Aggregating Earth Surface data using GEE and R

Francesco Pirotti <francesco.pirotti@unipd.it>

#Summer Webinar Series — Sept. 29, 2020

Contents

Code & Copyleft

Code available in  GITHUB here

Agenda

- From a regular lattice of points create squares
- In each square assign values from aggregating data (e.g. height above sea level of Earth surface)

R CRAN

Starters

We use RStudio IDE. Create a .R file (ours is called *webinar1_2020_07_29.R*) and write the following lines of code. Ctrl-Return to run the line that cursor is on or run all highlighted lines. Or use the button in top right part of the editor.

To add functionalities to your .R script you must load the specific libraries. **REMEMBER:** make sure that they are installed in your system. If not you can install through RStudio on **menu => Tools => Install Packages** or directly through R console with command `install.packages("NAME OF PACKAGE")`

Below we add three libraries which add functionalities that we need

```
library(sf)
library(mapview)
library(raster)
```

Coordinate Reference System

My custom Coordinate Reference System (CRS) projection, Lambert Conical Conformal (LCC), NOT secant but tangent at lat=45.827 and lon=11.625 - for more info:

- PROJ library
- Wikipedia
- Wolfram Mathworld
- John P. Snyder "Map projections: A working manual" (pp. 104-110)

```
myproj <- "+proj=lcc +lat_1=45.827  +lat_2=45.827
           +lat_0=45.827 +lon_0=11.625
           +x_0=4000000
           +y_0=2800000 +ellps=GRS80
           +towgs84=0,0,0,0,0,0,0 +units=m +no_defs"
```

Read lattice

My points (in lat long) over the regular grid/lattice these points are 666.67 m apart

```
grid.points <- st_read( "data/webinar1_2020_07_29/points.shp" )
```

```
## Reading layer `points' from data source `/archivio/R/shared/webinars/data/webinar1_2020_07_29/points
## Simple feature collection with 1315 features and 1 field
## geometry type:  POINT
```

```
## dimension:       XY
## bbox:            xmin: 11.83267 ymin: 46.33037 xmax: 12.14322 ymax: 46.55611
## geographic CRS: WGS 84
```

Transform CRS

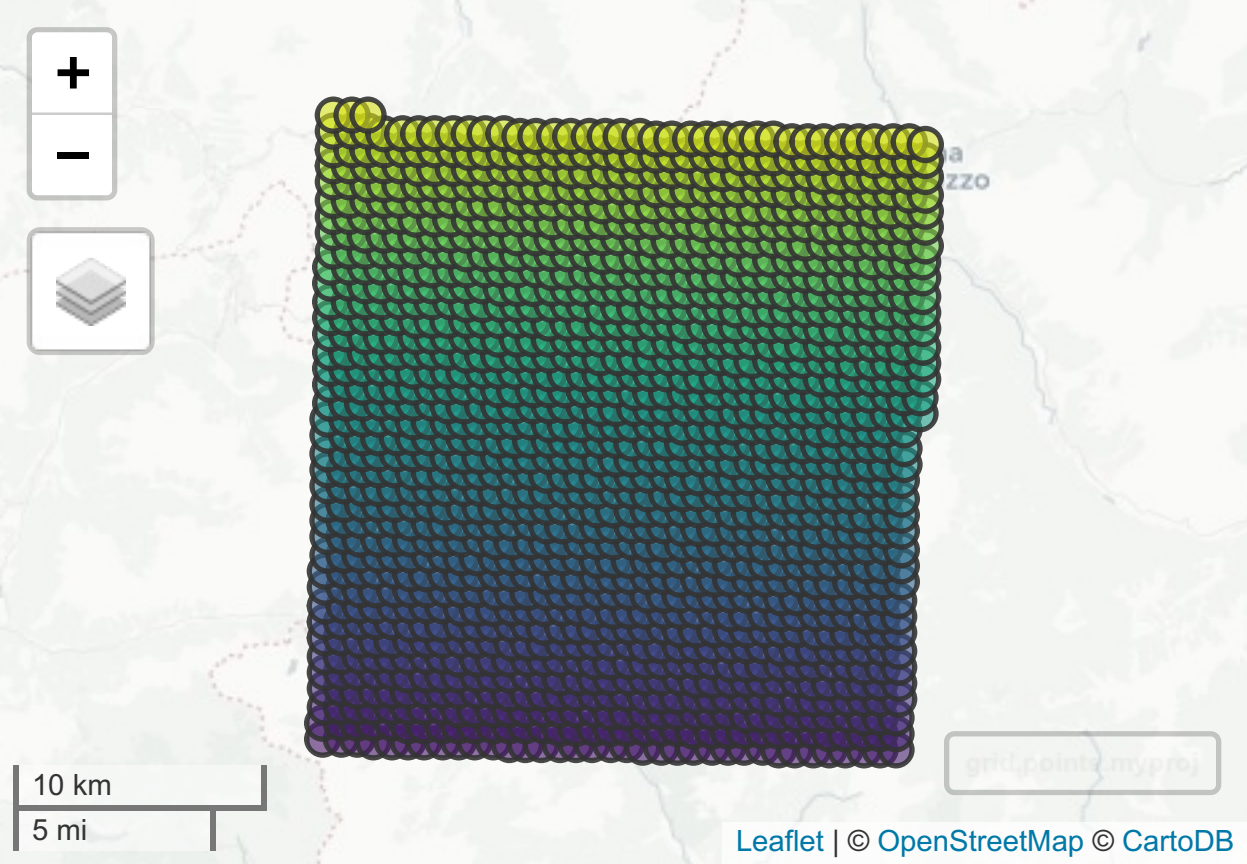
Convert my points from a Geographic (latitude and longitude degrees) to my custom CRS projection

```
grid.points.myproj <-   grid.points %>% st_transform(myproj)
```

