Comparing Population Maps

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

Contents

From git@github.com:dd-harp/population_comparison_bioko.git on Thu Apr 9 21:34:55 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.

These maps are estimates of the number of people in each pixel.

```
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 coarse	HRSL_coarse_BIMEP.tif	BIMEP on HRSL coarse	coarse	BIMEP
HRSL on HRSL coarse	HRSL_coarse_HRSL.tif	HRSL on HRSL coarse	coarse	HRSL
BIMEP on HRSL fine	HRSL_fine_BIMEP.tif	BIMEP on HRSL fine	fine	BIMEP
HRSL on HRSL fine	HRSL_fine_HRSL.tif	HRSL on HRSL fine	fine	HRSL
BIMEP on LandScan coarse	LandScan_coarse_BIMEP.tif	BIMEP on LandScan coarse	coarse	BIMEP
LandScan on LandScan coarse	LandScan_coarse_LandScan.tif	LandScan on LandScan coarse	coarse	LandSca
BIMEP on LandScan fine	LandScan_fine_BIMEP.tif	BIMEP on LandScan fine	fine	BIMEP
LandScan on LandScan fine	LandScan_fine_LandScan.tif	LandScan on LandScan fine	fine	LandSca
BIMEP on WorldPop coarse	WorldPop_coarse_BIMEP.tif	BIMEP on WorldPop coarse	coarse	BIMEP
WorldPop on WorldPop coarse	WorldPop_coarse_WorldPop.tif	WorldPop on WorldPop coarse	coarse	WorldPo
BIMEP on WorldPop fine	WorldPop_fine_BIMEP.tif	BIMEP on WorldPop fine	fine	BIMEP
WorldPop on WorldPop fine	WorldPop_fine_WorldPop.tif	WorldPop on WorldPop fine	fine	WorldPo

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_population <-
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 coarse"]]
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)
```

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. *Loading from `density_dir` means we are using the disc.*

```
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"           "side_meters"
##  [4] "maximum"         "max_density"     "empty_percent"
##  [7] "pareto_fraction" "urban_num"       "urban_den"
## [10] "urban_raw"       "urban_fit_num"   "urban_fit_den"
## [13] "urban_fit"       "na_percent"      "filename"
## [16] "resolution"      "source"          "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(sprintf("%.1f", summary_df$side_meters)),
    total = sprintf("%.0f", summary_df$total),
    max_density = sprintf("%.0f", summary_df$max_density),
    empty_percent = sprintf("%.4g", summary_df$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)),
    stringsAsFactors = FALSE
  )
}
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
source	BIMEP	BIMEP	HRSL	HRSL
grid	HRSL	HRSL	HRSL	HRSL
resolution	985.3	30.8	985.3	30.8
total	239056	239056	231210	231210
max_density	17130	22212	6085	7525
empty_percent	88.61	99.13	86.32	98.92
na_percent	44.99	47.86	44.99	47.86
urban_raw	2.007	0.8733	3.175	1.076
urban_fit	2.007	2.029	3.175	3.283
pareto_fraction	3.875	3.904	5.696	5.643

```
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	985.3	30.8	985.3	30.8
total	239056	239056	231210	231210
max_density	17130	22212	6085	7525
empty_percent	88.61	99.13	86.32	98.92
na_percent	44.99	47.86	44.99	47.86
urban_raw	1.494	0.8254	2.381	1.076
urban_fit	1.494	1.637	2.381	2.665
pareto_fraction	3.875	3.904	5.696	5.643

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
```

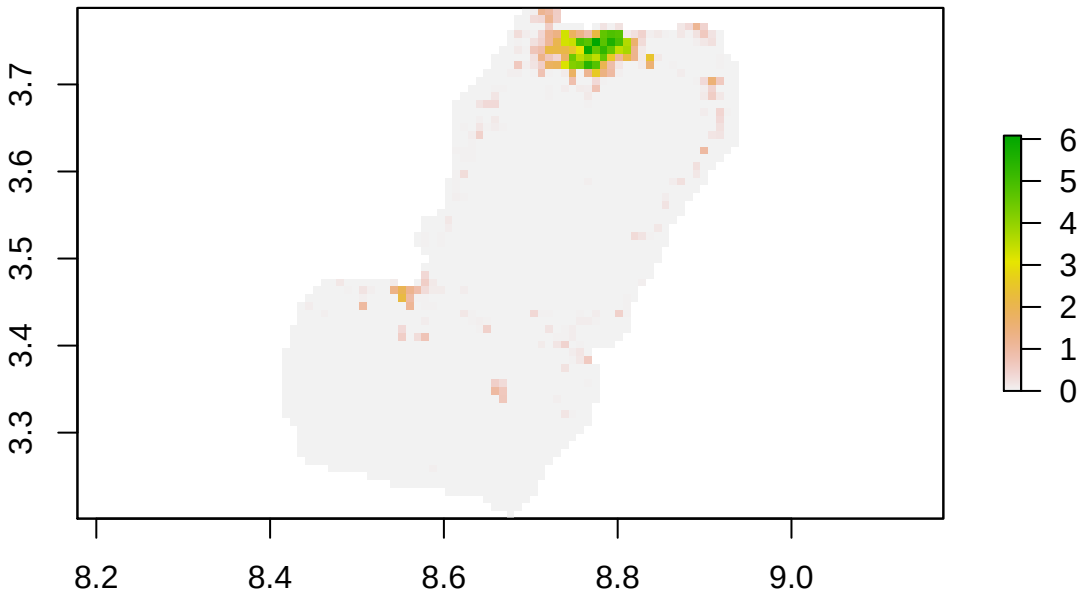
```
## [1] 970768.1 970221.0
```

```
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
```

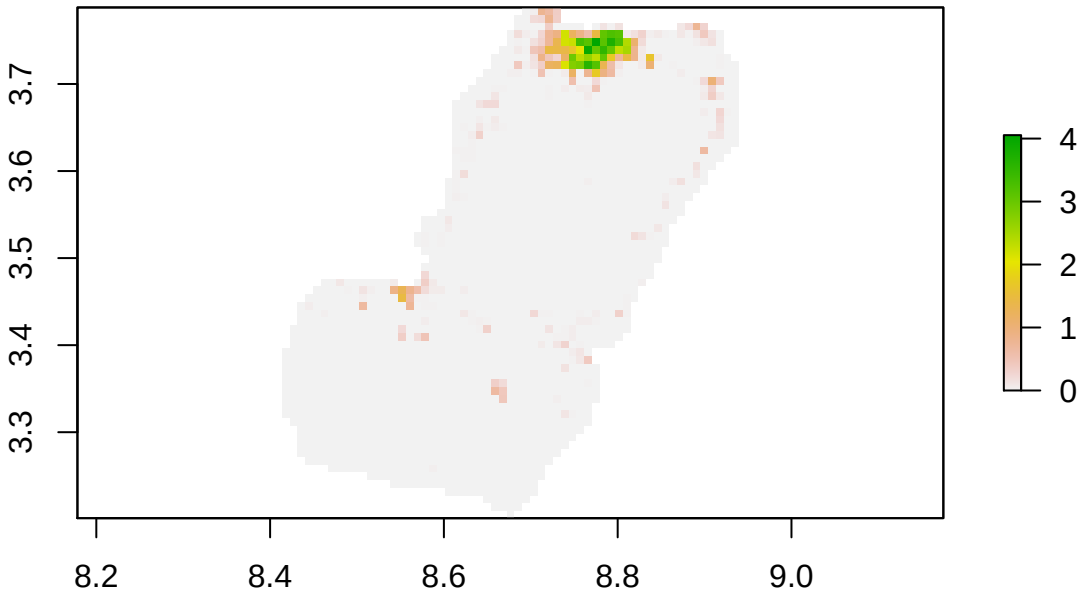
urban_per_meter_sq	raw	density
0.0010	970.7681	970.221
0.0015	1456.1521	1455.332

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

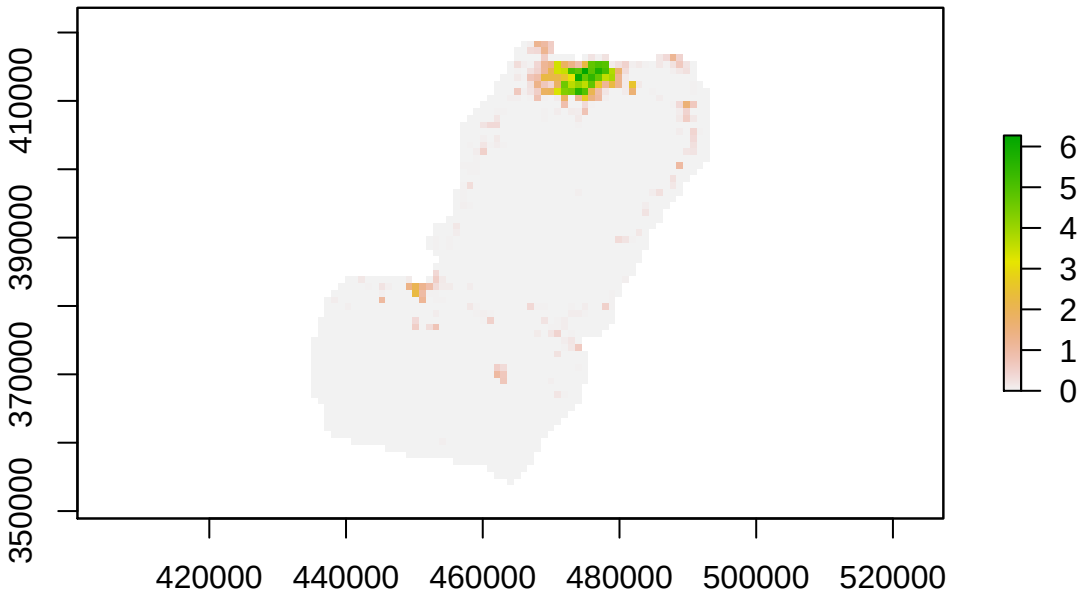
```
plot(maps[[2]] / cutoffs[1, "raw"])
```



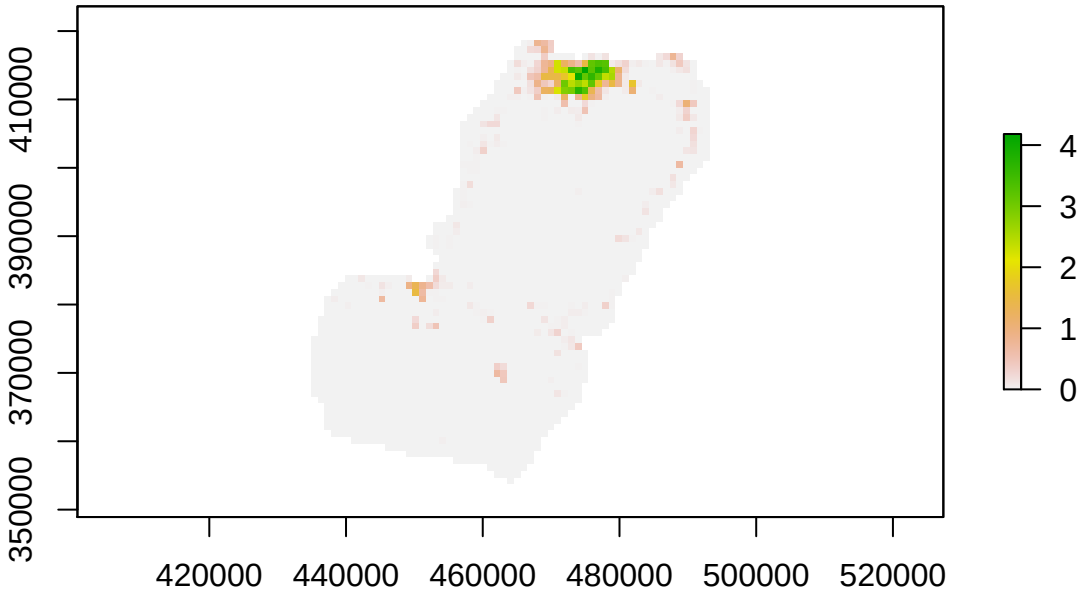

```
plot(maps[[2]] / cutoffs[2, "raw"])
```




```
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	985.3	30.8	985.3	30.8
total	239056	239056	231210	231210
max_density	17130	22212	6085	7525
empty_percent	88.61	99.13	86.32	98.92
na_percent	44.99	47.86	44.99	47.86

	1	2	7	8
urban_raw	1.494	0.8254	2.381	1.076
urban_fit	1.494	1.637	2.381	2.665
pareto_fraction	3.875	3.904	5.696	5.643

```
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	1016.0	92.4	1016.0	92.4
total	239056	239056	204820	204820
max_density	18847	20972	1018	1017
empty_percent	88	98.07	0.6536	1.137
na_percent	45.02	47.82	45.57	47.92
urban_raw	1.692	1.285	0	0
urban_fit	1.692	1.642	0	0
pareto_fraction	4.131	3.945	24.13	24.7

