# Local Spatial Autocorrelation 1

R Notes

Luc Anselin and Grant Morrison*

06/30/2019

## Contents



*University of Chicago, Center for Spatial Data Science – anselin@uchicago.edu,morrisonge@uchicago.edu

## Introduction

This notebook cover the functionality of the Local Spatial Autocorrelation section of the GeoDa workbook. We refer to that document for details on the methodology, references, etc. The goal of these notes is to approximate as closely as possible the operations carried out using GeoDa by means of a range of R packages.

The notes are written with R beginners in mind, more seasoned R users can probably skip most of the comments on data structures and other R particulars. Also, as always in R, there are typically several ways to achieve a specific objective, so what is shown here is just one way that works, but there often are others (that may even be more elegant, work faster, or scale better).

For this notebook, we use Cleveland house price data. Our goal in this lab is show how to assign spatial weights based on different distance functions.

### Objectives

After completing the notebook, you should know how to carry out the following tasks:

- Identify clusters with the Local Moran cluster map and significance map
- Identify clusters with the Local Geary cluster map and significance map
- Identify clusters with the Getis-Ord Gi and Gi* statistics
- Identify clusters with the Local Join Count statistic
- Interpret the spatial footprint of spatial clusters
- Assess potential interaction effects by means of conditional cluster maps
- Assess the significance by means of a randomization approach
- Assess the sensitivity of different significance cut-off values
- Interpret significance by means of Bonferroni bounds and the False Discovery Rate (FDR)

### R Packages used

- **sf**: To read in the shapefile and make queen contiguity weights
- **spdep**: To create spatial weights structure from neighbors structure
- **robustHD**: To compute standarized scores for variables and lag variables
- **tmap**: To construct significance and cluster maps with custom functions
- **tidyverse**: To manipulate the data
- **RColorBrewer**: To create custom color palattes that mirror the GeoDa cluster and significance maps

### R Commands used

Below follows a list of the commands used in this notebook. For further details and a comprehensive list of options, please consult the R documentation.

- **Base R**: `install.packages`, `library`, `setwd`, `summary`, `attributes`, `lapply`, `class`, `length`, `rev`, `cut`, `mean`, `sample`, `as.data.frame`, `matrix`, `unique`, `as.character`, `which`, `order`, `data.frame`, `ifelse`, `sum`, `rep`, `set.seed`
- **sf**: `st_read`, `st_relate`
- **spdep**: `nb2listw`, `lag.listw`

- **robustHD**: `standardized`
- **tmap**: `tm_shape`, `tm_borders`, `tm_fill`, `tm_layout`, `tm_facets`
- **tidyverse**: `filter`, `mutate`
- **RColorBrewer: `brewer.pal`

## Preliminaries

Before starting, make sure to have the latest version of R and of packages that are compiled for the matching version of R (this document was created using R 3.5.1 of 2018-07-02). Also, optionally, set a working directory, even though we will not actually be saving any files.[1]

### Load packages

First, we load all the required packages using the `library` command. If you don't have some of these in your system, make sure to install them first as well as their dependencies.[2] You will get an error message if something is missing. If needed, just install the missing piece and everything will work after that.

```
library(sf)
library(spdep)
library(tmap)
library(tidyverse)
library(RColorBrewer)
library(robustHD)
library(geodaData)
library(Matrix)
library(biclust)
```

### geodaData

All of the data for the R notebooks is available in the **geodaData** package. We loaded the library earlier, now to access the individual data sets, we use the double colon notation. This works similar to to accessing a variable with `$`, in that a drop down menu will appear with a list of the datasets included in the package. For this notebook, we use `guerry`.

```
guerry <- geodaData::guerry
```

### Making the weights

To start we create a function for queen contiguity, which is just `st_relate` with the specified pattern for queen contiguity which is `F***T****`

```
st_queen <- function(a, b = a) st_relate(a, b, pattern = "F***T****")
```

We apply the queen contiguity function to the voronoi polygons and see that the class of the output is **sgbp**. This structure is close to the **nb** structure, but has a few difference that we will need to correct to use the rest of **spdep** functionality.

```
queen.sgbp <- st_queen(guerry)
class(queen.sgbp)
```

[1] Use `setwd(directorypath)` to specify the working directory.
[2] Use `install.packages(packagename)`.

```
## [1] "sgbp"
```

This function converts type **sgbp** to **nb**. It is covered in more depth in the Contiguity Based Weight notebook. In short, it explicitly changes the name of the class and deals with the observations that have no neighbors.

```
as.nb.sgbp <- function(x, ...) {
  attrs <- attributes(x)
  x <- lapply(x, function(i) { if(length(i) == 0L) 0L else i } )
  attributes(x) <- attrs
  class(x) <- "nb"
  x
}
```

```
queen.nb <- as.nb.sgbp(queen.sgbp)
```

To go from neighbors object to weights object, we use `nb2listw`, with default parameters, we will get row standardized weights.

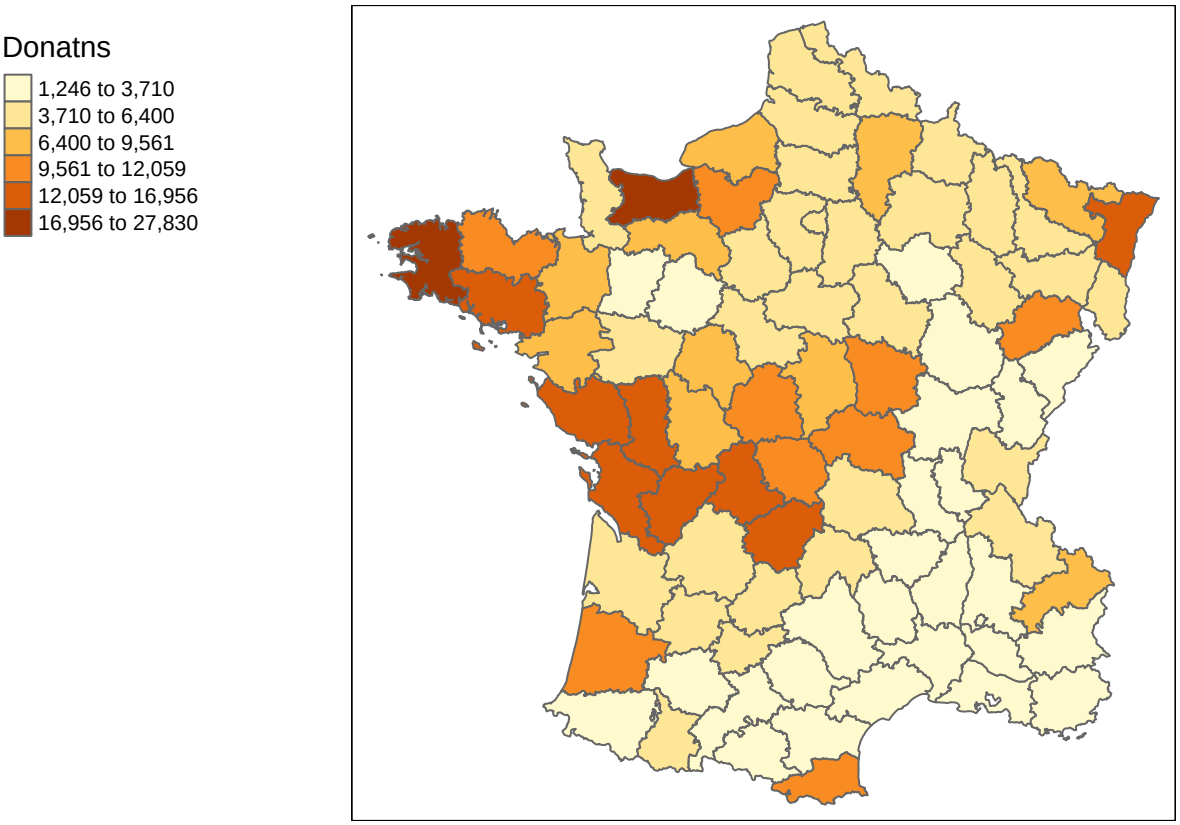
```
queen.weights <- nb2listw(queen.nb,style = "B", zero.policy = TRUE)
```

### Univariate analysis

Throughout the notebook, we will focus on the variable **Donatns**, which is charitable donations per capita. Before proceeding with the local spatial statistics and visualizations, we will take preliminary look at the spatial distribution of this variable. This is done with **tmap** functions. We will not go into too much detail on these because there is a lot to cover local spatial statistics and this functionality was covered in a previous notebook. Please the Basic Mapping notebook for more information on basic **tmap** functionality

For the univariate map, we use the natural breaks or jenks style to get a general sense of the spatial distribution for our variable.

```
tm_shape(guerry) +
  tm_fill("Donatns", style = "jenks", n = 6) +
  tm_borders() +
  tm_layout(legend.outside = TRUE, legend.outside.position = "left")
```




## Local Moran

### Principle

The local Moran statistic was suggested in Anselin(1995) as a way to identify local clusters and local spaital outliers. Most global spatial autocorrelation can be expressed as a double sum over i and j indices, such as $\Sigma_i \Sigma_j g_{ij}$. The local form of such a statistic would then be, for each observation(location)i, the sum of the relevant expression over the j index, $\Sigma_j g_{ij}$.

