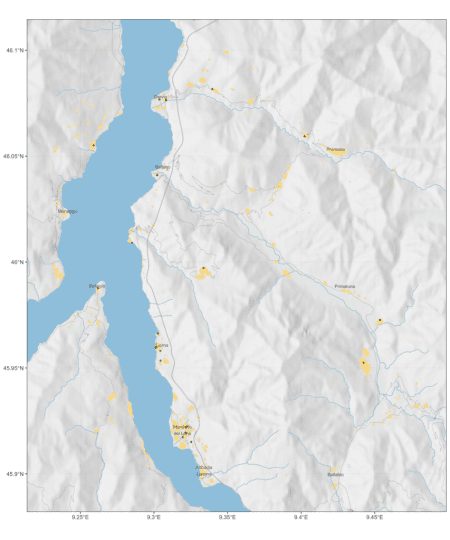
[[](http://www.francescobailo.net/wordpress/wp-content/uploads/2018/08/Enrich-your-map-with-natural-and-anthropic-details.png)](http://www.francescobailo.net/wordpress/wp-content/uploads/2018/08/Enrich-your-map-with-natural-and-anthropic-details.png" \t "_blank)

In this post I show how to enrich a ggplot map with data obtained from the Open Street Map (OSM) API. After adding elevation details to the map, I add water bodies and elements identifying human activity. To highlight the areas more densely inhabitated, I propose to use a density-based clustering algorithm of OSM features.

**Data**

Let’s first start with some data: a few points somewhere in Italy.

my\_points.df <-

data.frame(lon = c(9.442366, 9.30222, 9.304297, 9.322077,

9.304432, 9.302683, 9.321543, 9.301541, 9.402329, 9.285115, 9.319564,

9.33959, 9.33367, 9.261872, 9.308274, 9.3011, 9.45355, 9.32527,

9.25892, 9.303647),

lat = c(45.952427, 46.041, 45.958035, 45.922367,

45.953438, 45.966248, 45.919487, 45.959721, 46.05933, 46.009094,

45.917286, 46.081585, 45.997231, 45.98764, 46.076529, 45.959641,

45.972649, 45.9151, 46.055069, 46.07671))

I will use the great [sf](https://cran.r-project.org/web/packages/sf/index.html) package so let’s create a sf spatial object with

library(sf)

my\_points.sf <- st\_as\_sf(my\_points.df, coords = c("lon", "lat"), crs = 4326)

crs = 4326 sets the [Coordinate Reference System](https://en.wikipedia.org/wiki/Spatial_reference_system) using an integer [EPSG](https://www.epsg-registry.org/) code. [EPSG:4326](https://epsg.io/4326) uses the World Geodetic System 1984 (WGS 84) that is used in GPS and its unit is the *degree* (not the meter!).

In this post I show how to enrich a ggplot map with data obtained from the Open Street Map (OSM) API. After adding elevation details to the map, I add water bodies and elements identifying human activity. To highlight the areas more densely inhabitated, I propose to use a density-based clustering algorithm of OSM features.

As a first step let’s define a bounding box that includes my points.

my\_bbox <- c(xmin = min(my\_points.df$lon),

xmax = max(my\_points.df$lon),

ymin = min(my\_points.df$lat),

ymax = max(my\_points.df$lat))

Once I have set the points, I can create a polygon with the function sf::st\_polygon(), which takes a two-column matrix for each polygon and create a simple feature (sf) object, and the function sf::st\_geometry(), which takes a simple future object and get its geometry. Points need to be five because we always need to close our polygons! So the last (or 5th) row will have the same pair of coordinates of the first row. st\_crs will set the CRS for the geometry my\_bbox.sf.

