Get Data

Contents

1 Overview

This notebook retrieves input data and converts it into gridded data about population density. This notebook should retrieve and clean data so that we can pass that to another notebook to do statistics.

The two sections here regrid in two different ways. The first section uses BIMEP GPS data, so we know where each house is. It re-grids the GPS data to each gridded dataset with which we will compare. You'll see that this matters for the 100m datasets but not for the 1km datasets. The second section uses the BIMEP area and section grids. It realigns HRSL, LandScan, and WorldPop to those grids.

1.1 Concerns

I have an initial concern about population counts near ocean borders. The population in the pixel will be entirely within the land border, and this is handled differently by different datasets. This could lead to different border population densities, and most of the population is near the water. We will see how much we can address this as we go.

We could also rescale populations according to the growth rate for Bioko. A commonly-used adjustment is to multiply values by $\exp(rt)$, where r is the yearly growth rate. This wouldn't affect the structure of zeroes, though.

1.2 Conventions

Pixels in the water should be marked as NA and excluded from calculations later.

All work will use pixels in latitude and longitude (lat-long). This is also known as an unprojected space. We'll need to transform the BIMEP projection to lat-long, but the rest are already in lat-long, and the BIMEP coarse and fine grids are rectilinear in lat-long.

We will ensure each dataset has NA values where it's ocean and 0 values inland of the ocean.

2 Download Datasets

Specify a directory into which to put the data. The `inst/extdata` subdirectory is a popular place.

```r
data_dir <- params$data_directory
if (!dir.exists(data_dir)) {
  dir.create(data_dir, recursive = TRUE)
}
```

The data will be stored in /home/adolgert/dev/popbioko/inst/extdata.

The Bioko shapefile has one geometry, a polygon outline of Bioko. We will use it to project population from rasters.

```r
latitude_longitude_projection <- "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"
bioko_sf <- sf::st_read(fs::path(data_dir, "source", "bioko.shp"))
#> Reading layer `bioko' from data source `/home/adolgert/dev/popbioko/inst/extdata/source
#> Simple feature collection with 1 feature and 1 field
#> geometry type:  POLYGON
#> dimension:      XY
#> bbox:           xmin: 8.414334 ymin: 3.208967 xmax: 8.938047 ymax: 3.788305
#> epsg (SRID):    4326
#> proj4string:    +proj=longlat +datum=WGS84 +no_defs
```

If you have an account at IHME, the following will copy files to your local drive, under the data directory. Look at the help for `data_configuration` to see about creating a config file for the download.

```r
download_worldpop(local_directory = data_dir)
download_bioko_grids(local_directory = data_dir)
download_hrsl_points(local_directory = data_dir)
```

3 Align to Grids from BIMEP GPS Data

This section uses the original BIMEP GPS data, so it's a longitude-latitude for each house an an integer count of the number of people. That means there is no section or area grid to consider. We align BIMEP to any location.

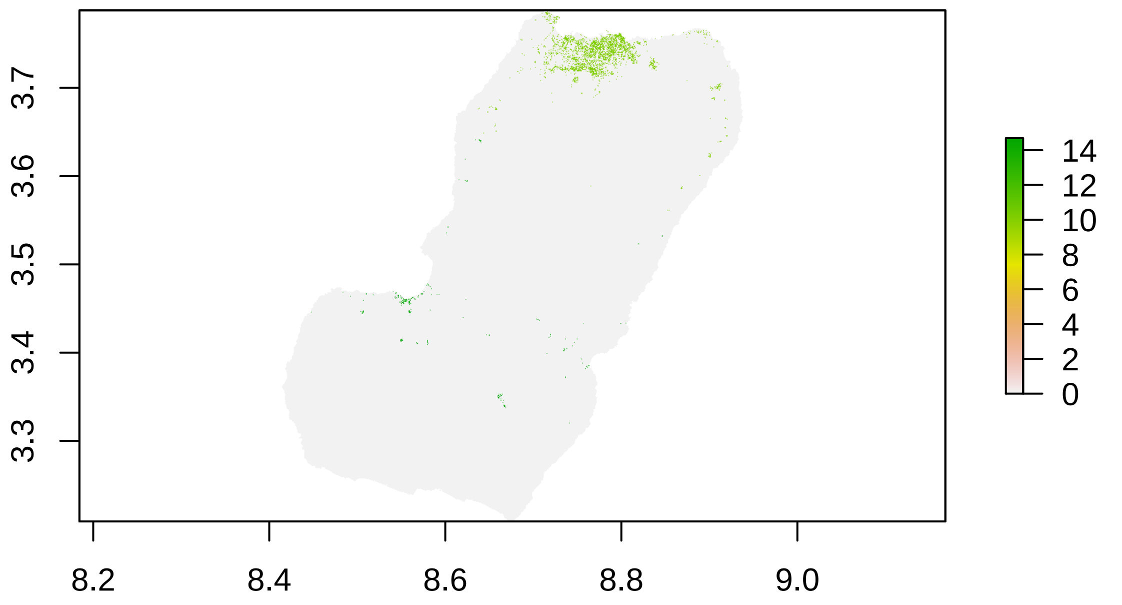
The output of this section is a set of GeoTIFFS in the `aligned` subdirectory of your data directory (likely `inst/extdata/aligned`). The names are of the form `<grid><resolution>_<source>.tif`. So the HRSL is on a 100m grid. It is saved as `HRSL100_HRSL.tif`. Bioko data on that grid is saved as `HRSL100_Bioko.tif`.

For each grid, the code to project the GPS data is the same.

