Comparing Population Maps

David L Smith, Carlos Guerra, Andrew Dolgert, Brendan Fries

Contents

From git@github.com:dd-harp/population_comparison_bioko.git on Wed Mar 11 08:35:18 2020, generated by adolgert.

1 Overview

This notebook calculates metrics from a population density map by comparing them with a gold standard map.

2 Maps

The maps were made by the "get-data" vignette. We load them all at once here for convenience. The naming scheme is dataset-on-grid, where grids are 100m or 1km, according to where they come from.

```
data_dir <- params$data_directory
map_root <- fs::path(data_dir, "aligned")
files <- list.files(map_root, pattern = "*.tif$")
files_df <- filenames_to_description(files)
maps <- lapply(files, function(x) raster::raster(fs::path(map_root, x)))
names(maps) <- rownames(files_df)
files_df
```

	filename	name	resolution	source
BIMEP on HRSL 100m	HRSL100_BIMEP.tif	BIMEP on HRSL 100m	100	BIMEP
HRSL on HRSL 100m	HRSL100_HRSL.tif	HRSL on HRSL 100m	100	HRSL
BIMEP on HRSL 1km	HRSL1000_BIMEP.tif	BIMEP on HRSL 1km	1000	BIMEP
HRSL on HRSL 1km	HRSL1000_HRSL.tif	HRSL on HRSL 1km	1000	HRSL
BIMEP on LandScan 100m	LandScan100_BIMEP.tif	BIMEP on LandScan 100m	100	BIMEP
LandScan on LandScan 100m	LandScan100_LandScan.tif	LandScan on LandScan 100m	100	LandScan
BIMEP on LandScan 1km	LandScan1000_BIMEP.tif	BIMEP on LandScan 1km	1000	BIMEP
LandScan on LandScan 1km	LandScan1000_LandScan.tif	LandScan on LandScan 1km	1000	LandScan
BIMEP on WorldPop 100m	WorldPop100_BIMEP.tif	BIMEP on WorldPop 100m	100	BIMEP
WorldPop on WorldPop 100m	WorldPop100_WorldPop.tif	WorldPop on WorldPop 100m	100	WorldPop
BIMEP on WorldPop 1km	WorldPop1000_BIMEP.tif	BIMEP on WorldPop 1km	1000	BIMEP

	filename	name	resolution	source
WorldPop on WorldPop 1km	WorldPop1000_WorldPop.tif	WorldPop on WorldPop 1km	1000	WorldPop

There are 12 rows for (3 data sources) x (2 resolutions each) x (data source and BIMEP comparison).

We will need some canonical projections, too. Universal Transverse Mercator (UTM) is a projection that will give us the island measured in meters.

```
utm_projection <- "+proj=utm +zone=32N +ellps=WGS84  +no_defs +units=m +datum=WGS84"
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
```

3 Summary Statistics

The first figure is summary statistics, done at the pixel level.

3.1 Urban Fraction

How should we calculate urban fraction? There isn't a single, standard definition. Countries each choose their own ways to assess the important demographic movement from rural to urban. These assessments often combine population density, administrative boundaries, resource availability (like sewer and water), and functional use patterns. We have one kind of data here, population density, and we're interested in how this data contributes to more nuanced urban metrics, but let's stick to density.

Equatorial Guinea defines urban population density as 1500 people per square kilometer. That's 1500 people per pixel for a grid with 1 km pixels and 15 people per pixel for a grid with 100 m pixels. This is equivalent to 0.0015 people per meter squared.

We know we want to measure population density, but there are three ways to measure population density.

1. Count each pixel that reaches the threshold.
2. Count each pixel that has 1500 people within a circle whose area is a square kilometer.
3. Use a kernel density estimator to treat pixels as samples from a population density surface.

3.1.1 Raw Urban Fraction

We measure pixel threshold as a number of people per square meter. We need to know the size of each pixel in order to do that. It helps to know how many pixels we're talking about, so we return the numerator and denominator of each fraction.

```
urban_fraction <-
function(density_raster, urban_per_kilometer_sq) {
  urban_per_meter_sq <- urban_per_kilometer_sq / 10^6
  urban_per_pixel_sq <- urban_per_meter_sq * square_meters_per_pixel.raster(density_raster)
  vals <- raster::getValues(density_raster)
  c(sum(vals > urban_per_pixel_sq, na.rm = TRUE), sum(!is.na(vals)))
}
```

We can reuse this function for the kernel density estimation by applying it to the estimated density instead of the raw density.

3.1.2 Kernel Density Estimation

Kernel density estimation is a way to determine density from a point set. We have a grid, not points, so we take a few steps.

1. Convert the grid into a rate per unit area.
2. Sample points from that grid using a poisson process for each cell.
3. Project the points into a plane measured in meters.
4. Estimate percent urban from the resulting points.

