sf:
SPATIAL DATA ... SIMPLIFIED.
ATTRIBUTES + GEOMETRIES
GLUE
tape
Sticky geometries: for people who love their maps and sanity.
Horst '18

(Illustration (c) 2018 by Allison Horst)

The “sf” Package

What is the “sf” package

“sf” stands for Simple Features - a set of standards that specify a common storage and access model of geographic features e.g., point, line, polygon

It represents natively in R all 17 simple feature types for all dimensions (XY, XYZ, XYM, XYZM)

Other related packages: dplyr, raster, terra, sp

It is a package for handling and processing spatial data

It is good for processing shapefiles in particular

Why I use it

Like most of us, I have had to match multiple data sources to one common spatial scale for covariate selection and model fitting.

Most of the time this involves matching points (my data) to shapefiles (my spatial scale want to use).

“sf” helps you to match different data streams and combinations of points and polygons.

My concerns of the package use

The package is excellent, its functionality is amazing, I recommend checking out it’s documentation for just how much it can do:

https://cran.r-project.org/web/packages/sf/sf.pdf

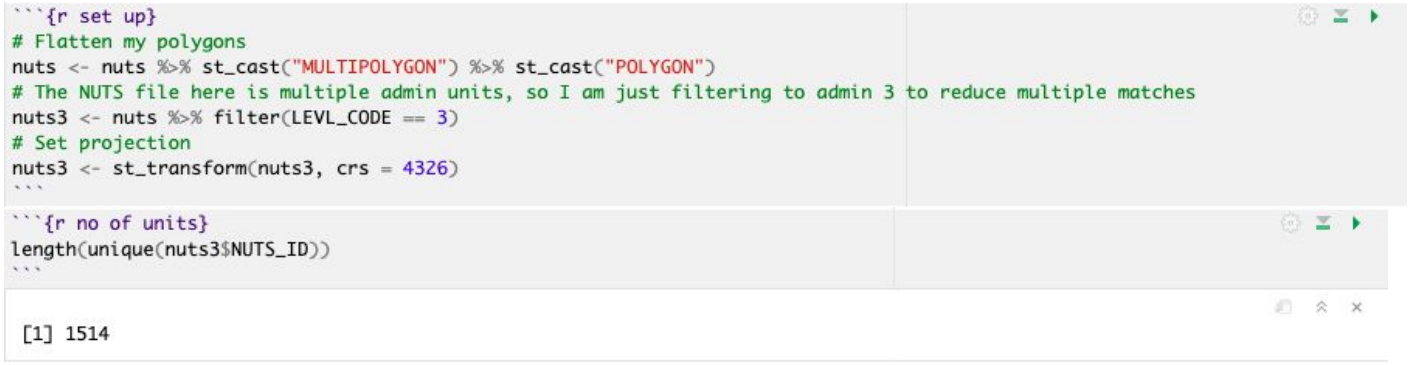
It is very well documented and used, and my experience is that people just blindly trust it, because it's the used packaged.

But it does make mistakes, nearly every time

Concerningly, if you have multiple millions of points, you can't check them all, and I think there must be a lot of epi studies which are wrong because people aren't checking.

Setting up my points and shapes

Here and in the markdown I am using an example of matching a NUTS shapefile to the ERA5-Land grid centroid coordinates.



~~~
```{r set up}
# Flatten my polygons
nuts <- nuts %>% st_cast("MULTIPOLYGON") %>% st_cast("POLYGON")
# The NUTS file here is multiple admin units, so I am just filtering to admin 3 to reduce multiple matches
nuts3 <- nuts %>% filter(LEVL_CODE == 3)
# Set projection
nuts3 <- st_transform(nuts3, crs = 4326)
```
~~~

~~~
```{r no of units}
length(unique(nuts3$NUTS_ID))
```
~~~

```
[1] 1514
```

To avoid some issues:

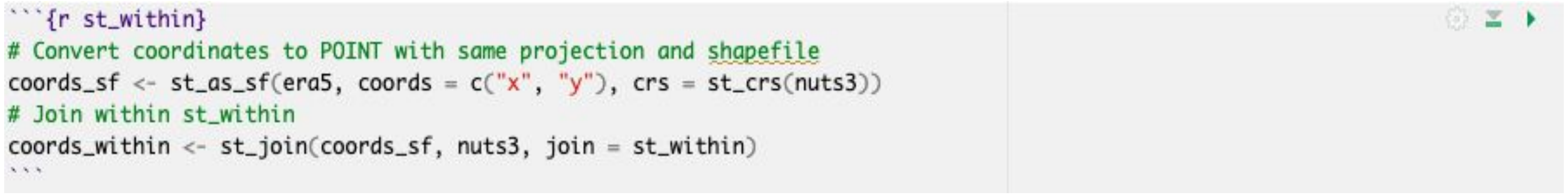
1. Go from MUTLIPOLYS to POLYS
2. Check projections
3. Check the number of spatial units you want to join to

Option 1: Using st_within

st_join is the sf package's join function and operates similar to the join functions in dplyr.

You can select your join within st_join, here I am using st_within

You need to convert your dataframe of x and y coordinates to an sf object, or POINT

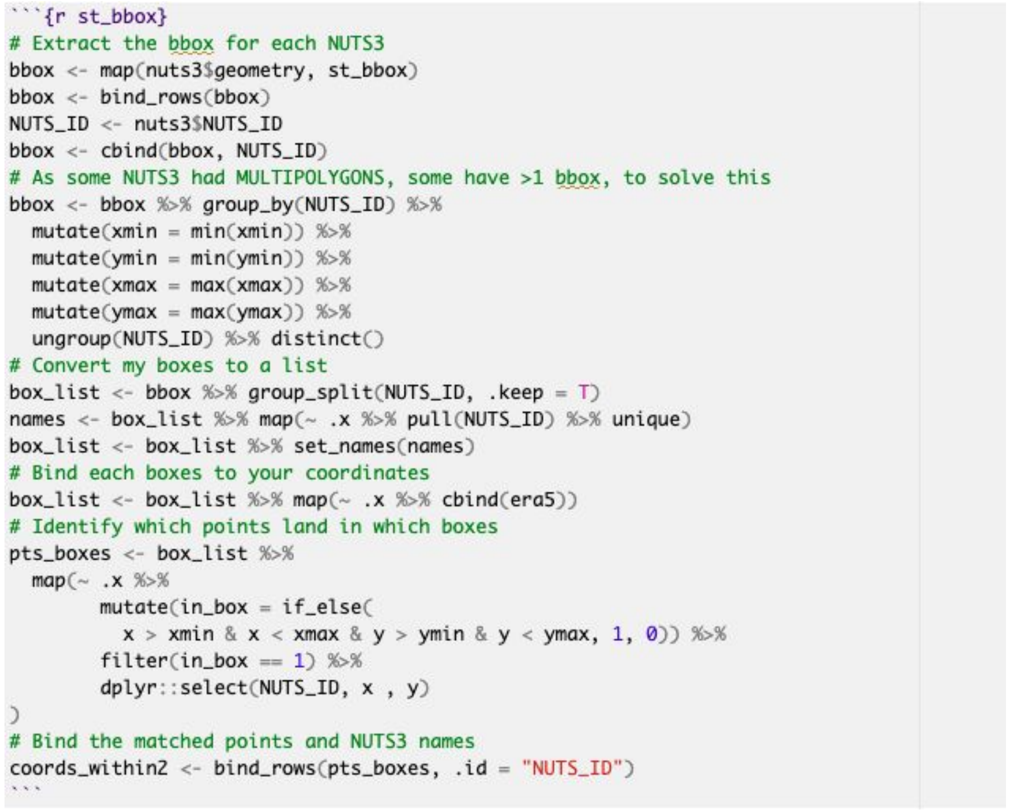


~~~
```{r st_within}
# Convert coordinates to POINT with same projection and shapefile
coords_sf <- st_as_sf(era5, coords = c("x", "y"), crs = st_crs(nuts3))
# Join within st_within
coords_within <- st_join(coords_sf, nuts3, join = st_within)
```
~~~

Options 2: Using st_box

Instead of using one of the join functions within sf, it can be easier to troubleshoot issues if you convert your polygons to bounding boxes and match them using dplyr instead.