my\_bbox.m <-

matrix(c(my\_bbox['xmin'], my\_bbox['xmin'], my\_bbox['xmax'], my\_bbox['xmax'], my\_bbox['xmin'],

my\_bbox['ymax'], my\_bbox['ymin'], my\_bbox['ymin'], my\_bbox['ymax'], my\_bbox['ymax']),

ncol = 2)

my\_bbox.sf <- st\_geometry(st\_polygon(x = list(my\_bbox.m)))

st\_crs(my\_bbox.sf) <- 4326

To give some margin to my points, I create an additional buffered bbox polygon around my\_bbox.sf. I want to specify the buffer distance in meters, so I need to transform my polygon to a meter-based coordinate system such as [EPSG:32632](https://epsg.io/32632) and then back to EPSG:4326. EPSG:32632 uses the same geodetic CRS of EPSG:4326 (WGS 84) but crucially its unit is the *meter* (not the degree!) so all calculations are meter-based. EPSG:32632 is the 32N zone of the [Universal Transverse Mercator](https://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system) (UTM) coordinate system. The world grid that can help you find the best UTM zone for your project is [here](http://www.dmap.co.uk/utmworld.htm).

my\_bbox\_buff\_2500.sf <-

my\_bbox.sf %>%

st\_transform(crs = 32632) %>%

st\_buffer(dist = 2500) %>% # 2.5 kilometers

st\_transform(crs = 4326)

my\_bbox\_buff\_5000.sf <-

my\_bbox.sf %>%

st\_transform(crs = 32632) %>%

st\_buffer(dist = 5000) %>% # 5 kilometers

st\_transform(crs = 4326)

my\_bbox\_buff\_25000.sf <-

my\_bbox.sf %>%

st\_transform(crs = 32632) %>%

st\_buffer(dist = 25000) %>% # 25 kilometers

st\_transform(crs = 4326)

This is how my point data look like in its essence.

library(ggplot2)

my\_world\_map <- map\_data('world')

my\_world\_map <- my\_world\_map[my\_world\_map$region %in% c("Italy","Switzerland"),]

ggplot() +

geom\_sf(data = my\_points.sf) +

geom\_sf(data = my\_bbox\_buff\_2500.sf, fill = NA) +

coord\_sf(xlim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['xmin'], st\_bbox(my\_bbox\_buff\_25000.sf)['xmax']),

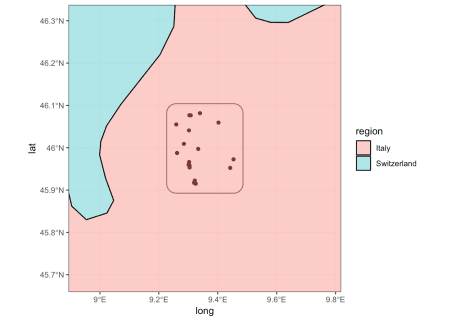
ylim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['ymin'], st\_bbox(my\_bbox\_buff\_25000.sf)['ymax'])) +

geom\_polygon(data = my\_world\_map,

aes(x=long, y = lat, group = group,

fill = region), colour = 'black', alpha = .4) +

theme\_bw()

[](https://i0.wp.com/www.francescobailo.net/wordpress/wp-content/uploads/2018/08/unnamed-chunk-7-1.png)

To give some geographic context to the data I plotted also the country polygons of Italy and Switzerland from the ggplot2::map\_data('world'). coord\_sf allows to set the limits of the plot which are extracted from the bounding box of my\_bbox\_buff\_25000.sf with the function sf::st\_bbox().

**Add elevation details**

Depending on the specific geography of the place you need to map, it might be helpful to provide an idea of the elevation. Elevation differences (that is, hills, mountains and valleys) are crucial constraints in the development of human and animal activities. To map the elevation around our points, I need to download data of a Digital Elevation Model (DEM). There are different options, I will use here data obtained by the [Shuttle Radar Topography Mission](https://en.wikipedia.org/wiki/Shuttle_Radar_Topography_Mission) (SRTM). The package [raster](https://cran.r-project.org/web/packages/raster/index.html) ships with the function (raster::getData()) to download the data directly from the R console. The arguments lat and lon are used to get the data file that is needed for the project.

library(raster)

dem.raster <- getData("SRTM", lat = 46.0146, lon = 9.344197, download = TRUE)

The SRTM data file is actually pretty large so it probably a good idea to crop it to reduce plotting time. The function raster::crop() takes an sp object (form the [sp](https://cran.r-project.org/web/packages/sp/index.html) package) to be used as mask to determine the extent of the resulting (cropped) raster. I need to convert my bbox first with as(my\_bbox\_buff\_25000.sf, 'Spatial').

