Dasymetric Mapping: An Introduction in R

Jeremiah J. Nieves, Geographic Data Science Lab - University of Liverpool

January 12, 2022

Contents

1 Overview

In this R-based practical, we will explain what dasymetric disaggregation and modelling is. We will then proceed to walk through the basic procedures of dasymetrically modelling gridded population datasets, from census-based data, and some of the procedures to get the gridded population data ready for further uses. We will also briefly cover an “intelligent” dasymetric disaggregation which utilises machine learning to inform the redistribution of population into grids and explore potential ways of integrating other statistical modelling engines, various parameterisations, and niche applications.

1.1 Objectives

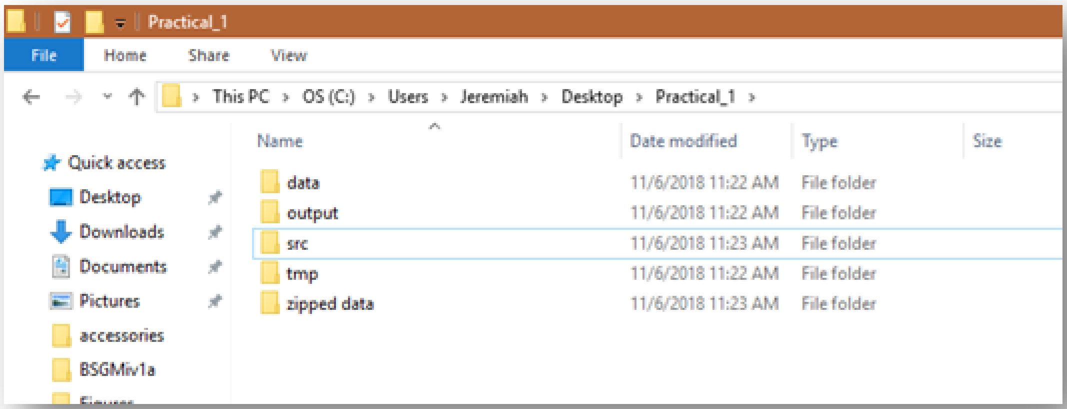
- Understand the differences between a “people per pixel” (ppp) and a “people per Hectare” (pph) gridded population dataset and acquire the skills to change between these population count (ppp) and population density (pph) datasets
- Learn about dasymetric disaggregation of count data and the use of “intelligent” dasymetric disaggregation as applied to gridded population modelling.

1.2 Prerequisites

Basic literacy with computers with regards to folder and file navigation, unzipping compressed folders, and basic familiarity with R, such as running a command in the interactive window.

2 Preparation

Open up the `../Practical_1/` folder and familiarize yourself with the structure and layout as the remainder of the instructions make reference to these folders and the files contained within them. Here the `..` refers to the directory where you have the unzipped workshop folder (`C:/Users/Jeremiah/Desktop/` in the example of Figure 1).



Name	Date modified	Type	Size
data	11/6/2018 11:22 AM	File folder	
output	11/6/2018 11:22 AM	File folder	
src	11/6/2018 11:23 AM	File folder	
tmp	11/6/2018 11:22 AM	File folder	
zipped data	11/6/2018 11:23 AM	File folder	

Figure 1: Example of file structure within the unzipped, supplied workshop folder.

- `../Practical_1/data/` contains the data we will be using throughout this exercise
- `../Practical_1/output/` is where we will place our finalized, desirable output data
- `../Practical_1/tmp/` is where we will place any intermediary files we have to make
- `../Practical_1/src/` is for the advanced practical and can be ignored for this exercise
- `../Practical_1/zipped_data.zip/` contains a compressed copy of the data in case you need to start over because somehow your original data became overwritten

We will go ahead and set the base folder path to the `GISRUK_2022_Training` folder and refer to it as `root`. For instance on my computer I set:

`‘root <- C:/Users/Jeremiah/Desktop/Practical_1/‘`

2.1 Installing Packages

We need to install the following packages prior to carrying out the practical exercise:

- **raster**
- **sf**
- **randomForest**
- **dplyr**
- **fasterize**

Type the following commands one at a time in the interactive window and hit `Enter` after each, waiting for each command to complete, i.e. the `>` symbol reappears in the interactive window. Or you can run them in the interactive code block below.

```
install.packages("raster")
install.packages("sf")
install.packages("randomForest")
install.packages("dplyr")
install.packages("fasterize")
```

From here on through the rest of the practical, you can either type the given commands in the interactive window, type them in a new script and running it line by line by highlighting each line and hitting `Ctrl+Enter`, or in the provided interactive code blocks within the R notebook (`.rmd`) document. The R Notebook (`GISRUK_2022_Training.rmd`) is available in the `../GISRUK_2022_Training/GISRUK_Dasy/` folder. Typing the commands in your own script will allow you to save the script for later reference and allow you to type comments by preceding your text with a `#`.

Once the packages have been successfully installed, load the packages:

```
library(raster)
library(sf)
library(randomForest)
library(dplyr)
library(fasterize)
```

3 Understanding Gridded Population Datasets

There are numerous gridded population datasets with global coverage that are produced. The popgrid consortiuum is a collaborative group of such data producers. Leyk et al. (2019))gives a peer-reviewed assessment of many of these and talks about how each has different application purposes for which they are best suited. However, there are even more peer-reviewed gridded population datasets being produced every day, such as Palacios-Lopez *et al.* (2022).

Regardless of the method of production the format of most of these gridded data are either in a people per pixel (ppp) format, where each pixel represents the number of people in the pixel in an *unprojected* raster (i.e. latitude and longitude), or represents population density format, where each (typically) *projected* pixel represents the number of people for a given normalising area.

A commonly used gridded population dataset producer is WorldPop. They typically produce their datasets as ppp and people per hectare (pph), the latter being a population density measure. For instance, while not always the case, pph is presented in pixels that have an area of a hectare (approximately a 100m x 100m pixel). However, there is nothing stopping a people per km density in a 100m x 100m pixel. Therefore it is very important that you understand how the data is made and what units are inferred on the values within your gridded dataset!

Sticking with the WorldPop ppp and pph data example, these two datasets should look identical when opened in a GIS, but only because the density measure matches the resolution of the projected raster for pph. Again, the ppp datasets are *unprojected*, i.e. uses the wGS 84 geographic coordinate system and position is given in degrees of latitude and longitude. Conversely, the pph datasets are *projected* in an equal area coordinate system of meters from a specified origin, in most cases a Universal Transverse Mercator (UTM) projection, that is appropriate for the given country.

We'll explore some of these differences with some data from Rwanda (three-letter iso code of "RWA").

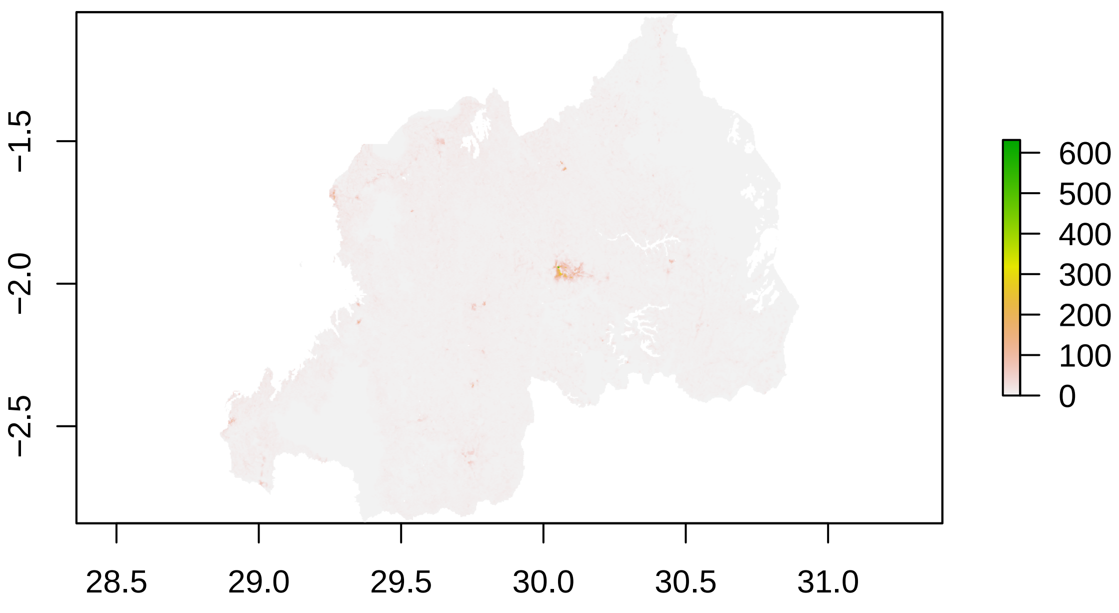
3.1 Spatial Characteristics of ppp and pph

In the `../data/SectionI/` folder lies our example population data rasters. Add the `RWA_pph_2002_v2.tif` and the `RWA_ppp_2002_v2.tif` population rasters to environment using the following two commands:

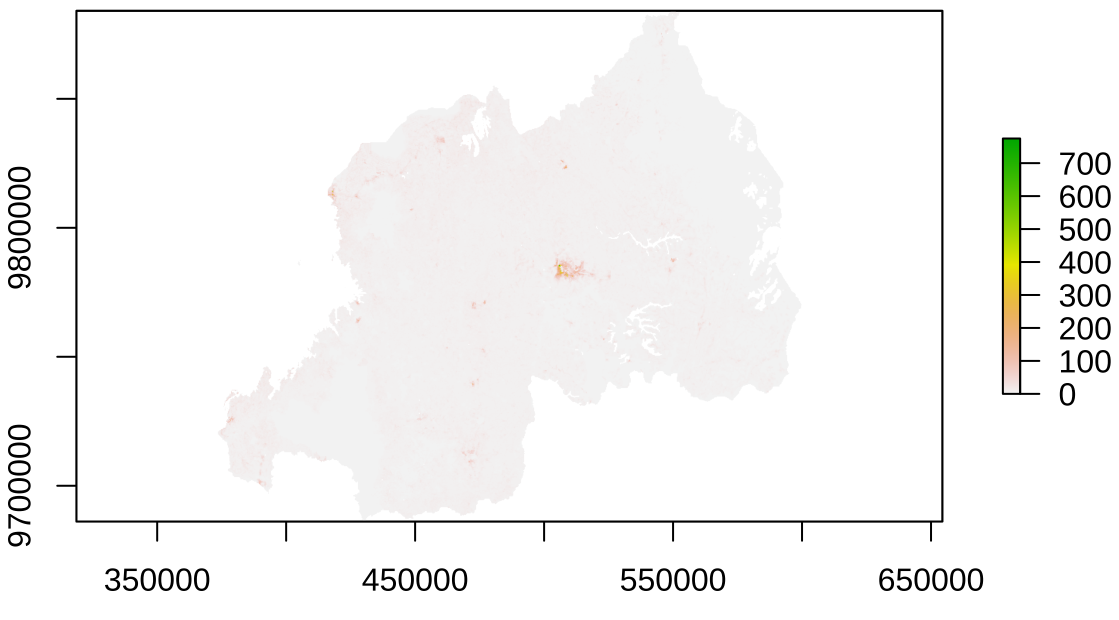
```r
ppp <- raster(paste0(root,"data/SectionI/RWA_ppp_2002_v2.tif"))
pph <- raster(paste0(root,"data/SectionI/RWA_pph_2002_v2.tif"))
```

You can now view the raster datasets interactively if you type (one command at a time):

```
plot(ppp)
```



```
plot(pph)
```



We want some basic information regarding the two rasters we've loaded. We can get this by simply typing the variable name we assigned one of the rasters to and hitting `Enter` in the interactive window. For example, if I type `pph` and hit `Enter` I get the following output in the interactive window.

```
pph
#> class      : RasterLayer
#> dimensions : 1980, 2265, 4484700  (nrow, ncol, ncell)
#> resolution : 100, 100  (x, y)
#> extent     : 373303, 599803, 9686121, 9884121  (xmin, xmax, ymin, ymax)
#> crs        : +proj=tmerc +lat_0=0 +lon_0=30 +k=0.9996 +x_0=500000 +y_0=10000000 +datum=WGS84 +units=
#> source     : RWA_pph_2002_v2.tif
#> names      : RWA_pph_2002_v2
#> values     : 0, 976.4255  (min, max)
```

Similarly if you type `ppp` and hit `Enter` we'll get the following.

```
ppp
#> class      : RasterLayer
#> dimensions : 2152, 2447, 5265944  (nrow, ncol, ncell)
#> resolution : 0.0008333, 0.0008333  (x, y)
#> extent     : 28.86092, 30.90001, -2.840613, -1.047351  (xmin, xmax, ymin, ymax)
#> crs        : +proj=longlat +datum=WGS84 +no_defs
#> source     : RWA_ppp_2002_v2.tif
#> names      : RWA_ppp_2002_v2
#> values     : 0, 713.5794  (min, max)
```

Notice how the coordinate reference systems (`coord. ref.`) are different and given in a standardized text format known as PROJ.4 (even if using the more updated PROJ.6 format). For the `ppp` dataset, it is the unprojected (meaning its units are in degrees of latitude and longitude) coordinate system WGS 1984. For the pph dataset, the coordinate reference system notes that it is in the datum of WGS 1984 (`+datum=WGS84`), but it is projected as a transverse Mercator (`+proj=tmerc`) whose origin latitude is +30° (`+lon_0=30`) and has units in meters (`+units=m`), that this dataset is a custom Universal Transverse Mercator (UTM) projection that lies between the "standard" UTM zones 35 and 36 (http://www.dmap.co.uk/utmworld.htm).

Also given here is information regarding the spatial resolution (resolution), the area size on the ground each pixel represents, of the rasters. We can see the ppp data has a value of (`0.0008333`,`0.0008333`) for the x and y-axes and similarly the `pph` dataset has (`100`,`100`). These values are the length and width of each pixel in the units of their coord. ref., meaning the ppp dataset has pixels with area equal to 0.0008333° x 0.0008333° and the `pph` dataset has pixels with area equal to 100m x 100x (0.1km sq also known as a "hectare").

These differences have some implication for users. Because WGS 1984 is a spherical representation of the surface of Earth, one degree of latitude becomes smaller the further away you move from the Equator. This means that the southernmost pixel in the Rwanda ppp population dataset represents and area smaller than the northernmost pixel in the same dataset! The reverse is true for datasets north of the Equator. The pixels of the `ppp` (0.0008333°) and `pph` (100m) datasets represent approximately the same area at the Equator. Within a given dataset, the issue of unprojected pixels representing smaller areas the further from the Equator is a relatively larger problem for countries that cover a wide range of latitudes, Chile for example, if we were to calculate population densities directly from the `ppp` dataset without accounting for the above issues.

3.2 Proper Resampling and Aggregating of pph

Let's say you wish to have the pph population density data have a different population density per unit area to match other data, for example people per sq. km, but still have every pixel be 100m x 100m. This is a simple algebraic unit conversion. Let's say we had a pixel with a value of 5.5 people per hectare and wanted to know the equivalent in sq. km. Given that one hectare is 0.01 sq km we use the equation:

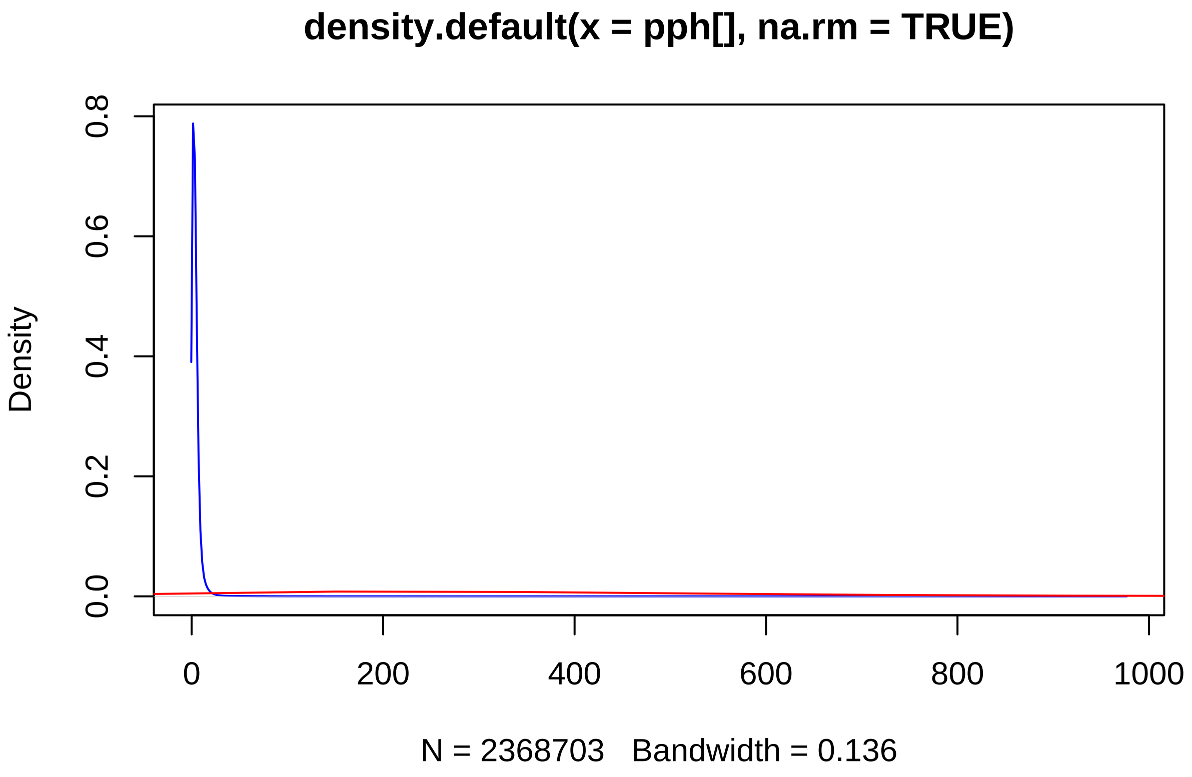
$$\text{people per km}^2 = \frac{5.5\text{ people}}{1\text{ hectare}} * \frac{1\text{ hectare}}{0.01\text{ km}^2}$$