View in webmap

Run this line to view the points

```
  mapview( grid.points.myproj, legend = F  )
```



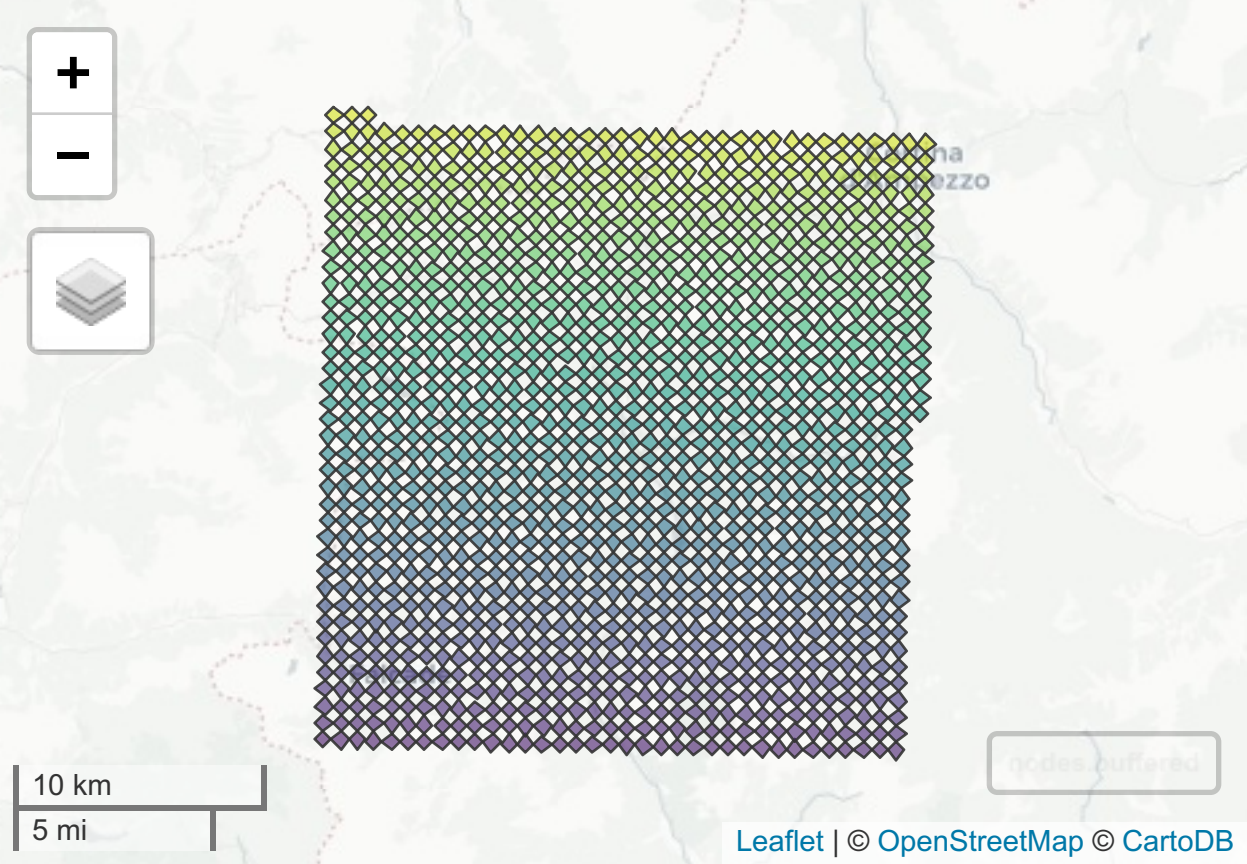
Points to Squares

Points to a square Polygon areas. How? It is trivial, but faster way was to create a rhomboid with a certain distance from the point ("radius") using a **buffer** of half the distance between points (333.33 m) . The minimum bounding box of the buffer with that radius will be a square with sides 2x the size of the radius of the buffer.

NB: nQuadSegs is 1 which means a chord(segment) at each quadrand, i.e. 1/4th of a circunference; this give a rhomboid which is a "very very rough circle" - it saves a lot of memory from drawing with a defaul value of *nQuadSegs=30* - it makes a difference when processing a large number of points.

We buffer each feature (point) and view results

```
nodes.buffered<-st_buffer(grid.points.myproj, 333.33,  nQuadSegs = 1)
mapview( nodes.buffered, legend = F  )
```



Then we apply a function to each "rhomboid" for creating the square. It is a bit more complex as the function includes another function inside.

```
st_bbox_by_feature = function(geom) {

  ## make sure that object "geom" becomes a geometry object;
  ## geom is your data set with all the single geometries/polygons
  geom2 = st_geometry(geom)

  #' Function to:
```

```
  #' (a) take a single geometry,
  #' (b) create a bounding box with st_bbox and
  #' (c) convert the bbox object to a new geometry (a square)
  f <- function(single.geom) {
    st_as_sfc( st_bbox(single.geom,), crs=myproj)
  }

  #' This line calls the function above LOOPING over each single geometry (lapply)
  #'  in the geom2 object
  do.call("c", lapply(geom2, f))
}
```

This line calls the above function over our data (rhomboids)

```
tiles <- st_bbox_by_feature( nodes.buffered )
```

Assign the CRS to the new dataset as it got lost. To check the CRS of our data we can use `{r} st_crs(tiles)`