3.1 HRSL

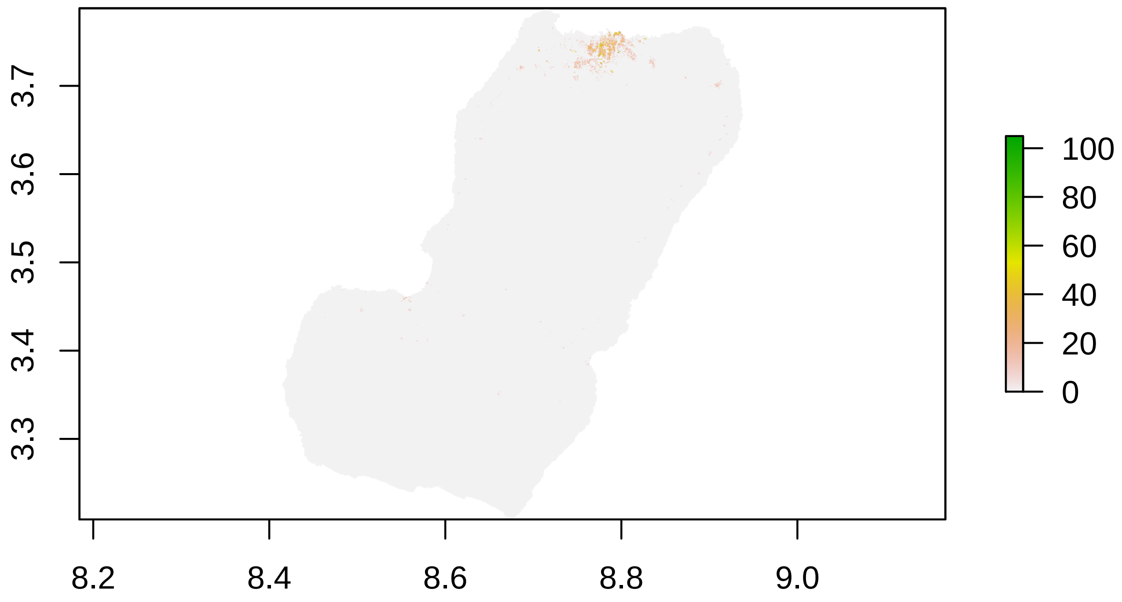
The HRSL marks where houses are with values and everywhere else with NA. The incoming data is for all of Equatorial Guinea, so we crop it. We need to have zeroes where there is land, so add the zeroes to the HRSL. Pixels with no land are marked NA. Otherwise, we don't modify values.

```r
hrsl_raster <- read_hrsl(local_directory = data_dir)
hrsl_raster_crop <- raster::crop(hrsl_raster, bioko_sf, snap = "out")
hrsl_zero_mask <- raster::rasterize(bioko_sf, hrsl_raster_crop, field = 0)
hrsl_raster_zero <- raster::cover(hrsl_raster_crop, hrsl_zero_mask)
plot(hrsl_raster_zero)
```



Now put Bioko data on that grid:

```r
bimep_on_hrsl <- bimep_on_grid(hrsl_raster_zero, bioko_sf, local_directory = data_dir)
plot(bimep_on_hrsl)
```



```r
popbioko::write_aligned_raster(
  hrsl_raster_zero,
  list(source = "HRSL", grid = "HRSL", resolution = 30),
  local_directory = data_dir
)
popbioko::write_aligned_raster(
  bimep_on_hrsl,
  list(source = "BIMEP", grid = "HRSL", resolution = 30),
  local_directory = data_dir
)
```

```
aggregate_and_write <- function(raster, name) {
  pixel_area <- square_meters_per_pixel.raster(raster)
  desired_area <- 10^6
  fact <- round(sqrt(desired_area / pixel_area))
  cat(paste("aggregating factor for", name, "is", fact, "with pixel side", sqrt(pixel_area
  aggregated <- raster::aggregate(raster, fact = fact, fun = sum, expand = TRUE)
  agg_pixel_side <- sqrt(square_meters_per_pixel.raster(raster))
  popbioko::write_aligned_raster(
    aggregated,
    list(source = name$source, grid = name$grid, resolution = agg_pixel_side),
    local_directory = data_dir)
}
aggregate_and_write(
  hrsl_raster_zero,
  list(source = "HRSL", grid = "HRSL", resolution = 30)
)
#> aggregating factor for HRSL is 32 with pixel side 30.789804349904
#>  aggregating factor for HRSL is 32 with pixel side 30.789804349904
#>  aggregating factor for 30 is 32 with pixel side 30.789804349904
aggregate_and_write(
  bimep_on_hrsl,
  list(source = "BIMEP", grid = "HRSL", resolution = 30)
)
#> aggregating factor for BIMEP is 32 with pixel side 30.789804349904
#>  aggregating factor for HRSL is 32 with pixel side 30.789804349904
#>  aggregating factor for 30 is 32 with pixel side 30.789804349904
```

3.2 LandScan

LandScan is 1 km data, so we don't aggregate it.

```
landscan <- read_landscan(local_directory = data_dir)
landscan <- raster::crop(landscan, bioko_sf, snap = "out")
bimep_on_landscan <- bimep_on_grid(
  landscan, bioko_sf, local_directory = data_dir)
popbioko::write_aligned_raster(
  landscan,
  list(source = "LandScan", grid = "LandScan", resolution = 1000),
  local_directory = data_dir
)
popbioko::write_aligned_raster(
  bimep_on_landscan,
  list(source = "BIMEP", grid = "LandScan", resolution = 1000),
  local_directory = data_dir
)
```

Actually, let's disaggregate it, to see what happens if we use it at 100m.