dem.raster <- crop(dem.raster, as(my\_bbox\_buff\_25000.sf, 'Spatial'), snap='out')

Because I need to add my raster as a ggplot layer, I need to transform it first into a matrix of points with raster::rasterToPoints() and finally into a data.frame object. The third column of the dem.df contains the altitude value (alt).

dem.m <- rasterToPoints(dem.raster)

dem.df <- data.frame(dem.m)

colnames(dem.df) = c("lon", "lat", "alt")

I can already plot dem.df with

ggplot() +

geom\_raster(data = dem.df, aes(lon, lat, fill = alt), alpha = .45) +

scale\_fill\_gradientn(colours = terrain.colors(100)) +

geom\_sf(data = my\_bbox\_buff\_2500.sf, fill = NA) +

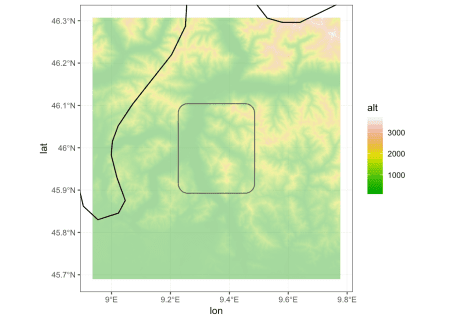
coord\_sf(xlim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['xmin'], st\_bbox(my\_bbox\_buff\_25000.sf)['xmax']),

ylim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['ymin'], st\_bbox(my\_bbox\_buff\_25000.sf)['ymax'])) +

geom\_polygon(data = my\_world\_map,

aes(x=long, y = lat, group = group), fill = NA, colour = 'black') +

theme\_bw()

[](https://i2.wp.com/www.francescobailo.net/wordpress/wp-content/uploads/2018/08/unnamed-chunk-11-1.png)

Now, instead of colouring each cell of the raster based on its altitude, I want to convey the same information with the hill shade. First, I extract the terrain characteristics (the slope and the aspect in this case) with raster::terrain(). Then, I compute the hill shade with raster::hillShade(), setting the elevation angle (of the sun) to 40 and the direction angle of the light to 270. I then transform the resulting hill.raster into a data.frame as before.

slope.raster <- terrain(dem.raster, opt='slope')

aspect.raster <- terrain(dem.raster, opt='aspect')

hill.raster <- hillShade(slope.raster, aspect.raster, 40, 270)

hill.m <- rasterToPoints(hill.raster)

hill.df <- data.frame(hill.m)

colnames(hill.df) <- c("lon", "lat", "hill")

And here we go

ggplot() +

geom\_raster(data = hill.df, aes(lon, lat, fill = hill), alpha = .45) +

scale\_fill\_gradientn(colours = grey.colors(100)) +

geom\_sf(data = my\_bbox\_buff\_2500.sf, fill = NA) +

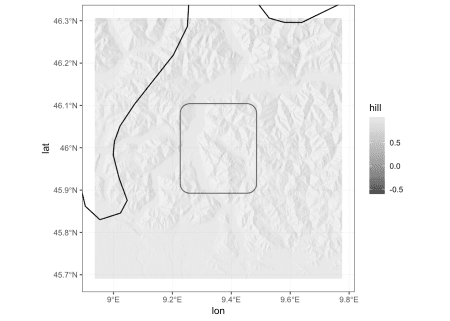
coord\_sf(xlim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['xmin'], st\_bbox(my\_bbox\_buff\_25000.sf)['xmax']),

ylim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['ymin'], st\_bbox(my\_bbox\_buff\_25000.sf)['ymax'])) +

geom\_polygon(data = my\_world\_map,

aes(x=long, y = lat, group = group), fill = NA, colour = 'black') +

theme\_bw()

[](https://i1.wp.com/www.francescobailo.net/wordpress/wp-content/uploads/2018/08/unnamed-chunk-13-1.png)

**Add bodies of water**

As you might have guessed from the presence of a few flat surfaces, there are lakes in the region. We can use the excellent [osmdata](https://cran.r-project.org/web/packages/osmdata/index.html) package to query the [Open Street Map (OSM)](https://www.openstreetmap.org/) [Overpass API](https://wiki.openstreetmap.org/wiki/Overpass_API) to download information about lakes and rivers. With the function osmdata::opq() we define the bbox for the query and with osmdata::add\_osm\_feature() we extract specific OSM features using the [tags](https://wiki.openstreetmap.org/wiki/Tags#Keys_and_values) (key and value).