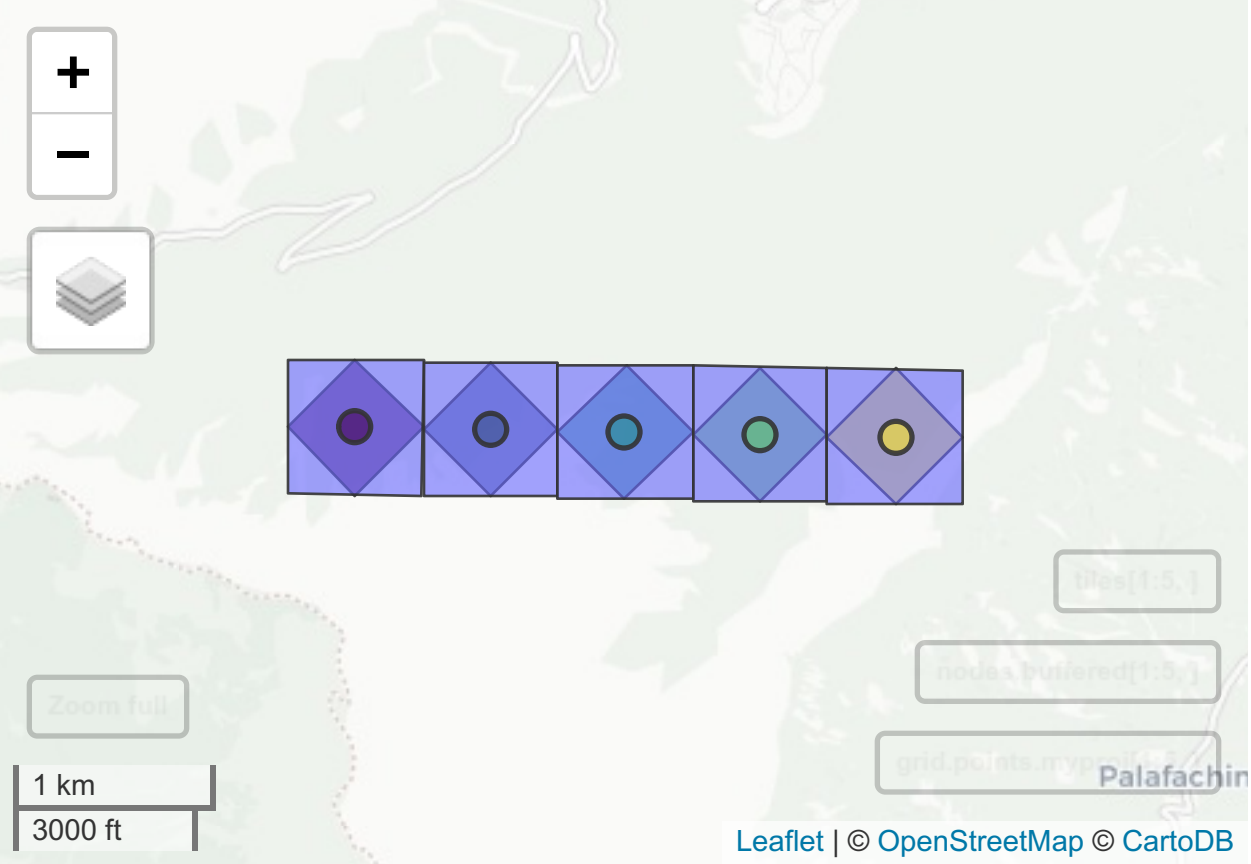
```
tiles <- tiles %>% st_set_crs(myproj)
```

We view all results all over the same map (small subset of data - the first 20 features).

PS you might notice a mis-alignment of the geometries, this is due to the webgis using a CRS called "earth universal transverse mercator" projection which as small deformations when representing data that is in other CRS.

For more reading see Wikipedia.

```
mapview( grid.points.myproj[1:5,], legend = F ) + mapview( nodes.buffered[1:5,], legend = F  )  + mapvi
```



Save result

Convert to Latitude and Longitude and save result to shapefile for future upload to Google Earth Engine

```
tiles.latlng <- tiles %>% st_transform("+init=epsg:4326")
st_write(tiles.latlng, "data/tiles.shp" )
```

GOOGLE EARTH ENGINE (GEE)

Import shapefile files "tiles.XXX" by left panel "assets" and click "NEW" and choose shapefile from dropdown menu.