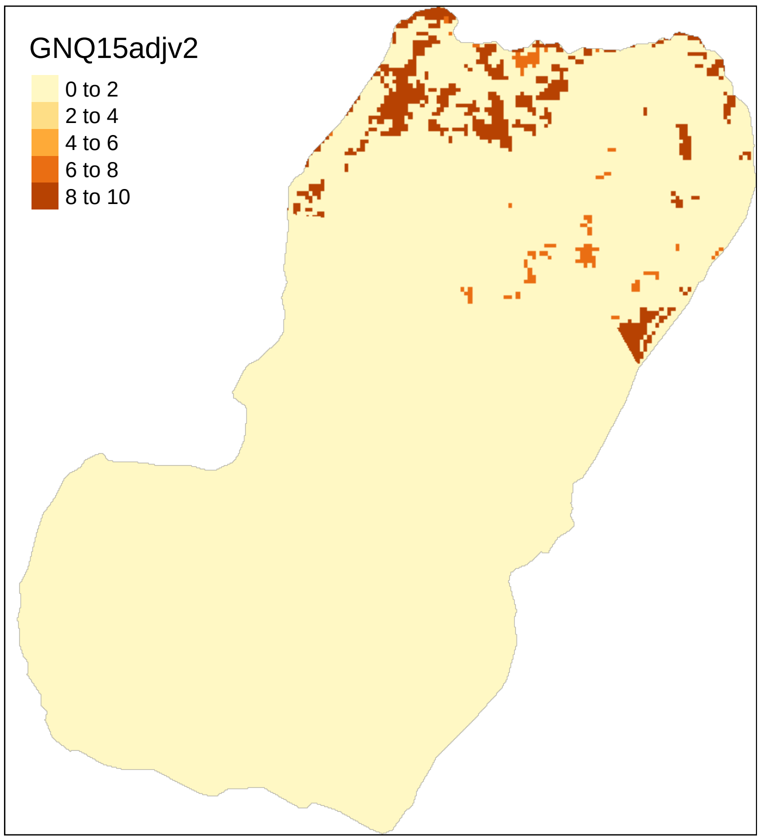
```
factor <- 10
landscan100 <- raster::disaggregate(landscan, fact = factor) / factor^2
bimep_on_landscan100 <- bimep_on_grid(
  landscan100, bioko_sf, local_directory = data_dir)
popbioko::write_aligned_raster(
  landscan100,
  list(source = "LandScan", grid = "LandScan", resolution = 100),
  local_directory = data_dir
)
popbioko::write_aligned_raster(
  bimep_on_landscan100,
  list(source = "BIMEP", grid = "LandScan", resolution = 100),
  local_directory = data_dir
)
```

3.3 WorldPop

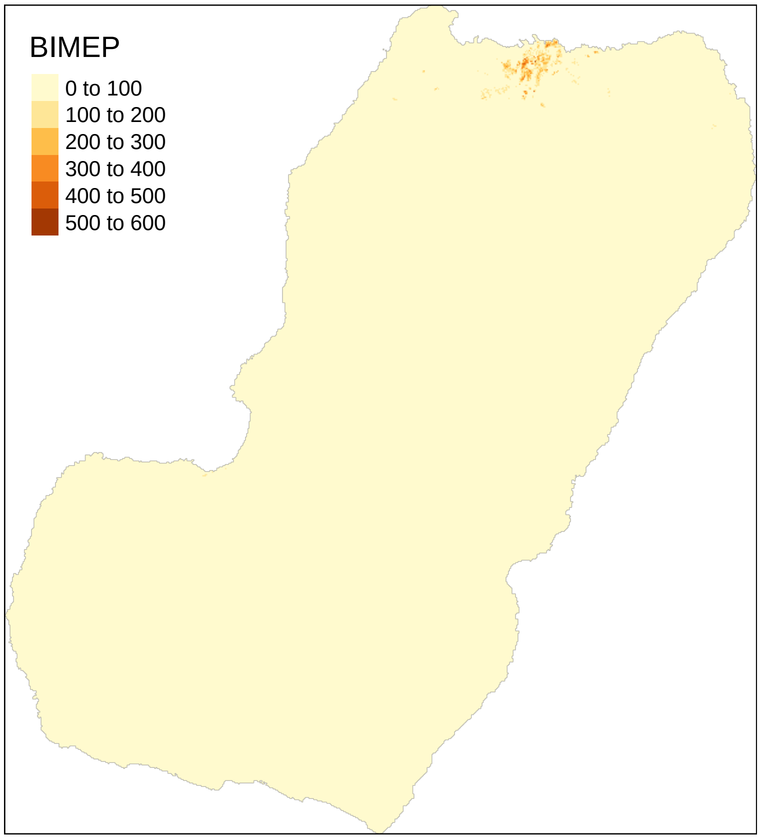
WorldPop is 100m, like the HRSL data, so we aggregate it.

```
wp_options <- list(raw = "GNQ15v2.tif", adjusted = "GNQ15adjv2.tif")
worldpop <- raster::raster(fs::path(
  data_dir, "Equatorial_Guinea_100m_Population", wp_options[["adjusted"]]))
worldpop <- raster::crop(worldpop, bioko_sf, snap = "out")
worldpop_id <- list(source = "WorldPop", grid = "WorldPop", resolution = 100)
bimep_id <- worldpop_id
bimep_id$source <- "BIMEP"
popbioko::write_aligned_raster(
  worldpop,
  worldpop_id,
  local_directory = data_dir
)
bimep_on_worldpop <- bimep_on_grid(
  worldpop, bioko_sf, local_directory = data_dir)
popbioko::write_aligned_raster(
  bimep_on_worldpop,
  bimep_id,
  local_directory = data_dir
)
aggregate_and_write(
  worldpop,
  worldpop_id
)
#> aggregating factor for WorldPop is 11 with pixel side 92.3657551178394
#>  aggregating factor for WorldPop is 11 with pixel side 92.3657551178394
#>  aggregating factor for 100 is 11 with pixel side 92.3657551178394
aggregate_and_write(
  bimep_on_worldpop,
  bimep_id
)
#> aggregating factor for BIMEP is 11 with pixel side 92.3657551178394
#>  aggregating factor for WorldPop is 11 with pixel side 92.3657551178394
#>  aggregating factor for 100 is 11 with pixel side 92.3657551178394
```

```
tm_shape(worldpop) + tm_raster()
```



```r
tm_shape(bimep_on_worldpop) + tm_raster()
```



4 Make other datasets align to BIMEP study grids

The BIMEP study uses two grids, at 100m and 1km resolution. This section reads population datasets and regrids them onto the BIMEP study datasets.