Don't do this if you have Issue 2

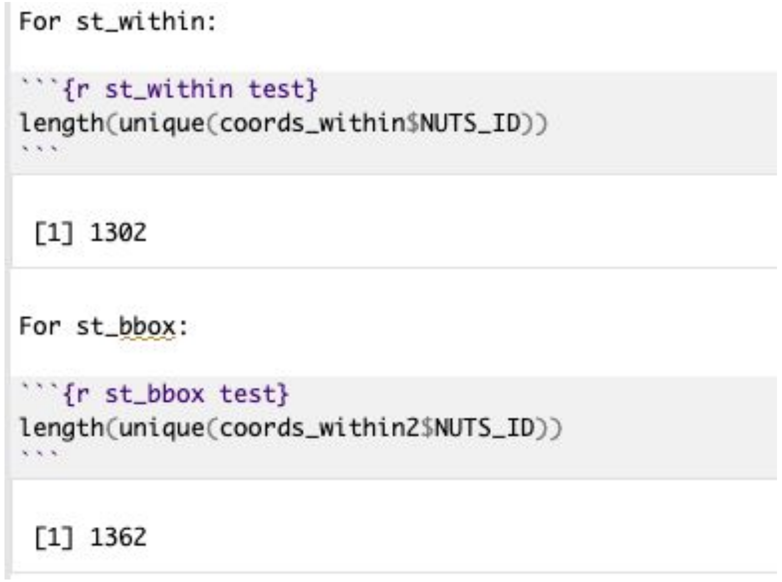


~~~
```{r st_bbox}
# Extract the bbox for each NUTS3
bbox <- map(nuts3$geometry, st_bbox)
bbox <- bind_rows(bbox)
NUTS_ID <- nuts3$NUTS_ID
bbox <- cbind(bbox, NUTS_ID)
# As some NUTS3 had MULTIPOLYGONS, some have >1 bbox, to solve this
bbox <- bbox %>% group_by(NUTS_ID) %>%
  mutate(xmin = min(xmin)) %>%
  mutate(ymin = min(ymin)) %>%
  mutate(xmax = max(xmax)) %>%
  mutate(ymax = max(ymax)) %>%
  ungroup(NUTS_ID) %>% distinct()
# Convert my boxes to a list
box_list <- bbox %>% group_split(NUTS_ID, .keep = T)
names <- box_list %>% map(~ .x %>% pull(NUTS_ID) %>% unique)
box_list <- box_list %>% set_names(names)
# Bind each boxes to your coordinates
box_list <- box_list %>% map(~ .x %>% cbind(era5))
# Identify which points land in which boxes
pts_boxes <- box_list %>%
  map(~ .x %>%
        mutate(in_box = if_else(
          x > xmin & x < xmax & y > ymin & y < ymax, 1, 0)) %>%
        filter(in_box == 1) %>%
        dplyr::select(NUTS_ID, x , y)
)
# Bind the matched points and NUTS3 names
coords_within2 <- bind_rows(pts_boxes, .id = "NUTS_ID")
```
~~~

Testing st_within vs st_bbox

There should be 1,514 admin 3 units

To explore these I usually plot those missing, extract the area, and the size of the bounding box.