This work will use the `raster` and `spatstat` packages for statistics, and it will use the `proj4` package to project the points from latitude-longitude to UTM.

```
pop_raster <- maps[["LandScan on LandScan 1km"]]
sum(raster::values(pop_raster) > 1500, na.rm = TRUE) / nrow(pop_raster) / ncol(pop_raster)
```

```
## [1] 0.006919643
```

```
population_count_estimator <-
function(projected_raster) {
  # Make a point pattern
  raster_sum <- as.integer(raster::cellStats(projected_raster, stat = "sum", na.rm = TRUE))
  pop_raster_im <- maptools::as.im.RasterLayer(projected_raster)
  point_pattern <- spatstat::rpoint(raster_sum, pop_raster_im)
  stopifnot(point_pattern$n == raster_sum)
  density <- spatstat::density.ppp(
    point_pattern,
    sigma = spatstat::bw.diggle,
    dimyx = c(raster::ncol(projected_raster), raster::nrow(projected_raster))
    )
  density * raster_sum / sum(density)
}
```

It will help to use the Bioko outlines for windowing in spatstat. You need to translate sf to sp to sp geometry to spatstat owin.

```
bioko_sp <- as(bioko_sf, Class = "Spatial")
bioko_sp_polygon <- as(bioko_sp, "SpatialPolygons")
# Cannot make an owin from an unprojected space. Apparently.
bioko_proj_sf <- sf::st_transform(bioko_sf, crs = utm_projection)
# The projected space will have points outside. Let's make a little
# buffer to catch those points.
meters <- 1
bioko_buffer_sf <- sf::st_buffer(bioko_proj_sf, 500 * meters)
bioko_proj_sp <- as(bioko_proj_sf, Class = "Spatial")
bioko_proj_polygon_sp <- as(bioko_proj_sp, "SpatialPolygons")
bioko_owin <- as.owin(bioko_proj_polygon_sp)
```

```
pop_raster <- maps[["LandScan on LandScan 1km"]]
ls1km_density <- urban_fraction_by_point_density(pop_raster, utm_projection, 1500)
ls1km_density
```

Landscan measures people in thousands, so urban is pixels above 1.5×10^{-6}. The population density is on a map in meters, so let's count the pixel size in square meters.

Let's make population maps for every incoming map. This step approximates the map values by projecting them from lat-long to UTM coordinates. It uses bilinear interpolation. We avoid this kind of approximation through most of our work. This step is less sensitive to interpolation because it will construct point process models during the `population_count_estimator`.

```
kde_dir <- fs::path(params$data_directory, "kde_raster")
if (!dir.exists(kde_dir)) {
  dir.create(kde_dir)
}
for (map_idx in 1:nrow(files_df)) {
  filename <- fs::path(kde_dir, files_df[map_idx, "filename"])
  if (!file.exists(filename)) {
    # or method = "ngb" for nearest-neighbor
    proj_raster <- raster::projectRaster(maps[[map_idx]], crs = utm_projection, method = "b
    # Bilinear interpolation, which can cause negative values.
    proj_raster <- raster::clamp(proj_raster, lower = 0)
    pop_density_im <- population_count_estimator(proj_raster)
    pop_density_raster <- im_to_raster(pop_density_im, utm_projection)
    raster::writeRaster(pop_density_raster, filename = filename, format = "GTiff")
  }
}
```

This makes density maps in a way that's very clear. For each pixel, it averages density within a circular, 1 square km neighborhood.

```
density_dir <- fs::path(params$data_directory, "density_raster")
if (!dir.exists(density_dir)) {
  dir.create(density_dir)
}
for (map_idx in 1:nrow(files_df)) {
  filename <- fs::path(density_dir, files_df[map_idx, "filename"])
  if (!file.exists(filename)) {
    # or method = "ngb" for nearest-neighbor
    proj_raster <- raster::projectRaster(maps[[map_idx]], crs = utm_projection, method = "n
    pop_density_raster <- density_from_disc(proj_raster)
    raster::writeRaster(pop_density_raster, filename = filename, format = "GTiff")
  }
}
```

The density maps are saved as GeoTIFFs in the `density_raster` subdirectory, so load them here before making the summary table.

```
density <- lapply(files_df$filename, function(x) raster::raster(fs::path(density_dir, x)))
names(density) <- names(maps)
```

3.2 Construct the Summary Statistics Table

This function makes the first table in the paper. It makes data for two urban densities, 1000 people per square km and 1500 people per square km. We use the former to compare with the current numbers, but the latter is what Equatorial Guinea uses to define urban.

```
summary_help <- function(urban_cutoff) {
  function(map_idx) {
    popbioko::summary_statistics(
      maps[[map_idx]], density[[map_idx]], urban_per_kilometer_sq = urban_cutoff
      )
```

```
  }
}


make_summary_df <- function(urban_cutoff) {
  summary_stats <- lapply(1:length(maps), summary_help(urban_cutoff))
  summary_list <- do.call(rbind, summary_stats)
  rownames(summary_list) <- names(maps)
  summary_df <- as.data.frame(summary_list)
  summary_df["name"] <- rownames(summary_list)
  summary_df <- merge(summary_df, files_df, by = "name")
}
summary1000_df <- make_summary_df(1000)
summary1500_df <- make_summary_df(1500)
names(summary1500_df)
```

```
##  [1] "name"            "total"           "maximum"
##  [4] "empty_percent"   "pareto_fraction" "urban_num"
##  [7] "urban_den"       "urban_raw"       "urban_fit_num"
## [10] "urban_fit_den"   "urban_fit"       "na_percent"
## [13] "filename"        "resolution"      "source"
## [16] "grid"
```