The output is all GeoTIFFs with names of the form data-on-grid. The data are BIMEP, WorldPop, HRSL, and LandScan. The grids are

- area, a 1km grid used by BIMEP.
- worldpop, a 100m grid used by WorldPop
- 1kworldpop, a 1km grid that aggregates and aligns with the WorldPop grid.
- landscan, the 1km LandScan grid
- hrsl, the 100m HRSL grid.

4.1 Load BIMEP Early

The BIMEP data will have a "pop" feature for the population in the pixel. Load it now so that we have its bounding box.

```
bimep_feature <- bimep_population_as_points(local_directory = data_dir)
names(bimep_feature)
#> [1] "pop"                              "st_as_sfc.projected_cell_center."
```

We see the population as `pop` and the other column, which is the geometry for each point.

4.2 Align HRSL and BIMEP to BIMEP 1k Grid

The Bioko grids define cells where BIMEP assigns residents of the island. One is a 1 km grid, the other a 100 m grid. If we read those grids, we can see that the input files are projected into UTM 32N, which makes all distances in meters. That's very convenient, but the original grids were defined as a regular grid in latitude and longitude.

```
grids <- read_bioko_grids(local_directory = data_dir)
#> Reading layer `secs' from data source `/home/adolgert/dev/popbioko/inst/extdata/Bioko_g
#> Simple feature collection with 197086 features and 16 fields
#> geometry type:  POLYGON
#> dimension:      XY
#> bbox:           xmin: 435458.1 ymin: 354672.8 xmax: 493205.8 ymax: 418789.2
#> epsg (SRID):    32632
#> proj4string:    +proj=utm +zone=32 +datum=WGS84 +units=m +no_defs
#> Reading layer `mapareas_grid' from data source `/home/adolgert/dev/popbioko/inst/extdat
#> Simple feature collection with 3894 features and 19 fields
#> geometry type:  POLYGON
#> dimension:      XY
#> bbox:           xmin: 435449.4 ymin: 353872.1 xmax: 494405 ymax: 419501.4
#> epsg (SRID):    32632
#> proj4string:    +proj=utm +zone=32 +datum=WGS84 +units=m +no_defs
grid_info <- parameters_of_km_grid(grids$coarse, unproject = TRUE)
```

You can see from the description above that the coarse and fine grids are shapefiles. Are these grids complete, or are they partial?

```
area_per_pixel <- function(grid) {
  bbox <- st_bbox(grid)
  (bbox["xmax"] - bbox["xmin"]) * (bbox["ymax"] - bbox["ymin"])
}
cat("area per pixel coarse", area_per_pixel(grids$coarse) / 3894, "\n")
#> area per pixel coarse 993635.7
cat("area per pixel fine", area_per_pixel(grids$fine) / 197086, "\n")
#> area per pixel fine 18786.61
```

We see that the area per pixel for the coarse is about 10^6 m^2, which means it's probably a complete grid. The fine grid looks far from complete because we expect 10^4 m^2 per pixel.

If we want to make such a grid, we could use

```
coarse_grid_copy <- make_fresh_grid(grid_info$bbox, grid_info$dimensions)
```

We can show the original and copy in unprojected space in order to see that they overlap.

```
grid_1km <- sf::st_transform(grids$coarse, crs = latitude_longitude_projection)
tm_shape(coarse_grid_copy) +
  tm_polygons(alpha = 0.1, border.col = "red") +
  tm_shape(grid_1km) +
  tm_polygons(alpha = 0.0, border.col = "blue")
```

What if we directly make a raster version of the 1km grid?

```
km_raster <- raster::raster(
  ncol = grid_info$dimensions[1],
  nrow = grid_info$dimensions[2],
  xmn = grid_info$bbox["xmin"],
  xmx = grid_info$bbox["xmax"],
  ymn = grid_info$bbox["ymin"],
```