Specifically, the local Moran statistic takes the form $cz_i \Sigma_j w_{ij} z_j$, with z in deviations from the mean. The scalar c is the same for all locations and therefore does not play a role in the assessment of significance. The latter is obtained by means of a conditional permutation method, where, in turn, each $z_i$ is held fixed, and the remaining z-values are randomly permuted to yield a reference distribution for the statistic. This operates in the same fashion as for the global Moran's I, except that the permutation is carried out for each observation in turn. The result is a pseudo p-value for each location, which can then be used to assess significance. Note that this notion of significance is not the standard one, and should not be interpreted that way (see the discussion of multiple comparisons below).

Assessing significance in and of itself is not that useful for the Local Moran. However, when an indication of significance is combined with the location of each observation in the Moran Scatterplot, a very powerful interpretation becomes possible. The combined information allows for a classification of the significant locations as high-high and low-low spatial clusters, and high-low and low-high spatial outliers. It is important to keep in mind that the reference to high and low is relative to the mean of the variable, and should not be interpreted in an absolute sense.

### Implementation

Throughout many of the mapping function we make in this notebook, the weights from *spdep* need to be convert to a full size spatial weights matrix for statistical computations. Here we make a function `convert_matrix` to accomplish this. We use `as` to convert to type `symmetricMatrix`.

```
convert_matrix <- function(weights){
 W  <- as(weights, "symmetricMatrix")
 W  <- as.matrix(W/Matrix::rowSums(W))
 W[which(is.na(W))] <- 0
 W
}
W <- convert_matrix(queen.weights)
```

```
## Registered S3 methods overwritten by 'spatialreg':
##   method                    from
##   residuals.stsls           spdep
##   deviance.stsls            spdep
##   coef.stsls                spdep
##   print.stsls               spdep
##   summary.stsls             spdep
##   print.summary.stsls       spdep
##   residuals.gmsar           spdep
##   deviance.gmsar            spdep
##   coef.gmsar                spdep
##   fitted.gmsar              spdep
##   print.gmsar               spdep
##   summary.gmsar             spdep
##   print.summary.gmsar       spdep
##   print.lagmess             spdep
##   summary.lagmess           spdep
##   print.summary.lagmess     spdep
##   residuals.lagmess         spdep
##   deviance.lagmess          spdep
##   coef.lagmess              spdep
##   fitted.lagmess            spdep
##   logLik.lagmess            spdep
##   fitted.SFResult           spdep
##   print.SFResult            spdep
##   fitted.ME_res             spdep
##   print.ME_res              spdep
##   print.lagImpact           spdep
##   plot.lagImpact            spdep
##   summary.lagImpact         spdep
##   HPDinterval.lagImpact     spdep
##   print.summary.lagImpact   spdep
##   print.sarlm               spdep
##   summary.sarlm             spdep
##   residuals.sarlm           spdep
##   deviance.sarlm            spdep
##   coef.sarlm                spdep
##   vcov.sarlm                spdep
##   fitted.sarlm              spdep
##   logLik.sarlm              spdep
##   anova.sarlm               spdep
```

```
##   predict.sarlm             spdep
##   print.summary.sarlm       spdep
##   print.sarlm.pred          spdep
##   as.data.frame.sarlm.pred spdep
##   residuals.spautolm        spdep
##   deviance.spautolm         spdep
##   coef.spautolm             spdep
##   fitted.spautolm           spdep
##   print.spautolm            spdep
##   summary.spautolm          spdep
##   logLik.spautolm           spdep
##   print.summary.spautolm    spdep
##   print.WXImpact            spdep
##   summary.WXImpact          spdep
##   print.summary.WXImpact    spdep
##   predict.SLX               spdep
```

To compute the observed value of the local moran statistc, we make a function.

```
univariate_moran <- function(x, W){
        xs <- standardize(x)
        local  <- (xs*W%*%xs)
        local
}
```

```
lmoran <- univariate_moran(guerry$Donatns,W)
```

Throughout this notebook, we will be taking a conditional random approach to assess significance with each local spatial statistic as outlined in (Anselin 1995). The basic approach here is to compute a reference distribution for each location. This is done by holding the value constant at each location then taking a random samples from the rest of the obseravtions for the neighbors. With spatially random draws for the neighbors, we then calculate the statistic for the permutation. The runtimes for these computations are not ideal in R, especially when using more than 10,000 permuations. When we absolutely need more permuations, there are work arounds, but they are not ideal.

Before computing the spatially random permutations for each location, we set the seed to 123456789, which is the random seed used in GeoDa.

```
set.seed(123456789)
```

```
univariate_moran_sim <- function(x, W, permutations){

        n   <- nrow(W)
        id  <- 1:n

        #place to store results
        local.sims  <- matrix(NA, nrow = n, ncol=permutations)
        x.sample = matrix(NA, nrow = n, ncol = permutations)

        # filling each column of the sample matrix
        for(i in 1:n){
          sample.indices <- sample(id[-i], permutations, replace = TRUE)
          x.sample[i,] <- x[sample.indices]
        }

        #standardizing the y sample values
        x.sample <- (x.sample - apply(x.sample, 1, mean))/apply(x.sample, 1, sd)
```

```
        #standardizing x
        xs <- standardize(x)
        W[which(is.na(W))] <- 0

        #calculating the local statistics
        local.sims  <- (xs*W%*%x.sample)
        local.sims
}
```

```
moran_ref <- univariate_moran_sim(guerry$Donatns,W,999)
```

With the reference distributions and observed statistics for each location, we can compute a pseudo pvalue for each location. For this, we just loop through each location and calculate the number of permuations that are greater than the observed statistic and store them in a vector. The pvalue is then calculated as a fraction of **1 + number greater** divided by the number of permuations plus one. Since the significance is two-sided, we will need to account for p-values close to one. For this, we use `ifelse` to assign (1- value) if the value is greater than .5. This allows us to account for both ends of the conditional distribution.

```
get_p_value <- function(mat, observed,type = "one-sided") {
  nperm <- ncol(mat)
  nlocs <- nrow(mat)
  p_value <- rep(NA,nlocs)
  for(i in 1:nlocs){
    num_greater <- length(which(mat[i,] >= observed[i]))
    p_value[i] <- (num_greater + 1) / (nperm + 1)
  }
  if (type == "two-sided"){
    p_value <- ifelse(p_value > .5, 1-p_value, p_value)
    p_value
  } else {
    p_value
  }
}
```

```
p_value <- get_p_value(moran_ref, lmoran,type="two-sided")
p_value
```

```
##  [1] 0.021 0.285 0.340 0.038 0.062 0.001 0.398 0.390 0.077 0.133 0.013 0.002
## [13] 0.204 0.355 0.024 0.047 0.119 0.325 0.369 0.010 0.027 0.054 0.363 0.052
## [25] 0.079 0.482 0.078 0.001 0.038 0.273 0.198 0.016 0.313 0.101 0.458 0.046
## [37] 0.278 0.095 0.431 0.127 0.043 0.084 0.452 0.397 0.370 0.018 0.136 0.038
## [49] 0.288 0.200 0.428 0.263 0.139 0.024 0.200 0.399 0.464 0.384 0.111 0.440
## [61] 0.243 0.384 0.020 0.081 0.410 0.193 0.132 0.018 0.231 0.301 0.358 0.338
## [73] 0.137 0.399 0.053 0.458 0.002 0.026 0.002 0.001 0.076 0.037 0.035 0.313
## [85] 0.284
```

Here we add coumn in the sf data frame, so we can use **tmap** functions to make significance and cluster maps.