```
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	923.7	92.4	923.7	92.4
total	239056	239056	218044	218044
max_density	20611	21144	34344	30640
empty_percent	88.53	98.09	56.29	56.29
na_percent	48.82	49.15	48.37	48.37
urban_raw	1.701	1.282	1.513	1.513
urban_fit	1.701	1.645	1.513	1.604
pareto_fraction	3.925	3.945	5.275	6.244

3.3.4 Compare across resolutions

This groups data with the same resolution.

```
as.data.frame(t(show1500_df[as.numeric(show1500_df$resolution) < 500, ]))
```

	2	4	6	8	10	12
source	BIMEP	BIMEP	BIMEP	HRSL	LandScan	WorldPop
grid	HRSL	LandScan	WorldPop	HRSL	LandScan	WorldPop
resolution	30.8	92.4	92.4	30.8	92.4	92.4
total	239056	239056	239056	231210	218044	204820
max_density	22212	21144	20972	7525	30640	1017
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	1.637	1.645	1.642	2.665	1.604	0
pareto_fraction	3.904	3.945	3.945	5.643	6.244	24.7

```
as.data.frame(t(show1500_df[as.numeric(show1500_df$resolution) > 500, ]))
```

	1	3	5	7	9	11
source	BIMEP	BIMEP	BIMEP	HRSL	LandScan	WorldPop
grid	HRSL	LandScan	WorldPop	HRSL	LandScan	WorldPop
resolution	985.3	923.7	1016.0	985.3	923.7	1016.0
total	239056	239056	239056	231210	218044	204820
max_density	17130	20611	18847	6085	34344	1018
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	1.494	1.701	1.692	2.381	1.513	0
pareto_fraction	3.875	3.925	4.131	5.696	5.275	24.13

3.4 Error Bounds

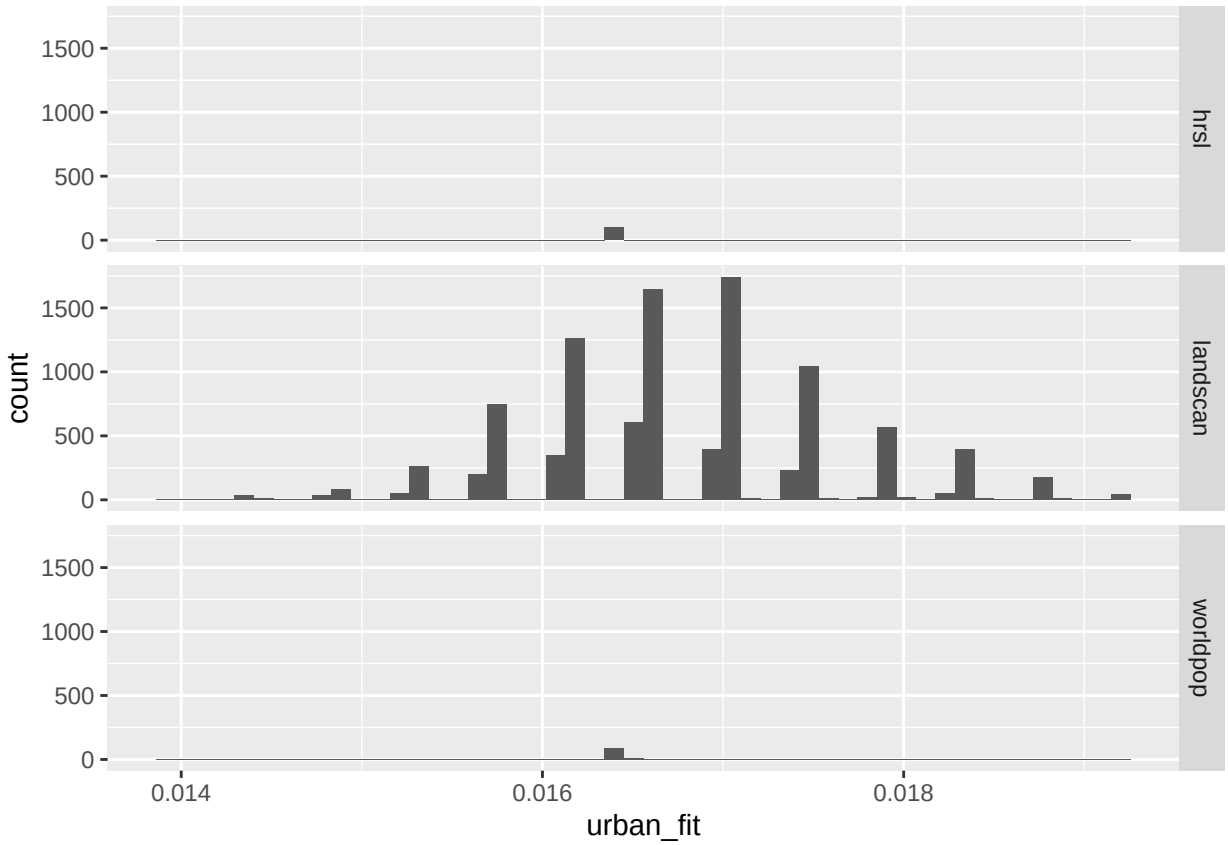
Carlos asked how much the exact alignment of the grid matters. The gold data is point data. Grids that are shifted slightly will have slightly different values in the tables. So let's shift each major grid a bunch of times and see what the values are.

```
# With side_len = 10, this can take two hours to run.
error_dir <- fs::path(params$data_directory, "errors")
side_len <- 10
for (grid in c("landscan", "hrsl", "worldpop")) {
  stats_file <- fs::path(error_dir, paste(grid, "csv", sep = "."))
  popbioko::summary_statistics_multiple_grids(grid, side_len, stats_file)
}
```

```
error_dir <- fs::path(params$data_directory, "errors")
csvs <- do.call(
  rbind,
  lapply(
    list.files(error_dir, pattern = ".csv$", full.names = TRUE),
    function(csv) {read.csv(csv, header = TRUE, stringsAsFactors = TRUE)}
    ),
)
```

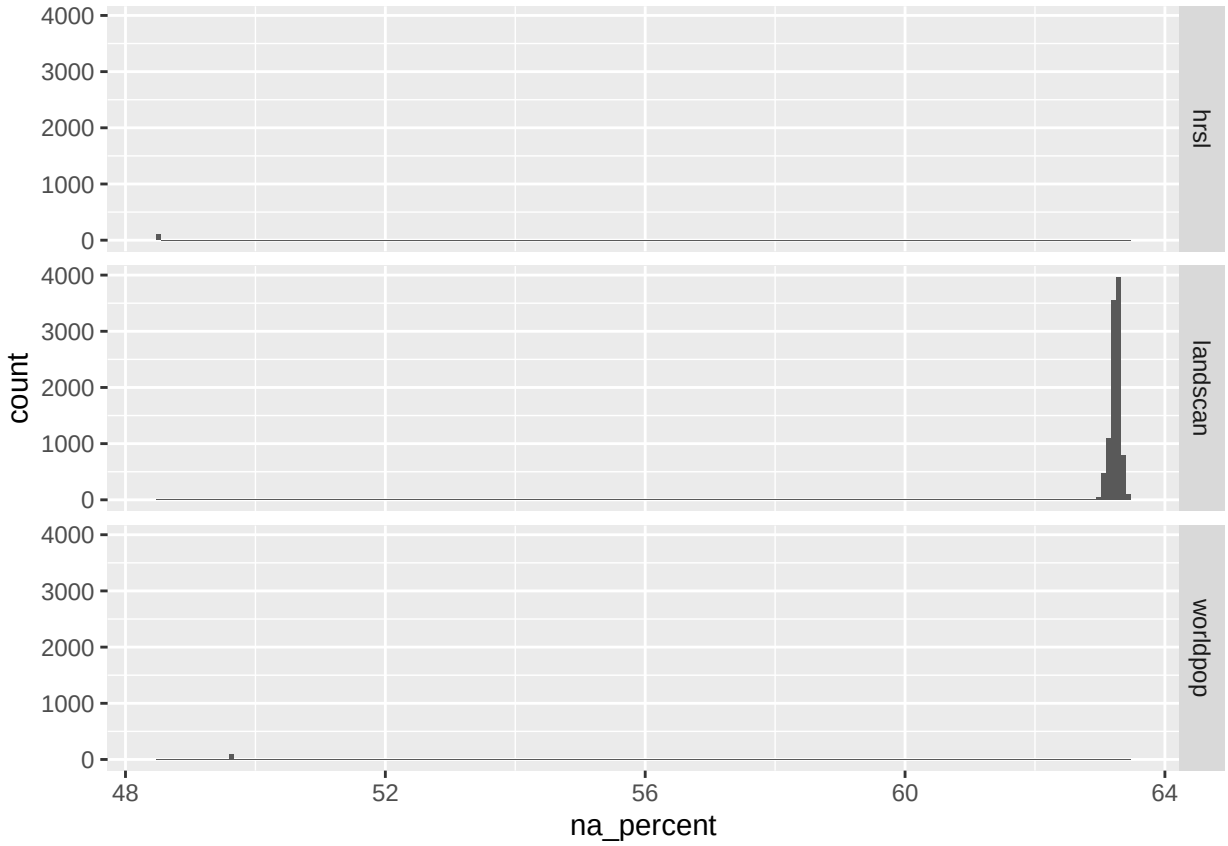
I'm curious how much urban fit varies. What I see is that the mean doesn't change much but the more coarse grids have a larger standard deviation.

```
ggplot(csvs, aes(x = urban_fit, group = grid)) +
  geom_histogram(bins = 50) +
    facet_grid(grid ~ .)
```



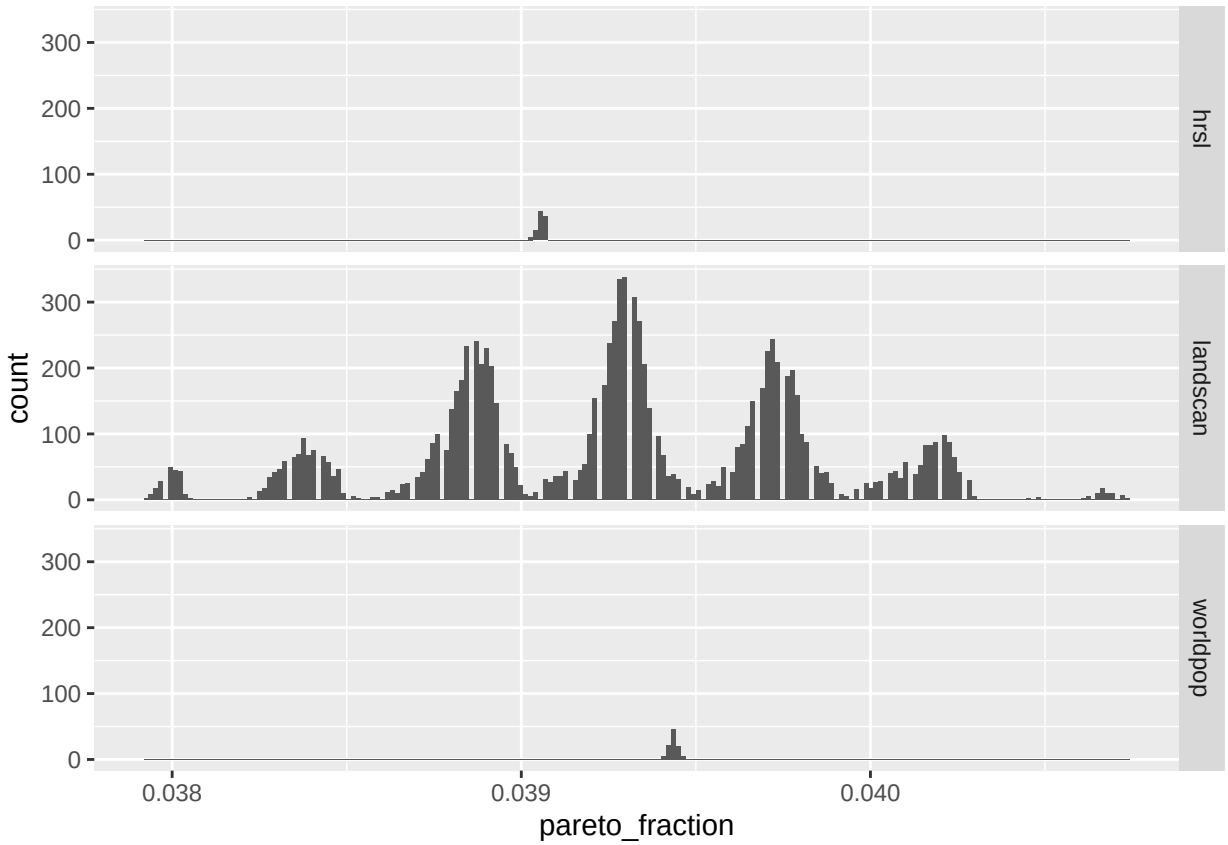
The percent of NA values is the size of the border around the island. This is kind of a control. We see it doesn't vary a lot.

```
ggplot(csvs, aes(x = na_percent, group = grid)) +
  geom_histogram(bins = 200) +
    facet_grid(grid ~ .)
```