Those dataframes are both strings and numeric. When we rotate them, they will be all strings, so let's do the conversion by hand.

```
for_show <- function(summary_df) {
  urban_raw_percent <- as.numeric(summary_df[, "urban_raw"]) * 100
  urban_percent <- as.numeric(summary_df[, "urban_fit"]) * 100
  data.frame(
    source = summary_df$source,
    grid = summary_df$grid,
    resolution = as.character(summary_df$resolution),
    total = sprintf("%.0f", summary_df$total),
    maximum = sprintf("%.0f", summary_df$maximum),
    empty_percent = sprintf("%.4g", summary_df$empty_percent),
    na_percent = sprintf("%.4g", summary_df$na_percent),
    urban_raw = sprintf("%.4g", urban_raw_percent),
    urban_fit = sprintf("%.4g", urban_percent),
    pareto_fraction = sprintf("%.4g", 100 * as.numeric(summary_df$pareto_fraction))
  )
}
show1000_df <- for_show(summary1000_df)
show1500_df <- for_show(summary1500_df)
```

3.3 Comparison Tables

3.3.1 Compare two urban fractions

We're looking for trends that show there are fewer urban areas when we raise the urban criteria from 1000 people to 1500 people per square kilometer.

```
as.data.frame(t(show1000_df[show1000_df$grid == "HRSL", ]))
```

	1	2	7	8

	1	2	7	8
source	BIMEP	BIMEP	HRSL	HRSL
grid	HRSL	HRSL	HRSL	HRSL
resolution	100	1000	100	1000
total	239056	239056	231210	231210
maximum	120	16620	15	5904
empty_percent	99.13	88.61	98.92	86.32
na_percent	47.86	44.99	47.86	44.99
urban_raw	0.8733	2.007	1.076	3.175
urban_fit	1.077	0.82	1.692	1.297
pareto_fraction	2.593	2.098	3.714	2.918

```
as.data.frame(t(show1500_df[show1500_df$grid == "HRSL", ]))
```

	1	2	7	8
source	BIMEP	BIMEP	HRSL	HRSL
grid	HRSL	HRSL	HRSL	HRSL
resolution	100	1000	100	1000
total	239056	239056	231210	231210
maximum	120	16620	15	5904
empty_percent	99.13	88.61	98.92	86.32
na_percent	47.86	44.99	47.86	44.99
urban_raw	0.8254	1.494	1.076	2.381
urban_fit	0.8642	0.6102	1.345	0.9725
pareto_fraction	2.593	2.098	3.714	2.918

3.3.2 Compare maps of raw and estimated urban fractions

We would like to see whether the estimated urban fraction has a reasonable-looking gaussian kernel size.

```
sq_m_per <- c(square_meters_per_pixel.raster(maps[[2]]), square_meters_per_pixel.raster(den
sq_m_per
```

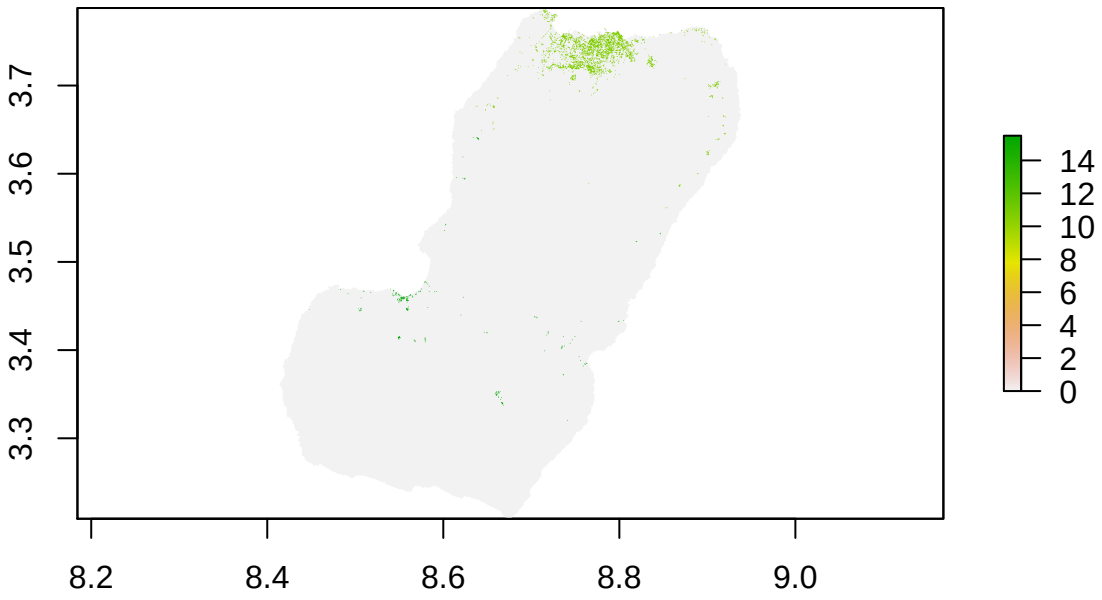
```
## [1] 948.0121 948.6300
```

```
cutoffs <- data.frame(urban_per_meter_sq = c(1000, 1500) / 10^6)
cutoffs["raw"] <- cutoffs[["urban_per_meter_sq"]] * square_meters_per_pixel.raster(maps[[2
cutoffs["density"] <- cutoffs[["urban_per_meter_sq"]] * square_meters_per_pixel.raster(dens
cutoffs
```