For st_within:

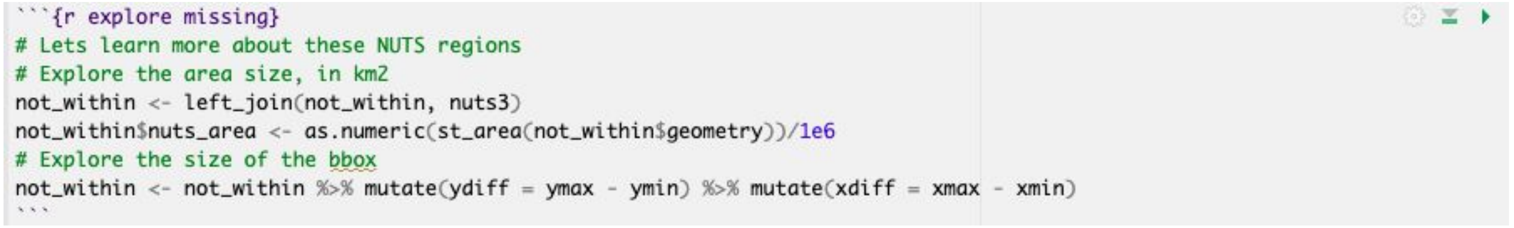
~~~
```{r st_within test}
length(unique(coords_within$NUTS_ID))
```
~~~

```
[1] 1302
```

For st_bbox:

~~~
```{r st_bbox test}
length(unique(coords_within2$NUTS_ID))
```
~~~

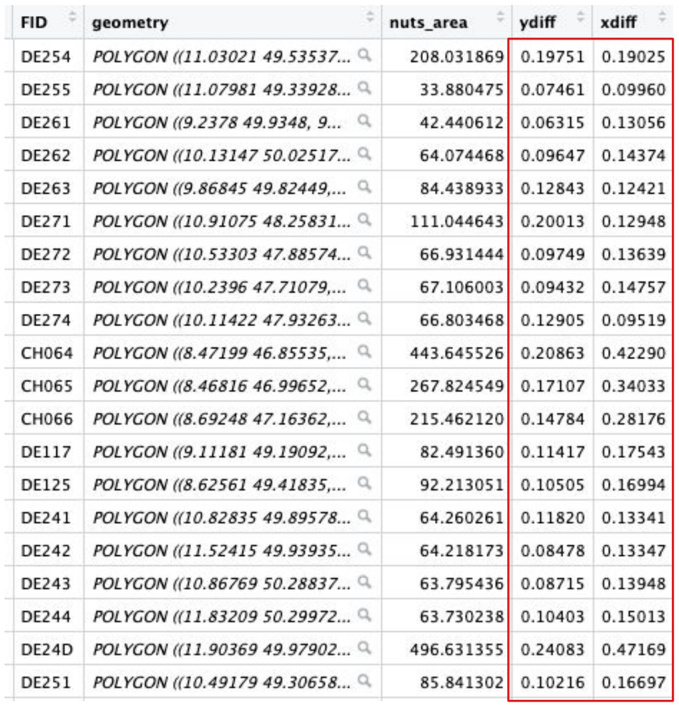
```
[1] 1362
```



~~~
```{r explore missing}
# Lets learn more about these NUTS regions
# Explore the area size, in km2
not_within <- left_join(not_within, nuts3)
not_within$nuts_area <- as.numeric(st_area(not_within$geometry))/1e6
# Explore the size of the bbox
not_within <- not_within %>% mutate(ydiff = ymax - ymin) %>% mutate(xdiff = xmax - xmin)
```
~~~

Here, those which did not match were for two reasons:

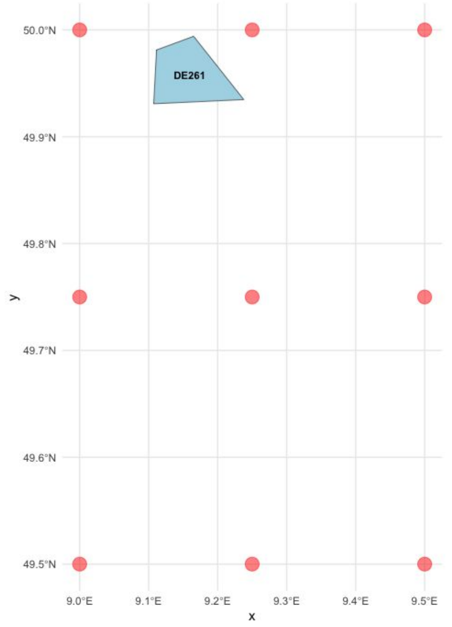
1. The bounding box was smaller than 0.25
2. They were European territories not in Europe such as Guadeloupe and La Réunion