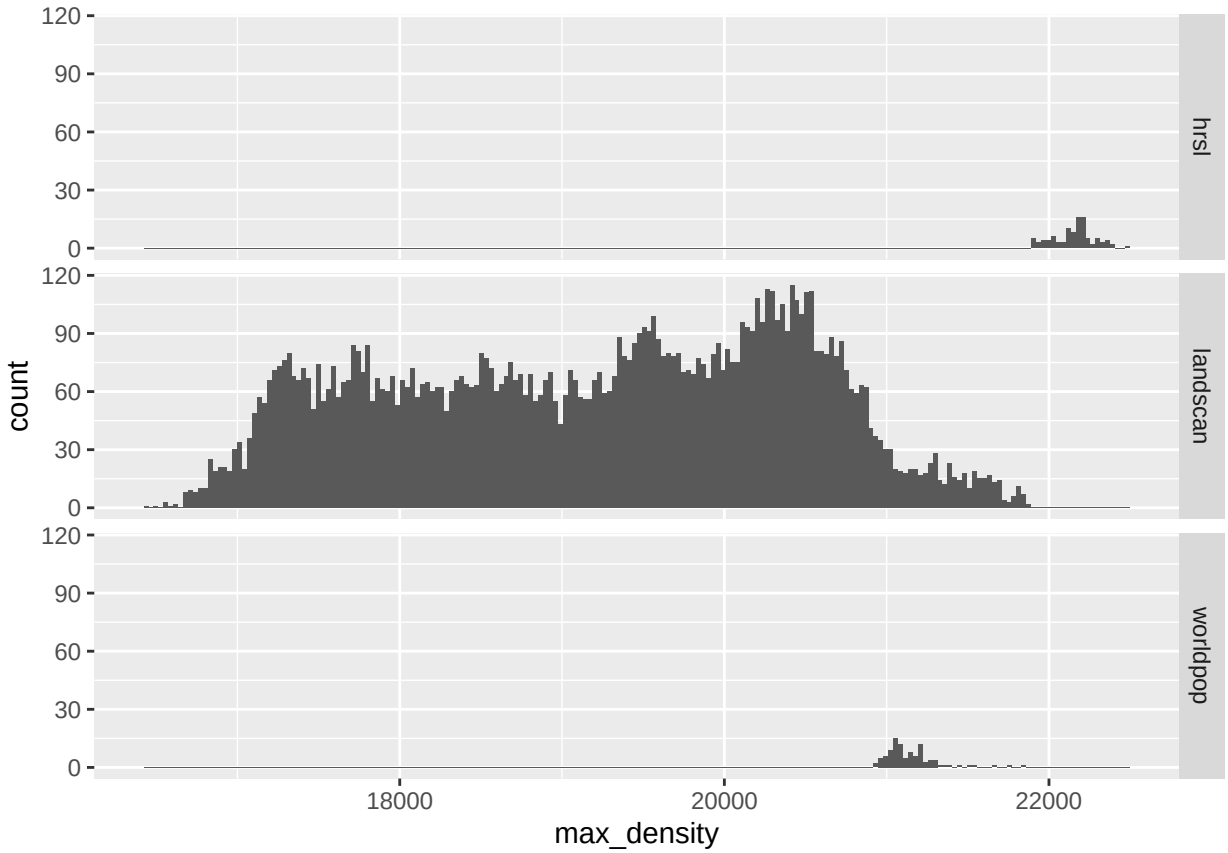
I don't know how I expect pareto fraction to vary as the grid size changes. I feel kind of surprised. I think this is a result of having small statistics and using a non-statistical estimator.

```
ggplot(csvs, aes(x = pareto_fraction, group = grid)) +
  geom_histogram(bins = 200) +
    facet_grid(grid ~ .)
```



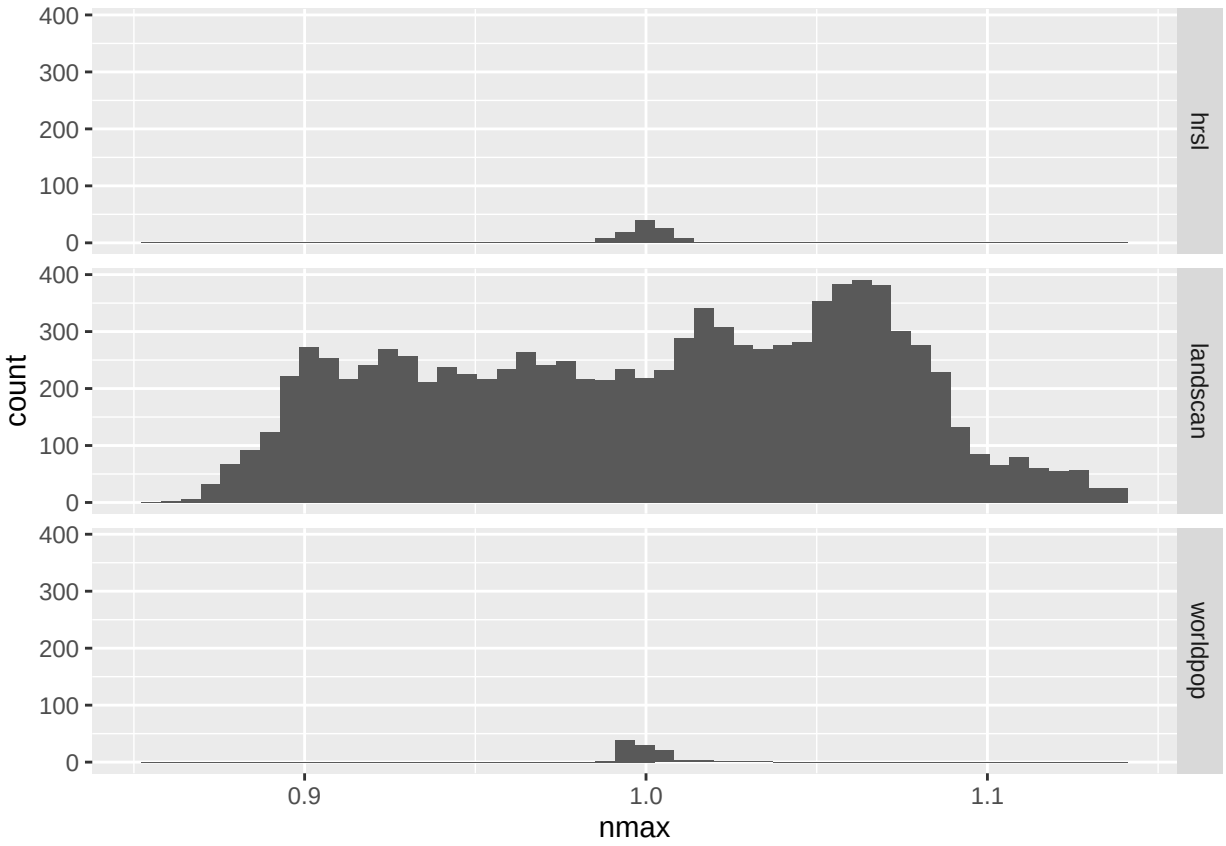
The maximum pixel value, of the population, not the population density. It looks like there's more noise in the coarse grid.

```
ggplot(csvs, aes(x = max_density, group = grid)) +
  geom_histogram(bins = 200) +
    facet_grid(grid ~ .)
```



Let's look more closely by dividing by the mean in each case. We see that they all have about the same uncertainty in the maximum when you divide by the mean.

```
mean_max <- aggregate(max_density ~ grid, FUN = mean, data = csvs)
names(mean_max)[names(mean_max) %in% "max_density"] <- "mean_max"
with_mean <- merge(csvs, mean_max)
with_mean["nmax"] <- with_mean$max_density / with_mean$mean_max
ggplot(with_mean, aes(x = nmax, group = grid)) +
  geom_histogram(bins = 50) +
    facet_grid(grid ~ .)
```



OK, so the plots aren't terribly skewed. We should use regular confidence intervals instead of bootstrapping. Maybe small values should use a Wald. Let's make values with confidence intervals for each of these datasets and take a look.

```
add_ci_to_column <- function(df, column) {
  names(df)[names(df) %in% column] <- "target"
  counted <- aggregate(target ~ grid, data = df, FUN = function(x) sum(!is.na(x)))
  names(counted)[names(counted) %in% "target"] <- "n"

  mean_col <- aggregate(target ~ grid, FUN = mean, data = df)
  names(mean_col)[names(mean_col) %in% "target"] <- "mean_val"

  sd_col <- aggregate(target ~ grid, FUN = sd, data = df)
  names(sd_col)[names(sd_col) %in% "target"] <- "sd_val"

  with_mean <- merge(merge(counted, mean_col), sd_col)
  lower <- paste(column, "low", sep = "_")
  upper <- paste(column, "up", sep = "_")
  with_mean[lower] <- with(with_mean, mean_val - 1.96 * sd_val / sqrt(n))
  with_mean[upper] <- with(with_mean, mean_val + 1.96 * sd_val / sqrt(n))
  names(with_mean)[names(with_mean) %in% "mean_val"] <- column
  with_mean[names(with_mean) %in% c(column, lower, upper, "grid")]
}

add_sd_to_column <- function(df, column) {
  names(df)[names(df) %in% column] <- "target"
  counted <- aggregate(target ~ grid, data = df, FUN = function(x) sum(!is.na(x)))
```

```
  names(counted)[names(counted) %in% "target"] <- "n"

  mean_col <- aggregate(target ~ grid, FUN = mean, data = df)
  names(mean_col)[names(mean_col) %in% "target"] <- column

  sd_col <- aggregate(target ~ grid, FUN = sd, data = df)
  sd_name <- paste(column, "sd", sep = "_")
  names(sd_col)[names(sd_col) %in% "target"] <- sd_name
  together <- merge(mean_col, sd_col, by = "grid")
  together[names(together) %in% c(column, sd_name, "grid")]
}

zdf <- data.frame(
  rate = c(rnorm(10, mean = 0.5, sd = 0.2), rnorm(10, mean = 0.7, sd = 0.1)),
  grid = c(rep("HRSL", 10), rep("LandScan", 10)),
  stringsAsFactors = FALSE
)

# A little test
zzdf <- add_ci_to_column(zdf, "rate")
stopifnot(nrow(zzdf) == 2)
stopifnot(all(zzdf$rate_low < zzdf$rate))
stopifnot(all(zzdf$rate < zzdf$rate_up))

zzdf <- add_sd_to_column(zdf, "rate")
zzdf
```

grid	rate	rate_sd
HRSL	0.5047082	0.1743487
LandScan	0.7083836	0.1060346

```
consider_error <- c("max_density", "empty_percent", "urban_fit", "pareto_fraction")
stopifnot(all(consider_error %in% names(csvs)))
single_ci <- function(col) add_sd_to_column(csvs, col)
ans_df <- single_ci(consider_error[1])
for (col in consider_error[2:length(consider_error)]) {
  ans_df <- merge(ans_df, single_ci(col))
}
ans_df
```

grid	max_density	max_density_sd	empty_percent	empty_percent_sd	urban_fit	urban_fit_sd	pareto_fr
hrsl	22153.31	124.5085	99.12567	0.0014593	0.0163673	0.0000069	0.0
landscan	19179.29	1233.6366	88.45034	0.1822979	0.0167807	0.0008261	0.0
worldpop	21149.80	160.1162	98.08220	0.0088654	0.0164180	0.0000270	0.0

3.5 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.