The second element of the equation is what is known as a *conversion factor*. Thankfully, R and the **`raster`** package allow for super simple commands for doing this sort of "Raster Math" across every pixel in the raster at once. Type and execute the following command:

```
ppk <- pph / 0.01
```

Simple, right? Let's see how the values changed by typing and executing the following commands in order to compare the two raster's distribution of values. Here we'll create two overlapping density curves (similar to histograms) of the `pph`, in blue, and the `ppk` data, in red.

```
plot(density(pph[], na.rm = TRUE), col = "blue")
lines(density(ppk[], na.rm = TRUE), col = "red")
```



or by calling the two rasters themselves to look at their range of values.

```
pph
#> class      : RasterLayer
#> dimensions : 1980, 2265, 4484700  (nrow, ncol, ncell)
#> resolution : 100, 100  (x, y)
#> extent     : 373303, 599803, 9686121, 9884121  (xmin, xmax, ymin, ymax)
#> crs        : +proj=tmerc +lat_0=0 +lon_0=30 +k=0.9996 +x_0=500000 +y_0=10000000 +datum=WGS84 +units=
#> source     : RWA_pph_2002_v2.tif
#> names      : RWA_pph_2002_v2
#> values     : 0, 976.4255  (min, max)
ppk
#> class      : RasterLayer
#> dimensions : 1980, 2265, 4484700  (nrow, ncol, ncell)
#> resolution : 100, 100  (x, y)
#> extent     : 373303, 599803, 9686121, 9884121  (xmin, xmax, ymin, ymax)
#> crs        : +proj=tmerc +lat_0=0 +lon_0=30 +k=0.9996 +x_0=500000 +y_0=10000000 +datum=WGS84 +units=
#> source     : memory
#> names      : RWA_pph_2002_v2
#> values     : 0, 97642.55  (min, max)
```

We can see that the values shifted in scale to much larger ones in the `ppk` data (by a factor of 100). Additionally, this served to spread the distribution of the `pph` values across a larger range in the `ppk` data, which is reflected in the *almost* flat red line of the density plots.The red curve appears flat only because of the scale of the plot, but is actually has a slight convex curve with a local maximum around 220 on the x-axis. The pixels are still only 100m x 100m (0.01 sq km), but the units of population density in the `ppk` data are different.

This approach can be used to convert to any area unit desired for the population density; all that is needed is the correct conversion factor.

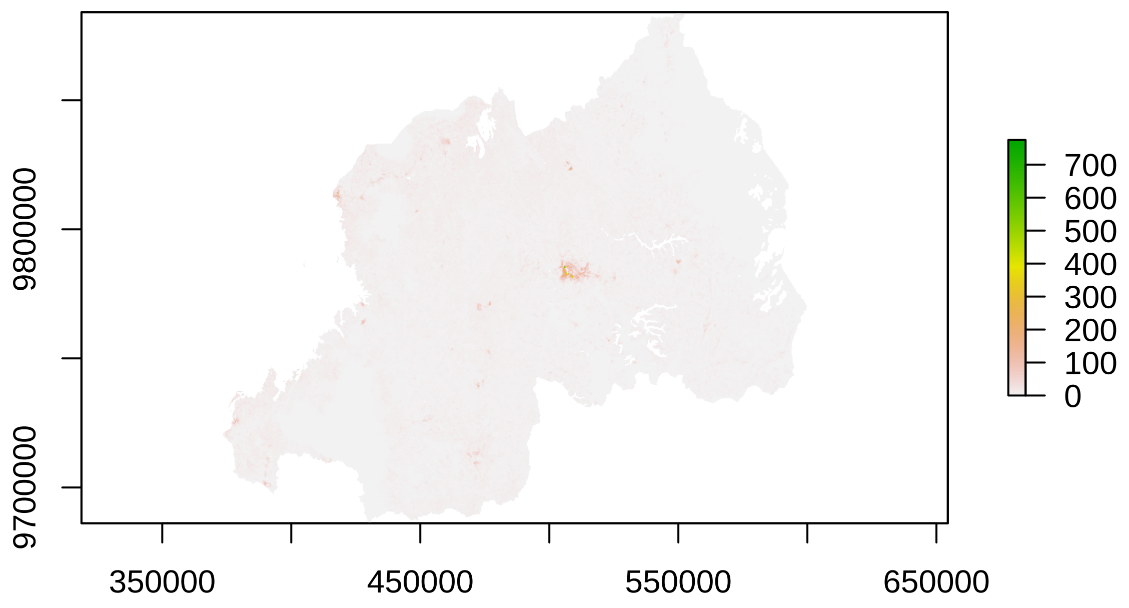
But what if you need to have your population density data at a coarser resolution to match another dataset? Let's say you have some data on urban extents at 1 sq. km resolution and need to have your population density information at the same spatial resolution. Here we can use the R function `aggregate()`. This function takes your original raster and aggregates the pixels into larger pixels using a specified mathematical function to aggregate the many values into 1 value per new, larger pixel. Type and execute `?raster::aggregate` in the interactive window for more function information.

Let's aggregate the `pph` data into pixels that are actually 1 sq. km in resolution.

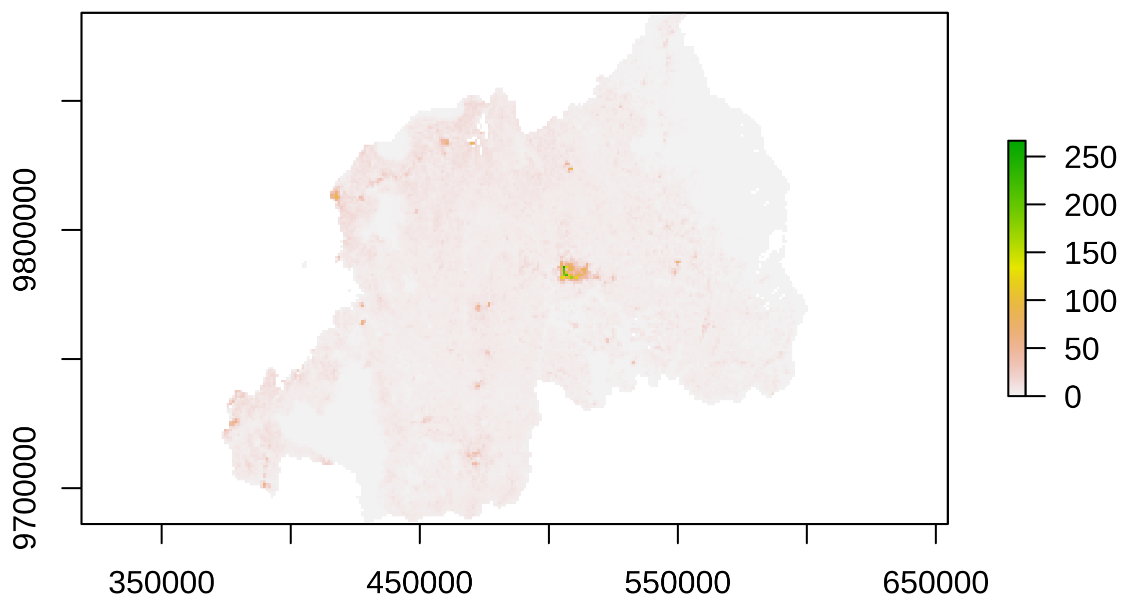
```
pph_1k <- aggregate(pph, fact = 10, fun = mean, expand = T, na.rm = TRUE)
```

Visually compare the `pph` raster and the `pph_1k` raster by using the plot function.

```
plot(pph)
```



```
plot(pph_1k)
```



You should now have a data set with a spatial resolution/cell size of 1000m x 1000m (check the layer properties to ensure). What are the units of this new raster's values though? Because we had R take the average or mean of the input data (`pph`), every pixel in the output dataset represents the average (notice the `fun = "mean"` parameter in the above code block) people per hectare of the input cells. If you desired people per sq. km at 1 sq. km resolution, then the conversion procedure would also need to be done either prior to aggregation.

Note: Be aware of the modifiable areal unit problem (MAUP) when aggregating. See Openshaw (1984) for more information. This is an R Markdown Notebook. When you execute code within the notebook, the results appear beneath the code.

Alternatively, you can resample to change the resolution of the data, although there are serious caveats with this if you resample to a smaller spatial resolution since that would be disaggregating without any corresponding finer scale information, as with dasymetric disaggregation (See Section 4), and therefore committing an *ecological fallacy*.

Let's say you need to resample to ~300m resolution to match some projected European Space Agency Climate Change Initiative land cover data. We will use the `resample()` function from the raster package to do this. Deciding what resampling method to use, bilinear or bicubic, is based upon use and preference. See here for a visual description of how the resampling methods work. Because our data are continuous we will use bilinear interpolation.

R prefers to have a raster that already has the desired resolution to use as a "template" for converting your input dataset. In this scenario we can just give it the landcover dataset.