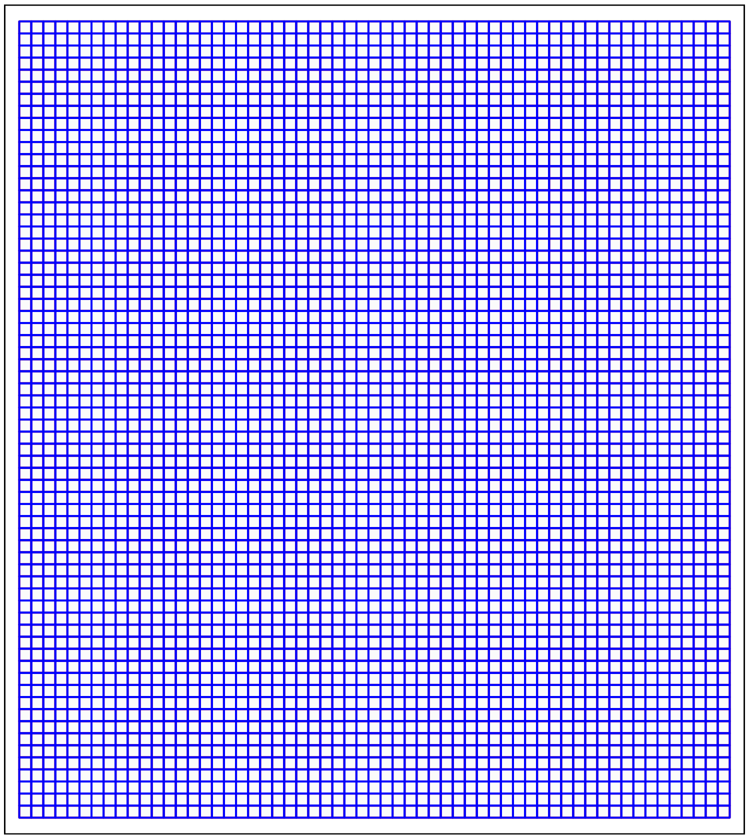
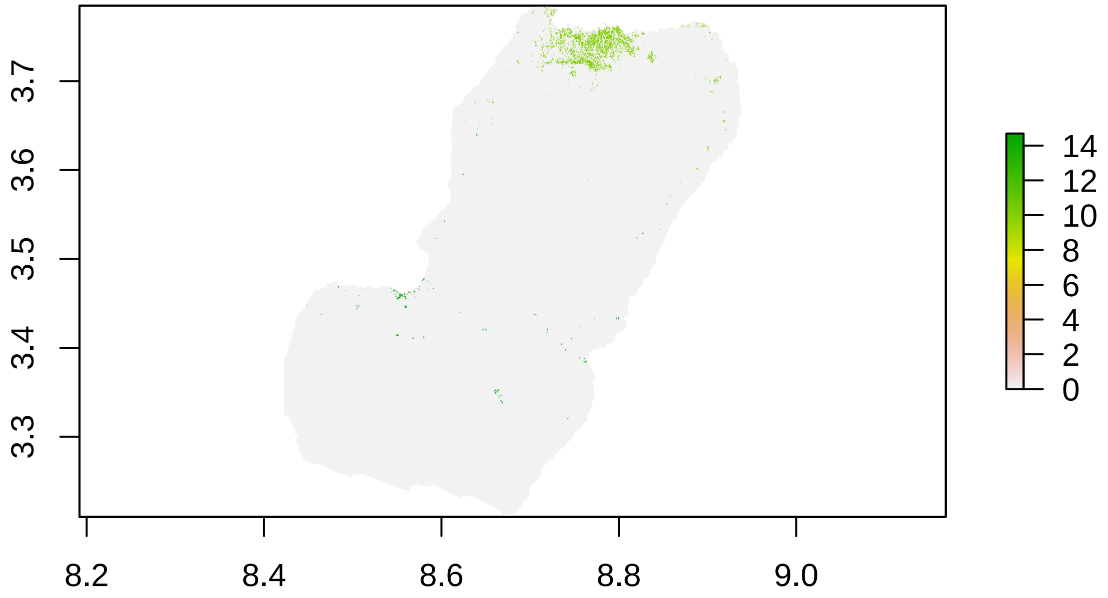


Figure 1: The original and our copy overlap.

```
  ymx = grid_info$bbox["ymax"]
  )
raster::projection(km_raster) <- latitude_longitude_projection
```

Let's get some points to project to that raster. The HRSL gives the location of a roof, or NA values. It doesn't say where there could be population but isn't. Other population maps assign a population of 0 to all non-ocean areas, including lakes. LandScan says it extends the land area by a bit, too. We will use a shapefile for Bioko in order to assign zero values to the HRSL.

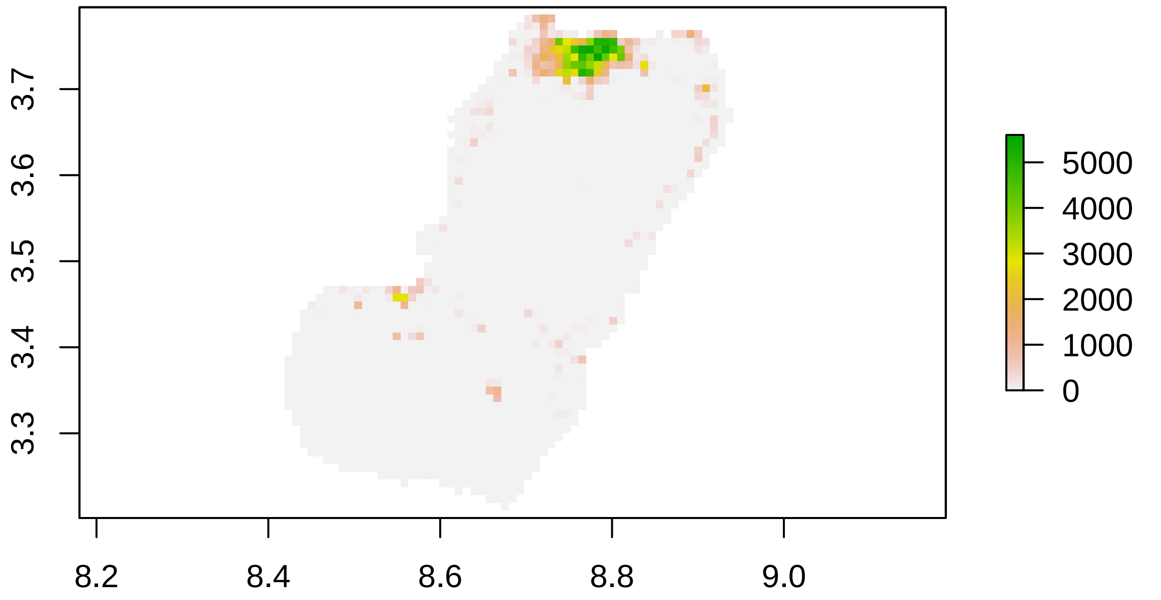
```
hrsl_raster <- read_hrsl(local_directory = data_dir)
hrsl_raster_crop <- raster::crop(hrsl_raster, bimep_feature, snap = "out")
hrsl_zero_mask <- raster::rasterize(bioko_sf, hrsl_raster_crop, field = 0)
hrsl_raster_zero <- raster::cover(hrsl_raster_crop, hrsl_zero_mask)
hrsl_points_sf <- hrsl_points(hrsl_raster_crop)
# hrsl_points_sf[is.na(hrsl_points_sf[[1]]), 1] <- 0
plot(hrsl_raster_zero)
```