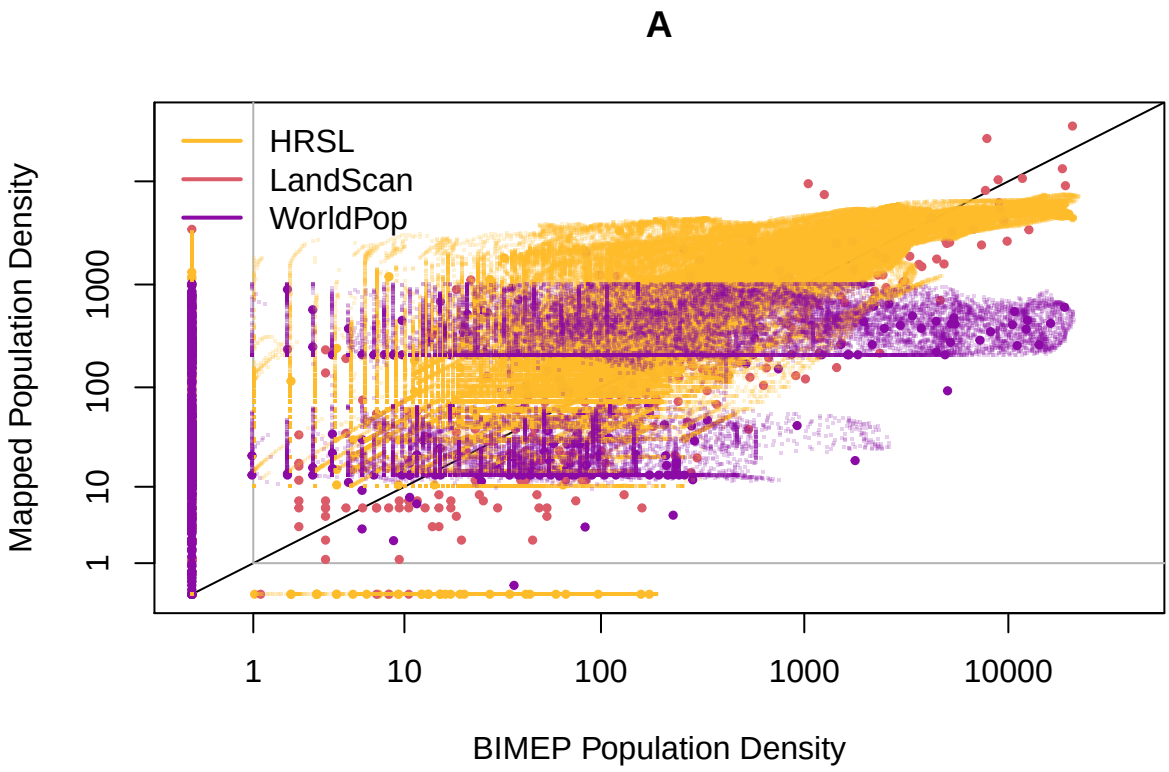
Set up consistent colors for the different data sets.

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

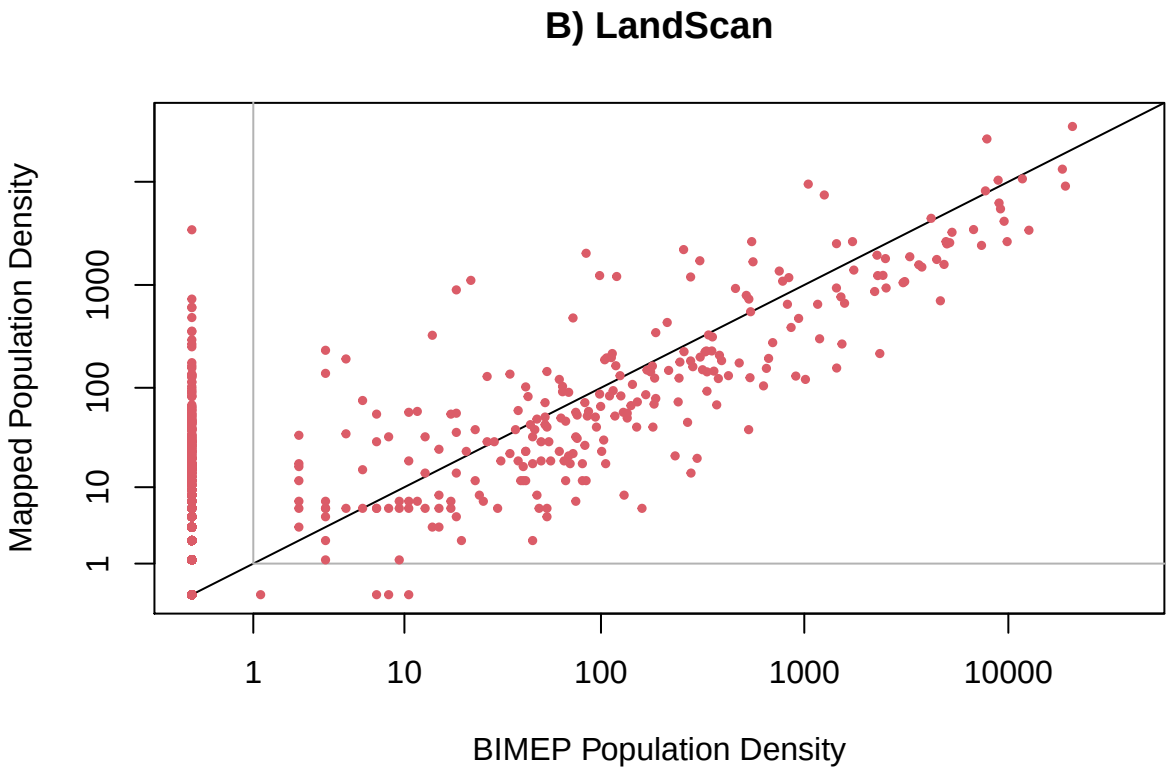
```
logoff <- log_offset(1)
symbol_size <- c(0.5, 0.25)
names(symbol_size) <- c("coarse", "fine")
symbol_choice <- c(19, 15)
names(symbol_choice) <- c("coarse", "fine")


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, counts = 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"])

  if (is.null(counts)) {
    data_function <- density_from_source_resolution
  } else {
    data_function <- xy_from_source_resolution
  }

  for (plot_idx in 1:length(y_name)) {
    y_source <- y_name[plot_idx]
    y_resolution <- resolution[plot_idx]
    xy <- data_function(y_source, y_resolution)
    cex <- symbol_size[as.character(y_resolution)]
    pch <- symbol_choice[as.character(y_resolution)]
    if (resolution[plot_idx] == "coarse") {
      color <- colors[y_source]
    } else {
      color <- scales::alpha(colors[y_source], 0.2)
    }
    if (!is.null(background) & plot_idx %in% background) {
      color <- colors["grey"]
    }
    points(xy, col = color, cex = cex, pch = pch)
  }
}
```

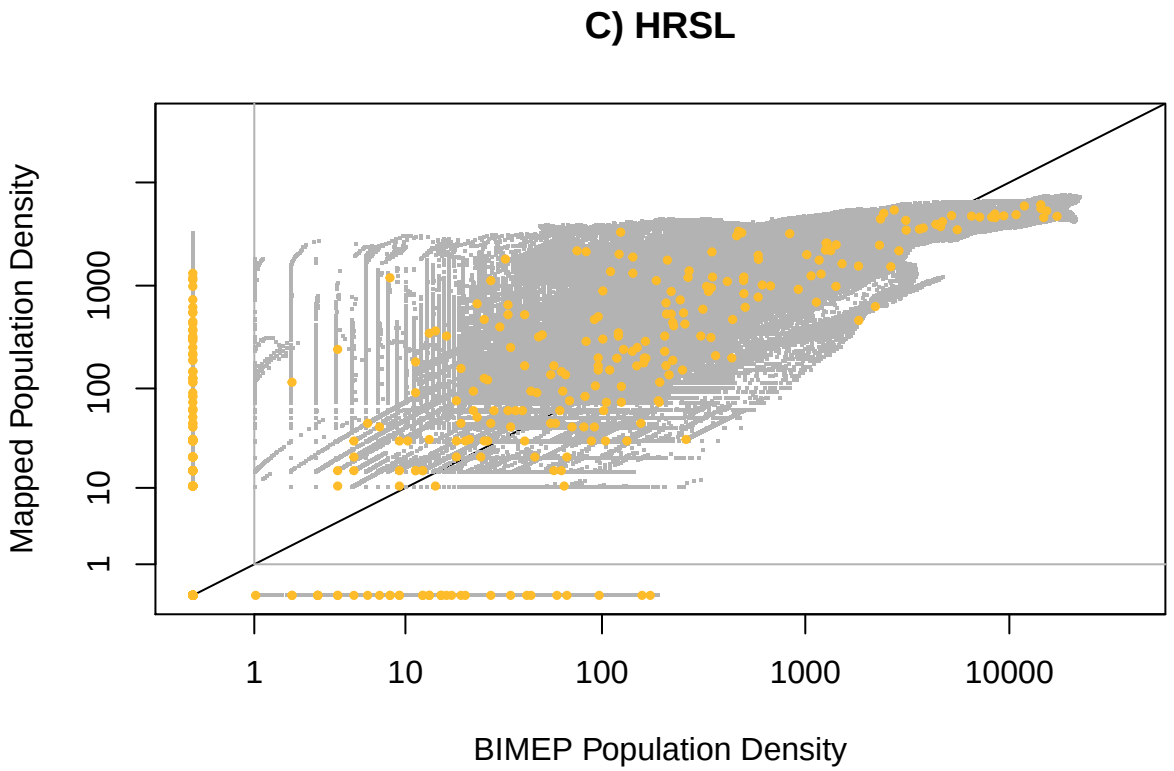
```
density_label <- c("BIMEP Population Density", "Mapped Population Density")
maximum1k <- 1.01 * max(density_from_source_resolution("LandScan", "coarse")$y, na.rm = TRU

scatter1k_together <- function() {
  maximum1k <- 1.01 * max(density_from_source_resolution("LandScan", "coarse")$y, na.rm = T
  the_three <- c("HRSL", "LandScan", "WorldPop")
  pop_scatter(
    c(the_three, "HRSL", "WorldPop"), c(rep("coarse", 3), rep("fine", 2)),
    density_label, "A", maximum1k)
  legend("topleft", legend =  the_three, col = colors[the_three],
              lty = 1, lwd = 2, cex = 1, bg = "white", bty = "n")
}
scatter1k_together()
```



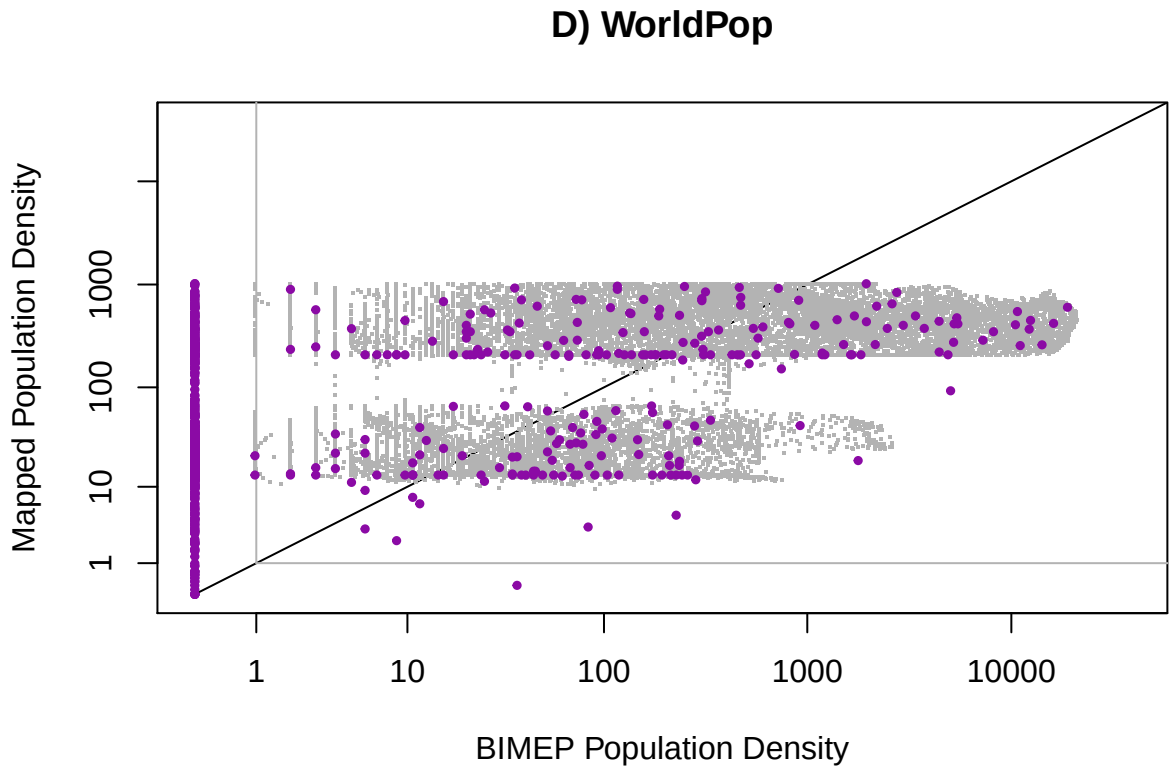

```
scatter1k_ls <- function() pop_scatter(
  "LandScan", "coarse",
  density_label, "B) LandScan", maximum1k)
scatter1k_ls()
```