Type and execute the following to load the landcover raster:

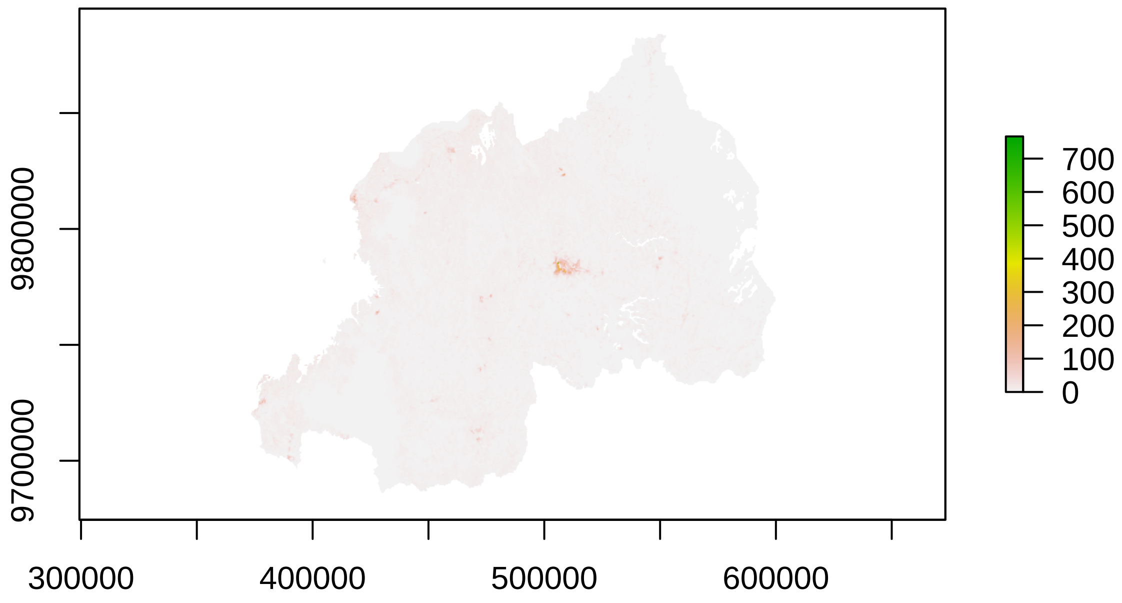
```
rwa_lc_300 <- raster(paste0(root, "data/SectionI/rwa_lc_300_projected.tif"))
```

Then type and execute the following to carry out the resample:

```
pph_300m <- resample(pph, rwa_lc_300, method = "bilinear")
```

Use the plot function to see how the results carried out.

```
plot(pph_300m)
```



Take caution, if you do any of the above with the population count data, once you have aggregated, using `fun="sum"`, or by resampling, they will no longer add back up to the population total of the census-based unit they were disaggregated from! That eliminates a key advantage and property of dasymetric mapping – its "volume preserving" characteristic.

3.3 Going from ppp to Spatially Projected Population Density

So, we've seen how to do some manipulations with the pph-formatted data. But what if we have unprojected ppp-formatted data and we want to acqurie gridded population data where our values correspond to some populaiton density?

The key is the pixel area grid dataset that is provided with this dataset. The pixel area raster is a 0.0008333° resolution dataset whose pixel values represent the area of a given pixel in square meters. This means that if we have the spatially aligned ppp dataset, it becomes another simple algebra problem using raster math where:

$$\text{population density in pixel } i = (\text{population count}_i)/(\text{pixel area}_i)$$

First we need to get a raster where each pixel contains the value of the its area in some squared linear units. Thankfully, the **raster** package already has a convenient function for this: `area()` (Although it is not suitable for *super precise* applications; see `?area` for more details). The units the `area()` function return are in sq. km. So, let's load a ppp dataset for Rwanda (RWA) and calculate the pixel area (`px_area`):

```
ppp_rwa <- raster(paste0(root,"data/SectionI/ppp_prj_2002_RWA.tif"))
px_area <- area(ppp_rwa)
```

We could then get the population density per sq. km, in approx 100m x 100m pixels, by simply dividing the `ppp_rwa` by `px_area`. But, let's say we want to convert this `ppp` dataset to have density units of people per sq. mile (mi). For this we'd need a conversion factor of 1 sq. km to 0.386102 sq. mi. For each pixel i the equation for its value would look like:

$$\text{ppmi} = \frac{ppp\_rwa}{px\_area_{km^2}} * \frac{1 \text{ km}^2}{0.386102 \text{ mi}^2}$$

We could then execute the following:

```
ppmi <- ppp_rwa/px_area * (1/0.386102)
```

You should now have a population density layer, in WGS 1984 coordinate system still, with units of people per square mile. If you desired another area unit for the density value, you could use the same equation above and modify it by simply multiplying by the correct conversion factor. Feel free to explore this dataset as before with plotting and other graphs.

4 “Intelligent” Dasymetric Redistirbution

4.1 What is Dasymetric Redistribution?

Dasymetric redistribution is the disaggregation of values from count data a larger *source area* to a series of, smaller, *target areas* that lie within the given source area based upon *ancillary data*. Ancillary data is information related to the phenomena, which is being disaggregated, and is used to inform, proportionally, which target units the counts being distributed should be assigned (Wright 1936; Eicher and Brewer 2001; J. Mennis 2003; J. Mennis and Hultgren 2006; Jeremy Mennis 2009). The ancillary data must be at the same spatial resolution of the target areas and the disaggregated values assigned to the target areas should add back up to the original value of the source area they were distributed from (J. Mennis 2003; J. Mennis and Hultgren 2006; Jeremy Mennis 2009). This is similar to the more simple procedure of areal reweighting (Figure 2) which redistributes count values based upon the proportional relationships between the areas of the targets to the source (Eicher and Brewer 2001).

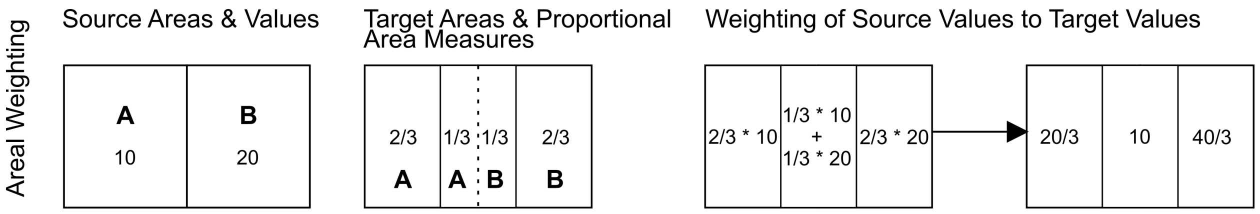


Figure 2: Example of areal reweighting. Adapted from Nieves et al. (2022) DOI: 10.13140/RG.2.2.24822.93763.

For instance, if we are disaggregating population counts from a county down to the admin units that make up the county, we could use land cover as an ancillary dataset, as shown in an example within Figure 3. This is because we know, or assume, things about the relationship between population counts and land cover, e.g. few to no people live in water (land cover class “200” in example), more people live in built land cover (land cover class “190” in example) than in agriculture land cover (land cover class “11” in example), and more people are likely to live in agriculture land cover than forest land cover (land cover class “40” in example) (J. Mennis 2003; J. Mennis and Hultgren 2006; Nieves et al. 2017).

How we go about communicating this relationship mathematically is by assigning weights to the different ancillary data. This can either be done by manually assigning weights based upon expertise or theoretical concepts (J. Mennis 2003), or so-called “intelligent” dasymetric mapping (J. Mennis and Hultgren 2006) where the weights are determined by some statistical function or model such as linear regression (Figure 3). It is important to note that these weights are used relative to each source area. So, if a target area has a given weight, we first normalize it, if the weights in the source area do not already add up to 1, relative to all the other target areas within its source area prior to multiplying the source area count by the weights to get the final disaggregated target area counts (Figure 3).