And do the projection using a builtin function of the raster package.

```
km_hrsl_raster <- raster::rasterize(
  hrsl_points_sf,
  km_raster,
  field = "population_gnq_2018.10.01",
  fun = sum
  )
area_zero_raster <- raster::rasterize(bioko_sf, km_hrsl_raster, field = 0)
km_hrsl_raster <- raster::cover(km_hrsl_raster, area_zero_raster)
```

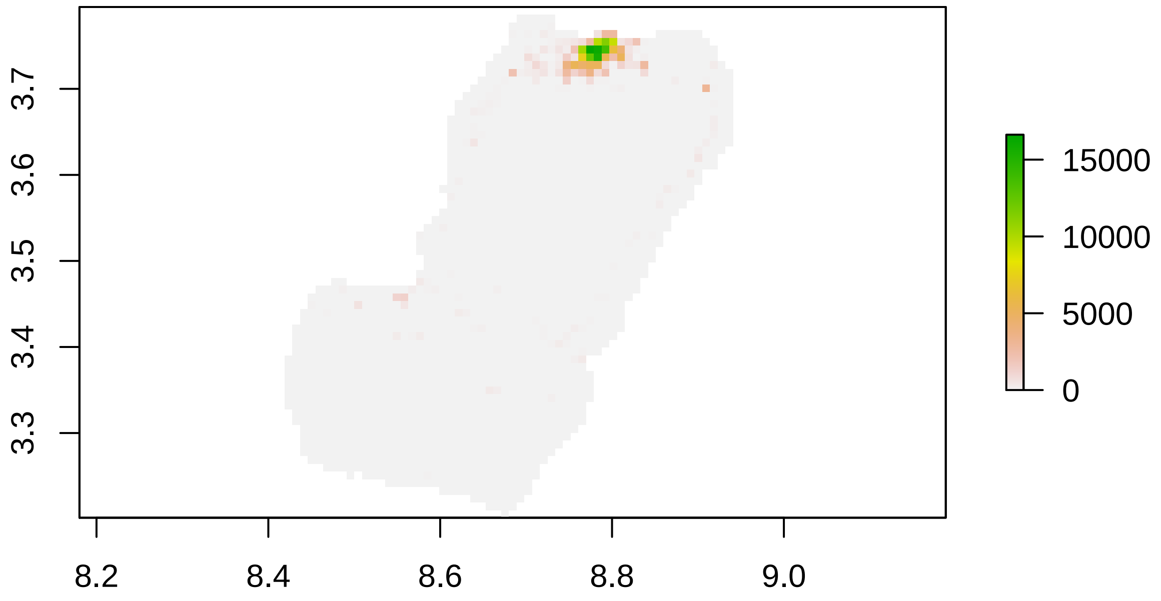
```
plot(km_hrsl_raster)
```



Rasterize that pop feature, summing it per pixel.

```r
bimep_raster <- raster::rasterize(
  bimep_feature,
  km_raster,
  field = "pop",
  fun = sum
  )
names(bimep_raster)
#> [1] "layer"
```

```r
plot(bimep_raster)
```



We will write all raster layers the same way.

```r
write_one <- function(layer, basename) {
  raster::writeRaster(
    layer,
    filename = fs::path(data_dir, basename),
    format = "GTiff",
    overwrite = TRUE
    )
}
```

Write these as layers to a GeoTIFF.

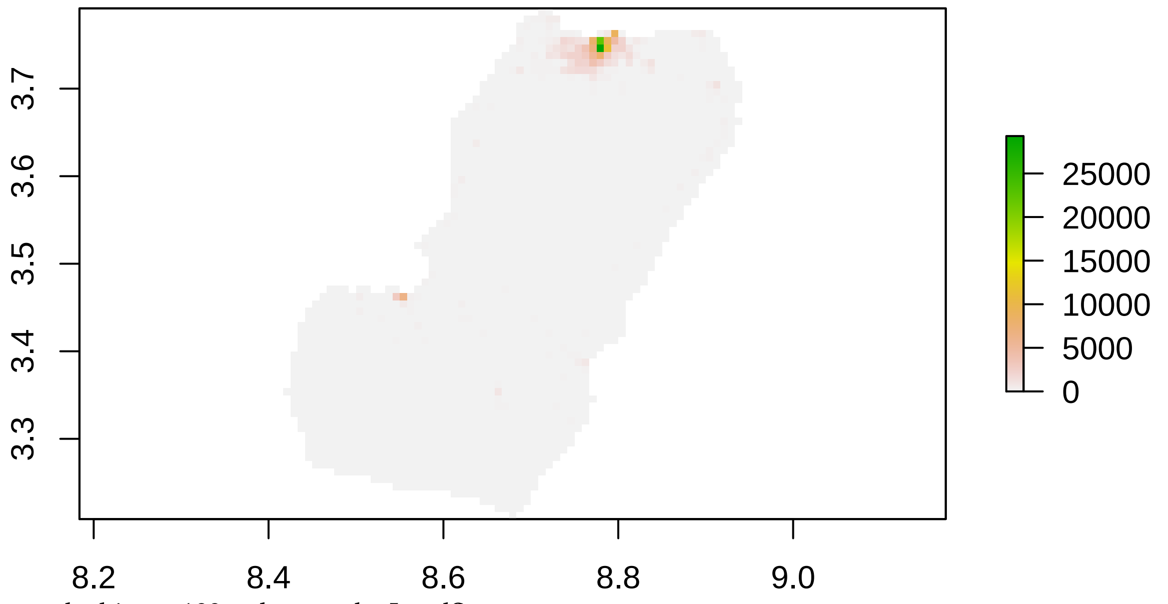
```
names(km_hrsl_raster) <- c("HRSL")
names(bimep_raster) <- c("BIMEP")
write_one(km_hrsl_raster, "hrsl_on_area")
write_one(bimep_raster, "bimep_on_area")
write_one(hrsl_raster_zero, "hrsl_on_hrsl")
```

```
grids$coarse["hrsl"] <- hrsl_points_sf[[1]]
tm_shape(grids$coarse[grids$coarse$hrsl > 0,]) +
  tm_polygons("hrsl", title = "HRSL", palette = "Greens", style = "log10") +
  tm_layout(legend.title.size = 1, legend.text.size = 1)
```