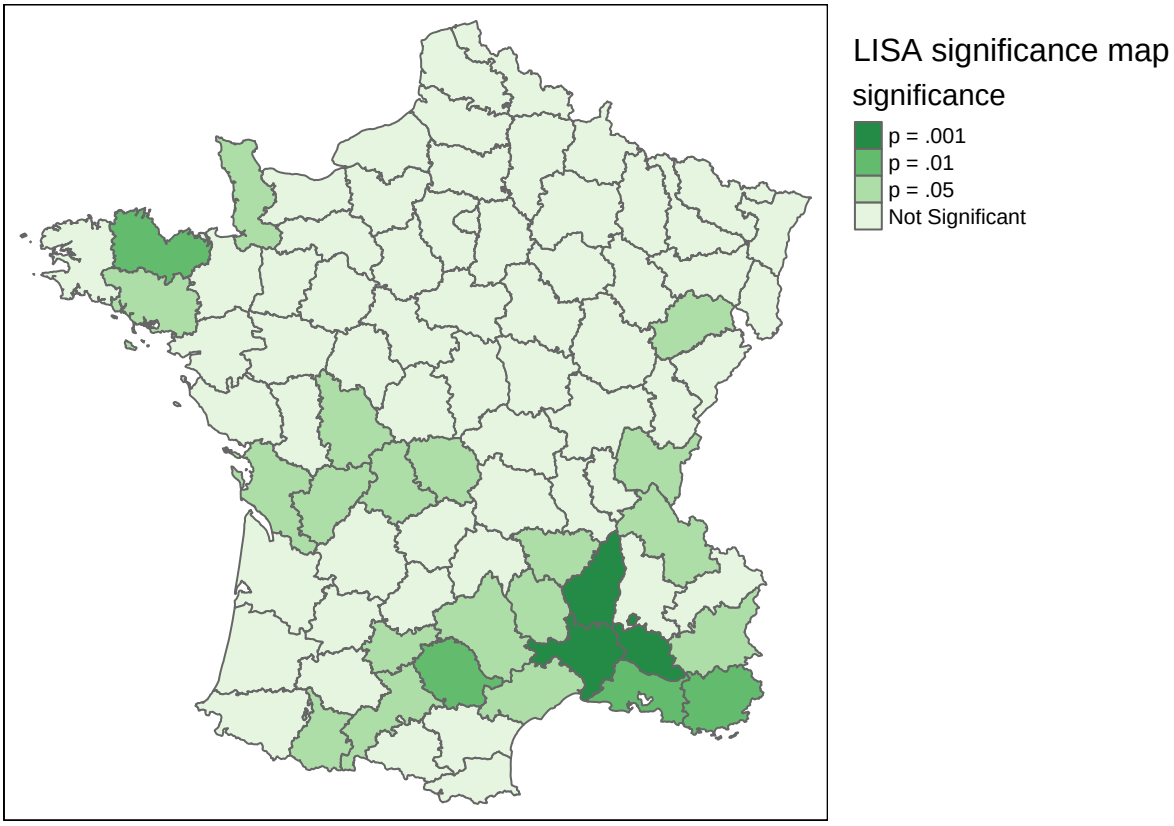
With the p-values, we can now move on to mapping. For the LISA significance map and the LISA cluster map, we will need to know which areas have significant p-values. We will need to assign categorical labels to indicate this for mapping purposes. There are many ways to assign these labels for instance it can be done with `ifelse` statements, but it is very messy. I found that using `cut` is the most concise way to accomplish this task. In `cut`, we specify breaks that correspond with the desired level of significance. In our case the minimum level of significance is .05. If we wanted the minimum classification of significance to be .01, the last interval in the breaks paramter would be .01 to 1. Additionaly, if we want to show higher levels of significance

we would include the values in the breaks parameter. For example to add .0001, we would include it in between 0 and .001. When choosing the breaks, make sure the labels corrrespond to the correct level of significance.

```
guerry$significance <- cut(p_value,
                           breaks = c(0, .001, .01, .05, 1),
                           labels = c("p = .001", "p = .01", "p = .05", "Not Significant"))
```

With the significance variable, we can make a preliminary LISA significance map with **tmap**. This tutorial focuses on local spatial autocorrelation, so we will not go into too much depth on **tmap**, but for more indepth coverage please see the Basic Mapping Notebook or the **tmap** documentation.

```
tm_shape(guerry) +
  tm_fill("significance", palette = "-Greens") +
  tm_borders() +
  tm_layout(legend.outside = TRUE, title = "LISA significance map")
```



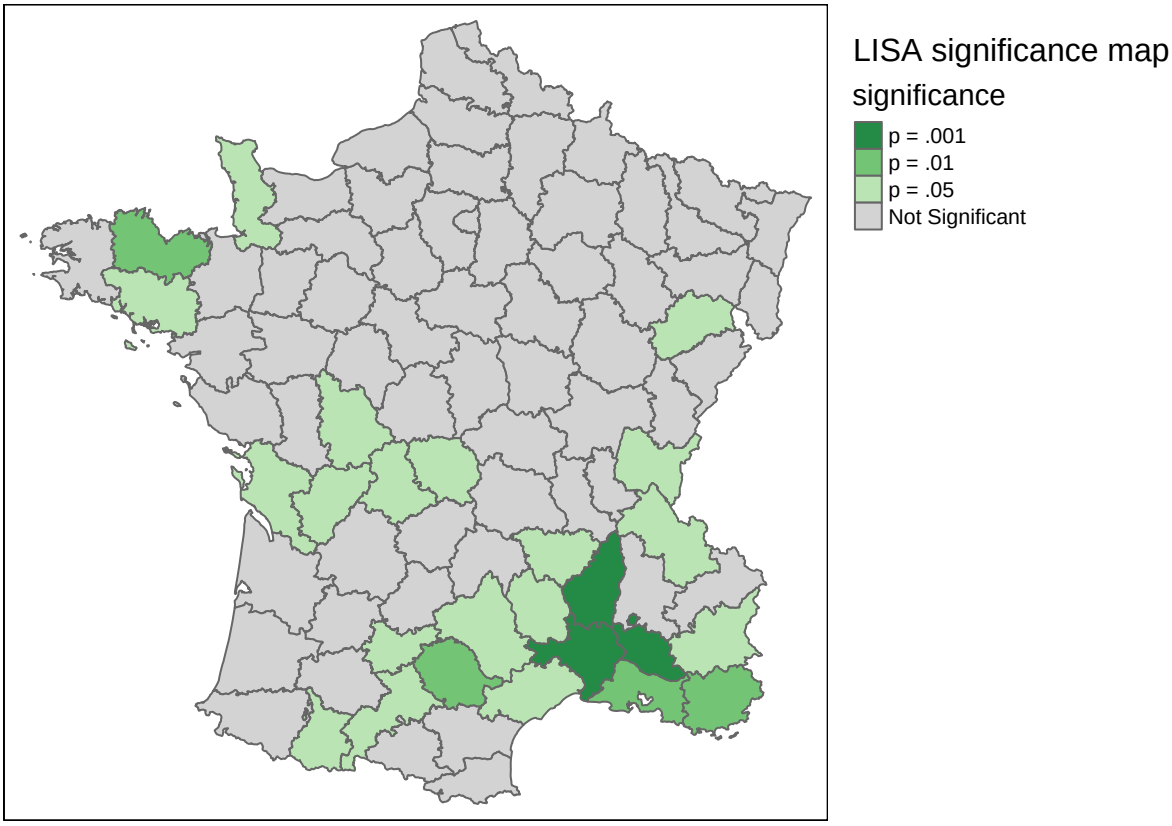

To get closer to the GeoDa mapping style, we can use **RColorBrewer** to create a specialized palette for the significance map. For this we will need to use `brewer.pal` to generate a palette of Greens. We will need to reverse the order of this palette to recreate the the plot, which is done with the base R function `rev`. At the end of the palette, we replace the last green shade with the hexadecimal number for grey.

```
pal <- rev(brewer.pal(4, "Greens"))
pal[4] <- "#D3D3D3"
```