library(osmdata)

osm\_lakes.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_25000.sf)) %>%

add\_osm\_feature(key = 'water', value = 'lake') %>%

osmdata\_sf()

osm\_lakes.sf <- osm\_lakes.sf$osm\_multipolygons

osm\_rivers.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_25000.sf)) %>%

add\_osm\_feature(key = 'waterway', value = 'river') %>%

osmdata\_sf()

osm\_rivers.sf <- osm\_rivers.sf$osm\_lines

ggplot() +

geom\_raster(data = hill.df, aes(lon, lat, fill = hill), alpha = .45) +

scale\_fill\_gradientn(colours = grey.colors(100)) +

geom\_sf(data = osm\_lakes.sf, fill = '#9ecae1', colour = NA) +

geom\_sf(data = osm\_rivers.sf, colour = '#9ecae1', size = 0.05) +

geom\_sf(data = my\_bbox\_buff\_2500.sf, fill = NA) +

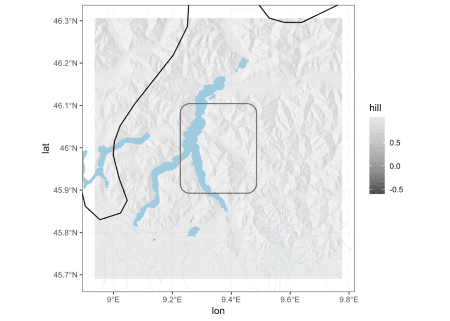
coord\_sf(xlim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['xmin'], st\_bbox(my\_bbox\_buff\_25000.sf)['xmax']),

ylim = c(st\_bbox(my\_bbox\_buff\_25000.sf)['ymin'], st\_bbox(my\_bbox\_buff\_25000.sf)['ymax'])) +

geom\_polygon(data = my\_world\_map,

aes(x=long, y = lat, group = group), fill = NA, colour = 'black') +

theme\_bw()

[](https://i1.wp.com/www.francescobailo.net/wordpress/wp-content/uploads/2018/08/unnamed-chunk-14-1.png)

**Add roads**

Similarly we can download details about roads. The road network on OSM might be very dense, so it is better to do some filtering using the OSM [tags](https://wiki.openstreetmap.org/wiki/Key:highway).

osm\_roads\_primary.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_5000.sf)) %>%

add\_osm\_feature(key = 'highway', value = 'trunk') %>%

osmdata\_sf()

osm\_roads\_primary.sf <- osm\_roads\_primary.sf$osm\_lines

osm\_roads\_secondary.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_5000.sf)) %>%

add\_osm\_feature(key = 'highway', value = 'secondary') %>%

osmdata\_sf()

osm\_roads\_secondary.sf <- osm\_roads\_secondary.sf$osm\_lines

osm\_roads\_tertiary.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_5000.sf)) %>%

add\_osm\_feature(key = 'highway', value = 'tertiary') %>%

osmdata\_sf()

osm\_roads\_tertiary.sf <- osm\_roads\_tertiary.sf$osm\_lines

ggplot() +

geom\_raster(data = hill.df, aes(lon, lat, fill = hill), alpha = .45) +

scale\_fill\_gradientn(colours = grey.colors(100)) +

geom\_sf(data = osm\_lakes.sf, fill = '#9ecae1', colour = NA) +

geom\_sf(data = osm\_rivers.sf, colour = '#9ecae1', size = 0.05) +

geom\_sf(data = osm\_roads\_primary.sf, colour = '#636363', size = 0.1) +

geom\_sf(data = osm\_roads\_secondary.sf, colour = '#636363', size = 0.05) +

geom\_sf(data = osm\_roads\_tertiary.sf, colour = '#636363', size = 0.02) +

geom\_sf(data = my\_bbox\_buff\_2500.sf, fill = NA) +

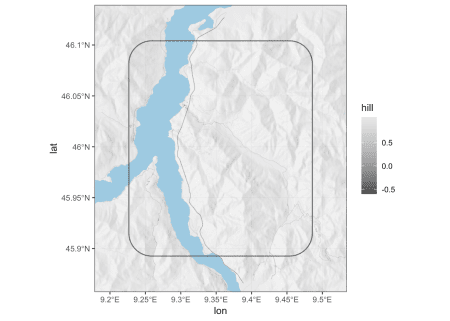
coord\_sf(xlim = c(st\_bbox(my\_bbox\_buff\_5000.sf)['xmin'], st\_bbox(my\_bbox\_buff\_5000.sf)['xmax']),