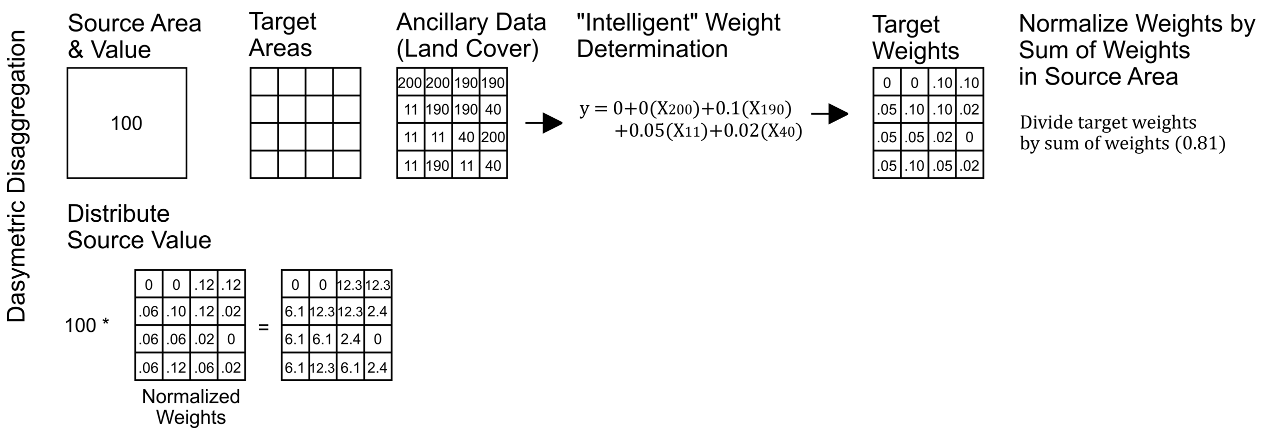

Dasymetric Disaggregation
Source Area & Value
100
Target Areas
Ancillary Data (Land Cover)
"Intelligent" Weight Determination
y = 0+0(X200)+0.1(X190) +0.05(X11)+0.02(X40)
Target Weights
Normalize Weights by Sum of Weights in Source Area
Divide target weights by sum of weights (0.81)
Distribute Source Value
100 *
Normalized Weights

Figure 3: Example of dasymetric disaggregation, specifically an "intelligent" version where the weights were determined by a linear regression. Adapted from Nieves et al. (2022) DOI: 10.13140/RG.2.2.24822.93763.

We are going to examine a method of random forest-informed dasymetric modelling of gridded population data, put forth by Stevens *et al.* (2015) that uses a random forest regression, a regression tree-based machine learning method, to create the weights that inform the dasymetric redistribution of population count values from source areas (typically census-based enumeration units) to our target areas (approximately 100m resolution pixels).

4.2 What is a Random Forest?

Random forests (RFs) are a non-parametric modelling method that creates an ensemble model by growing many independent decision trees through random covariate selection and a sampling with replacement method called bagging (Breiman 1996, 2001). As related to user input and parametrization, RFs do not require much, which is an attractive feature and allows it to be used on large datasets with varying attributes. A brief summary of strengths and weaknesses of RFs are presented in Table 1:

While fully understanding a RF is not necessary to utilize this data correctly, those who wish to know more can read (Breiman 1996, 2001; Liaw and Wiener 2002) and the document `Classifiers and Random Forests.pdf` in the `../docs/` folder. The important concept is that we use random forests to create the weighting layer used in our dasymetric redistribution of population count data by training it at the admin unit level using unit averages of covariates and population density and then predict population density per pixel based upon pixel level covariates.

Table 1: General strengths and limits of random forests

Strengths	Limits
All-purpose model that performs well on most problems including those with non-linear relationships, many complex interactions of covariates, and collinearity	Covariate relationships to outcome of interest largely, but not wholly, uninterpretable since it is composed of hundreds of decision trees, i.e. considered a 'black box' method
Handles noisy or missing data in addition to categorical and continuous data and is robust against over training	
Can be specified to only include the most important variables	
Can handle extremely large and extremely small datasets	
Can be used for either classification or regression	
Requires little to no manual tuning of parameters	
Scalable, i.e. can be parallelized for quicker performance	
Internal validation estimate	

4.3 Dasymetric Disaggregation with a Random Forest-informed Weighting Layer

Here we will provide you with the weighting layer created by a RF and have you manually go through the process of disaggregating population count data from vector-based administrative units to ~100m pixels. Let's start by adding our vector-based population data that is stored as a polygon shapefile.

Execute the following:

```
pop_shp <- st_read(dsn = paste0(root,"data/SectionII/adminpop.shp"),
                   layer = "adminpop",
                   stringsAsFactors = FALSE)
#> Reading layer `adminpop' from data source
#>   `E:\Workshops\Dasymetric Mapping\GISRUK_2022_Training\data\SectionII\adminpop.shp'
#>   using driver `ESRI Shapefile'
#> Simple feature collection with 9192 features and 6 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 28.86129 ymin: -2.840059 xmax: 30.89954 ymax: -1.047631
#> Geodetic CRS:  WGS 84
```

Take a bit of time to explore the attribute table of the `pop_shp` object paying attention to the `ADMINID` and `ADMINPOP` fields. The `ADMINID` field is the unique ID of each source area and the `ADMINPOP` field is the total population count for each area. If we execute the `summary()` function and call the `head()` function on the `pop_shp` object:

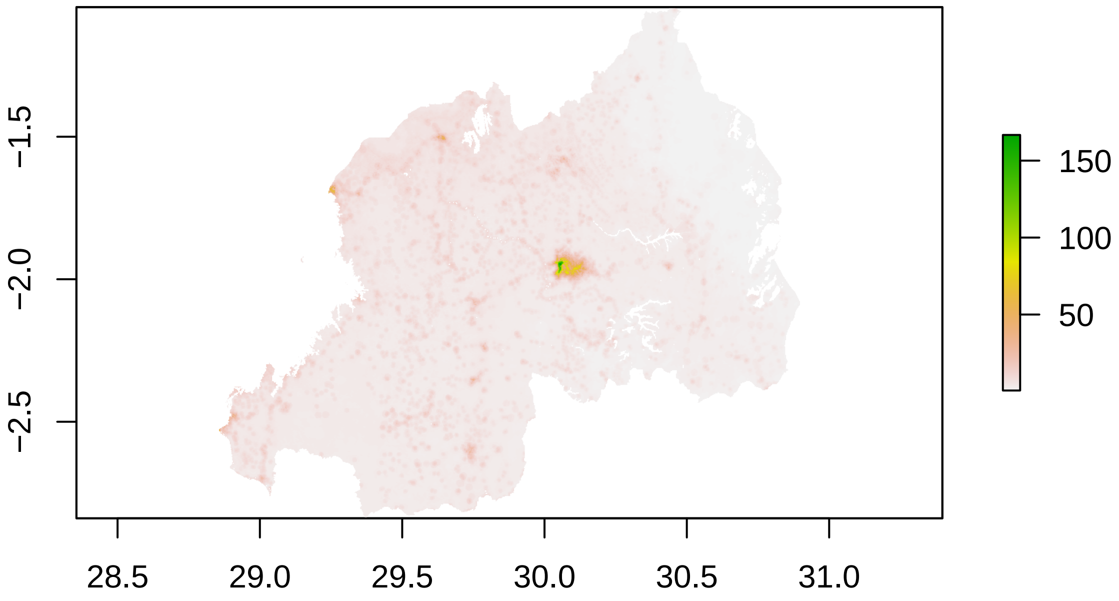
```
head(pop_shp)
#> Simple feature collection with 6 features and 6 fields
#> Geometry type: MULTIPOLYGON
#> Dimension:     XY
#> Bounding box:  xmin: 30.04958 ymin: -1.972594 xmax: 30.07124 ymax: -1.945095
#> Geodetic CRS:  WGS 84
#>   GRIDCODE ADMINID ADMINPOP YEARPOP ISO DIS_ID                       geometry
#> 1        1       1     5880    2002 RWA     15 MULTIPOLYGON (((30.06374 -1...
#> 2        2       2     3469    2002 RWA     15 MULTIPOLYGON (((30.06124 -1...
#> 3        3       3     3279    2002 RWA     15 MULTIPOLYGON (((30.05874 -1...
#> 4        4       4     3340    2002 RWA     15 MULTIPOLYGON (((30.06874 -1...
#> 5        5       5     6510    2002 RWA     15 MULTIPOLYGON (((30.06208 -1...
#> 6        6       6     3224    2002 RWA     15 MULTIPOLYGON (((30.05374 -1...
summary(pop_shp)
#>     GRIDCODE        ADMINID         ADMINPOP         YEARPOP
#>  Min.   :   1   Min.   :   1   Min.   :    0   Min.   :2002
#>  1st Qu.:2299   1st Qu.:2299   1st Qu.:  593   1st Qu.:2002
#>  Median :4596   Median :4596   Median :  839   Median :2002
#>  Mean   :4596   Mean   :4596   Mean   : 1032   Mean   :2002
#>  3rd Qu.:6894   3rd Qu.:6894   3rd Qu.: 1184   3rd Qu.:2002
#>  Max.   :9192   Max.   :9192   Max.   :19500   Max.   :2002
#>      ISO                DIS_ID              geometry
#>  Length:9192        Min.   : 1.0   MULTIPOLYGON :9192
#>  Class :character   1st Qu.: 7.0   epsg:4326    :   0
#>  Mode  :character   Median :17.0   +proj=long...:   0
#>                     Mean   :15.8
#>                     3rd Qu.:23.0
#>                     Max.   :30.0
```

We can see the attribute table of the has been read in using the **sf** package essentially as a R `data.frame` class with attached spatial geometry data. This means anything you can do to an R `data.frame` you can use to manipulate and view the data of the `pop_shp` object. We can also see that there are six field columns here plus a column for the geometry, but we are mainly interested in ADMINID and ADMINPOP fields.Now we'll add the prediction density layer by executing the following:

```
rf_weight_layer <- raster(paste0(root,
                                 "data/SectionII/",
                                 "predict_density_rf_pred_prj_2002_RWA.tif"))
```

Get an idea of the what the weighting layer looks like by calling:

```
plot(rf_weight_layer)
```



Looking at the plot legend, it is quickly apparent that that the sum of pixel values within any given source area will be much higher than 1, meaning we are not meeting one of the requirements for dasymetric weights and indicating we will need to normalize the values somehow first. This is because the RF outputs values of predicted average population density, which can have values from zero to, theoretically, infinity.