With the new palette, we can create the map. The difference here is that we use our palette `pal` instead of `"-Greens"`

```
tm_shape(guerry) +
  tm_fill("significance", palette = pal) +
```

```
  tm_borders() +
  tm_layout(legend.outside = TRUE, title = "LISA significance map")
```




Each spatial statistic in this tutorial comes with a significance mapping component. In order to avoid repetitive code, we will make a significance map function, using the process outlined above. It is a bit tricky in that we must get the proper breaks and labels for the map from the p-value data.

```
significance_map <- function(polys, pvalues, permutations, alpha){
  # function to create significance map
  # arguments:
  #    polys: sf dataframe
  #    pvalue_vector: a vector of p-values
  #    permutations: the number of permuations used to calculate the pvalues
  #    min.sig: the alpha level required for significance
  # returns:
  #    a significance map in GeoDa style

  target_p <- 1 / (1 + permutations)
  potential_brks <- c(.00001, .0001, .001, .01)
  brks <- potential_brks[which(potential_brks > target_p & potential_brks < alpha)]
  brks2 <- c(target_p, brks, alpha)
  labels <- c(as.character(brks2), "Not Significant")
  brks3 <- c(-.000001, brks2, 1)

  cuts <- cut(pvalues, breaks = brks3,labels = labels)
  polys <- polys %>% mutate(sig = cuts)
```

```
  pal <- rev(brewer.pal(length(labels), "Greens"))
  pal[length(pal)] <- "#D3D3D3"

  tm_shape(polys) +
    tm_fill("sig", palette = pal) +
    tm_borders() +
    tm_layout(title = "Significance Map")
}
```

To build the LISA cluster map, we will need to know which quadrant of the global Moran's I scatterplot each location is in. To do this, we will create the stardardized version of the the variable and the lag of this variable. High-high will correspond to positive values for both the original variable and the lag. High-low will be positive for the original variable and negative for the lag variable. Vice versa for low-high. We use `standardize` to standardize our variable.

```
xs <- standardize(guerry$Donatns)
```

There are many ways to assign the correct LISA patterns, but some are more concise and intuitive than others. We use interacction to assign TRUE and FALSE boolean values based on two condtionals. One for the standardized variable being greater than 0 and the other for the lagged standardized variable or $Wx$ being greater than zero. We use `as.character` to convert the booleans to characters. Next, we use `str_replace_all` to replace "TRUE" with "High" and "FASLE" with "Low". With this, we have the correct LISA cluster designations assigned to each location. The last step is assign "Not Significant" to the locations with p_values of greater than .05, which we computed earlier.

```
 patterns <- as.character( interaction(xs > 0, W%*%xs > 0) )
  patterns <- patterns %>%
        str_replace_all("TRUE","High") %>%
        str_replace_all("FALSE","Low")
  patterns[which(p_value > .05)] <- "Not Significant"
guerry <- guerry %>% mutate(patterns = patterns)
```

Before building the plot, we will need to know which classifications are significant. We will build a palette that contains colors for the significant classifications. All of the classifications will not always be significant in eahc plot. We check this with `unique` on the **patterns** variable.

```
unique(guerry$patterns)
```

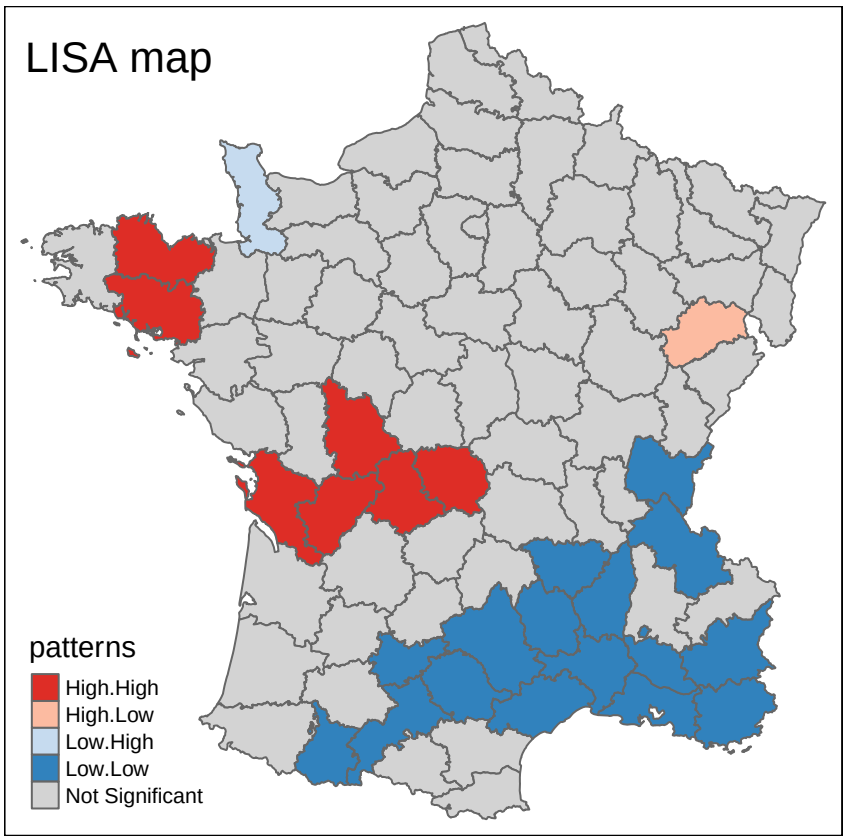
```
## [1] "Low.Low"         "Not Significant" "High.High"       "Low.High"
## [5] "High.Low"
```

Since only two of the classifications are significant, we will need to build a palette with three colors, red, blue, and grey. The simplest way to do this is just by building a vector of color strings. We could also use `brewer.pal` again to build this, but that is complicated than necessary.

```
pal <- c("#DE2D26","#FCBBA1","#C6DBEF", "#3182BD", "#D3D3D3")
```

Using the palette from above, we can create our LISA cluster map with the color scheme used in GeoDa.

```
tm_shape(guerry) +
  tm_fill("patterns", palette = pal) +
  tm_borders() +
  tm_layout("LISA map")
```