FID	geometry	nuts_area	ydiff	xdiff
DE254	POLYGON ((11.03021 49.53537...	208.031869	0.19751	0.19025
DE255	POLYGON ((11.07981 49.33928...	33.880475	0.07461	0.09960
DE261	POLYGON ((9.2378 49.9348, 9...	42.440612	0.06315	0.13056
DE262	POLYGON ((10.13147 50.02517...	64.074468	0.09647	0.14374
DE263	POLYGON ((9.86845 49.82449,...	84.438933	0.12843	0.12421
DE271	POLYGON ((10.91075 48.25831...	111.044643	0.20013	0.12948
DE272	POLYGON ((10.53303 47.88574...	66.931444	0.09749	0.13639
DE273	POLYGON ((10.2396 47.71079,...	67.106003	0.09432	0.14757
DE274	POLYGON ((10.11422 47.93263...	66.803468	0.12905	0.09519
CH064	POLYGON ((8.47199 46.85535,...	443.645526	0.20863	0.42290
CH065	POLYGON ((8.46816 46.99652,...	267.824549	0.17107	0.34033
CH066	POLYGON ((8.69248 47.16362,...	215.462120	0.14784	0.28176
DE117	POLYGON ((9.11181 49.19092,...	82.491360	0.11417	0.17543
DE125	POLYGON ((8.62561 49.41835,...	92.213051	0.10505	0.16994
DE241	POLYGON ((10.82835 49.89578...	64.260261	0.11820	0.13341
DE242	POLYGON ((11.52415 49.93935...	64.218173	0.08478	0.13347
DE243	POLYGON ((10.86769 50.28837...	63.795436	0.08715	0.13948
DE244	POLYGON ((11.83209 50.29972...	63.730238	0.10403	0.15013
DE24D	POLYGON ((11.90369 49.97902...	496.631355	0.24083	0.47169
DE251	POLYGON ((10.49179 49.30658...	85.841302	0.10216	0.16697

Issue 1: Small admin units

If your admin units are smaller than your grid where your centroids are taken from, you may struggle to match some points in any of the polygons



Here is an example of a NUTS3 region in Germany (DE261), along with the ERA5 centroids.

As you can see, the polygon is smaller than the grid size, and therefore, you don't get any matches.