ylim = c(st\_bbox(my\_bbox\_buff\_5000.sf)['ymin'], st\_bbox(my\_bbox\_buff\_5000.sf)['ymax'])) +

geom\_polygon(data = my\_world\_map,

aes(x=long, y = lat, group = group), fill = NA, colour = 'black') +

theme\_bw()

[](https://i0.wp.com/www.francescobailo.net/wordpress/wp-content/uploads/2018/08/unnamed-chunk-15-1.png)

**Identify densely populated areas**

Densely populated areas are certainly very important in every map that wants to describe human activities. Again, I use OSM data to identify these areas. First, I query for features that are usually associated with human residency: residential roads and buildings. On OSM roads and buildings are [not mapped with the same completeness everywhere](https://stefanodesabbata.com/2014/05/28/openstreetmap/) but the density of their presence might provide a good enough approximation of where people actually live. The number of such features (polygons and lines) is usually very large. But actually I don’t need *individual features*, I just need *the geography of densely populated areas*.

First, I query the OSM API for all the buildings (osm\_polygons), residential roads and unclassified roads (osm\_lines). Second, I transform polygons and lines into points with sf::st\_centroid() (which requires a CRS transformation). Finally, I store all the point simple feature objects into osm\_residential\_areas\_pnt.df, with two columns respectively for longitude and latitude, and create from it the spatial object osm\_residential\_areas\_pnt.sf.

osm\_buildings.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_2500.sf)) %>%

add\_osm\_feature(key = 'building') %>%

osmdata\_sf()

osm\_buildings.sf <- osm\_buildings.sf$osm\_polygons

osm\_buildings\_pnt.sf <-

osm\_buildings.sf %>%

st\_transform(crs = 32632) %>%

st\_centroid() %>%

st\_transform(crs = 4326)

osm\_roads\_residential.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_5000.sf)) %>%

add\_osm\_feature(key = 'highway', value = 'residential') %>%

osmdata\_sf()

osm\_roads\_residential.sf <- osm\_roads\_residential.sf$osm\_lines

osm\_roads\_residential\_pnt.sf <-

osm\_roads\_residential.sf %>%

st\_transform(crs = 32632) %>%

st\_centroid() %>%

st\_transform(crs = 4326)

osm\_roads\_unclassified.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_5000.sf)) %>%

add\_osm\_feature(key = 'highway', value = 'unclassified') %>%

osmdata\_sf()

osm\_roads\_unclassified.sf <- osm\_roads\_unclassified.sf$osm\_lines

osm\_roads\_unclassified\_pnt.sf <-

osm\_roads\_unclassified.sf %>%

st\_transform(crs = 32632) %>%

st\_centroid() %>%

st\_transform(crs = 4326)

osm\_residential\_areas\_pnt.df <-

rbind(

do.call(rbind,

st\_geometry(osm\_buildings\_pnt.sf)),

do.call(rbind,

st\_geometry(osm\_roads\_residential\_pnt.sf)),

do.call(rbind,

st\_geometry(osm\_roads\_unclassified\_pnt.sf))) %>%

as.data.frame()

colnames(osm\_residential\_areas\_pnt.df) <- c('lon', 'lat')

osm\_residential\_areas\_pnt.sf <-

st\_as\_sf(osm\_residential\_areas\_pnt.df, coords = c("lon", "lat"), crs = 4326)

osm\_residential\_areas\_pnt.sf <-

st\_transform(osm\_residential\_areas\_pnt.sf, 32632)