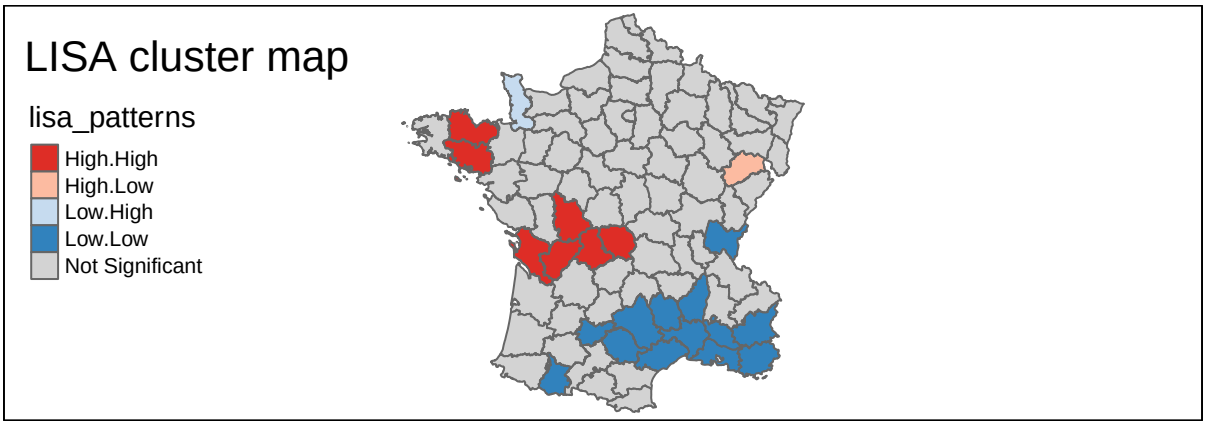


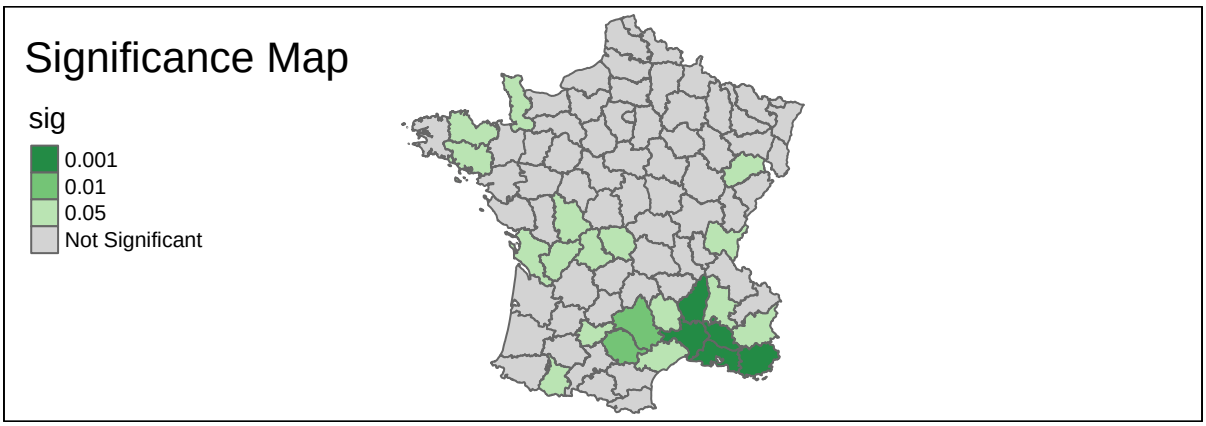

**Putting it all together**

```
univariate_lisa_map <- function(area,x, weights, permutations, alpha){
  # function to create a bivariate LISA map
  # arguments:
  #    area: sf dataframe
  #    x: vectors of x values
  #    y: a vector of y values
  #    nperms: the number of permuations used to calculate the pvalues
  #    alpha: the alpha level required for significance
  # returns:
  #    a LISA map in GeoDa style


  # Converting the weights
  W <- convert_matrix(weights)

  # Computing the observed statistics and reference distributions
  observed <- univariate_moran(x,W)
  sims <- univariate_moran_sim(x,W, permutations = permutations)


  #computing p-value from observed statistics and reference distribution
  p_value <- get_p_value(sims,observed, type = "two-sided")

  # Standardizing the x and y vars
```

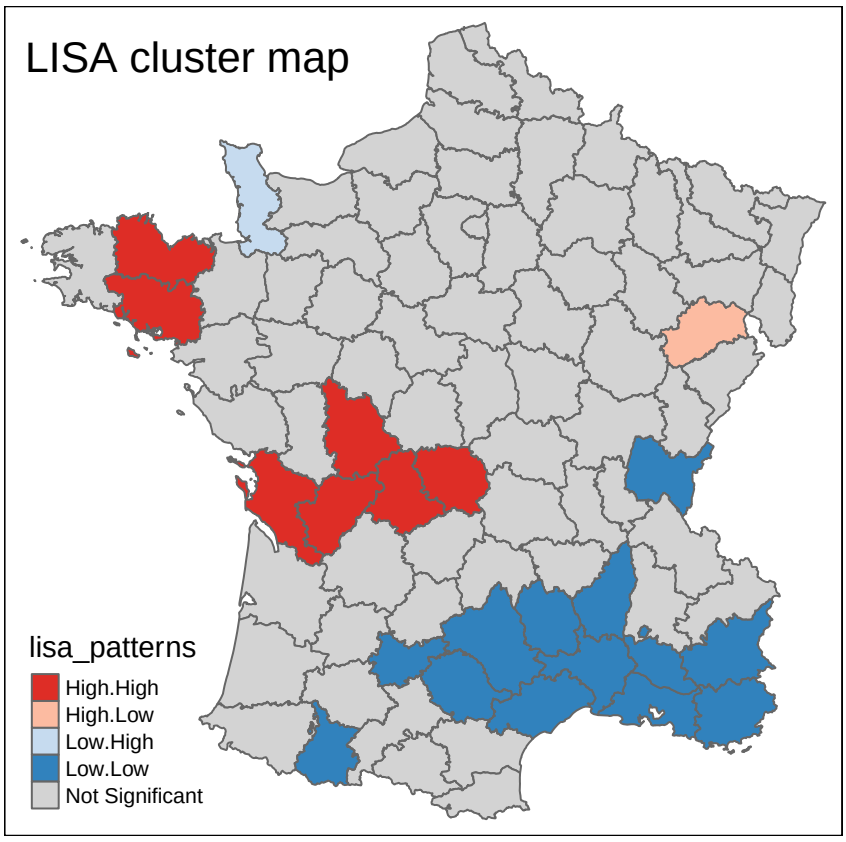
```
  xs <- standardize(x)

  #Assigning classifications
  lisa_patterns <- as.character( interaction(xs > 0, W%*%xs > 0) )
  lisa_patterns <- patterns %>%
        str_replace_all("TRUE","High") %>%
        str_replace_all("FALSE","Low")
  lisa_patterns[which(p_value > alpha)] <- "Not Significant"

  # Adding the classifications to the data frame
  area <- area %>% mutate(lisa_patterns)

  # Constructing the correct palette based on presense of classifications
  classes <- unique(lisa_patterns)
  patts <- c("High.High", "High.Low", "Low.High", "Low.Low", "Not Significant")
  colors <- c("#DE2D26","#FCBBA1","#C6DBEF", "#3182BD", "#D3D3D3")
  mat <- matrix(c(patts,colors), ncol = 2)
  logi <- patts %in% classes
  pre_col <- matrix(mat[logi], ncol = 2)
  pal <- pre_col[,2]

  # Making the LISA map
  p1 <- tm_shape(area) +
  tm_fill("lisa_patterns", palette = pal) +
  tm_borders() +
  tm_layout(title = "LISA cluster map")

  p2 <- significance_map(area, p_value, permutations = permutations,alpha = alpha)

  tmap_arrange(p1,p2, ncol = 1)
}
```

```
p <- univariate_lisa_map(guerry,guerry$Donatns,queen.weights,999,.05)
p
```

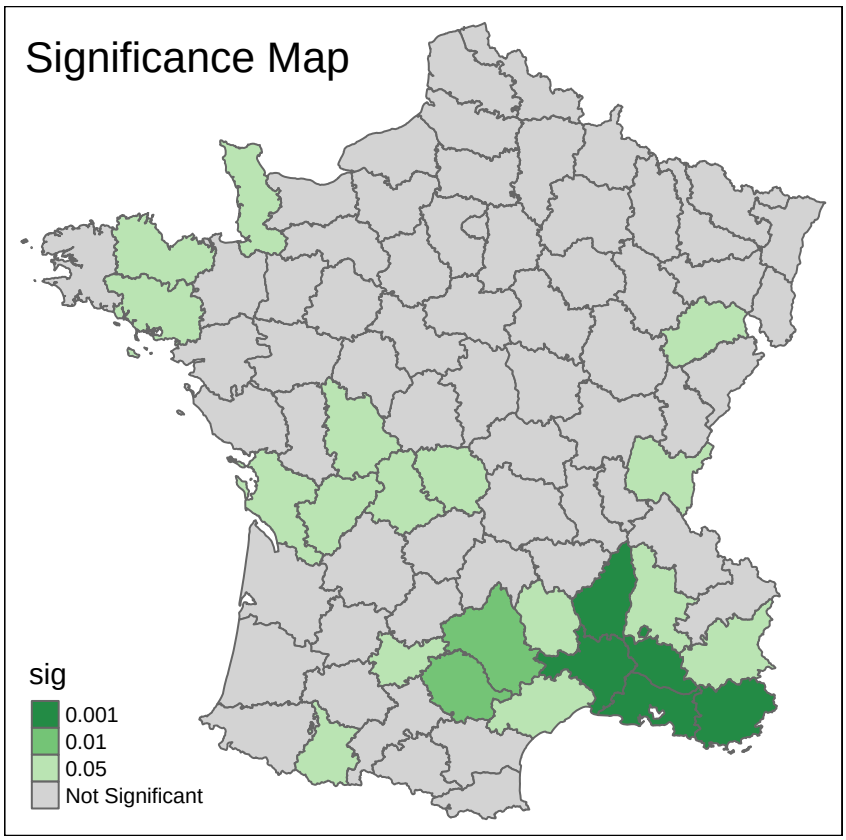

LISA cluster map
lisa_patterns
High.High
High.Low
Low.High
Low.Low
Not Significant