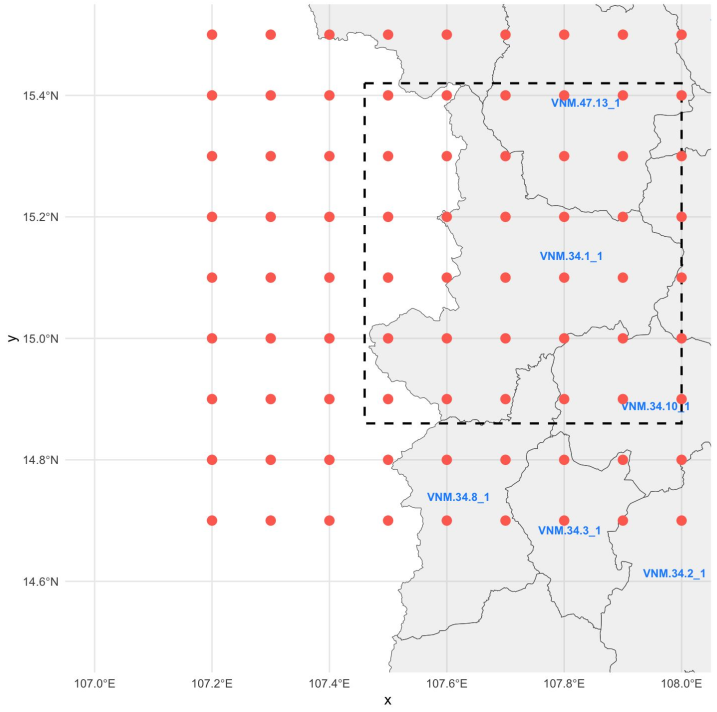
Issue 2: Funky shapes

If your shapefile is an odd shape, it struggles to match points and polygons

Using an example from Vietnam

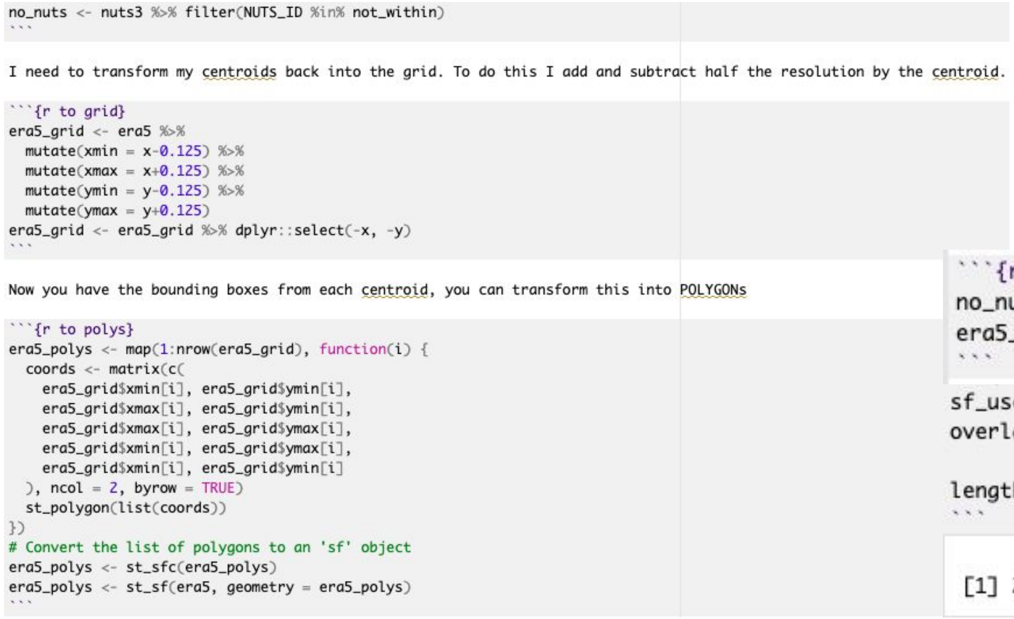
The plots is VNM.34.1_1, with its bounding box, a 0.1x0.1 centroid grid, and neighbouring admin units

Some of the points, particularly those near the border, can be picked up as part of VNM.34.1_1 because they are in the bbox



Solution: Using st_intersect

To solve these issues, I match in two stages. I first go for st_within, and then I use st_intersection. There were 213 missing NUTS after st_within



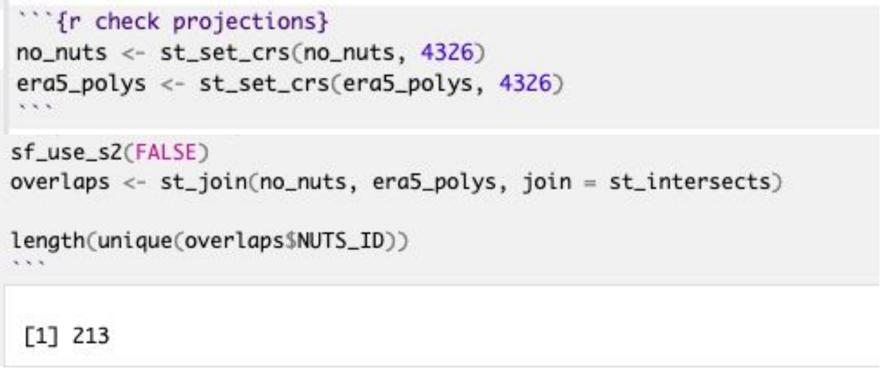
```
no_nuts <- nuts3 %>% filter(NUTS_ID %in% not_within)
```
```

I need to transform my centroids back into the grid. To do this I add and subtract half the resolution by the centroid.