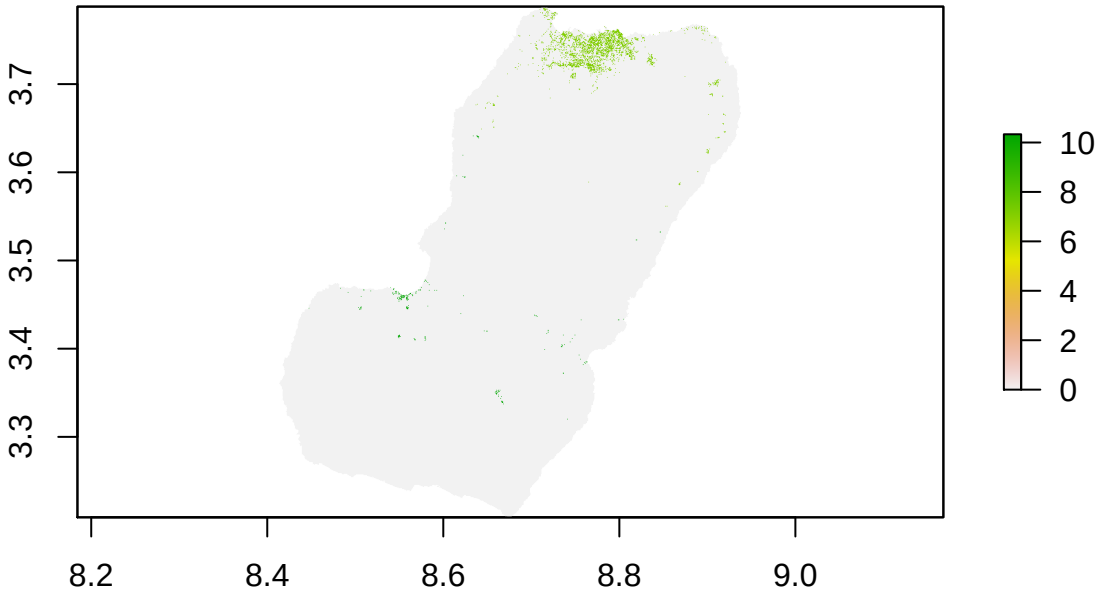
urban_per_meter_sq	raw	density
0.0010	0.9480121	0.948630
0.0015	1.4220181	1.422945

Plot these with cutoffs to see that the maps are similar.

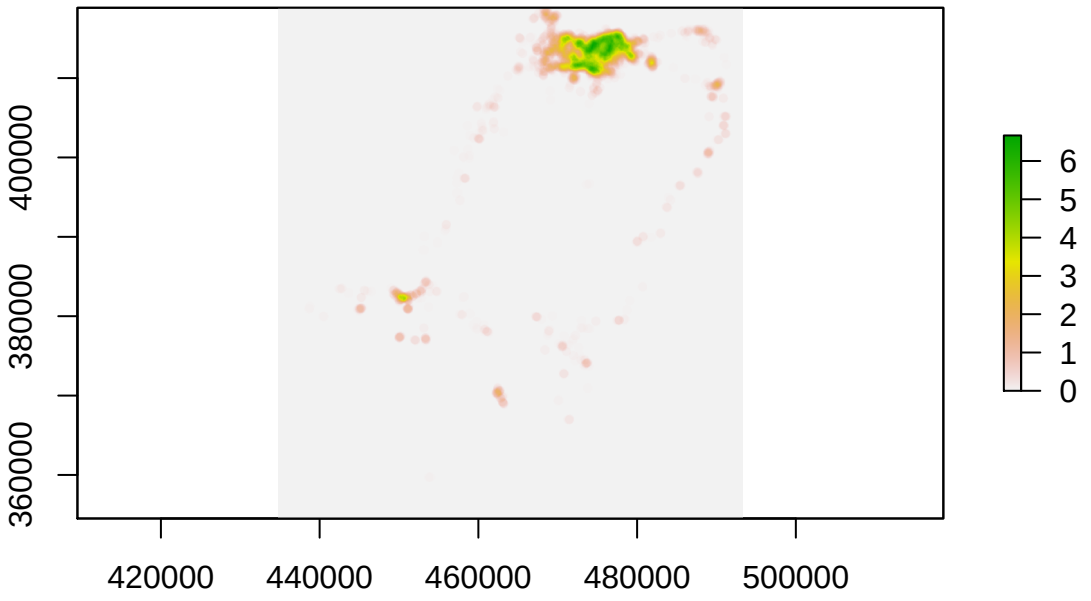
```r
plot(maps[[2]] / cutoffs[1, "raw"])
```