Significance Map
sig
0.001
0.01
0.05
Not Significant

```
p[[1]]
```


LISA cluster map
lisa_patterns
High.High
High.Low
Low.High
Low.Low
Not Significant

```
p[[2]]
```

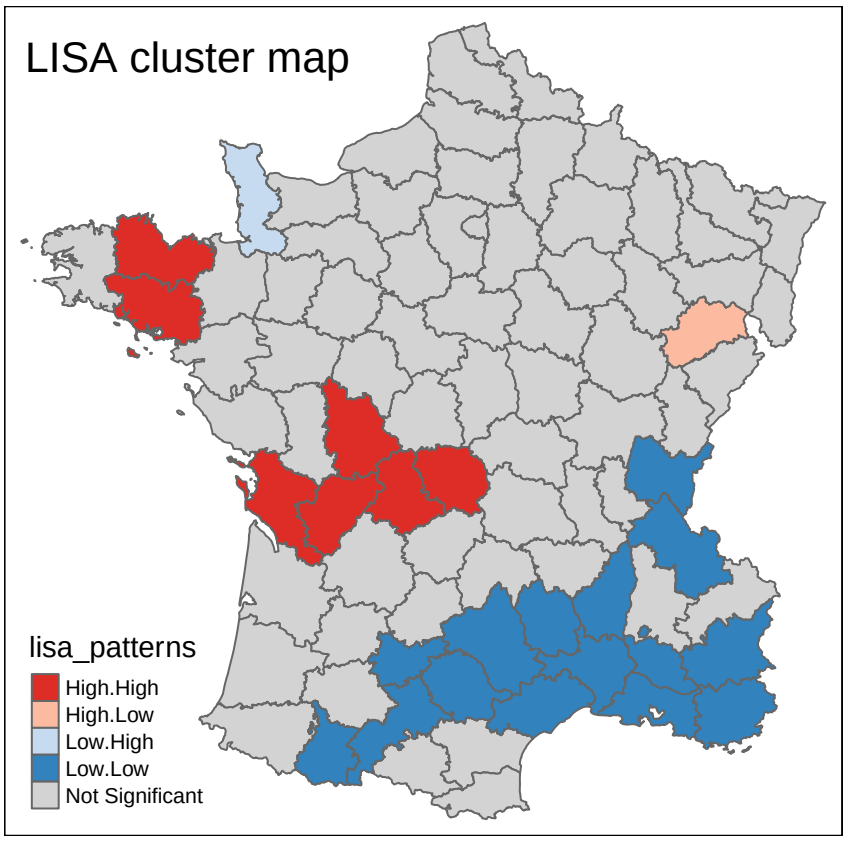



### Randomization Options

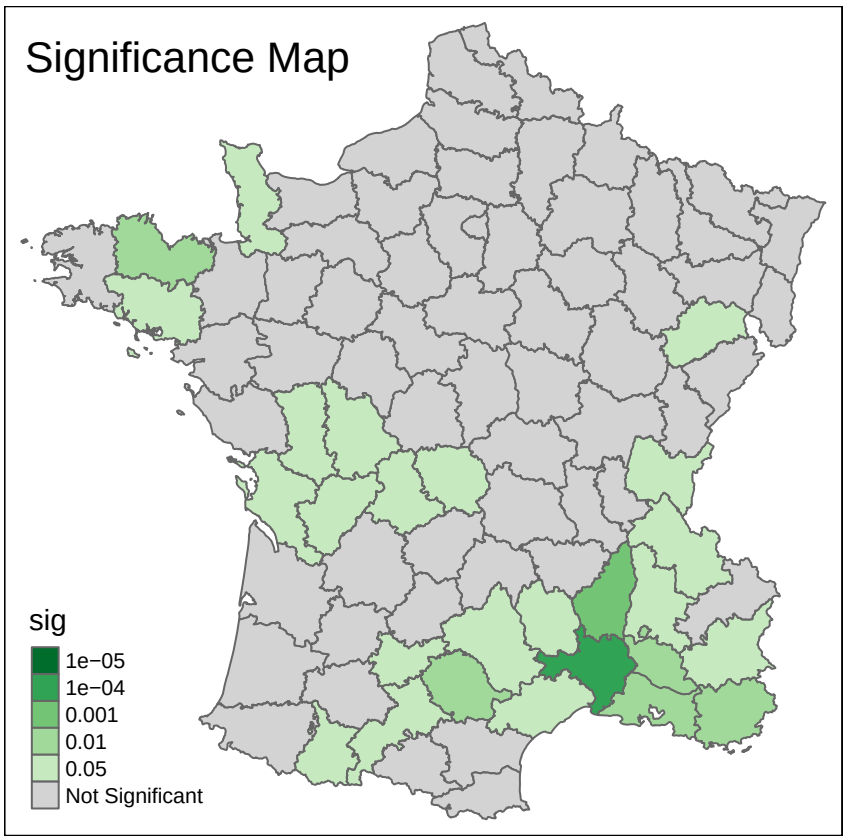
To obtain higher significance levels, we need to use more permutations in the calculation of the the local moran for each location. For instance, a pseudo pvalue of .00001 would require 999999 permutations.

To choose a set level of significance in the LISA plots, we do this when setting the breaks for LISA maps.

```
p <- univariate_lisa_map(guerry,guerry$Donatns,queen.weights,99999,.05)
p[[1]]
```


LISA cluster map
lisa_patterns
High.High
High.Low
Low.High
Low.Low
Not Significant

```
p[[2]]
```



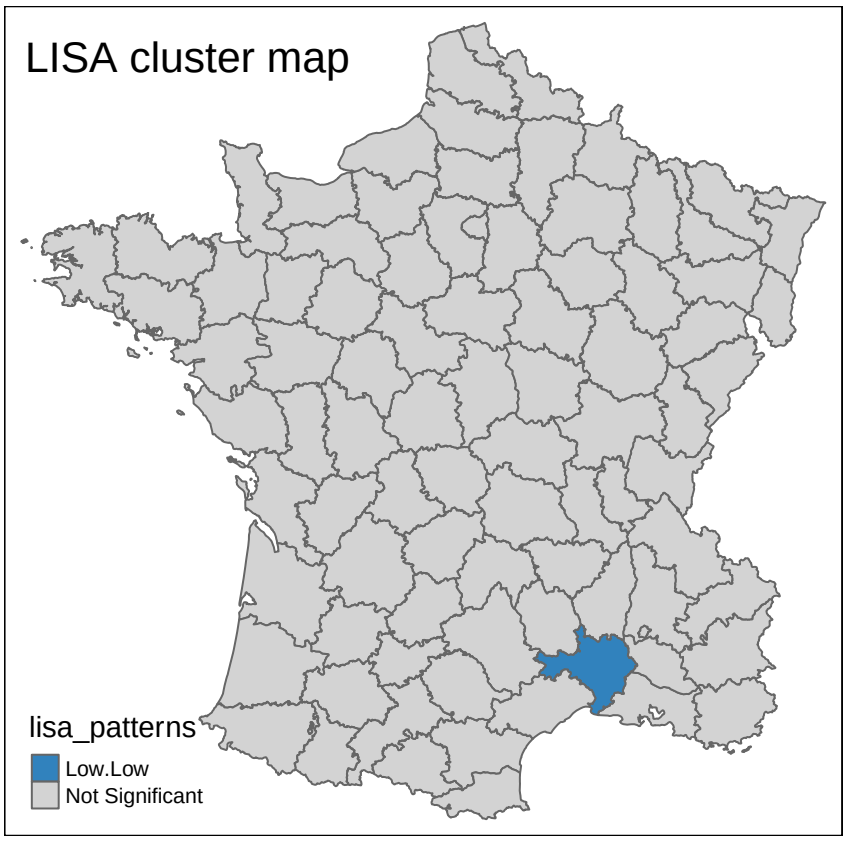

### Significance

An important methodological issue associated with the local spatial autocorrelation statistics is the selection of the p-value cut-off to properly reflect the desired Type I error. Not only are the pseudo p-values not analytical, since they are the result of a computational permutation process, but they also suffer from the problem of multiple comparisons (for a detailed discussion, see de Castro and Singer 2006). The bottom line is that a traditional choice of 0.05 is likely to lead to many false positives, i.e., rejections of the null when in fact it holds.