```
```{r to grid}
era5_grid <- era5 %>%
  mutate(xmin = x-0.125) %>%
  mutate(xmax = x+0.125) %>%
  mutate(ymin = y-0.125) %>%
  mutate(ymax = y+0.125)
era5_grid <- era5_grid %>% dplyr::select(-x, -y)
```
```

Now you have the bounding boxes from each centroid, you can transform this into POLYGONs

```
```{r to polys}
era5_polys <- map(1:nrow(era5_grid), function(i) {
  coords <- matrix(c(
    era5_grid$xmin[i], era5_grid$ymin[i],
    era5_grid$xmax[i], era5_grid$ymin[i],
    era5_grid$xmax[i], era5_grid$ymax[i],
    era5_grid$xmin[i], era5_grid$ymax[i],
    era5_grid$xmin[i], era5_grid$ymin[i]
  ), ncol = 2, byrow = TRUE)
  st_polygon(list(coords))
})
# Convert the list of polygons to an 'sf' object
era5_polys <- st_sfc(era5_polys)
era5_polys <- st_sf(era5, geometry = era5_polys)
```
```



```
```{r check projections}
no_nuts <- st_set_crs(no_nuts, 4326)
era5_polys <- st_set_crs(era5_polys, 4326)
```
```

```
sf_use_s2(FALSE)
overlaps <- st_join(no_nuts, era5_polys, join = st_intersects)

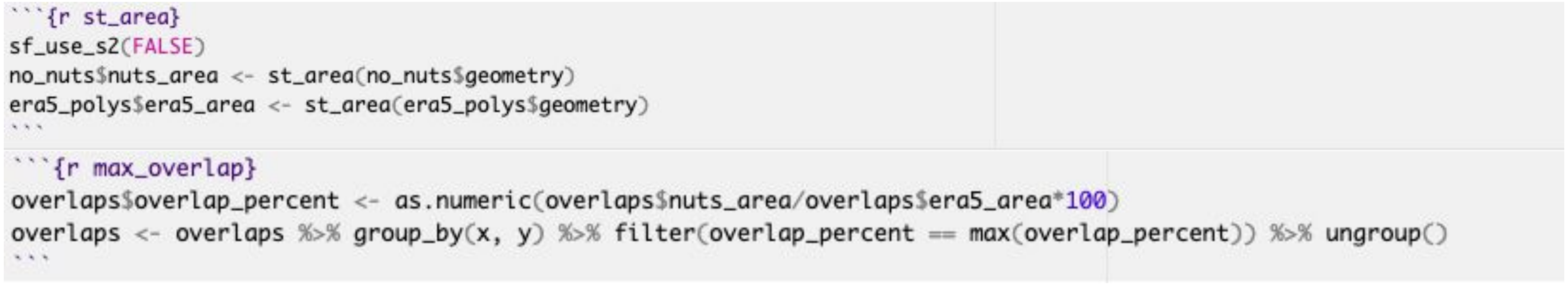
length(unique(overlaps$NUTS_ID))
```
```

```
[1] 213
```

Extra Step: st_area and max overlap

For some larger grid cells that you want to do this with, you might get a lot of overlap with multiple places.

So you can use a max overlap function to filter and reduce repeats



~~~
```{r st_area}
sf_use_s2(FALSE)
no_nuts$nuts_area <- st_area(no_nuts$geometry)
era5_polys$era5_area <- st_area(era5_polys$geometry)
```
~~~

~~~
```{r max_overlap}
overlaps$overlap_percent <- as.numeric(overlaps$nuts_area/overlaps$era5_area*100)
overlaps <- overlaps %>% group_by(x, y) %>% filter(overlap_percent == max(overlap_percent)) %>% ungroup()
```
~~~

Thanks!

I have a cheatsheet provided by sf

I also have an R Markdown if people would find the code snippets useful