DO THINGS in GEE

code shared: [click HERE to open your GEE code editor] but also reported here below

```
// IMPORTS
// these are in the "import" top part of the script editor in GEE,
// but can also be added to the script itself.

var srtm = ee.Image("CGIAR/SRTM90_V4");

var  srtm_viz = {"opacity":1,"bands":["elevation"],
      "min":1439.9060309998727,"max":2892.700269813135,
      "palette":["005aff","ff0000"]};
```

```
var tiles = ee.FeatureCollection("users/2020_Kanan/summer_webinar_series/Webinar1_tiles");
///
////////////////////////////////////////////////////////////////////////////////
///////////////// PART OF SUMMER WEBINAR SERIES 2020 ///////////////////////////
////////////////////////////////////////////////////////////////////////////////
/////  Francesco Pirotti - CIRGEO / TESAF Department University of Padova ///////////
////////////////////////////////////////////////////////////////////////////////
/////  https://www.cirgeo.unipd.it/shared/R/webinars/webinar1_2020_07_29.html  ////////
////////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////////



// "run" this script and then go to "tasks" button in the right panel to
// run also the export of the data.
// The last function in this script, "Export...",
// will create a "task" that you have to launch manually
// to put the final data  to your google drive


// this prints info on the "srtm" dataset that we imported
print(srtm)
// PS double click in the link "SRTM Digital Elevation Data Version 4." above in the
// "imports" to see what data we are reading and aggregating at tiles
////////////////////////////////////////////////////////////////////////////////

// this prints info on the "tiles" dataset that we imported from the
// shapefile created with R
print(tiles);
////////////////////////////////////////////////////////////////////////////////

// this function below applies the "reduceRegions" function to the "srtm" dataset (image)
// over each feature (square polygon in our "tiles" dataset) ...
// NB the "reducer" function applies 5 percentiles (10, 25, 50, 75, 90) thus allowing a good
// representation of distribution of height values inside the square.
// We could also use average and standard deviation as reducers.
var  height =  srtm.reduceRegions({
  reducer: ee.Reducer.percentile([10,25,50,75,90]),
  //reducer: ee.Reducer.mean(),
  collection: tiles
});

// this prints result
print(height);

// here we zoom to our tiles and add the layers
Map.centerObject(tiles, 12)
Map.addLayer(srtm, srtm_viz, "SRTM");
Map.addLayer(tiles, {}, "TILES");

// this exports the data to our Google drive
 Export.table.toDrive({
    collection: height,
```

```
    description:'tiles_withData',
    fileFormat: 'SHP'
  });
```

EXPORT THE RESULTS TO A SHAPEFILE CALLED tiles_withData.shp

Read the file exported from GEE

```
grid.points.gee <- st_read( "data/webinar1_2020_07_29/tiles_withData.shp" )
```

```
## Reading layer `tiles_withData' from data source `/archivio/R/shared/webinars/data/webinar1_2020_07_29
## Simple feature collection with 1315 features and 6 fields
## geometry type:  POLYGON
## dimension:      XY
## bbox:           xmin: 11.82833 ymin: 46.32735 xmax: 12.14759 ymax: 46.55912
## geographic CRS: WGS 84
```

R - Plot

Calculate new IQR field

Create a new attribute column with the interquartile range (The difference between p75 and p25, i.e. 75th and 25th percentile)

```
grid.points.gee$iqr<-grid.points.gee$p75 - grid.points.gee$p25
```

PLOT variables

```
pal = mapviewPalette("mapviewSpectralColors")
mapview( grid.points.gee, col.regions = pal(100), zcol = "p50",
         color=NULL, alpha.regions=0.8  )
```