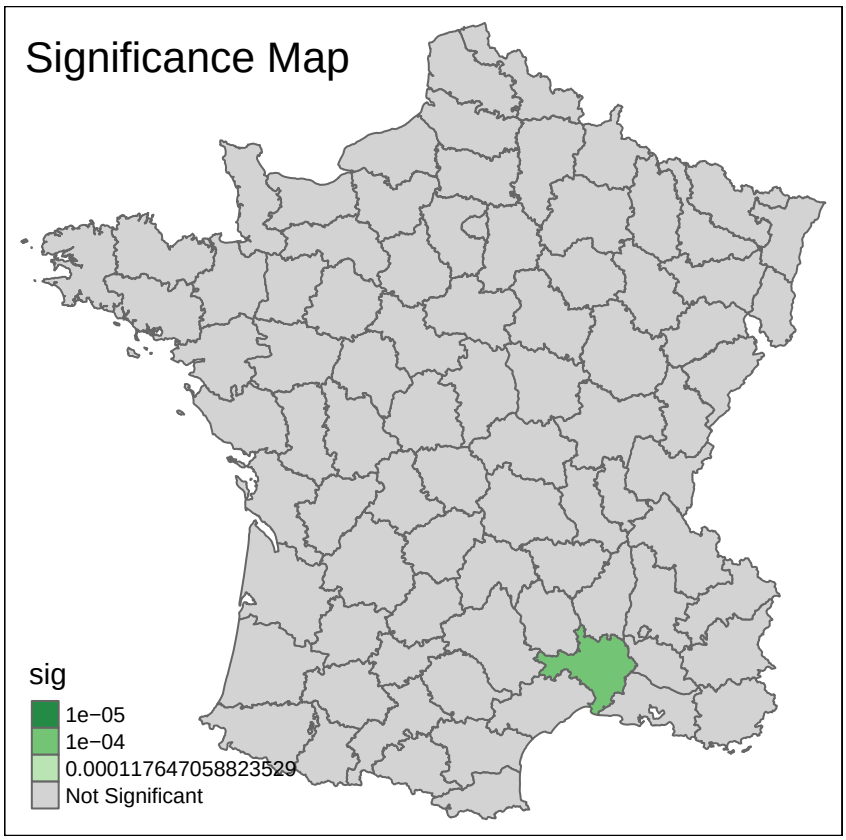
### Bonferroni bound

The Bonferroni bound constructs a bound on the overall p-value by taking $\alpha$ and dividing it by the number of comparisons. In our context, the latter corresponds to the number of observation, n. As a result, the Bonferroni bound would be $\alpha/n = .00012$, the cutoff p-value to be used to determine significance.

```
bonferroni <- .01 / 85
p <- univariate_lisa_map(guerry,guerry$Donatns,queen.weights,99999,bonferroni)
p[[1]]
```




```
p[[2]]
```




### Interpretation of significance

As mentioned, there is no fully satisfactory solution to deal with the multiple comparison problem. Therefore, it is recommended to carry out a sensitivity analysis and to identify the stage where the results become interesting. A mechanical use of 0.05 as a cut off value is definitely not the proper way to proceed.

Also, for the Bonferroni and FDR procedures to work properly, it is necessary to have a large number of permutations, to ensure that the minimum p-value can be less than $\alpha/n$. Currently, the largest number of permutations that GeoDa can support is 99999. In R, we can do more, however the runtime is not ideal. For the bonferroni criterion to yield significant values, we must implement a minimum number of permutations, otherwise we cannot assess significance. This is not due to a characteristic of the data, but to the lack of sufficient permutations to yield a pseudo p-value that is small enough.

### Interpretation of clusters

Strictly speaking, the locations shown as significant on the significance and cluster maps are not the actual clusters, but the cores of a cluster. In contrast, in the case of spatial outliers, they are the actual locations of interest.

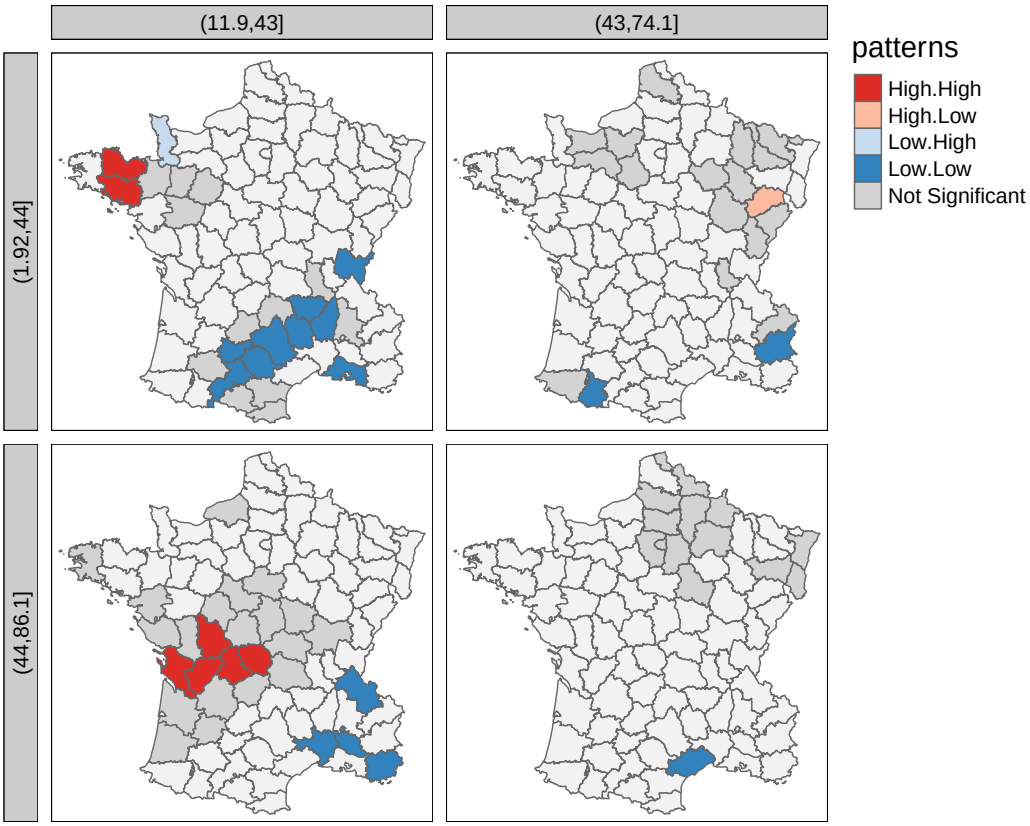
### Conditional local cluster maps

To make the conditional map, we first need to make two categorical variables, with two categories. `cut` breaks the data up into two equal pieces. With the two categorical variables, we can create facets with **tmap**.

```
guerry$cut.literacy <- cut(guerry$Litercy, breaks = 2)
guerry$cut.clergy <- cut(guerry$Clergy, breaks = 2)
```

To make the conditional maps, we use the same **tmap** function and palette as the LISA cluster map. The only addition is `tm_facets`, which will use the two categorical variables created above.

```
tm_shape(guerry) +
  tm_fill("patterns", palette = pal) +
  tm_borders() +
  tm_facets(by = c("cut.clergy", "cut.literacy"),free.coords = FALSE,drop.units=FALSE)
```




## Local Geary

### Principle

The Local Geary statistic, first outlined in Anselin (1995), and further elaborated upon in Anselin (2018), is a Local Indicator of Spatial Association (LISA) that uses a different measure of attribute similarity. As in its global counterpart, the focus is on squared differences, or, rather, dissimilarity. In other words, small values of the statistics suggest positive spatial autocorrelation, whereas large values suggest negative spatial autocorrelation.

Formally, the Local Geary statistic is

$$LG_i = \Sigma_j w_{ij}(x_i - x_j)^2$$

in the usual notation.

Inference is again based on a conditional permutation procedure and is interpreted in the same way as for the Local Moran statistic. However, the interpretation of significant locations in terms of the type of association is not as straightforward. In essence, this is because the attribute similarity is not a cross-product and thus

has no direct correspondence with the slope in a scatter plot. Nevertheless, we can use the linking capability within GeoDa to make an incomplete classification.

Those locations identified as significant and with the Local Geary statistic smaller than its mean, suggest positive spatial autocorrelation (small differences imply similarity). For those observations that can be classified in the upper-right or lower-left quadrants of a matching Moran scatter plot, we can identify the association as high-high or low-low. However, given that the squared difference can cross the mean, there may be observations for which such a classification is not possible. We will refer to those as other positive spatial autocorrelation.