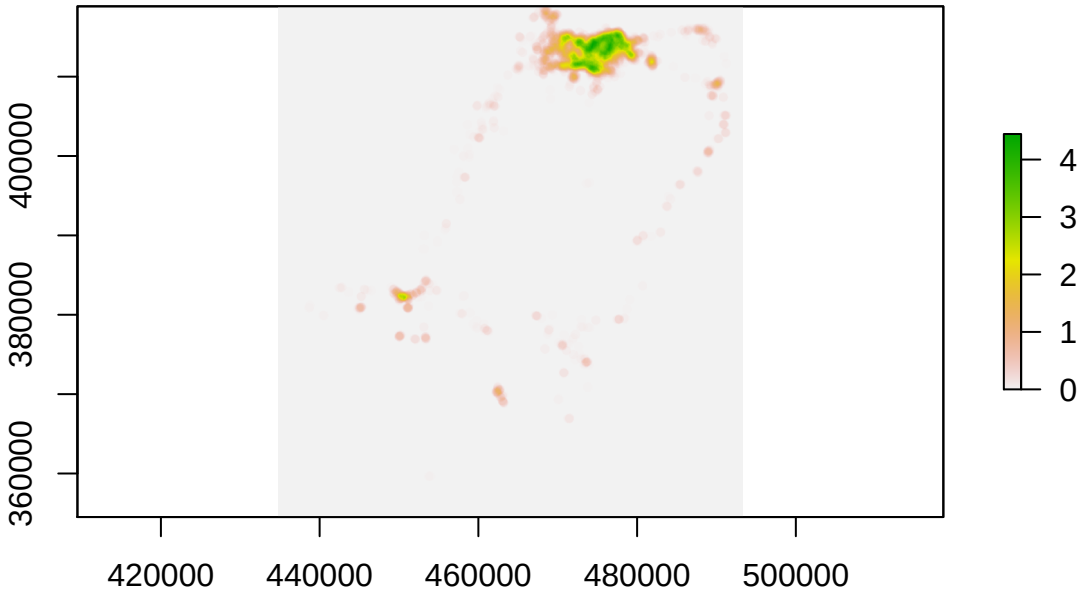

```r
plot(maps[[2]] / cutoffs[2, "raw"])
```




```r
plot(density[[2]] / cutoffs[1, "density"])
```




```
plot(density[[2]] / cutoffs[2, "density"])
```



3.3.3 Compare across source grids

This shows all HRSL grid together, then all WorldPop, then all LandScan.

```
as.data.frame(t(show1500_df[show1500_df$grid == "HRSL", ]))
```

	1	2	7	8
source	BIMEP	BIMEP	HRSL	HRSL
grid	HRSL	HRSL	HRSL	HRSL
resolution	100	1000	100	1000
total	239056	239056	231210	231210
maximum	120	16620	15	5904
empty_percent	99.13	88.61	98.92	86.32
na_percent	47.86	44.99	47.86	44.99

	1	2	7	8
urban_raw	0.8254	1.494	1.076	2.381
urban_fit	0.8642	0.6102	1.345	0.9725
pareto_fraction	2.593	2.098	3.714	2.918

```
as.data.frame(t(show1500_df[show1500_df$grid == "WorldPop", ]))
```

	5	6	11	12
source	BIMEP	BIMEP	WorldPop	WorldPop
grid	WorldPop	WorldPop	WorldPop	WorldPop
resolution	100	1000	100	1000
total	239056	239056	204820	204820
maximum	524	19416	9	1049
empty_percent	98.07	88	1.137	0.6536
na_percent	47.82	45.02	47.92	45.57
urban_raw	1.285	1.692	0	0
urban_fit	0.8506	0.6849	0	0
pareto_fraction	2.595	2.216	15.22	12.11

```
as.data.frame(t(show1500_df[show1500_df$grid == "LandScan", ]))
```

	3	4	9	10
source	BIMEP	BIMEP	LandScan	LandScan
grid	LandScan	LandScan	LandScan	LandScan
resolution	100	1000	100	1000
total	239056	239056	218044	218044
maximum	443	17578	293	29290
empty_percent	98.09	88.53	56.29	56.29
na_percent	49.15	48.82	48.37	48.37
urban_raw	1.282	1.701	1.513	1.513
urban_fit	0.829	0.6588	0.8212	0.5912
pareto_fraction	2.549	2.027	4.527	3.125

3.3.4 Compare across resolutions

This groups data with the same resolution.

```
as.data.frame(t(show1500_df[show1500_df$resolution == 100, ]))
```