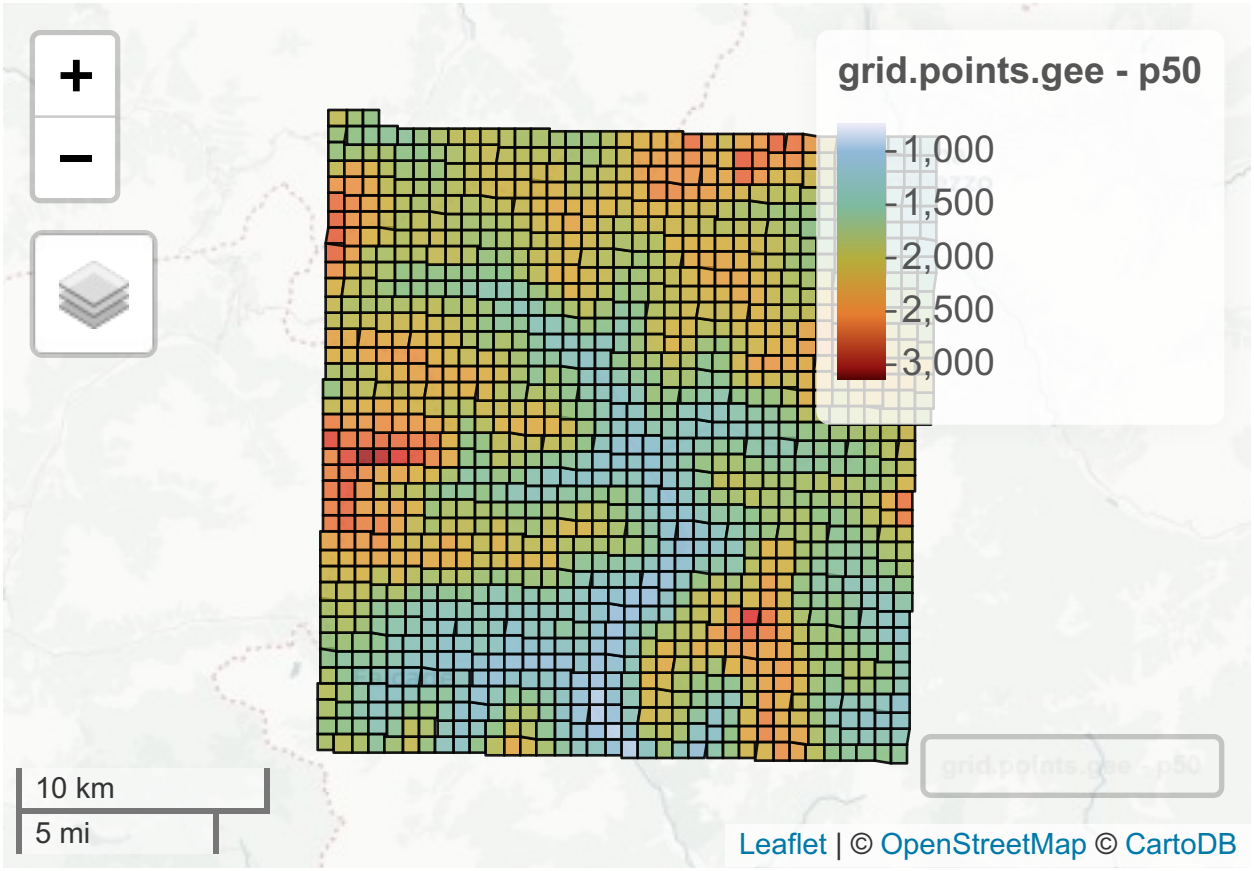


Figure 1 - 50th Percentile (median)

```
mapview( grid.points.gee, col.regions = pal(100), zcol = "p10",
         color=NULL, alpha.regions=0.8  )
```