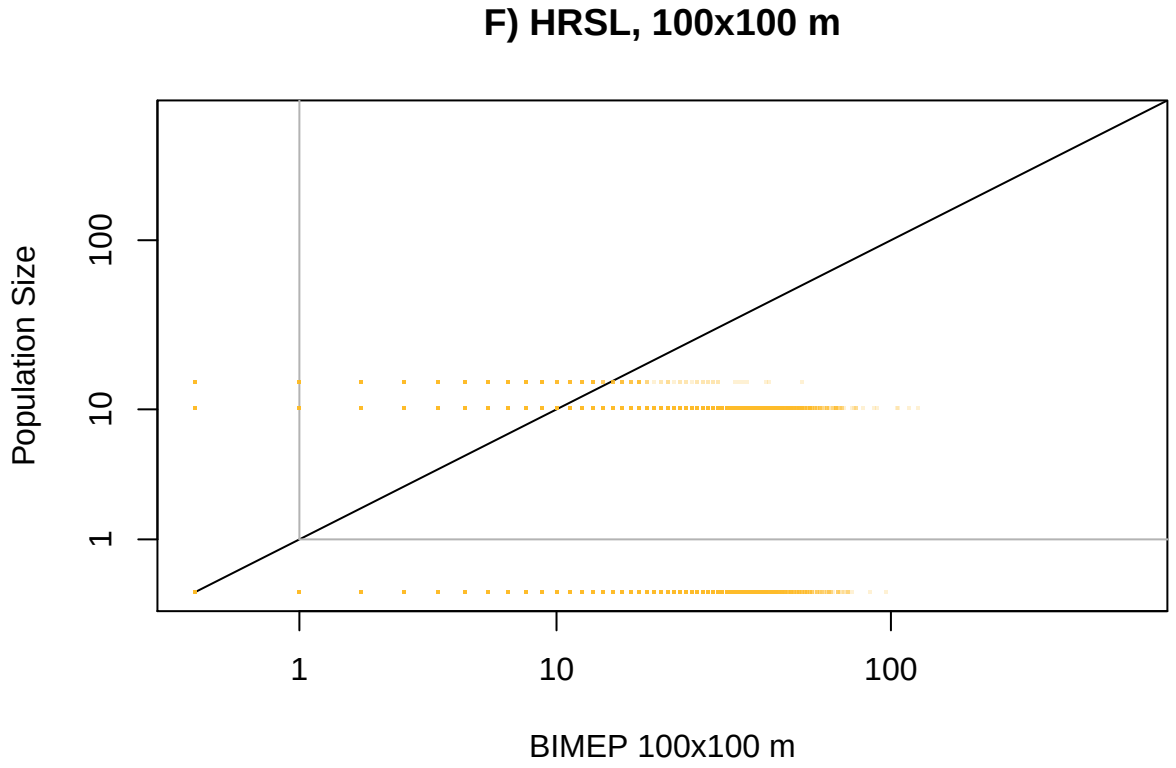
```
scatter1k_hrsl <- function() pop_scatter(
  rep("HRSL", 2), c("fine", "coarse"),
  density_label, "C) HRSL", maximum1k, background = 1)
scatter1k_hrsl()
```



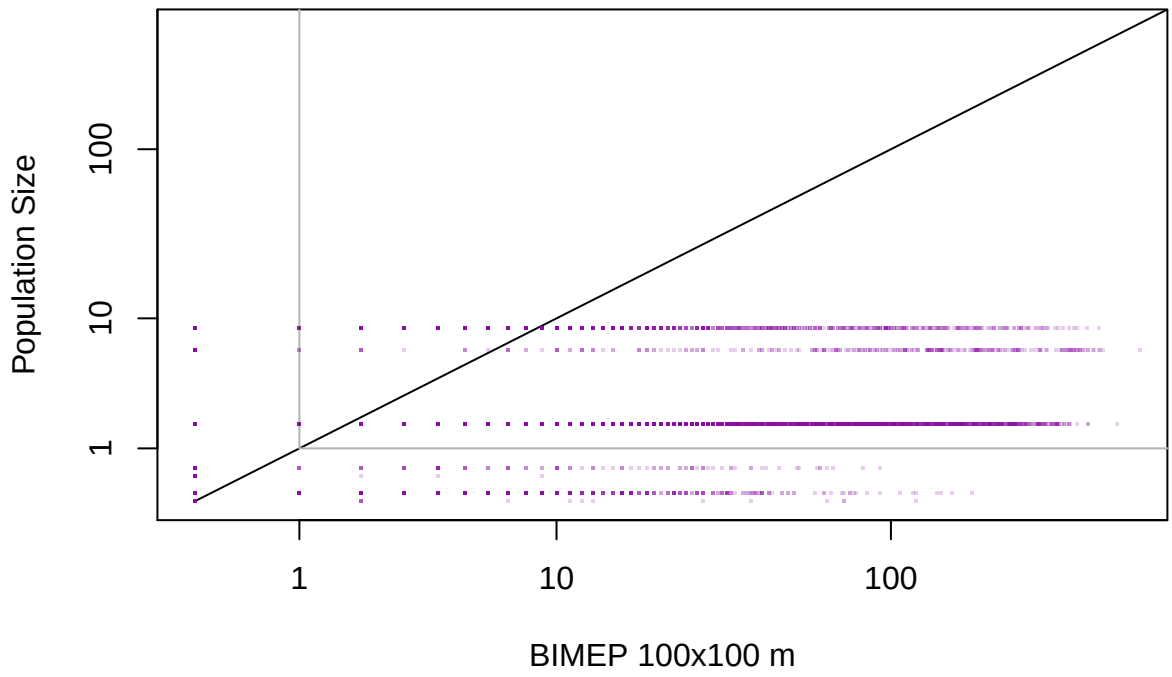

```
scatter1k_wp <- function() pop_scatter(
  rep("WorldPop", 2), c("fine", "coarse"),
  density_label, "D) WorldPop", maximum1k, background = 1)
scatter1k_wp()
```



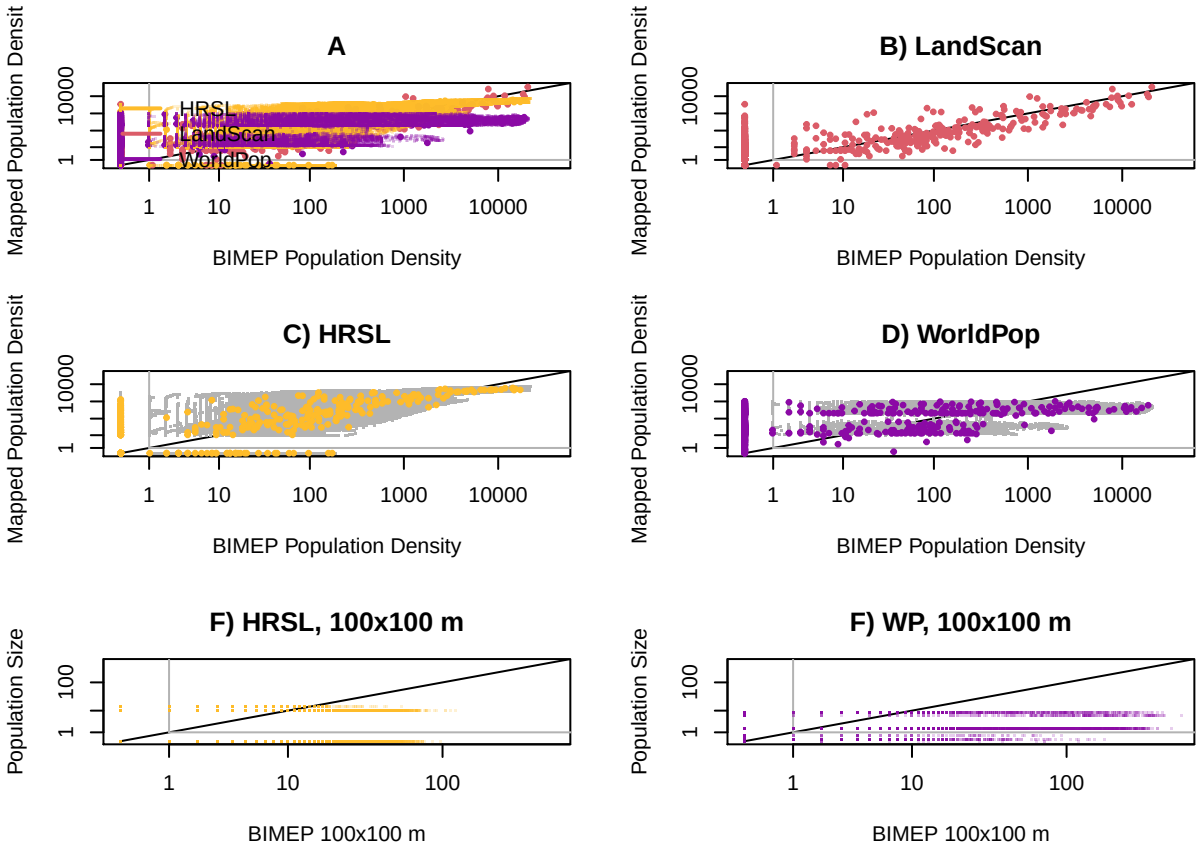

```
maximum100 <- 6.2
pop_scatter_label <- c("BIMEP 100x100 m", "Population Size")
scatter1k_hrsl_count <- function() pop_scatter(
  rep("HRSL", 1), c("fine"),
  pop_scatter_label, "F) HRSL, 100x100 m", maximum100, counts = TRUE)
scatter1k_hrsl_count()
```




```
maximum100 <- 6.2
pop_scatter_label <- c("BIMEP 100x100 m", "Population Size")
scatter1k_wp_count <- function() pop_scatter(
  rep("WorldPop", 1), c("fine"),
  pop_scatter_label, "F) WP, 100x100 m", maximum100, counts = TRUE)
scatter1k_wp_count()
```




```
make_figure <- function() {
  par(mfrow = c(3, 2), mar = c(5, 4.2, 3, 2))
  scatter1k_together()
  scatter1k_ls()
  scatter1k_hrsl()
  scatter1k_wp()
  scatter1k_hrsl_count()
  scatter1k_wp_count()
}
popbioko::save_plot(make_figure, "scatter1k")
make_figure()
```



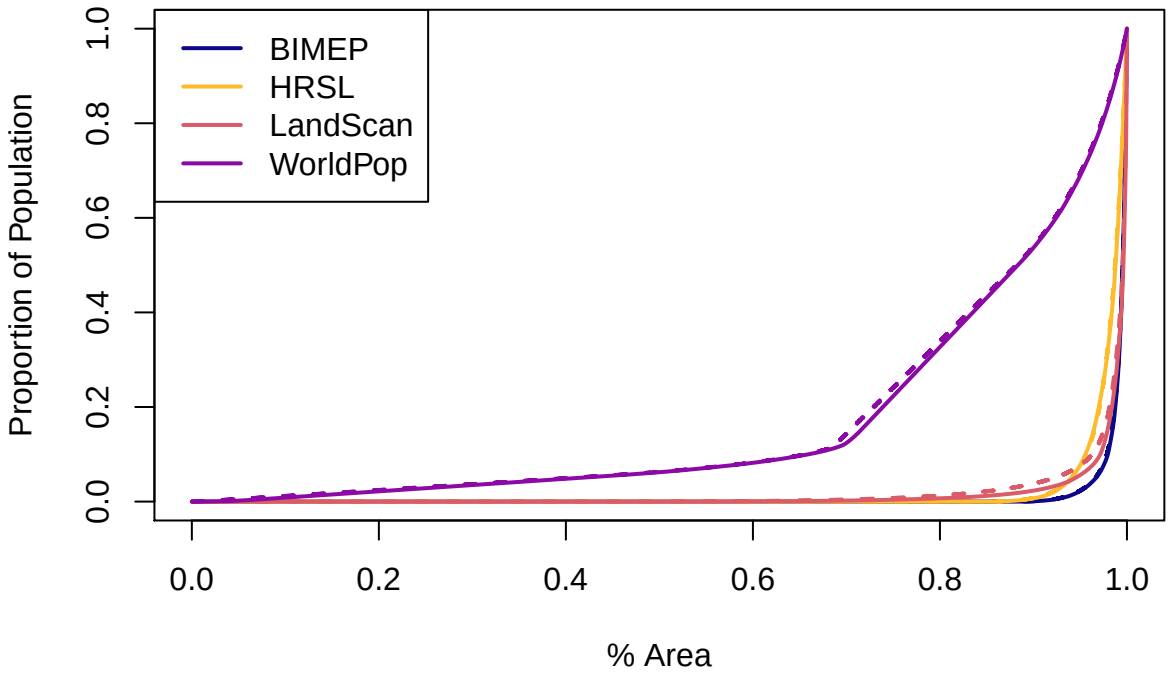
5 Cumulative Maps

This section builds the cumulative maps in figure 5. It separates the work into one part to generate x and y data and one part to decide axes and such for that data.

```
linear_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 = x, y = y)
}

sources <- c("BIMEP", "HRSL", "LandScan", "WorldPop")

# This draws each line for the plots, taking care of color and line type.
lines_for_datasets <- function(datasets, linewidth) {
  for (data_idx in 1:length(datasets)) {
    lines(
      datasets[[data_idx]][["x"]],
      datasets[[data_idx]][["y"]],
      type = "l",
      col = colors[datasets[[data_idx]][["source"]]],
      lwd = linewidth,
      lty = ifelse(datasets[[data_idx]][["resolution"]] == "coarse", 1, 2)
```