	1	3	5	7	9	11
source	BIMEP	BIMEP	BIMEP	HRSL	LandScan	WorldPop
grid	HRSL	LandScan	WorldPop	HRSL	LandScan	WorldPop
resolution	100	100	100	100	100	100
total	239056	239056	239056	231210	218044	204820
maximum	120	443	524	15	293	9
empty_percent	99.13	98.09	98.07	98.92	56.29	1.137
na_percent	47.86	49.15	47.82	47.86	48.37	47.92
urban_raw	0.8254	1.282	1.285	1.076	1.513	0
urban_fit	0.8642	0.829	0.8506	1.345	0.8212	0
pareto_fraction	2.593	2.549	2.595	3.714	4.527	15.22

```r
as.data.frame(t(show1500_df[show1500_df$resolution == 1000, ]))
```

	2	4	6	8	10	12
source	BIMEP	BIMEP	BIMEP	HRSL	LandScan	WorldPop
grid	HRSL	LandScan	WorldPop	HRSL	LandScan	WorldPop
resolution	1000	1000	1000	1000	1000	1000
total	239056	239056	239056	231210	218044	204820
maximum	16620	17578	19416	5904	29290	1049
empty_percent	88.61	88.53	88	86.32	56.29	0.6536
na_percent	44.99	48.82	45.02	44.99	48.37	45.57
urban_raw	1.494	1.701	1.692	2.381	1.513	0
urban_fit	0.6102	0.6588	0.6849	0.9725	0.5912	0
pareto_fraction	2.098	2.027	2.216	2.918	3.125	12.11

3.4 Discussion

- Using estimation to find the urban area matters for the 100m data but not the 1km data.
- I'm surprised at how much the BIMEP 100m numbers depend on which 100m grid is used. For instance, the BIMEP on HRSL has a higher empty percent than the others. I could understand this by regridding BIMEP data to a set of evenly-spaced grids.
- I don't know the implied year for HRSL here. The total populations are different enough, that maybe I should use a growth rate for correction.
- The empty percent was very sensitive to my choice of lower bound for empty. Should I make a graph of how different datasets get close to zero? It's a problem for both WorldPop and LandScan.
- The HRSL is 30m, not 100m. Make sure it's aggregated correctly to 1km.
- WorldPop has a large population. They provide an unscaled and a scaled version. Check which one we're using, what year it's supposed to be, and try the other one.
- The 1km data behaves like a smooth grid. The 100m and 30m data behave like points. I see that b/c the point smoothing makes a big difference on the urban numbers. At what resolution do we treat this data as point data?
- Using a kernel density estimator gives the largest estimate for urban. The 1 km^2 disc seems to give the smallest.

4 Population Scatter Plots

This section makes figure 4, scatter plots of population data versus ground truth population data. The density plots use maps of either resolution to plot estimated density per square km. The population size plots instead use the count of people in a pixel.

Set up consistent colors for the different data sets.

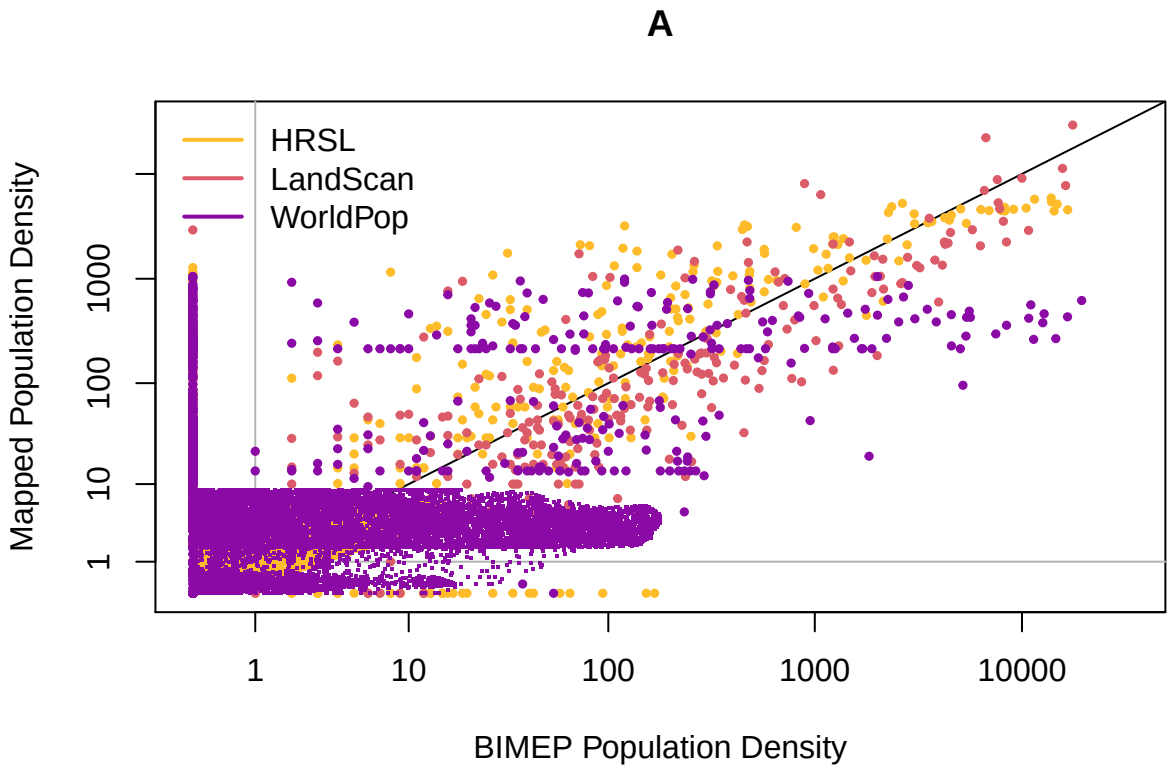
```r
log_offset <- function(offset) {
  function(x) {
    log(x + offset)
  }
}
```

```r
logoff <- log_offset(1)
symbol_size <- c(0.5, 0.25)
names(symbol_size) <- c("1000", "100")
symbol_choice <- c(19, 15)
names(symbol_choice) <- c("1000", "100")
```

