to set up our data to allow this:
  - We will split the data into X parts
  - A common number for X is 10, but let’s do 5

Cross validation

Sample data

Cross validation



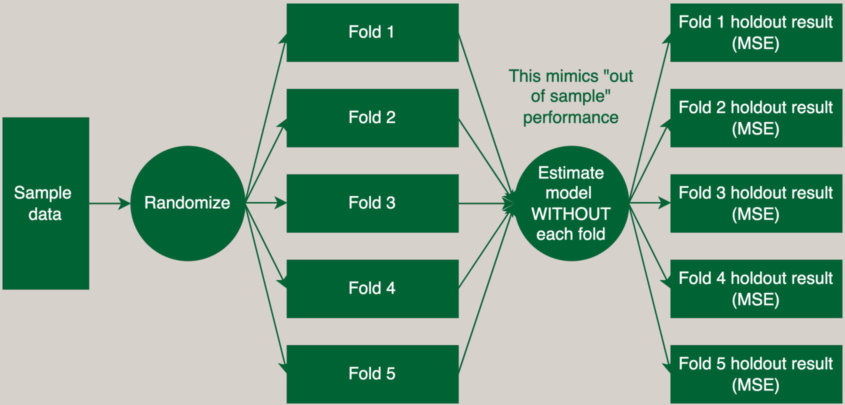
Cross validation - random folds

▾ Code

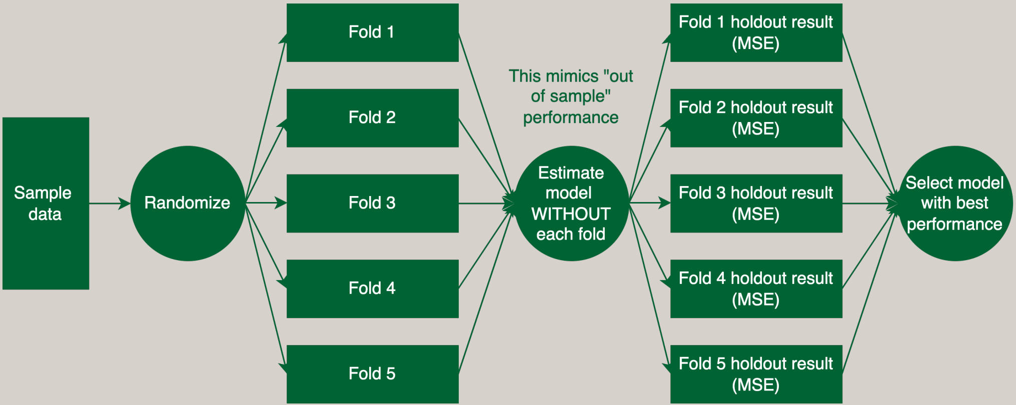
```r
surveycollapsed$fold <- sample(1:5, nrow(surveycollapsed), replace = TRUE)
head(surveycollapsed)
```

```
# A tibble: 6 × 5
   EA_CODE   poor total_weights total_obs  fold
     <dbl>  <dbl>         <dbl>     <dbl> <int>
1 10101006 0.230          5690.        16     5
2 10101011 0.444          7614.        16     4
3 10101027 0.0947         9441.        16     1
4 10101033 0.376          7486.        16     3
5 10101039 0.600          9147.        16     1
6 10101054 0.497          5351.        16     1
```

Cross validation



Cross validation



But what “models” are we going to fit?

- What are the models we are going to fit?
  - We want a way to select the best predictors
  - This will reduce the number of predictors and prevent overfitting (we hope)

- We are going to use a method called LASSO (or lasso)
  - It’s an acronym: **L**east **A**bsolute **S**hrinkage and **S**election **O**perator
  - No details, but it’s a way to select the best predictors
    - It “penalizes” the coefficients of the predictors
  - `R` package `glmnet` does this for us

The setup - with a transformed outcome

▼ Code

```r
library(glmnet)
set.seed(398465) # this is a random process, so we want to set the seed!

# we need to set up the data (combining the predictors and the outcome)
data <- surveycollapsed |>
  left_join(predictors, by = "EA_CODE")

# cv.glmnet will set up everything for us
lasso <- cv.glmnet(
  y = asin(sqrt(data$poor)), # the outcome
  x = data |> dplyr::select(starts_with("mosaik")) |> as.matrix(), # the predictors (as.matrix() is required)
  weights = data$total_weights, # the weights (sample weights)
  nfolds = 5) # number of folds (10 is the default)
lasso
```

```
Call:  cv.glmnet(x = as.matrix(dplyr::select(data, starts_with("mosaik"))),      y = asin(sqrt(data$poor)), weights = data$total_weights,      nfolds = 5) 

Measure: Mean-Squared Error 
```

```
        Lambda Index Measure       SE Nonzero
    min 0.02030    26 0.04227 0.006409       6
    1se 0.06493     1 0.04418 0.005811       0
```

What have we done?