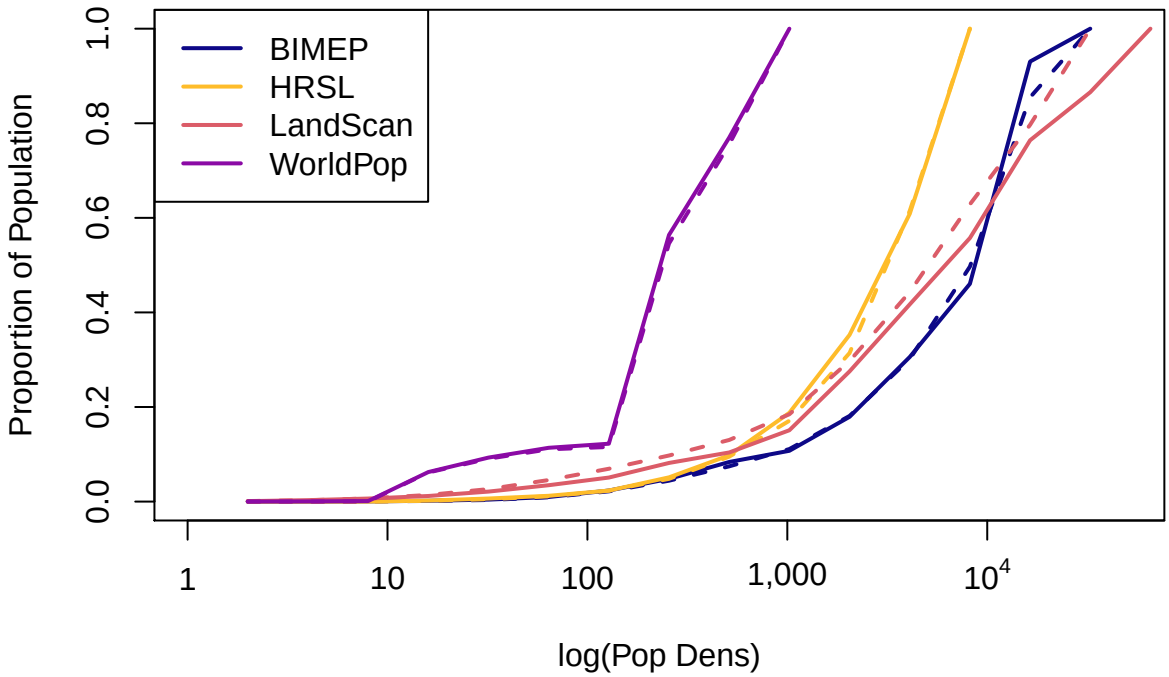
```
      )
  }
}
```

```
cdfAreaLines <- function(H, source, resolution, linewidth = 2) {
  H <- sort(H)
  area <-  c(1:length(H)) / length(H)
  lines(
    area,
    cumsum(H) / sum(H),
    col = colors[source],
    lwd = linewidth,
    lty = ifelse(resolution == "coarse", 1, 2)
    )
}

cdfAreaFigure <- function(llwd = 1, label = "") {
  plot(
    vector(mode = "numeric", length = 0),
    xlim = c(0, 1), ylim = c(0, 1),
    xlab =  "% Area", ylab =  "Proportion of Population",
    main = label
    )
  sources <- c("BIMEP", "HRSL", "LandScan", "WorldPop")
  for (source in sources) {
    for (resolution in c("coarse", "fine")) {
      if (source == "BIMEP") {
        data <- linear_from_source_resolution("HRSL", resolution, density)$x
      } else {
        data <- linear_from_source_resolution(source, resolution, density)$y
      }
      cdfAreaLines(data, source, resolution, linewidth = llwd)
    }
  }
  legend("topleft", legend =  sources, col = colors[sources],
         lty = 1, lwd = 2, cex = 1)
}

cdfAreaFigure(2)
```




```
cdfDens <- function(H, log_base) {
  H <- sort(H)
  mx <- ceiling(log(max(H), base = log_base))

  c2 <- 1:mx
  ixz <- which(H == 0)
  Pz <- length(ixz)
  for (i in c2) {
    ix <- which(H < log_base^i)
    Pz <- c(Pz, sum(H[ix]))
  }
  pop <- c(0, log_base^c2)
  cbind(pop, Pz)
}


logPopAreaLines <- function(H, source, resolution, log_base, linewidth = 2) {
  dd <- cdfDens(H,log_base)[-1,]
  P <- dd[,1]
  H <- dd[,2]
  l <- log10(max(H))

  lines(
    P,
    H / max(H),
    col = colors[source],
    lwd = linewidth,
    lty = ifelse(resolution == "coarse", 1, 2)
    )
}
```

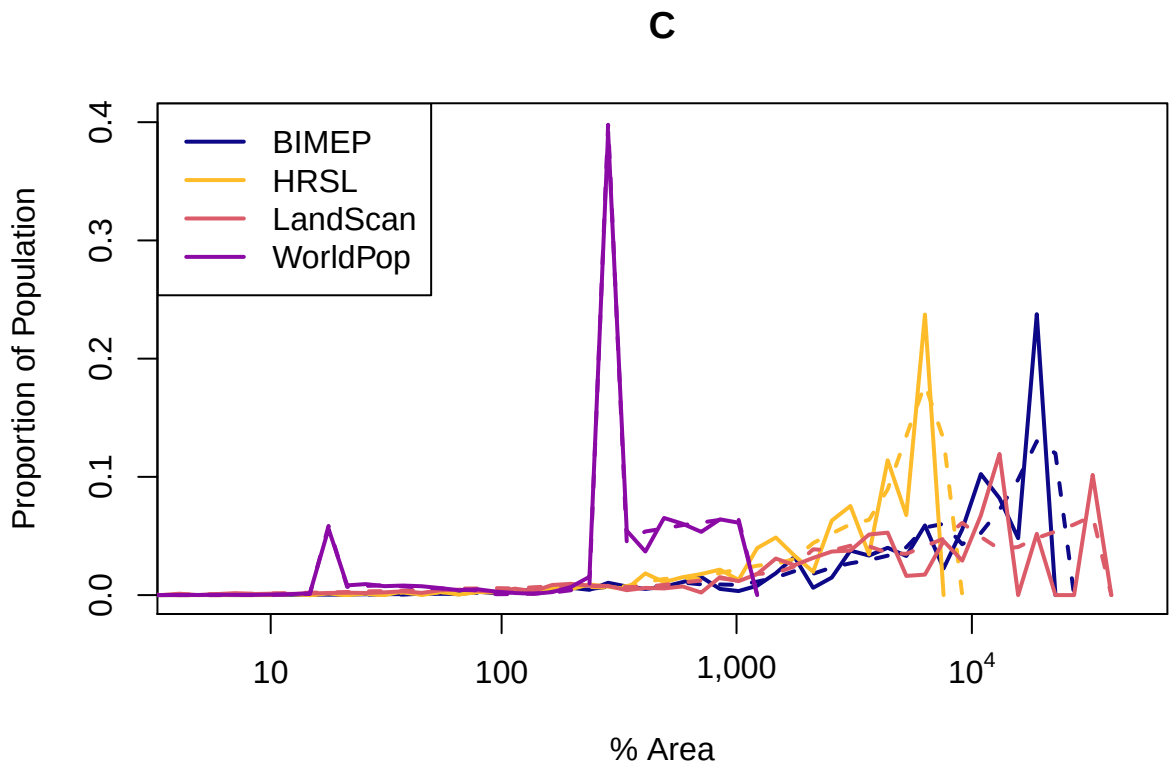
```r
logPopAreaFigure <- function(llwd = 1, log_base = 2, label = "") {
  plot(
    vector(mode = "numeric", length = 0),
    log = "x",
    xlim = c(1.05, 0.5 * 10^5),
    ylim = c(0, 1),
    xlab =  "log(Pop Dens)",
    ylab =  "Proportion of Population",
    xaxt = "n",
    main = label
    )

  log_max <- 0
  for (source in sources_ordered) {
    for (resolution in c("coarse", "fine")) {
      if (source == "BIMEP") {
        data <- linear_from_source_resolution("HRSL", resolution, density)$x
      } else {
        data <- linear_from_source_resolution(source, resolution, density)$y
      }
      log_max <- max(log_max, log10(max(data, na.rm = TRUE)))
      logPopAreaLines(data, source, resolution, log_base, linewidth = llwd)
    }
  }
  axis(1, 10^(0:(log_max + 1)), logarithmic_axis_labels[1:(log_max + 2)])
  legend("topleft", legend =  sources, col = colors[sources],
         lty = 1, lwd = 2, cex = 1)
}

logPopAreaFigure(2)
```



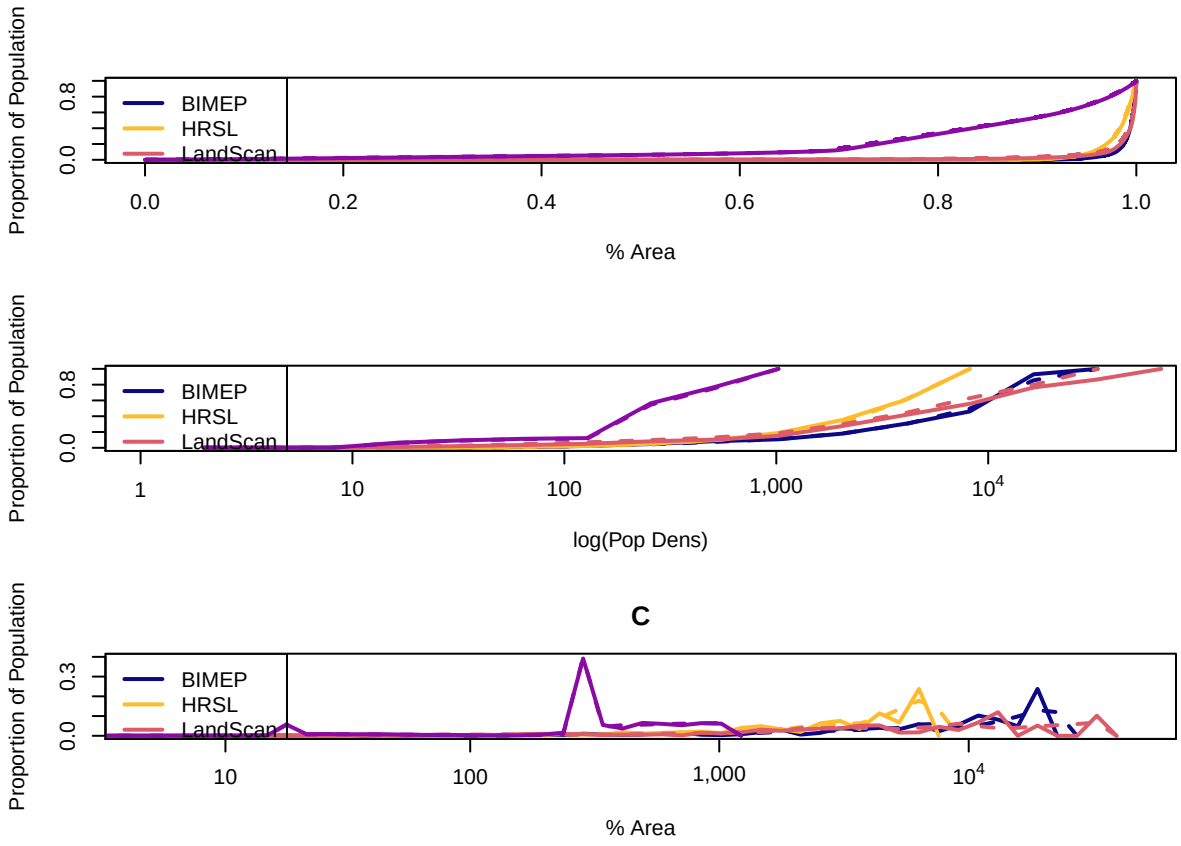
Now the third part of Figure 5, a density plot that is just density, not cumulative.

```r
log_density <- function(H, base_power = 2) {
  H <- H[!is.na(H)]
  c2 <- ceiling(max(log(H[H > 0], base = base_power)))
  ixz <- which(H <= 0)
  upper <- 0
  HH <- H[-ixz]
  Pz <- length(ixz)
  for (i in 1:c2) {
    upper <- c(upper, base_power^(i - 1))
    ix <- which(HH < base_power^(i - 1))
    if (length(ix) > 0) {
      Pz <- c(Pz, sum(HH[ix]))
      HH <- HH[-ix]
    } else {
      Pz <- c(Pz, 0)
    }
  }
  while (Pz[length(Pz)] == 0) {
    Pz <- Pz[-length(Pz)]
    upper <- upper[-length(Pz)]
  }
  cbind(round(Pz), upper)
}

single_test <- linear_from_source_resolution("LandScan", "coarse", density)$y
dens_test <- log_density(single_test, base_power = 1.2)


density_kernel_data <- function(base_power, offset = 0) {
  datasets <- vector(mode = "list", length = 2 * length(sources))
  running_idx <- 1
  for (source in sources_ordered) {
    for (resolution in c("coarse", "fine")) {
      if (source == "BIMEP") {
        data <- linear_from_source_resolution("HRSL", resolution, density)$x
      } else {
        data <- linear_from_source_resolution(source, resolution, density)$y
      }
      D <- log_density(data, base_power)[-1, 1]
      l <- length(D)
      x <- base_power^c(c(1:(l + 1))) * base_power^offset
      y <- c(D / sum(data, na.rm = TRUE), 0)
      datasets[[running_idx]] <- list(source = source, resolution = resolution, x = x, y =
      running_idx <- running_idx + 1
    }
  }
  datasets
}


density_kernel_figure <- function(base_power, linewidth = 1, label = "") {
  datasets <- density_kernel_data(base_power)
  l = max(sapply(datasets, function(ds) ds$l))
```

```
  plot(
    vector(mode = "numeric", length = 0),
    xaxt = "n",
    log = "x",
    xlim = c(5, base_power^(l + 2)),
    ylim = c(0, 0.4),
    xlab =  "% Area", ylab =  "Proportion of Population",
    main = label
    )
  lines_for_datasets(datasets, linewidth)
  L <- 4
  axis(1, 10^(0:(L + 1)) + 0.5, logarithmic_axis_labels[1:(L + 2)])
  legend("topleft", legend =  sources, col = colors[sources],
         lty = 1, lwd = 2, cex = 1)
}

density_kernel_figure(1.2, 2, "C")
```




```
cdf_figure <- function() {
  par(mfrow = c(3, 1), mar = c(5, 5, 3, 2))
  cdfAreaFigure(2)
  logPopAreaFigure(2)
  density_kernel_figure(1.2, 2, "C")
}
save_plot(cdf_figure, "CDF")
cdf_figure()
```