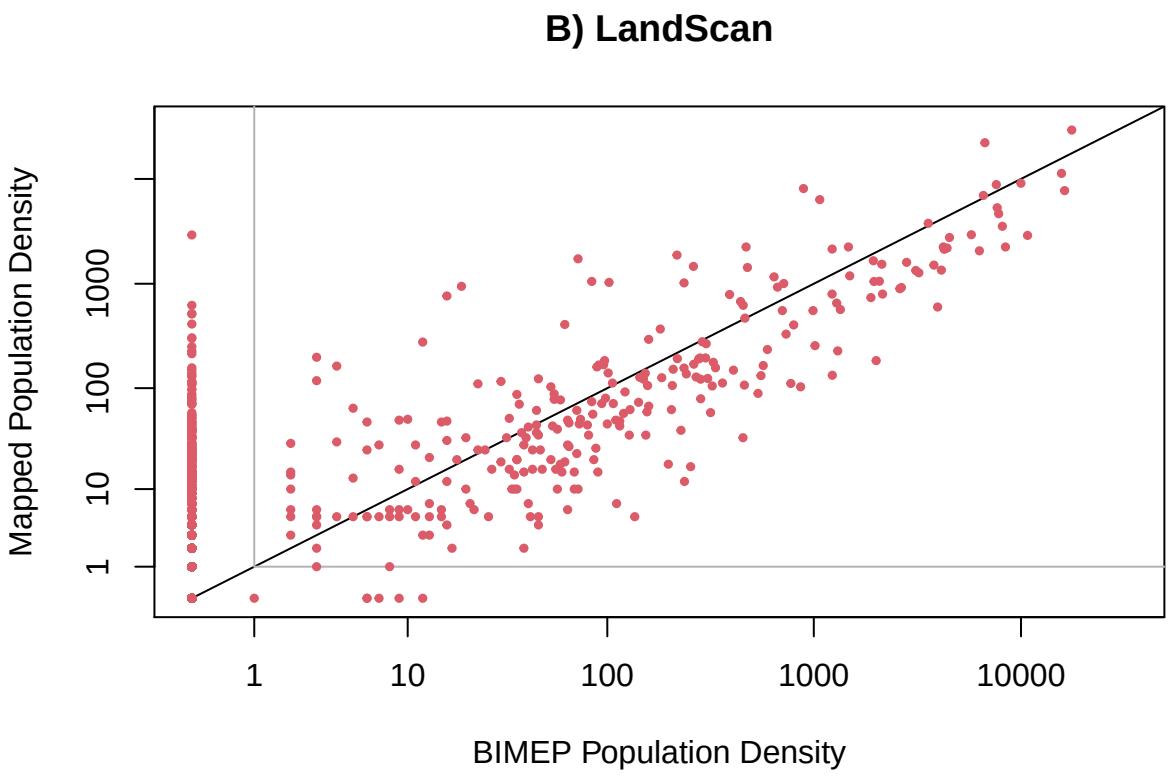
```
map_from_source_resolution <- function(source, resolution, map_list) {
  xy_df <- files_df[files_df$resolution == resolution & files_df$grid == source, ]
  x_map_name <- xy_df[xy_df$source == "BIMEP", "name"]
  y_map_name <- xy_df[xy_df$source == source, "name"]
  x <- raster::getValues(map_list[[x_map_name]])
  y <- raster::getValues(map_list[[y_map_name]])
  list(x = logoff(x), y = logoff(y))
}


#' This gets population counts, not density.
xy_from_source_resolution <- function(source, resolution) {
  map_from_source_resolution(source, resolution, maps)
}


density_from_source_resolution <- function(source, resolution) {
  map_from_source_resolution(source, resolution, density)
}


pop_scatter <- function(y_name, resolution, labels, title, maximum, background = NULL) {
  plot(
    vector(mode = "numeric", length = 0),
    xlim = c(0, maximum), ylim = c(0, maximum),
    xlab = labels[[1]], ylab = labels[[2]], main = title,
    xaxt = "n", yaxt = "n"
    )
  axis(1, logoff(10^c(0:5)), 10^c(0:5))
  axis(2, logoff(10^c(0:5)), 10^c(0:5))
  segments(logoff(0), logoff(0), logoff(10^5), logoff(10^5))
  segments(logoff(1), logoff(1), logoff(1), logoff(10^5), col = colors["grey"])
  segments(logoff(1), logoff(1), logoff(10^5), logoff(1), col = colors["grey"])

  for (plot_idx in 1:length(y_name)) {
    y_source <- y_name[plot_idx]
    y_resolution <- resolution[plot_idx]
    xy <- density_from_source_resolution(y_source, y_resolution)
    cex <- symbol_size[as.character(y_resolution)]
    pch <- symbol_choice[as.character(y_resolution)]
    color <- colors[y_source]
    if (!is.null(background) & plot_idx %in% background) {
      color <- colors["grey"]
    }
    points(xy, col = color, cex = cex, pch = pch)
  }
}
```

```
density_label <- c("BIMEP Population Density", "Mapped Population Density")
maximum1k <- 1.01 * max(xy_from_source_resolution("LandScan", 1000)$y, na.rm = TRUE)
the_three <- c("HRSL", "LandScan", "WorldPop")
pop_scatter(
  c(the_three, "HRSL", "WorldPop"), c(rep(1000, 3), rep(100, 2)),
```