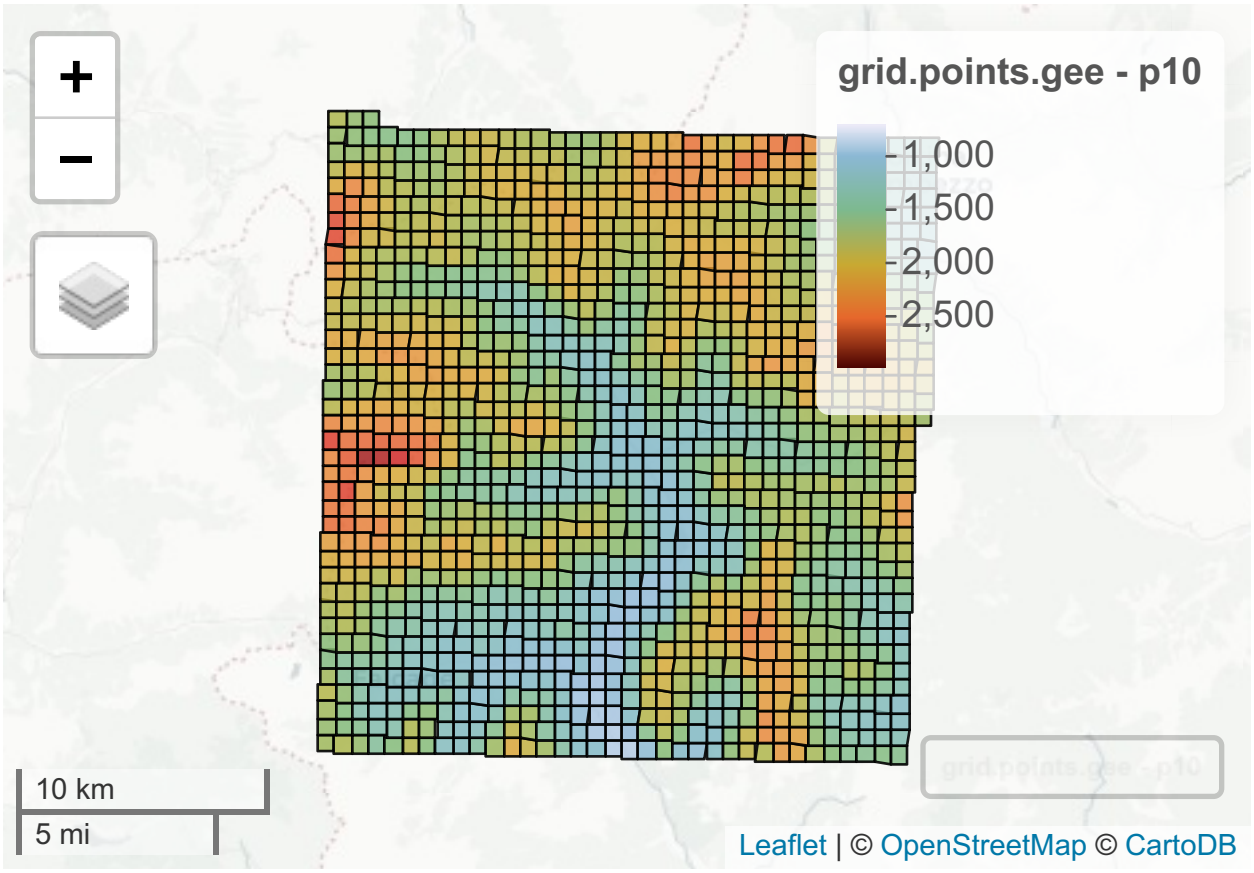


Figure 2 - 10th Percentile

```
mapview( grid.points.gee, col.regions = pal(100), zcol = "iqr",
         color=NULL, alpha.regions=0.8)
```