4.3 Align BIMEP to LandScan Grid

The LandScan grid is an Arc/Info Binary Grid, so it's the adf type. It uses pixel sizes of 0.008333 in both dimensions. It's the whole world, so we will subset it.

```
landscan <- read_landscan(local_directory = data_dir)
landscan <- raster::crop(landscan, bimep_feature, snap = "out")
plot(landscan)
```

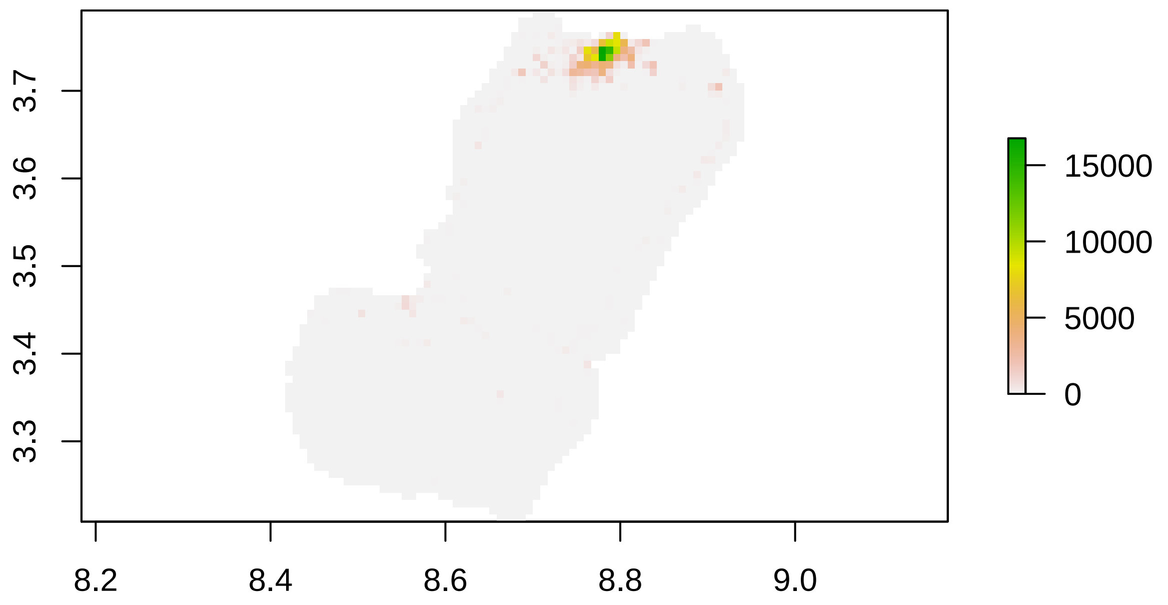


Now put the bimep 100m data on the LandScan raster.

```
bimep_landscan_raster <- raster::rasterize(
  bimep_feature,
  landscan,
  field = "pop",
  fun = sum
  )
names(bimep_landscan_raster)
```

```
#> [1] "layer"
```

```
plot(bimep_landscan_raster)
```

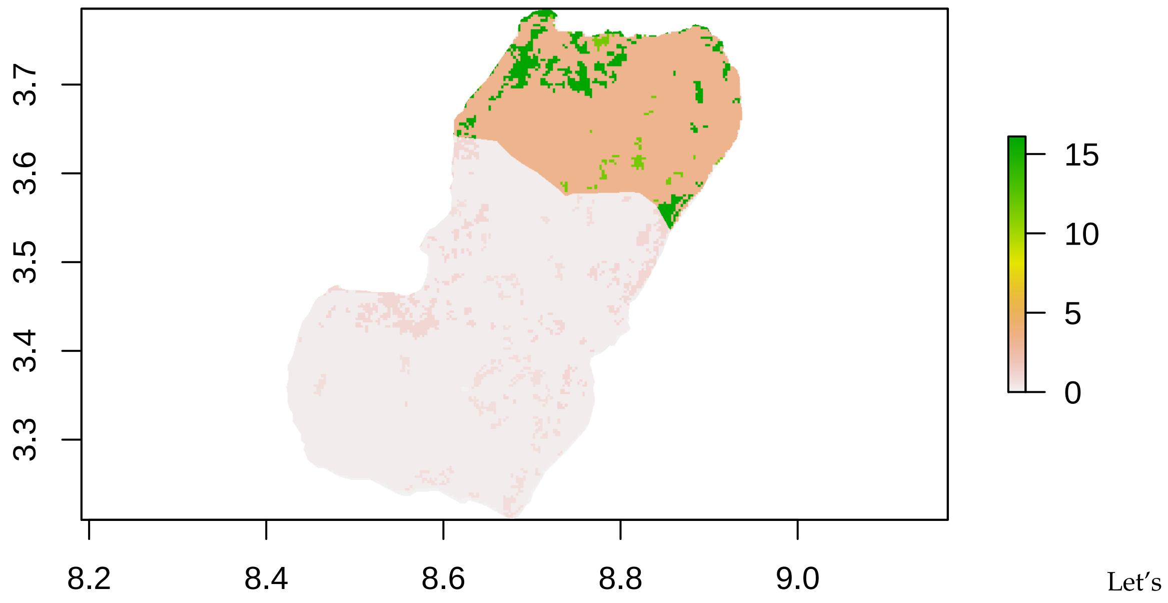


```
names(bimep_landscan_raster) <- c("HRSL")
names(landscan) <- c("LandScan")
write_one(landscan, "landscan_on_landscan")
write_one(bimep_landscan_raster, "bimep_on_landscan")
```

4.4 BIMEP Data on WorldPop Raster

The WorldPop pixel size is 100m, so it's (0.0008333,-0.0008333) in lat-long.