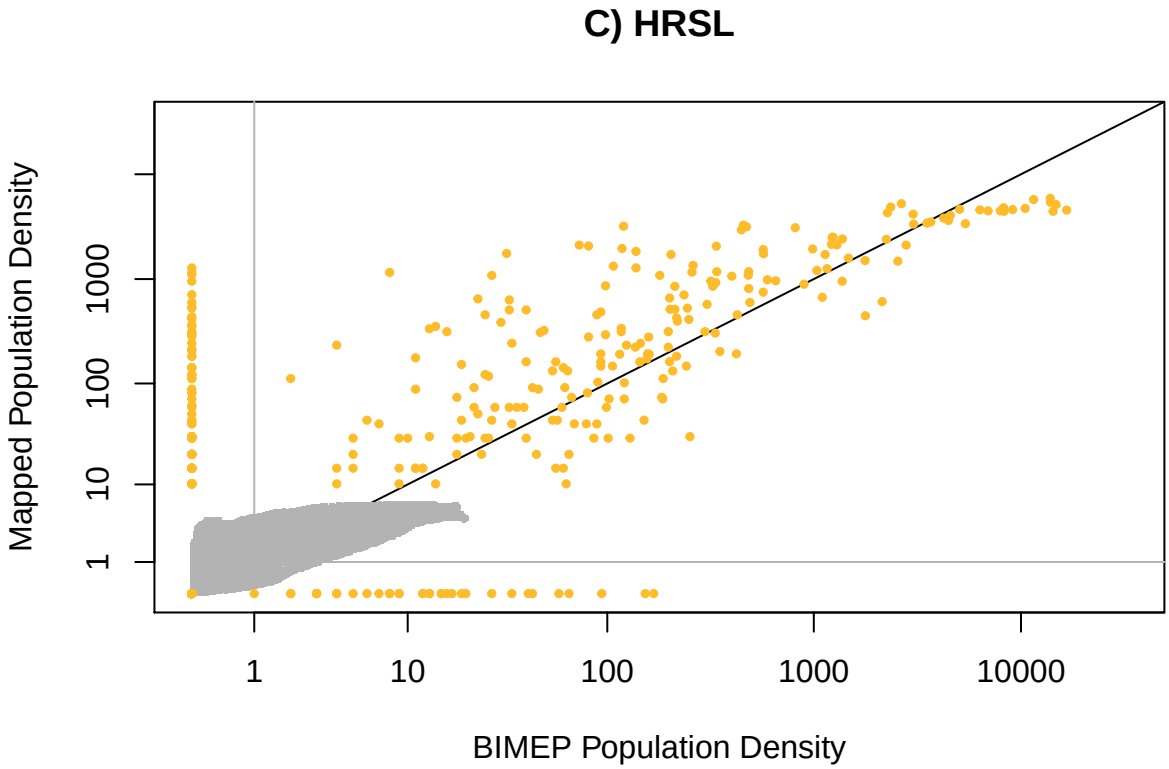
```
  density_label, "A", maximum1k)
legend("topleft", legend =  the_three, col = colors[the_three],
           lty = 1, lwd = 2, cex = 1, bg = "white", bty = "n")
```




```
pop_scatter(
  "LandScan", 1000,
  density_label, "B) LandScan", maximum1k)
```




```
pop_scatter(
  rep("HRSL", 2), c(100, 1000),
  density_label, "C) HRSL", maximum1k, background = 1)
```



```
pop_scatter(
  rep("WorldPop", 2), c(100, 1000),
  density_label, "D) WorldPop", maximum1k, background = 1)
```