Lets focus on normalising the weights relative to each unit. We can think of this in mathematical terms where for a given source area i we divide every target pixel j within that given source area by the sum of all pixels j in the source area i. Or written as a function:

$$\text{normalised weight}_{ij} = \frac{RF\_predicted\_value_{ij}}{\sum RF\_predicted\_value_{ij}}$$

So, to get our normalised weighting layer we need to do two things: 1) find the sum of all pixel values within each source area of the `.shp` layer, and, 2) Divide all the pixel values of the `.tif` weighting raster by those sums. First, we'll find the sum of weight values and create a raster of those sums that we can normalise by. So, we would normally create a raster of zones to carry out our sum procedure within by calling the `rasterize()` function from the raster package to put the ADMINID values into spatially coincident raster cells. However, the `rasterize()` function is relatively inefficient, so we will use the more efficient `fasterize()` function that can leverage some of the greater efficiencies of the **sf** package.

```r
zone_ras <- fasterize(pop_shp,
                      rf_weight_layer,
                      field = "ADMINID",
                      fun = "first",
                      background = NA)
```

Go ahead and save the zonal raster to file with the following command:

```r
writeRaster(zone_ras,
            filename = paste0(root,"data/SectionII/zonal_raster.tif"),
            format = "GTiff",
            overwrite = TRUE,
            options = c("COMPRESS=LZW"))
```

We now have our raster where every pixel value indicates the unique ID of the source polygon area the pixel lies in. Now, let's find the total sum of pixel level weights per admin unit. Type and execute the following to get the sum:

```r
weight_sum_table <- zonal(rf_weight_layer,
                          zone_ras,
                          fun = "sum",
                          na.rm = TRUE)
```

This gives us a matrix that has one column for the unique ID of each zone and the corresponding sum of weights per zone in the second column. But this data is no longer spatial, so we need to get it back into a spatially explicit format again, specifically in a raster. So first, let's convert the matrix to a `data.frame` and make the names line up for a join to the shapefile. Execute the following two lines one after the other:

```r
weight_sum_table <- as.data.frame(weight_sum_table)
names(weight_sum_table) <- c("ADMINID", "RFW_SUM")
```

We then join the data to the shapefile by the matching `ADMINID` value:

```r
pop_shp <- merge(pop_shp, weight_sum_table, by = "ADMINID")
```

Check that the values have been transferred to the shapefile by calling:

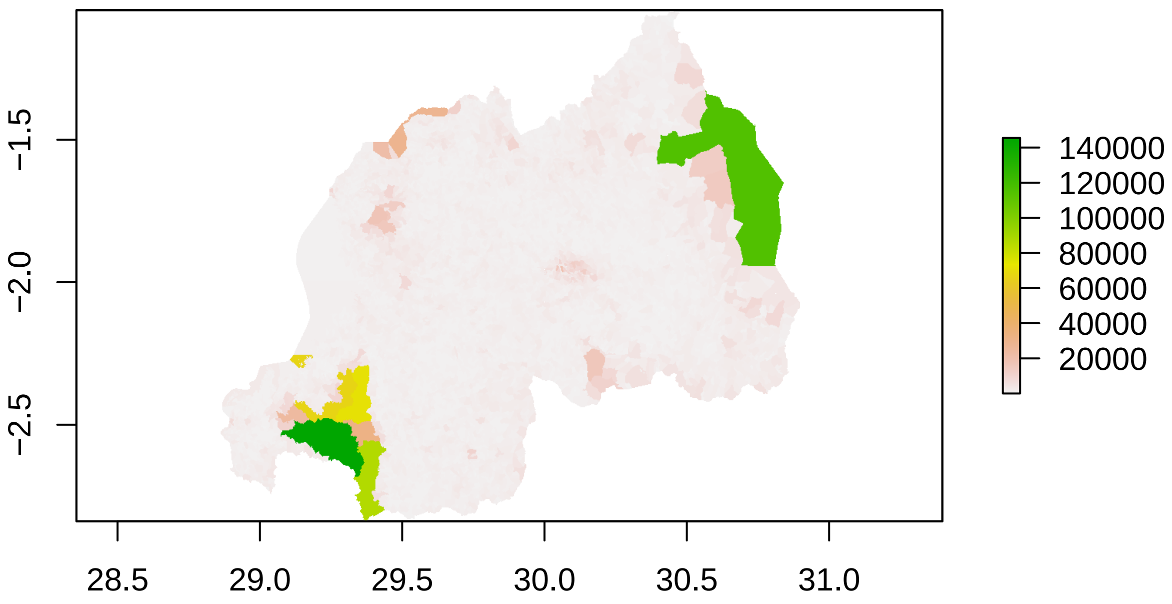
```r
head(pop_shp$RFW_SUM)
#> [1] 2502.028 1164.864 1165.418 1801.993 2298.034 1768.341
```

Now we'll turn those weight sums into a raster using `fasterize()` again:

```r
weight_sum_ras <- fasterize(pop_shp,
                            rf_weight_layer,
                            field = "RFW_SUM",
                            fun = "first",
                            background = NA)
```

Go ahead and plot the raster to see how it did.

```
plot(weight_sum_ras)
```



Let’s normalise the weights; execute the following:

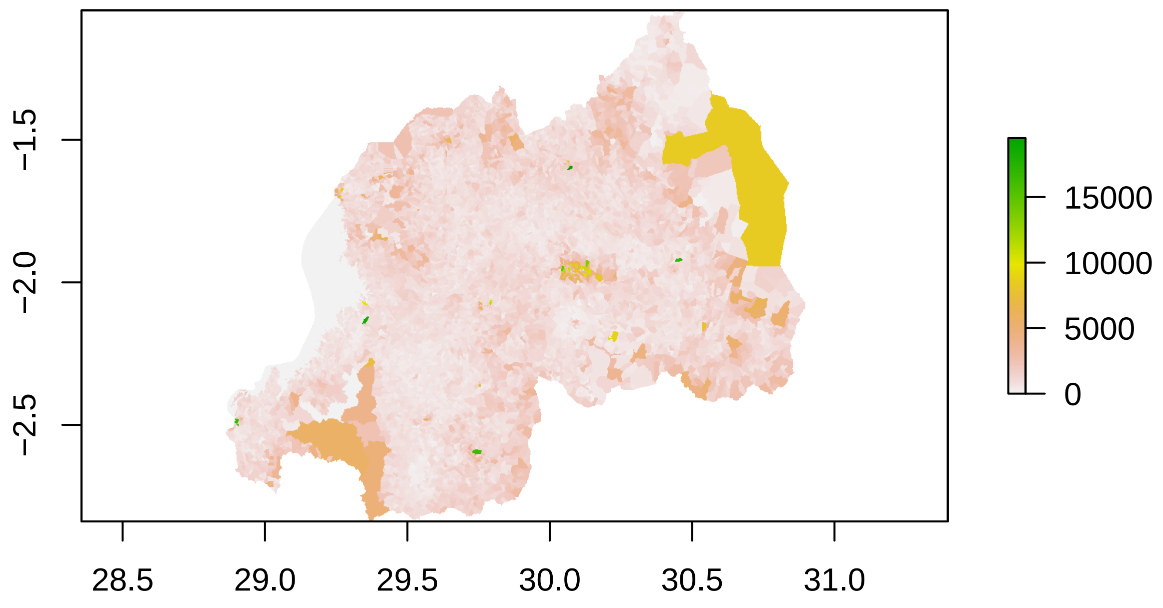
```
norm_weights <- rf_weight_layer / weight_sum_ras
```

Now that we have our normalised weights in raster format we can focus on disaggregating our population count data from the `adminpop.shp` data. However, we will need to convert the population data from vector to raster and make sure we have it aligned with our normalised weighting raster, again using `fasterize()`.

```
pop_sum_ras <- fasterize(pop_shp,
                         rf_weight_layer,
                         field = "ADMINPOP",
                         fun = "first",
                         background = NA)
```

Give this raster a plot to see how coarse the population data is relative to what we are about to do.

```
plot(pop_sum_ras)
```