6 Proportion, Accuracy, Recall, and Precision

Figure 6 is proportion, accuracy, recall, and precision.

6.1 Define the values

The following functions in compute the specificity and sensitivity of a map compared with a gold standard:

```
breakpoints <- c(
  empty = 0,
  rural1 = 1,
  rural2 = 50,
  periurban = 250,
  urban = 1000
)
break_colors <- c(
  empty = grey(0.95),
  rural1 = "ivory",
  rural2 = "lavender",
  periurban = "cornsilk",
  urban = "aliceblue"
)
stopifnot(all(names(breakpoints) == names(break_colors)))

lessThanCat <- function(tau, map, gold, km2 = 1) {
  tau <- tau * km2
```

```
  accuracy_profile(truth_table(function(observed) {observed < tau}, gold, map))
}

greaterThanCat <- function(tau, map, gold, km2 = 1) {
  tau <- tau * km2
  accuracy_profile(truth_table(function(observed) {tau <= observed}, gold, map))
}

betweenCat <- function(L, U, map, gold, km2 = 1) {
  L <- L * km2
  U <- U * km2
  # Choose comparison so that it is cadlag.
  accuracy_profile(truth_table(function(obs) {L <= obs & obs < U }, gold, map))
}

fullCat <- function(map, gold, km2 = 1) {
  with(as.list(breakpoints), {
    data.frame(as.vector(rbind(
      empty = lessThanCat(rural1, map, gold, km2),
      rural1 = betweenCat(rural1, rural2, map, gold, km2),
      rural2 = betweenCat(rural2, periurban, map, gold, km2),
      periurban = betweenCat(periurban, urban, map, gold, km2),
      urban = greaterThanCat(urban, map, gold, km2)
    )))
  })
}
```

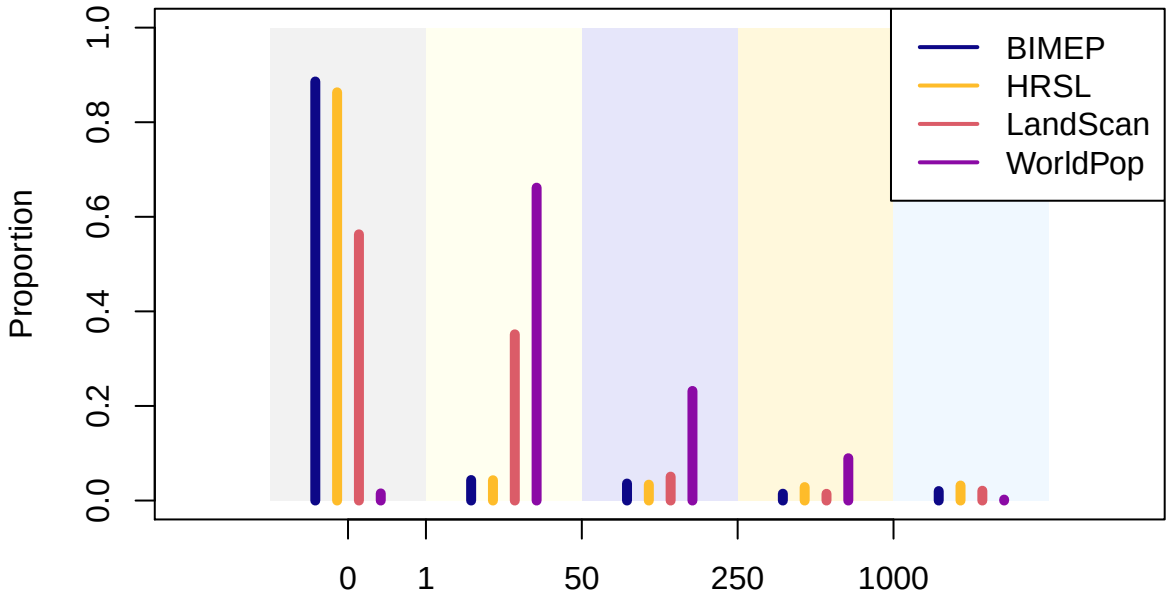
6.2 Proportion

```
landAA <- function(H, breakpoints, area = 1) {
  stopifnot(class(breakpoints) == "numeric")
  breakpoints <- c(breakpoints, Inf) / area

  Pz <- rep(0, length(breakpoints))
  for (cut_idx in 1:length(breakpoints)) {
    ixz <- which(H < breakpoints[cut_idx])
    Pz[cut_idx] <- length(ixz)
    if (Pz[cut_idx] > 0) {
      H <- H[-ixz]
    }  # Nothing to remove
  }
  Pz / sum(Pz)
}

over_arrays <- function(transform) {
  datasets <- vector(mode = "list", length = 2 * length(sources))
  running_idx <- 1
  for (source in sources) {
    for (resolution in c("coarse", "fine")) {
      if (source == "BIMEP") {
        data <- linear_from_source_resolution("HRSL", resolution, density)$x
      } else {
        data <- linear_from_source_resolution(source, resolution, density)$y
```

```
      }
      datasets[[running_idx]] <- list(source = source, resolution = resolution, x = transfo
      running_idx <- running_idx + 1
    }
  }
  datasets
}

proportion_data <- function(breakpoints) {
  over_arrays(function(data) {
    landAA(data, breakpoints)
  })
}
example_data <- proportion_data(c(1, 50, 250, 1000))
```

```
#' Make shaded backgrounds for all four figures.
#' @param splits The sides of the boxes, where colors change.
make_boxes <- function(splits) {
  stopifnot(length(splits) == length(break_colors) + 1)
  boxes <- vector(mode = "list", length = length(break_colors))
  for (color_idx in 1:length(break_colors)) {
    boxes[[color_idx]] <- list(splits[color_idx], splits[color_idx + 1], break_colors[colo
  }
  makebox <- function(x1, x2, color) {
    xx <- c(x1, x1, x2, x2)
    yy <- c(0,1,1,0)
    polygon(cbind(xx, yy), border = NA, col = color)
  }
  for (box_idx in 1:length(boxes)) do.call(makebox, boxes[[box_idx]])
}


pdfAreaFigure <- function(lb = "") {
  # Start from 2 because this function uses the upper cutoff.
  prop_data <- proportion_data(breakpoints[2:length(breakpoints)])
  plot(
    vector(mode = "numeric", length = 0),
    xlim = c(1, 7),
    ylim = c(0, 1),
    xaxt = "n",
    xlab = "",
    ylab = "Proportion",
    main = ""
    )
  axis(1, c(2, 2:5 + 0.5), c(0,1,50,250,1000))

  make_boxes(1:6 + 0.5)

  offset_idx <- 0
  y_data <- NULL
  for (find_source in sources_ordered) {
    for (data_idx in 1:length(prop_data)) {
      y_data <- with(prop_data[[data_idx]], {
        if ((source == find_source) & (resolution == "coarse")) {
```

```
            return(x)
          }
        })
        if (!is.null(y_data)) break
      }
      offset <- -.21 + 0.14 * offset_idx
      stopifnot(!is.null(y_data))
      lines(2:6 + offset, y_data, type = "h", lwd = 5, col = colors[find_source], lty = 1)
      offset_idx <- offset_idx + 1
    }

    legend("topright", legend = sources_ordered, col = colors[sources_ordered],
           lty = 1, lwd = 2, cex = 1, bg = "white")
}

pdfAreaFigure()
```



6.3 Precision

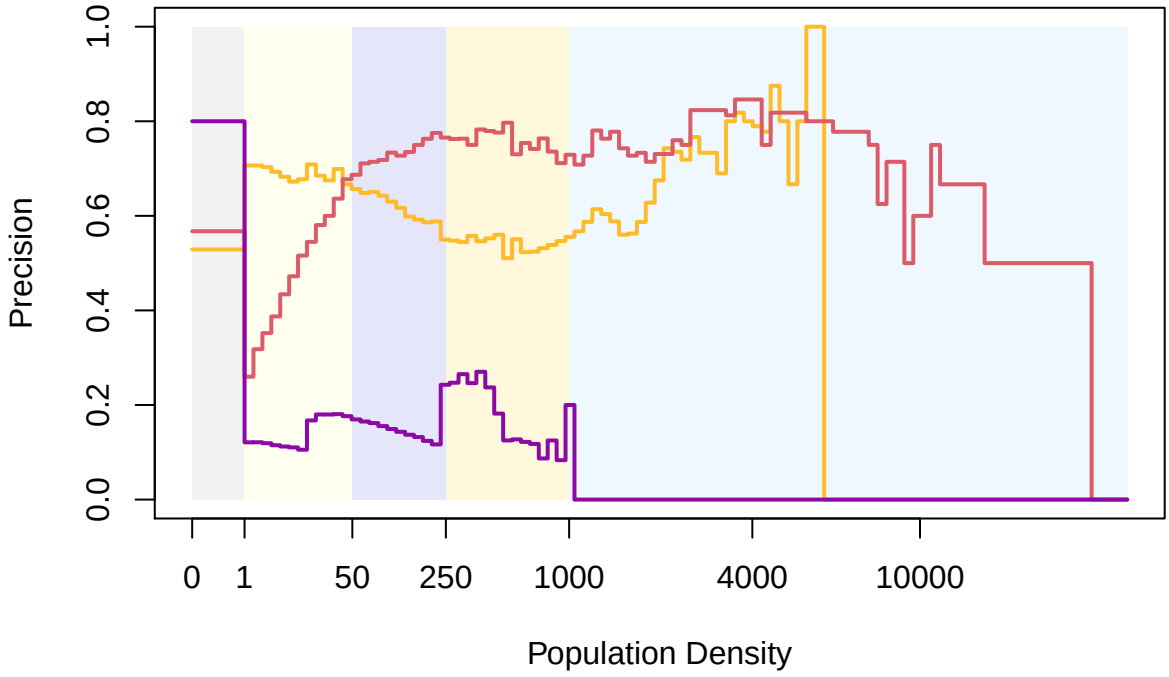
Precision is also known as positive predictive value, so it's the ppv variable.

```
get_measure <- function(x, map, gold, measure) {
  if (x == 0) {
    lessThanCat(x, map, gold)[[measure]]
  } else {
    greaterThanCat(x, map, gold)[[measure]]
  }
}
getPPV <- function(x, map, gold) get_measure(x, map, gold, "ppv")
```

```
compare_sources <- sources[!sources %in% "BIMEP"]
ppv_data <- sapply(compare_sources, function(x) {
  linear_from_source_resolution(x, "coarse", density)
})
dimnames(ppv_data) <- list(c("x", "y"), compare_sources)
```