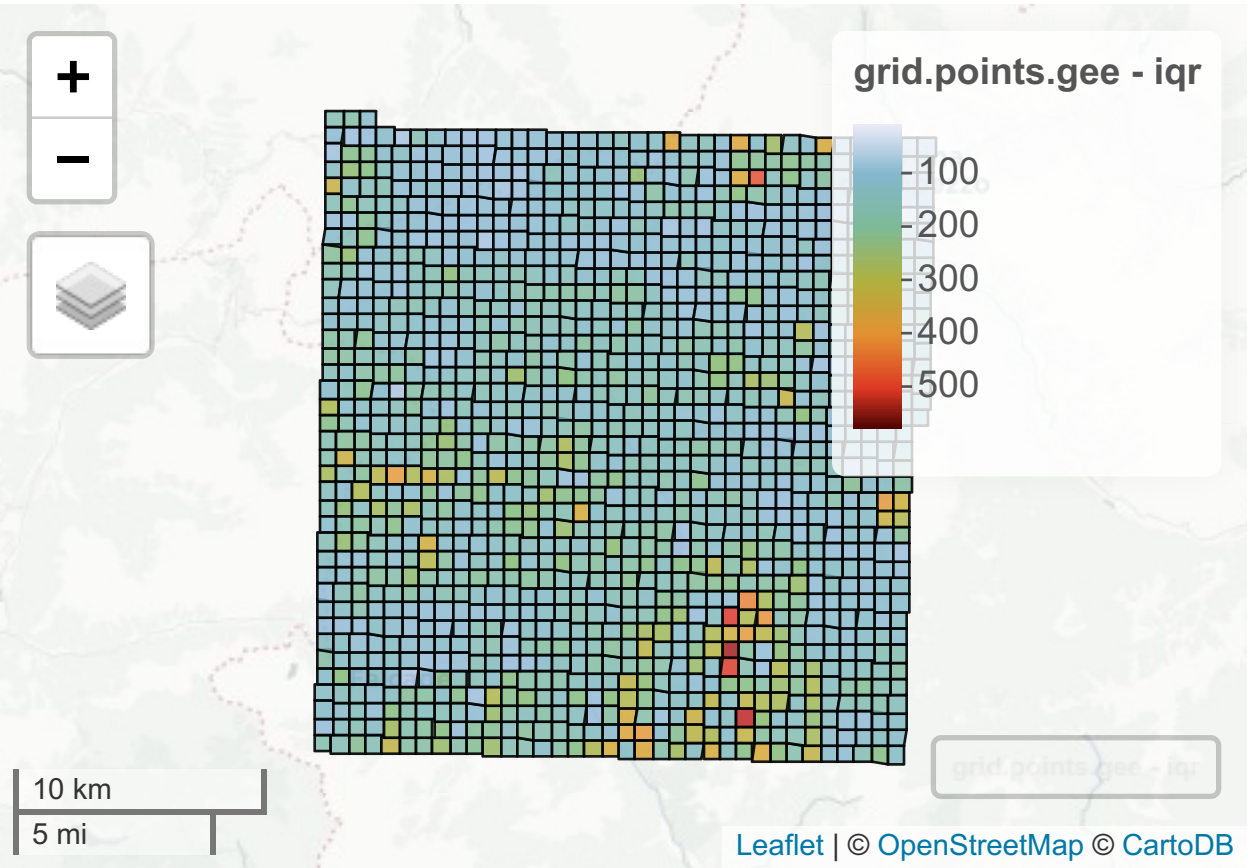


Figure 3 - Inter Quartile Range

```
sessionInfo()
```

```
## R version 3.6.3 (2020-02-29)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.1 LTS
##
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/atlas/libblas.so.3.10.3
## LAPACK: /usr/lib/x86_64-linux-gnu/atlas/liblapack.so.3.10.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
```

```
## [1] raster_3.3-13 sp_1.4-1      mapview_2.7.8 sf_0.9-5
##
## loaded via a namespace (and not attached):
##  [1] tidyselect_0.2.5   xfun_0.10          purrr_0.3.3
##  [4] lattice_0.20-40    colorspace_1.4-1   htmltools_0.4.0
##  [7] stats4_3.6.3       viridisLite_0.3.0  yaml_2.2.0
## [10] base64enc_0.1-3    rlang_0.4.5        leafpop_0.0.1
## [13] e1071_1.7-2        later_1.0.0        pillar_1.4.2
## [16] glue_1.3.1         DBI_1.1.0          stringr_1.4.0
## [19] munsell_0.5.0      htmlwidgets_1.5    codetools_0.2-16
## [22] evaluate_0.14      knitr_1.25         callr_3.3.2
## [25] fastmap_1.0.1      ps_1.3.0           httpuv_1.5.2
## [28] crosstalk_1.0.0    class_7.3-15       leafem_0.1.1
## [31] Rcpp_1.0.3         KernSmooth_2.23-16 xtable_1.8-4
## [34] promises_1.1.0     scales_1.0.0       classInt_0.4-1
## [37] satellite_1.0.1    jsonlite_1.6       leaflet_2.0.2
## [40] webshot_0.5.1      mime_0.7           png_0.1-7
## [43] digest_0.6.21      stringi_1.4.6      processx_3.4.2
## [46] dplyr_0.8.5        shiny_1.4.0        grid_3.6.3
## [49] tools_3.6.3        magrittr_1.5       tibble_2.1.3
## [52] crayon_1.3.4       pkgconfig_2.0.3    assertthat_0.2.1
## [55] rmarkdown_2.2      R6_2.4.1           units_0.6-4
## [58] compiler_3.6.3
```