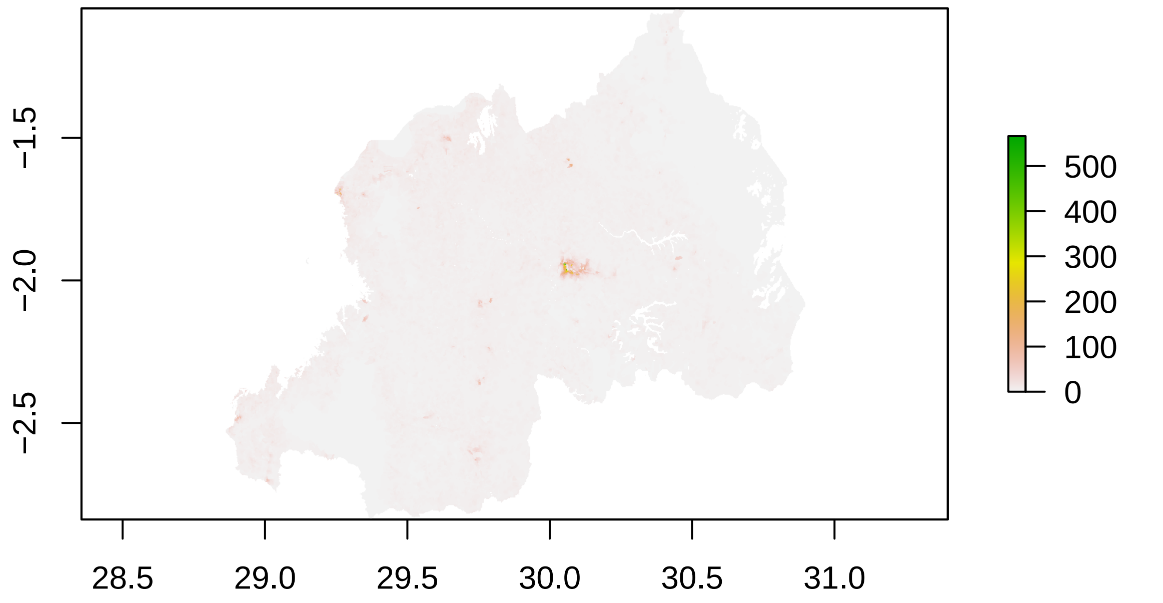


At last, we can disaggregate the population using the normalised weights. Execute the following:

```
rf_disaggregation <- norm_weights*pop_sum_ras
```

Now plot this raster to compare to the raster of the admin unit sums!

```
plot(rf_disaggregation)
```



We can now perform one more check to ensure that everything went correctly. Dasymetric redistributions should ideally be "volume preserving," meaning that all the values of the target areas should add back up to the source area they were disaggregated from. Let's add up the values using the `zonal()` function again and subtract the disaggregated sums from the admin unit values.

```
pop_check <- zonal(rf_disaggregation,
                   zone_ras,
                   fun = "sum",
                   na.rm = TRUE)
pop_difference_vector <- pop_shp$ADMINPOP - pop_check[,2]
```

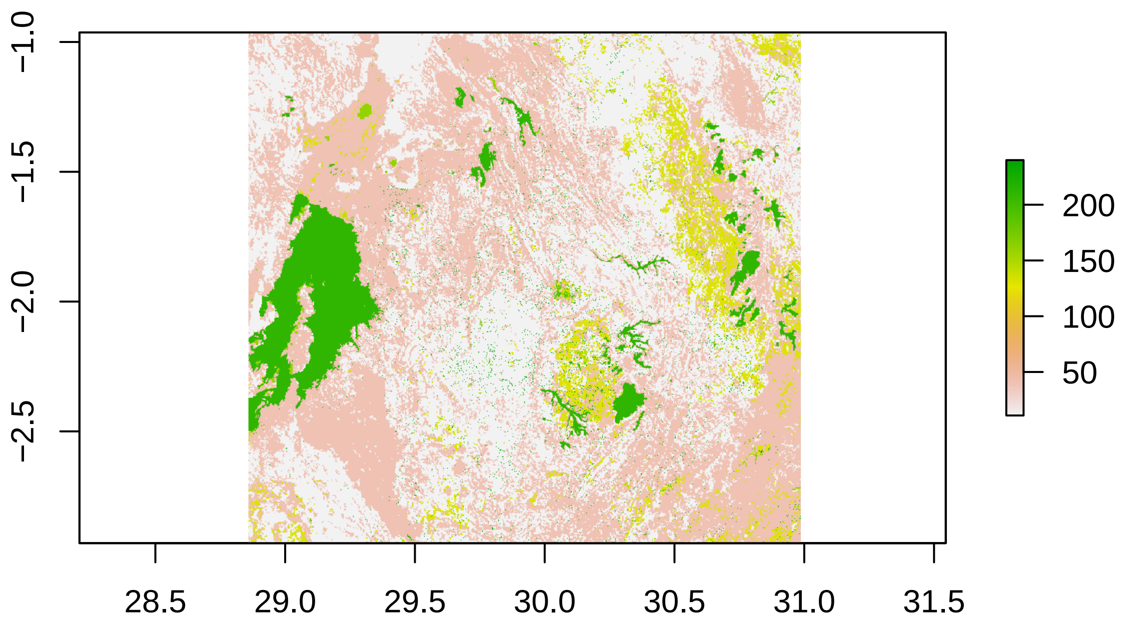
Now we can see what the difference is (if any) between the sum of our modelled values and the original source values:

```
summary(pop_difference_vector)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>       0       0       0       0       0       0
```

4.4 Creating Your Own Dasymetric Weighting Layer

Let's manually create a dasymetric weighting layer and see how this impacts the disaggregation. First load the `rwa_lc.tif` and view it by executing:

```
lc_ras <- raster(paste0(root,"data/SectionII/rwa_lc.tif"))
plot(lc_ras)
```



This raster is a thematic land cover dataset where every integer value represents one of the described land cover classes in the table below (Table 2). There are more classes possible in this dataset, but we only include the ones seen in Rwanda for conciseness.

Table 2: ESA CCI Land Cover classes, descriptions, and user assigned weights

Class Number	Land Cover Description	Assigned Weight
11	Cultivated and Managed Lands	
40	Natural and Semi-Natural Vegetation: Woody/Trees	
130	Natural and Semi-Natural Vegetation: Shrubs	
140	Natural and Semi-Natural Vegetation: Herbaceous/Grassy	
160	Natural and Semi-Natural Vegetation: Aquatic	
190	Built Areas: Urban	
200	Bare Areas	
210	Water	
240	Built Areas: Rural	

To turn this into a weighting layer, we first need to convert the land cover class values into weights. The catch is, with the `reclass()` function, we can only convert from one integer value to another integer value, so keep this in mind when selecting weights.

Using the "User Assigned Weights" column in Table 2 to record your choices, assign weighting values at your discretion based upon your beliefs of how population distribution is related to land cover classes. But remember that these values should have relative meaning amongst themselves. For instance, you may have reason to believe that water land cover (class 200) has zero population so you would want to give it a weight of "0." You also may believe that people are five times as likely to live in built land cover (class 190) as compared to cultivated land cover (class 11) and twice as likely to live in cultivated land cover (class 11) as they are to live in forested land cover (class 40). Understanding that the lowest non-zero weight you can assign is "1," you could then assign weights of 10, 2, and 1 to built, cultivated, and forest land cover, respectively.

Now that you have decided the weights you desire to use, we are going to create our initial weighting layer. We need to first create our reclassification matrix. A reclassification matrix has three columns: 1) the "from" column, 2) the "to" column and, 3), the "becomes" column. The first two columns define the class number you want to designate, and the third column indicates the value you want that range of values to become. I've filled in the first two columns of values in the code below and now you just need to fill in your selected weight values for the third column, where every `x` I've placed should be replaced with your chosen weight. Class numbers in the code are in the same order as the above table (Table **??**. NOTE: The sum of your weights should add to 1.0!

```
rcl_mat <- matrix(data = c(c(9,39,129,139,159,189,199,209,239),
                           c(12,41,131,141,161,191,201,211,241),
                           c(x,x,x,x,x,x,x,x,x)),
                  ncol = 3)
```

Now we can reclassify to get our weights raster by running the following:

```
lc_weight_ras <- reclassify(lc_ras,
                            rcl_mat,
                            filename = paste0(root,
                                              "output/SectionII/rwa_lc_manual_weights.tif"),
                            overwrite = TRUE)
```

We'll create a new *projected* zonal raster since the one we made previously was *unprojected*.

```
zone_ras <- fasterize(pop_shp, lc_weight_ras, field = "ADMINPOP")
```

Like with the RF-based weighting layer, we now need to normalise this weighting layer by getting the sum of weights per admin unit, converting that to a raster, and then dividing by the sum. Type and execute the following to get the sum of weights:

```
weight_sum_table <- zonal(lc_weight_ras,
                          zone_ras,
                          fun = "sum",
                          na.rm = TRUE)
```

First, let's convert the matrix to a `data.frame` and make the names line up for a join to the shapefile. Execute the following two lines one after the other:

```
weight_sum_table <- as.data.frame(weight_sum_table)
names(weight_sum_table) <- c("ADMINID","LCW_SUM")
```

We then join the data to the shapefile by the matching `ADMINID` value.

```
pop_shp <- merge(pop_shp, weight_sum_table, by = "ADMINID")
```

Check that the values have been transferred to the shapefile by calling:

```
head(pop_shp$LCW_SUM)
#> [1] 8.00 5.33 0.66 6.62 5.74 4.74
```

Now we'll turn those weight sums into a raster using `fasterize()` again.

```
weight_sum_ras <- fasterize(pop_shp,
                            lc_weight_ras,
                            field = "LCW_SUM",
                            fun = "first",
                            background = NA)
```

Normalise our land cover based weights:

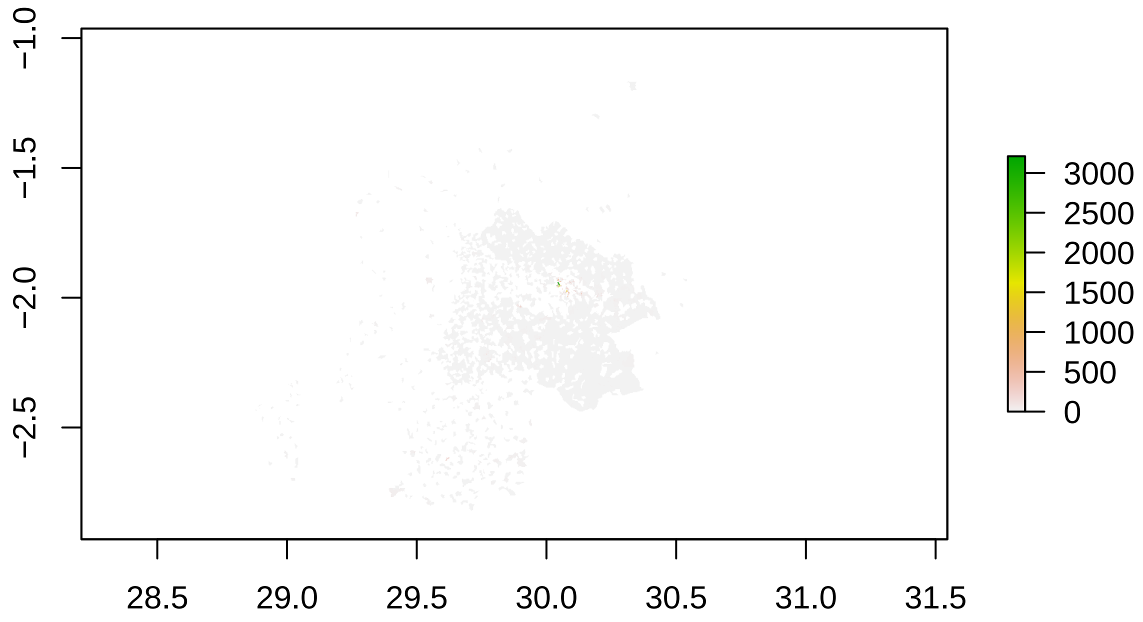
```r
lc_norm_weights <- lc_weight_ras / weight_sum_ras
```

We need to go ahead and create another raster representation of the admin unit level population count to make sure it lines up with our weighting layer.

```r
pop_sum_ras <- fasterize(pop_shp,
                         weight_sum_ras,
                         field = "ADMINPOP",
                         fun = "first",
                         background = NA)
```

We can now move to the final step of multiplying our population count raster by the weights.

```r
lc_disagg <- lc_norm_weights * pop_sum_ras
plot(lc_disagg)
```



Take a minute to plot and explore your dasymetrically redistributed population surface! Or save the `lc_disagg` object to a file (using the earlier `writeRaster` command) and open it up Arcmap/QGIS for an interactive exploration. If you desire, you can perform the aggregation check to make sure your population count sums up properly. Additionally, you can take the `RWA_ppp_manual_2002.tif` and `RWA_ppp_LC_manual_2002.tif` and subtract them using Raster Calculator type operations to examine how each disaggregation distributed the population counts differently.

5 Further Steps

Now that you have the basics of how to create and post-process these types of data, try exploring other implementations of dasymetric disaggregation. Some things to possibly explore:

- Utilise another statistical modelling method to create the dasymetric weights, e.g. artificial neural network (Hopfield 1988) using the **ann** package or generalised linear models (Nelder and Wedderburn 1972) using the **glm** package
- Read more on the limits of uncertainty in dasymetric modelling/disaggregation and one possible soultion in Nagle *et al.* (2014)
- Find some census-based data at two different spatial resolutions, e.g. census tracts (admin. level 3), block groups (admin. level 4), and blocks (admin. level 5) from the US census Bureau. Model gridded population by creating your weights using the coarser data (e.g. admin level 3 and 4) and then use the boundaries and counts of the spatially finest data (admin level 5) to compare the errors that get introduced by modelling with different coarsenesses of input data. Note: good error measures for this are the mean absolute error (MAE) and the root mean square error (RMSE), but it is better to calculate them using population density rather than counts as the former adjusts for the area of each admin unit. See Gaughan *et al.* (2014), Reed *et al.* (2018), and Stevens *et al.* (2020) for more details of looking at the errors of modelled population.
- Explore the **popRF** package vignette to see how you can quickly and easily construct your own random forest-informed gridded population datasets with an open online geospatial data library as well as your own input data

References

Breiman, L. 1996. “Bagging Predictors.” *Machine Learning* 24 (2): 123–40.

———. 2001. “Statistical Modeling: The Two Cultures.” *Statistical Science* 16 (3): 199–231.

Eicher, C L, and C A Brewer. 2001. “Dasymetric Mapping and Areal Interpolation: Implementation and Evaluation.” *Cartography and Geographic Information Science* 28: 125–38.

Gaughan, A E, F R Stevens, C Linard, N G Patel, and A J Tatem. 2014. “Exploring Nationally and Regionally Defined Models for Large Area Population Mapping.” *International Journal of Digital Earth*. https://doi.org/10.1080/17538947.2014.965761.

Hopfield, J. J. 1988. “Artificial Neural Networks.” *IEEE Circuits and Devices Magazine* 4 (5): 3–10. https://doi.org/10.1109/101.8118.

Leyk, Stefan, Andrea E. Gaughan, Susana B. Adamo, Alex de Sherbinin, Deborah Balk, Sergio Freire, Amy Rose, et al. 2019. “The spatial allocation of population: a review of large-scale gridded population data products and their fitness for use.” *Earth System Science Data* 11 (3): 1385–1409. https://doi.org/10.5194/essd-11-1385-2019.

Liaw, A, and M Wiener. 2002. “Classification and Regression by randomForest.” *R News* 3 (2): 18–22.

Mennis, J. 2003. “Generating Surface Models of Population Using Dasymetric Mapping.” *Professional Geographer* 55 (1): 31–42.

Mennis, Jeremy. 2009. “Dasymetric Mapping for Estimating Population in Small Areas.” *Geography Compass* 3 (2): 727–45. https://doi.org/10.1111/j.1749-8198.2009.00220.x.

Mennis, J, and T Hultgren. 2006. “Intelligent Dasymetric Mapping and its Application to Areal Interpolation.” *Cartography and Geographic Information Science2* 33: 179–94.

Nagle, Nicholas N., Barbara P. Buttenfield, Stefan Leyk, and S. Spielman. 2014. “Dasymetric Modeling and Uncertainty.” *Annals of the Association of American Geographers* 104: 80–94.

Nelder, J. A., and R. W. M. Wedderburn. 1972. “Generalized Linear Models.” *Journal of the Royal Statistical Society Series A* 135 (Part 3): 370–84.

Nieves, J J, Forrest R Stevens, A. E. Andrea E Gaughan, Catherine Linard, Alessandro Sorichetta, Graeme Hornby, Nirav N. N. Patel, and A. J. Andrew J Tatem. 2017. “Examining the correlates and drivers of human population distributions across low- and middle-income countries.” *Journal of The Royal Society Interface* 14 (137): 20170401. https://doi.org/10.1098/rsif.2017.0401.

Openshaw, S. 1984. “The modifiable areal unit problem.” *Concepts and Techniques in Modern Geography* 38.

Palacios-Lopez, Daniela, Thomas Esch, Kytt MacManus, Mattia Marconcini, Alessandro Sorichetta, Greg Yetman, Julian Zeidler, Stefan Dech, Andrew J. Tatem, and Peter Reinartz. 2022. “Towards an Improved Large-Scale Gridded Population Dataset: A Pan-European Study on the Integration of 3D Settlement Data into Population Modelling.” *Remote Sensing* 14 (2): 325. https://doi.org/10.3390/rs14020325.

Reed, Fennis, Andrea Gaughan, Forrest Stevens, Greg Yetman, Alessandro Sorichetta, and Andrew Tatem. 2018. “Gridded Population Maps Informed by Different Built Settlement Products.” *Data* 3 (3): 33. https://doi.org/10.3390/data3030033.

Stevens, F R, A E Gaughan, C Linard, and A J Tatem. 2015. “Disaggregating Census Data for Population Mapping Using Random Forests with Remotely-sensed Data and Ancillary Data.” *PLoS One* 10 (2): e0107042. https://doi.org/10.1371/journal.pone.0107042.

Stevens, Forrest R., Andrea E. Gaughan, Jeremiah J. Nieves, Adam King, Alessandro Sorichetta, Catherine Linard, and Andrew J. Tatem. 2020. “Comparisons of two global built area land cover datasets in methods to disaggregate human population in eleven countries from the global South.” *International Journal of Digital Earth* 13 (1): 78–100. https://doi.org/10.1080/

17538947.2019.1633424.

Wright, J K. 1936. “A Method of Mapping Densities of Population.” *The Geographical Review* 26: 103–110#.