▼ Code

```
lasso
```

```
Call:  cv.glmnet(x = as.matrix(dplyr::select(data, starts_with("mosaik"))),      y = asin(sqrt(data$poor)), weights = data$total_weights, 
nfolds = 5) 

Measure: Mean-Squared Error 

     Lambda Index Measure       SE Nonzero
min 0.02030    26 0.04227 0.006409       6
1se 0.06493     1 0.04418 0.005811       0
```

- What are the different “models”?
  - Different values of lambda
  - In this case, the “best” lambda is 0.02030
  - Note that some people prefer to use the `1se` value (it is more conservative). No details today.

Different values of lambda: different predictors!

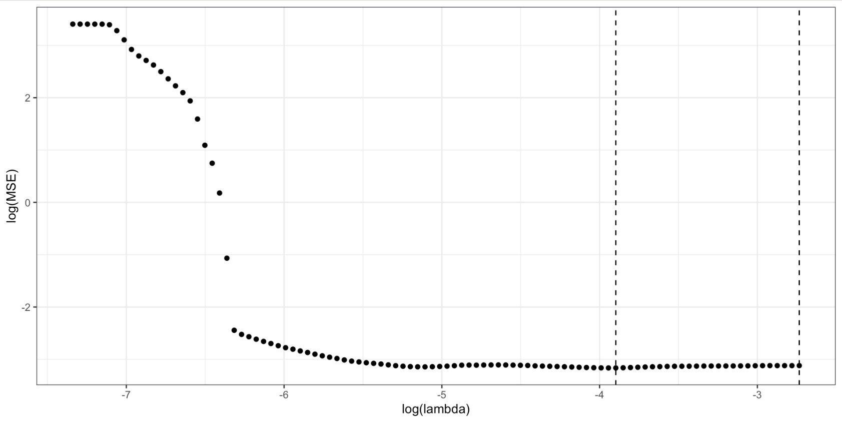
▼ Code

```
lasso
```

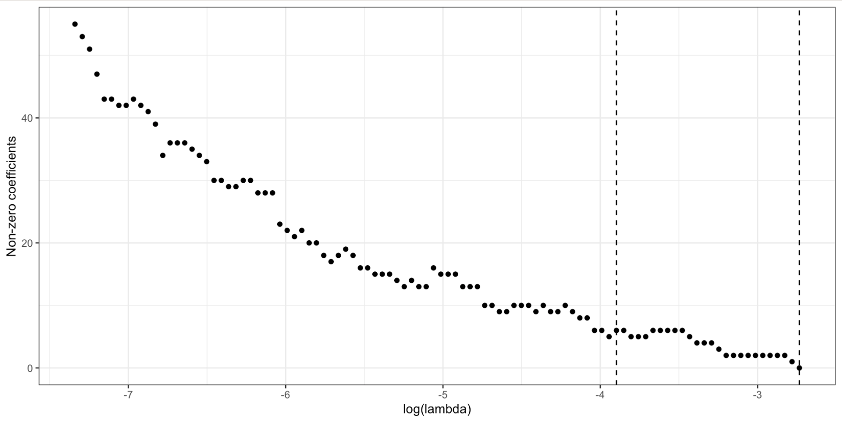
```
Call:  cv.glmnet(x = as.matrix(dplyr::select(data, starts_with("mosaik"))),      y = asin(sqrt(data$poor)), weights = data$total_weights,
nfolds = 5)

Measure: Mean-Squared Error

     Lambda Index Measure       SE Nonzero
min 0.02030    26 0.04227 0.006409       6
1se 0.06493     1 0.04418 0.005811       0
```

- At the “optimal” lambda, we have 6 predictors (non-zero coefficients)

Choosing based on mean-squared error (MSE)



Non-zero coefficients



Non-zero coefficients

▼ Code

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

```
501 x 1 sparse Matrix of class "dgCMatrix"
                       s1
(Intercept)   0.568658061
mosaik1       .
mosaik2       .
mosaik3       .
mosaik4       .
mosaik5       .
mosaik6       .
mosaik7       .
mosaik8       .
mosaik9       .
mosaik10      .
mosaik11      .
mosaik12      .
mosaik13      .
mosaik14      .
mosaik15      .
```

What we want: the non-zero variable names!