Once I have the coordinate data.frame, with one row for each OSM feature, I pass it to the function dbscan::dbscan() of the [dbscan](https://cran.r-project.org/web/packages/sf/index.html) package, which implements a *Density-based spatial clustering* algorithm to identify areas with the highest density of points . eps and minPts, respectively the size of the neighbourhood and the number of minimum points for each region, should be adjusted accordingly to the local geography. Once the number the of clusters are identified, I can create a polygon with the [convex hull](https://en.wikipedia.org/wiki/Convex_hull) of the clustered points with the function grDevices::chull(). Each created polygon is stored the spatial object pop\_dense\_areas.sf.

library(dbscan)

res <- dbscan(osm\_residential\_areas\_pnt.df, eps = 0.0005, minPts = 10)

pop\_dense\_areas.sf <-

st\_sf(id = 1:max(res$cluster),

geometry = st\_sfc(lapply(1:max(res$cluster), function(x) st\_geometrycollection())))

st\_crs(pop\_dense\_areas.sf) <- 4326

pop\_dense\_areas.sf <- st\_transform(pop\_dense\_areas.sf, 32632)

for (cl in 1:max(res$cluster)) {

these\_points <- osm\_residential\_areas\_pnt.df[which(res$cluster == cl),]

this\_chull <- chull(these\_points)

this\_mat <- as.matrix(these\_points[this\_chull,])

this\_mat <- rbind(this\_mat, this\_mat[1,])

this\_polygon <- st\_geometry(st\_polygon(x = list(this\_mat)))

st\_crs(this\_polygon) <- 4326

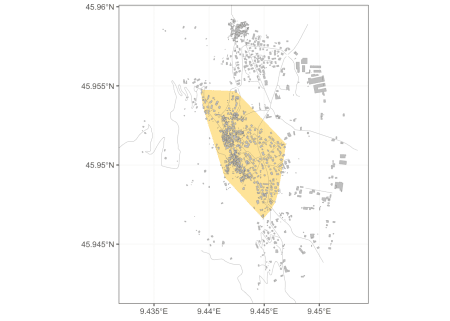
this\_polygon <- st\_transform(this\_polygon, 32632)

pop\_dense\_areas.sf$geometry[cl] <- this\_polygon

}

pop\_dense\_areas.sf <- st\_transform(pop\_dense\_areas.sf, 4326)

To give an idea of the actual result, you see below a densely populated area inferred from the cluster of points derived from building and residential roads. The algorithm successfully identified relatively more anthorpised regions. Tweaking of the two arguments of the density-based clustering function might be necessary depending on the size and human geography of the final map.

[](https://i1.wp.com/www.francescobailo.net/wordpress/wp-content/uploads/2018/08/unnamed-chunk-18-1.png)

The final enrichment of my map is the addition of a few names of populated places (*towns* and *villages* in the OSM tagging system). Again I use the OSM API. I filter the results for the more important places based on their relative population. To avoid overcrowding the map with labels, I only select the 10 larger places. geom\_sf doesn’t allow to label points (yet). It is then necessary to convert the spatial object into a simple data.frame with two coordinate columns. Names might be too long, so I add a \n (newline) after 10 characters with gsub('(.{1,10})(\\s|$)', '\\1\n', str).

library(dplyr)

osm\_villages.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_2500.sf)) %>%

add\_osm\_feature(key = 'place', value = 'village') %>%

osmdata\_sf()

osm\_villages.sf <- osm\_villages.sf$osm\_points

osm\_towns.sf <-

opq(bbox = st\_bbox(my\_bbox\_buff\_2500.sf)) %>%

add\_osm\_feature(key = 'place', value = 'town') %>%

osmdata\_sf()

osm\_towns.sf <- osm\_towns.sf$osm\_points

osm\_larger\_places.df <-

rbind(cbind(as.data.frame(osm\_villages.sf)[,c('name','population')],

st\_coordinates(osm\_villages.sf)),

cbind(as.data.frame(osm\_towns.sf)[,c('name','population')],

st\_coordinates(osm\_towns.sf))) %>%

mutate(population = as.numeric(as.character(population))) %>%

top\_n(10, population)

osm\_larger\_places.df$name <- gsub('(.{1,10})(\\s|$)', '\\1\n', osm\_larger\_places.df$name)