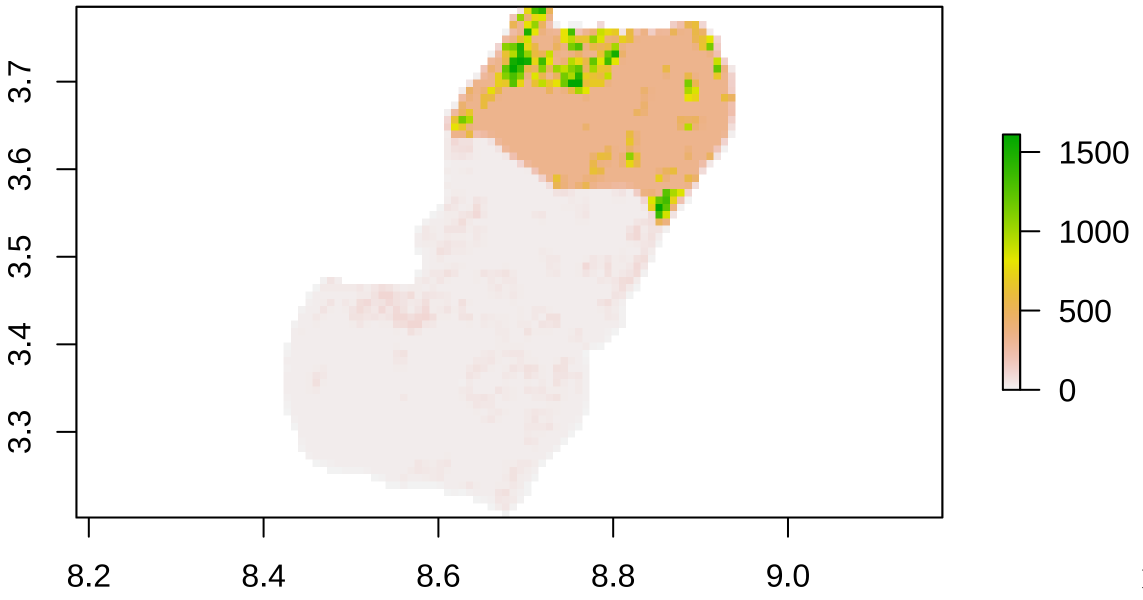
```
worldpop <- raster::raster(fs::path(data_dir, "Equatorial_Guinea_100m_Population", "GNQ15v
worldpop <- raster::crop(worldpop, bimep_feature, snap = "out")
plot(worldpop)
```



Let's turn the

worldpop into a 1km grid using aggregation.

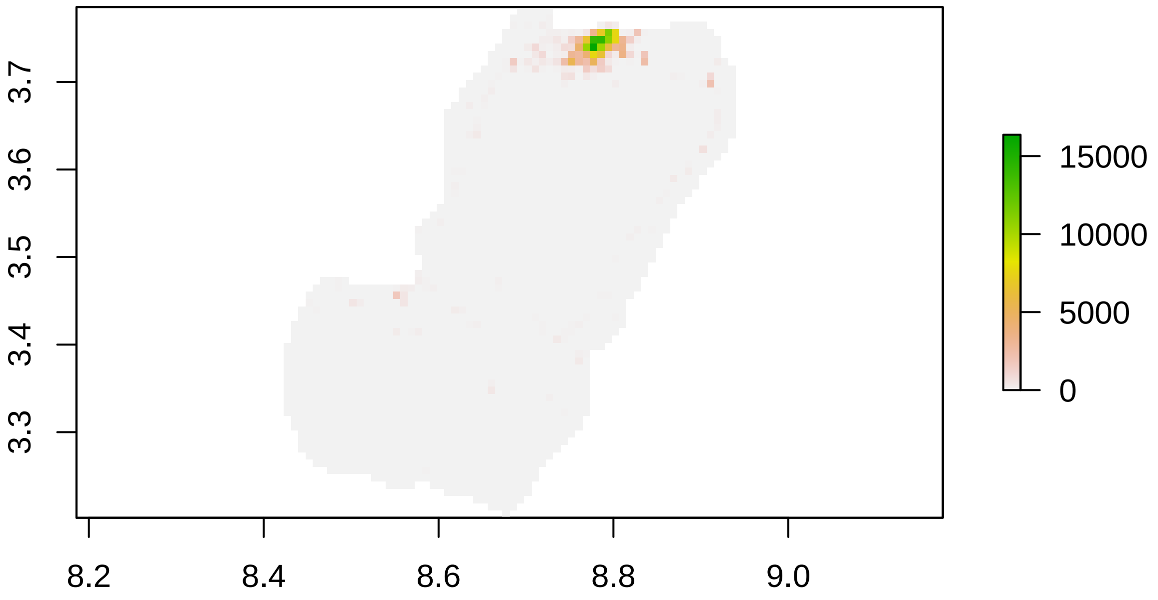
```
worldpop_1km <- raster::aggregate(worldpop, fact = 10, fun = sum, expand = TRUE)
plot(worldpop_1km)
```



Now put the BIMEP on this raster.

```
bimep_worldpop_raster <- raster::rasterize(
  bimep_feature,
  worldpop_1km,
  field = "pop",
  fun = sum
  )
names(bimep_worldpop_raster)
#> [1] "layer"
```

```
plot(bimep_worldpop_raster)
```



```r
names(worldpop_1km) <- "WorldPop"
names(bimep_worldpop_raster) <- "BIMEP"
wp_layers <- raster::addLayer(worldpop_1km, bimep_worldpop_raster)
write_one(worldpop_1km, "worldpop_on_1kworldpop")
write_one(bimep_worldpop_raster, "bimep_on_1kworldpop")
write_one(worldpop, "worldpop_on_worldpop")
```