- Getting the names of the variables is more complicated than it should be

▼ Code

```
# first, turn the coefs into a data.frame
coefs <- coef(lasso, s = "lambda.min") |>
  as.matrix() |>
  as.data.frame()
coefs
```

```
                     s1
(Intercept)   0.568658061
mosaik1       0.000000000
mosaik2       0.000000000
mosaik3       0.000000000
mosaik4       0.000000000
mosaik5       0.000000000
mosaik6       0.000000000
mosaik7       0.000000000
mosaik8       0.000000000
mosaik9       0.000000000
mosaik10      0.000000000
mosaik11      0.000000000
```

```
mosaik12      0.000000000
mosaik13      0.000000000
mosaik14      0.000000000
mosaik15      0.000000000
mosaik16      0.000000000
```

What we want: the non-zero variable names!

- Getting the names of the variables is more complicated than it should be

▼ Code

```
# Now, create variable that is the name of the rows
coefs$variable <- rownames(coefs)
head(coefs)
```

```
                   s1    variable
(Intercept) 0.5686581 (Intercept)
mosaik1     0.0000000     mosaik1
mosaik2     0.0000000     mosaik2
mosaik3     0.0000000     mosaik3
mosaik4     0.0000000     mosaik4
mosaik5     0.0000000     mosaik5
```

▼ Code

```
# non-zero rows
coefs <- coefs[coefs$s1!=0,]
# finally, the names of the variables
coefs$variable
```

```
[1] "(Intercept)" "mosaik39"    "mosaik234"   "mosaik277"   "mosaik280"
[6] "mosaik396"   "mosaik459"
```

One more step: remove the Intercept!

- We don’t want the name of the intercept
  - All of the packages we use will add that automatically

▼ Code

```
allvariables <- coefs$variable[-1]
allvariables
```

```
[1] "mosaik39"  "mosaik234" "mosaik277" "mosaik280" "mosaik396" "mosaik459"
```

How do we use this with ebp?

- In our SAE model, we need a `formula`
- How do we turn this into a formula?
  - We need to add the outcome variable (poor) and combine the predictors with +

▼ Code

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

```
poor ~ mosaik39 + mosaik234 + mosaik277 + mosaik280 + mosaik396 +
    mosaik459
```

`povmap` and EBP

- We are going to use an `EBP` (empirical best predictor) model
- We will use the package `povmap`:

▼ Code

```
install_github("SSA-Statistical-Team-Projects/povmap", ref = "david3")
```

Finally: estimating the model

▼ Code

```r
library(povmap) # I like to use povmap instead of emdi (personal preference)
# get "area" variable
predictors$TA_CODE <- substr(predictors$EA_CODE, 1, 5)
data$TA_CODE <- substr(data$EA_CODE, 1, 5)
ebp <- ebp(fixed = ebpformula, # the formula
  pop_data = predictors, # the population data
  pop_domains = "TA_CODE", # the domain (area) name in the population data
  smp_data = data, # the sample data
  smp_domains = "TA_CODE", # the domain (area) name in the sample data
  transformation = "arcsin", # I'm going to use the arcsin transformation
  weights = "total_weights", # sample weights
  weights_type = "nlme", # weights type
  MSE = TRUE, # variance? yes please
  L = 0) # this is a new thing in povmap: "analytical" variance estimates. much faster!
```

```
Time difference of 1.14 secs
```

▼ Code

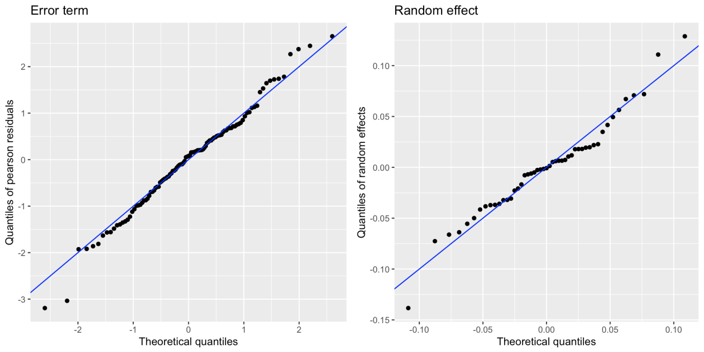
```r
head(ebp$ind)
```

	Domain	Mean	Head_Count	Poverty_Gap
1	10101	0.3669687	0.1446667	NA
2	10102	0.3526612	0.1677491	NA
3	10103	0.2829257	0.3078810	NA
4	10104	0.3255265	0.2415296	NA
5	10105	0.3564514	0.1773102	NA
6	10106	0.3747787	0.1587885	NA