And now let’s plot everything along with my initial data points:

ggplot() +

geom\_raster(data = hill.df, aes(lon, lat, fill = hill), alpha = .45) +

scale\_fill\_gradientn(colours = grey.colors(100)) +

geom\_sf(data = pop\_dense\_areas.sf, fill = '#fee08b', colour = NA, alpha = 0.9) +

geom\_sf(data = osm\_lakes.sf, fill = '#9ecae1', colour = NA) +

geom\_sf(data = osm\_rivers.sf, colour = '#9ecae1', size = 0.4) +

geom\_sf(data = osm\_roads\_primary.sf, colour = '#636363', size = 0.1) +

geom\_sf(data = osm\_roads\_secondary.sf, colour = '#636363', size = 0.08) +

geom\_sf(data = osm\_roads\_tertiary.sf, colour = '#636363', size = 0.08) +

geom\_sf(data = my\_points.sf, shape = 5, colour = 'black', size = .5) +

geom\_text(data = osm\_larger\_places.df, aes(x=X,y=Y,label=name), size = 2.5, alpha = .65) +

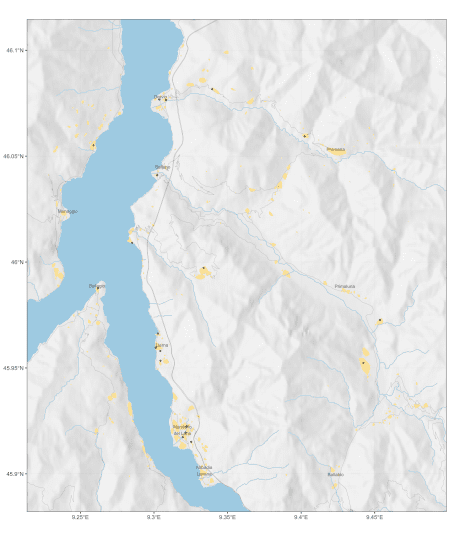
coord\_sf(xlim = c(st\_bbox(my\_bbox\_buff\_2500.sf)['xmin'], st\_bbox(my\_bbox\_buff\_2500.sf)['xmax']),

ylim = c(st\_bbox(my\_bbox\_buff\_2500.sf)['ymin'], st\_bbox(my\_bbox\_buff\_2500.sf)['ymax'])) +

guides(fill=FALSE)+

labs(x=NULL, y=NULL) +

theme\_bw()

[](https://i0.wp.com/www.francescobailo.net/wordpress/wp-content/uploads/2018/08/Enrich-your-map-with-natural-and-anthropic-details.png)

**References**

Hahsler, Michael, and Matthew Piekenbrock. 2018. *Dbscan: Density Based Clustering of Applications with Noise (Dbscan) and Related Algorithms*. [https://CRAN.R-project.org/package=dbscan](https://cran.r-project.org/package=dbscan).

Hijmans, Robert J. 2017. *Raster: Geographic Data Analysis and Modeling*. [https://CRAN.R-project.org/package=raster](https://cran.r-project.org/package=raster).

Padgham, Mark, Bob Rudis, Robin Lovelace, and Maëlle Salmon. 2018. *Osmdata: Import ’Openstreetmap’ Data as Simple Features or Spatial Objects*. [https://CRAN.R-project.org/package=osmdata](https://cran.r-project.org/package=osmdata).

Pebesma, Edzer. 2018. *Sf: Simple Features for R*. [https://CRAN.R-project.org/package=sf](https://cran.r-project.org/package=sf).

Wickham, Hadley, Winston Chang, Lionel Henry, Thomas Lin Pedersen, Kohske Takahashi, Claus Wilke, and Kara Woo. 2018. *Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. [https://CRAN.R-project.org/package=ggplot2](https://cran.r-project.org/package=ggplot2).

Wickham, Hadley, Romain Francois, Lionel Henry, and Kirill Müller. 2017. *Dplyr: A Grammar of Data Manipulation*. [https://CRAN.R-project.org/package=dplyr](https://cran.r-project.org/package=dplyr).

Xie, Yihui. 2018. *Knitr: A General-Purpose Package for Dynamic Report Generation in R*. [https://CRAN.R-project.org/package=knitr](https://cran.r-project.org/package=knitr).