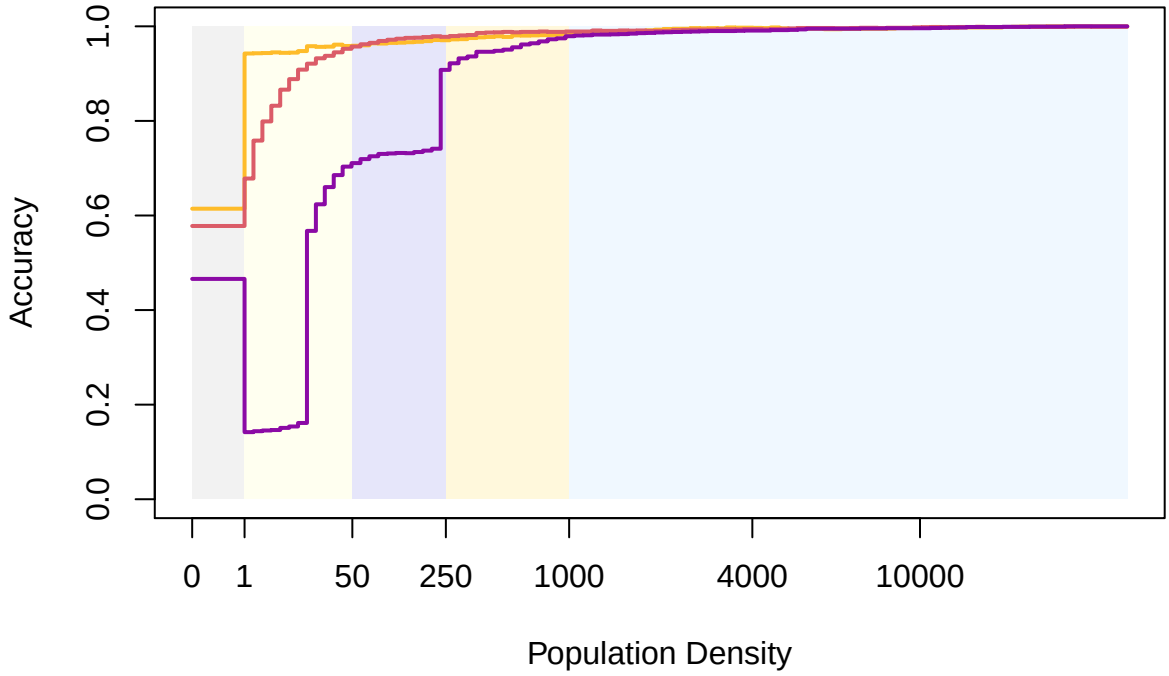
```
mn <- 1
ppv_mx <- 0.7 * max(ppv_data["y", "LandScan"][[1]], na.rm = TRUE)
tk <- c(breakpoints, 4000, 10000)

xx <- c(0, seq(sqrt(mn), sqrt(ppv_mx), length.out = 100)^2)
xxp <- sqrt(xx)
tk0 <- sqrt(tk)

pw <- 1 / 3.5
xx <- c(0, seq(mn^pw, ppv_mx^pw, length.out = 100)^(1 / pw))
xxp <- xx^pw
tk0 <- tk^pw

ppv_splits <- c(breakpoints, ppv_mx)^pw

generate_accuracy_y <- function(xx, measure) {
  ppv_y <- matrix(0, nrow = length(xx), ncol = length(compare_sources))
  for (source_idx in 1:length(compare_sources)) {
    source <- compare_sources[source_idx]
    gold <- ppv_data["x", source][[1]]
    observed <- ppv_data["y", source][[1]]
    ppv_y[1:length(xx), source_idx] <- as.vector(unlist(sapply(
      xx,
      function(x, map, gold) get_measure(x, map, gold, measure),
      map = observed,
      gold = gold
      )))
  }
  colnames(ppv_y) <- compare_sources
  ppv_y
}
ppv_y <- generate_accuracy_y(xx, "ppv")
```

```
PPVProfile <- function(ppv_y, name, lb = "") {
  plot(
    vector(mode = "numeric", length = 0),
    xlim = c(0, max(xxp)),
    ylim = c(0, 1),
    xaxt = "n",
    xlab = "Population Density",
    ylab = name,
    main = lb
    )
  axis(1, tk0, tk)

  make_boxes(ppv_splits)

  for (col_idx in 1:ncol(ppv_y)) {
    source <- colnames(ppv_y)[col_idx]
    lines(xxp, as.numeric(ppv_y[, col_idx]), type = "s", col = colors[source], lwd = 2)
  }
}
PPVProfile(ppv_y, "Precision")
```



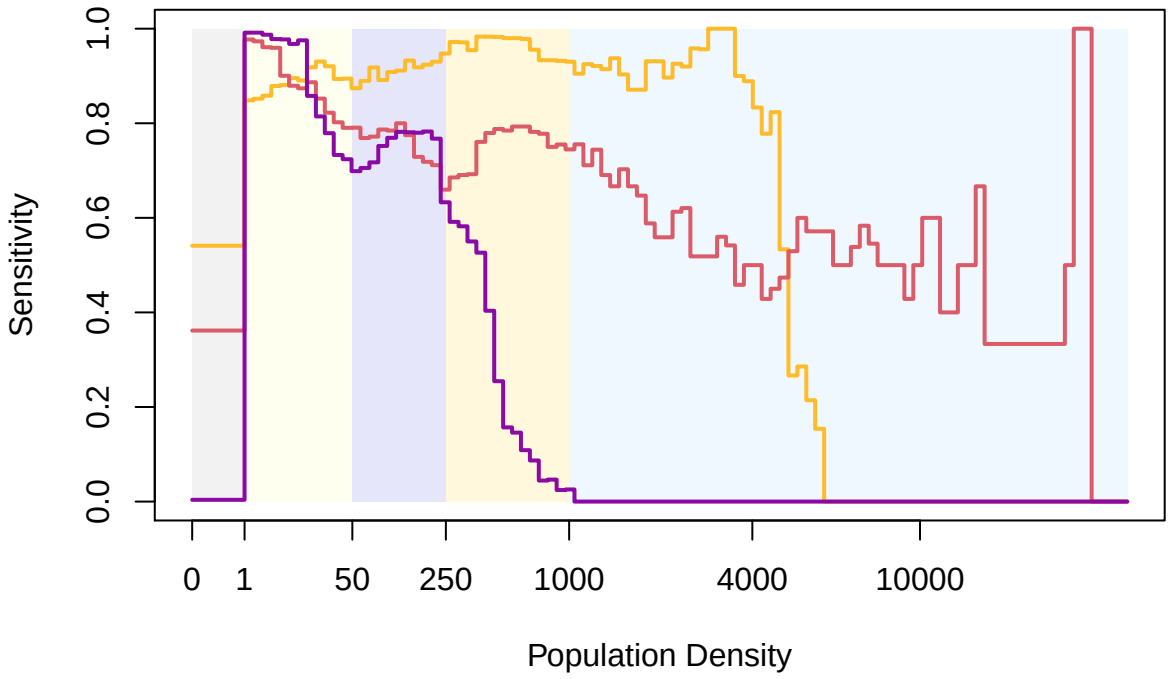
Accuracy

```
acc_y <- generate_accuracy_y(xx, "acc")
PPVProfile(acc_y, "Accuracy")
```

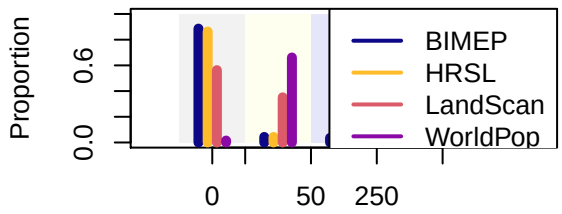


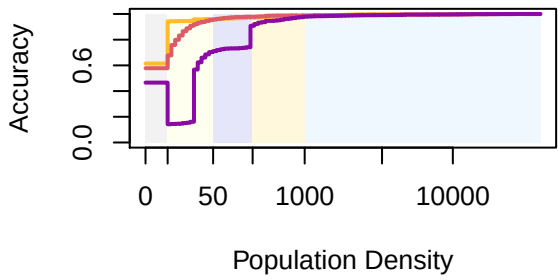
Recall

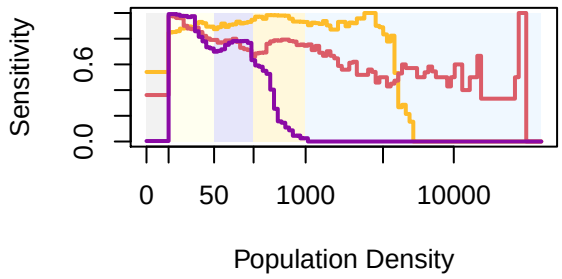
```
sens_y <- generate_accuracy_y(xx, "sens")
PPVProfile(sens_y, "Sensitivity")
```

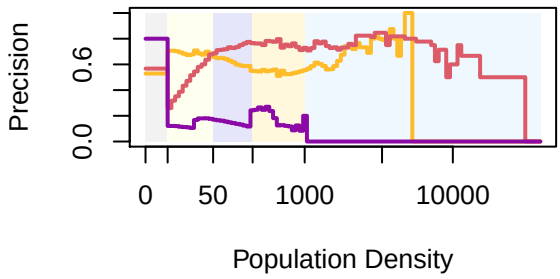



```
area_figure <- function() {
  par(new, mfrow = c(2, 2))
  pdfAreaFigure()
  PPVProfile(acc_y, "Accuracy")
  PPVProfile(sens_y, "Sensitivity")
  PPVProfile(ppv_y, "Precision")
}
save_plot(area_figure, "AccuracyProfile")
area_figure()
```









6.4 Table of ranges

This is table 3, which has accuracy, recall, and precision for ranges of values.

```
HRSL <- fullCat(ppv_data["y", "HRSL"][[1]], ppv_data["x", "HRSL"][[1]])
LS <- fullCat(ppv_data["y", "LandScan"][[1]], ppv_data["x", "LandScan"][[1]])
WP <- fullCat(ppv_data["y", "WorldPop"][[1]], ppv_data["x", "WorldPop"][[1]])

# ppv <- cbind(HRSL = as.vector(HRSL$ppv), LS = as.vector(LS$ppv), WP = as.vector(WP$ppv))
# sens <- cbind(HRSL = as.vector(HRSL$sens), LS = as.vector(LS$sens), WP = as.vector(WP$se
# acc <- cbind(HRSL = as.vector(HRSL$acc), LS = as.vector(LS$acc), WP = as.vector(WP$acc))

ppv <- cbind(HRSL = as.numeric(HRSL$ppv), LS = as.numeric(LS$ppv), WP = as.numeric(WP$ppv)
sens <- cbind(HRSL = as.numeric(HRSL$sens), LS = as.numeric(LS$sens), WP = as.numeric(WP$se
acc <- cbind(HRSL = as.numeric(HRSL$acc), LS = as.numeric(LS$acc), WP = as.numeric(WP$acc)
accuracy_as_table <- signif(cbind(acc, sens, ppv), 3)
rownames(accuracy_as_table) <- rownames(HRSL)
t(accuracy_as_table)
```

```
##       empty rural1 rural2 periurban  urban
## HRSL 0.9430 0.9410 0.9600    0.9700 0.9850
## LS   0.6720 0.6740 0.9530    0.9750 0.9890
## WP   0.1290 0.3440 0.7540    0.9120 0.9800
## HRSL 0.9550 0.3120 0.4160    0.4520 0.9070
## LS   0.6320 0.7500 0.5380    0.2270 0.7450
## WP   0.0124 0.4880 0.3010    0.5150 0.0256
## HRSL 0.9800 0.3150 0.4440    0.2300 0.5740
## LS   0.9970 0.0869 0.3770    0.3030 0.7290
## WP   0.9570 0.0312 0.0542    0.0955 0.2500
```