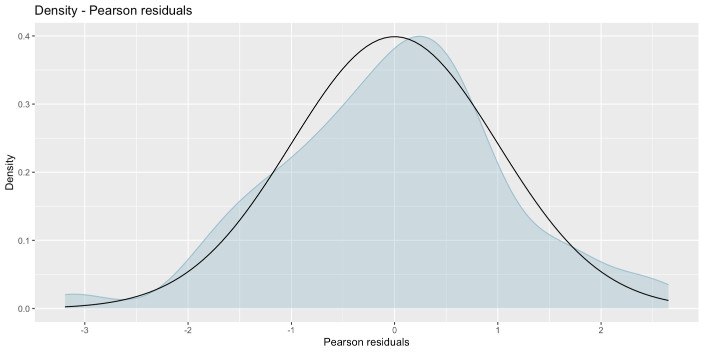
Some results

▼ Code

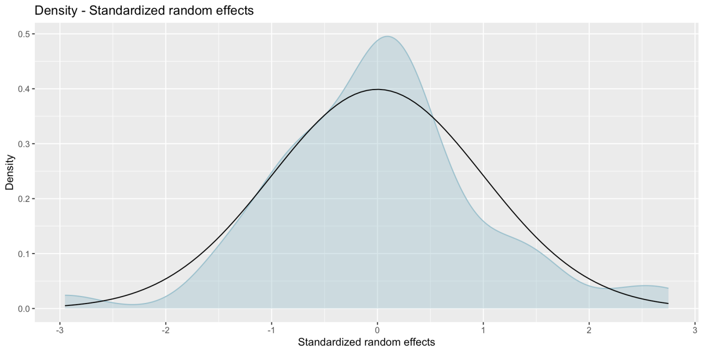
```
plot(ebp)
```




```
Press [enter] to continue
```




```
1 Press [enter] to continue
```




```
1 Press [enter] to continue
```

Some results

▼ Code

```r
summary(ebp)
```

```
Empirical Best Prediction

Call:
 ebp(fixed = poor ~ mosaik39 + mosaik234 + mosaik277 + mosaik280 + 
    mosaik396 + mosaik459, pop_data = predictors, pop_domains = "TA_CODE", 
    smp_data = data, smp_domains = "TA_CODE", L = 0, transformation = "arcsin", 
    MSE = TRUE, weights = "total_weights", weights_type = "nlme")

Out-of-sample domains:  27
In-sample domains:  49
Out-of-sample subdomains:  0
In-sample subdomains:  0

Sample sizes:
Units in sample:  107
Units in population:  2911
                   Min. 1st Qu. Median      Mean 3rd Qu. Max.
Sample_domains        1     1.0      2  2.183673       3    7
Population_domains    1     5.5     18 38.302632      44  300
```

Some results

▼ Code

```r
estimators(ebp, "Mean", MSE = TRUE, CV = TRUE)
```

```
Indicator/s: Mean
   Domain      Mean    Mean_MSE   Mean_CV
1   10101 0.3669687 0.002831292 0.1449984
2   10102 0.3526612 0.004281904 0.1855499
3   10103 0.2829257 0.005202843 0.2549458
4   10104 0.3255265 0.010839984 0.3198365
5   10105 0.3564514 0.005060076 0.1995621
6   10106 0.3747787 0.004249448 0.1739367
7   10107 0.3518191 0.006747908 0.2334883
8   10108 0.3545512 0.011015621 0.2960230
9   10109 0.3971006 0.005494159 0.1866595
10  10110 0.2786403 0.029178299 0.6130360
11  10120 0.3480978 0.010632100 0.2962158
12  10201 0.3514189 0.003631778 0.1714883
13  10202 0.3344865 0.005765226 0.2270019
14  10203 0.4354745 0.003538805 0.1366047
15  10204 0.3489391 0.003373777 0.1664595
16  10205 0.3882951 0.004288408 0.1686499
17  10206 0.2853691 0.028118716 0.5876122
```

Using `write.excel`

▼ Code

```r
write.excel(ebp, file = "results.xlsx", indicator = "Mean", MSE = TRUE, CV = TRUE)
```