For negative spatial autocorrelation (large values imply dissimilarity), it is not possible to assess whether the association is between high-low or low-high outliers, since the squaring of the differences removes the sign.

### Implementation

The local geary is not implemented in **spdep**, so we will have to compute the local statistics with base functionality and the `lag.listw` function from **spdep**. Before we begin, we will need to break up the formula for local Geary into managable pieces.

$$LG_i = \Sigma_j w_{ij}(x_i - x_j)^2$$

The above formula can be simplified into this:

$$LG_i = x_i^2 - 2x_i\Sigma_j w_{ij}x_j + \Sigma_j w_{ij}x_j^2$$

With this simplification, we have three managable parts that can be easily calculated with the function available to us. Notice the middle terms is 2 times the $x_i$ times the lag variable of x. The third term is the lag variable of x squared.

```
local_geary <- function(x,W){
  x2 <- x^2
  lg_x <- x2 - 2*x*(W%*%x) + W%*%x2
  lg_x
}
lg <- local_geary(guerry$Donatns,W)
```

As with the local moran, we will have to compute reference distributions for each location to compare the observed statistic with. We will follow a similar process with a few differences.

To store the local moran statistic for each location and permutation, we will use a data frame, as with the local moran.

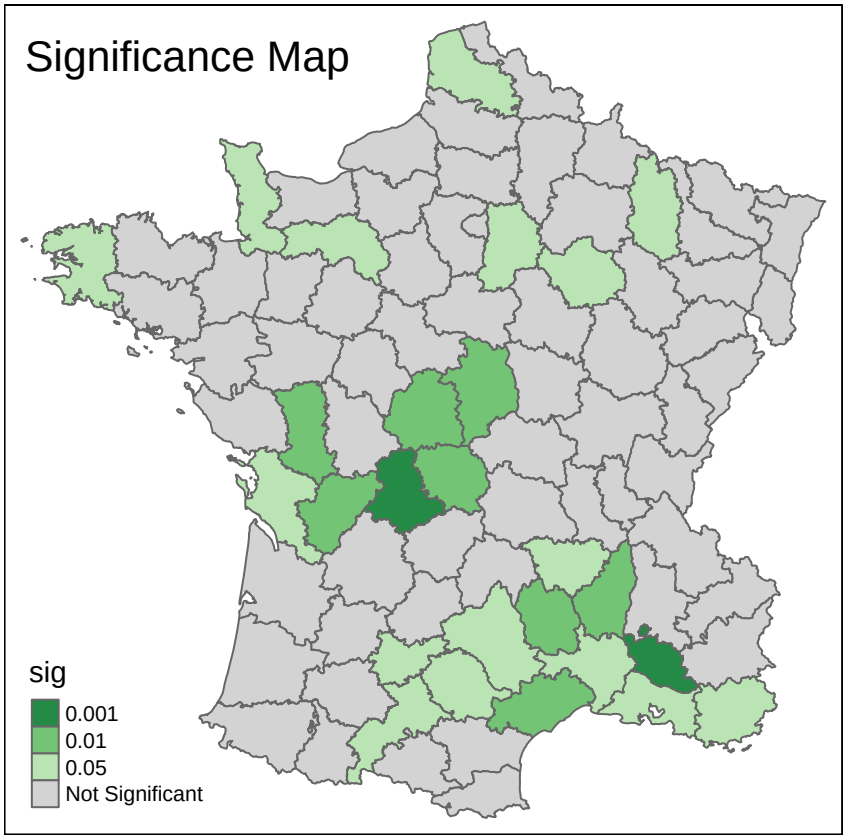
```
local_geary_sims <- function(x,W,permutations){
  n <- nrow(W)
  id  <- 1:n

  local.sims  <- matrix(NA, nrow = n, ncol=permutations)
  x.sample = matrix(NA, nrow = n, ncol = permutations)

  for(i in 1:n){
    sample.indices <- sample(id[-i], permutations, replace = TRUE)
    x.sample[i,] <- x[sample.indices]
  }
  x2 <- x^2
  x2.sample <- x.sample ^ 2
```

```
  local.sims <- (x2 - 2*x*W%*%x.sample + W%*%x2.sample)
  local.sims
}
```

```
sims <- local_geary_sims(guerry$Donatns,W,999)
```

```
p_value <- get_p_value(sims,lg,type = "two-sided")
```

```
significance_map(guerry,p_value,999,.05)
```




```
mean_lg <- rep(NA, 85)
for (i in 1:85){
  mean_lg[i] <- mean(sims[i,])
}
```

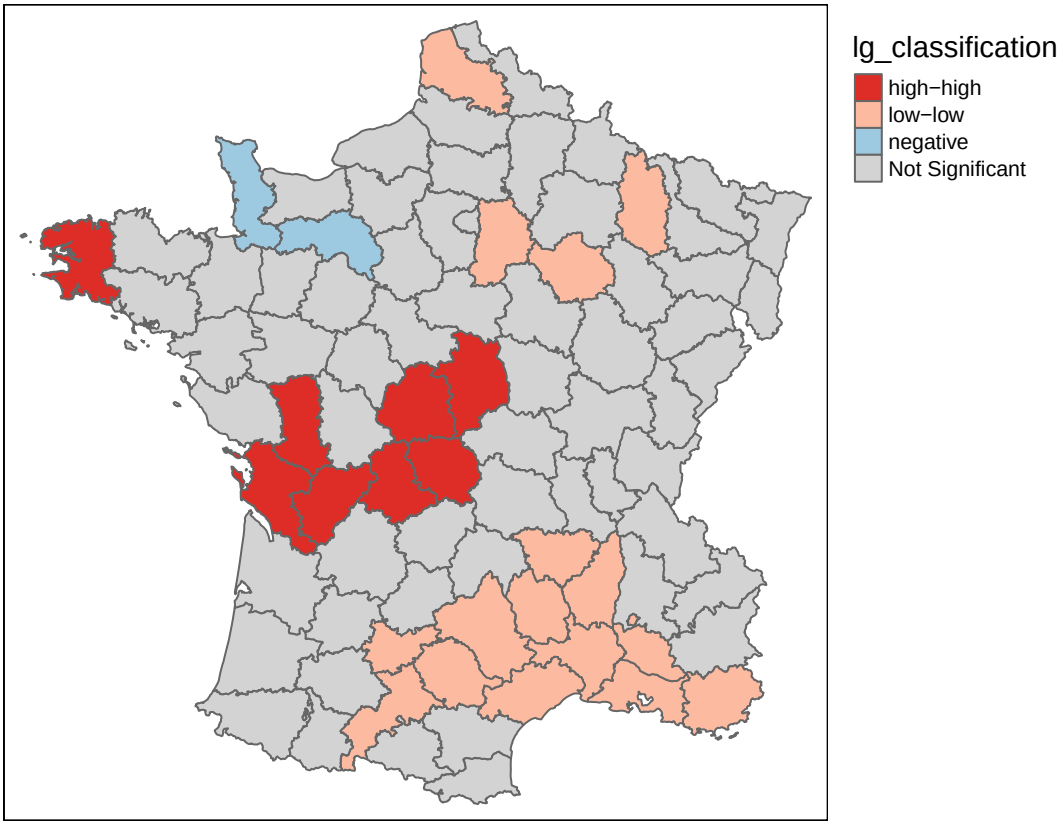
```
guerry[xs >= 0 & W%*%xs >= 0, "lg_classification"] <- "high-high"
guerry[xs <= 0 & W%*%xs <= 0, "lg_classification"] <- "low-low"
guerry[lg > mean_lg, "lg_classification"] <- "negative"
guerry[lg < mean_lg & xs*W%*%xs < 0,"lg_classification"] <- "other-positive"
guerry[p_value >.05, "lg_classification"] <- "Not Significant"
```

```
pal <- c("#DE2D26","#FCBBA1","#9ECAE1", "#D3D3D3", "#FEE5D9")
brewer.pal(6,"Reds")
```

```
## [1] "#FEE5D9" "#FCBBA1" "#FC9272" "#FB6A4A" "#DE2D26" "#A50F15"
```

```
tm_shape(guerry) +
  tm_fill("lg_classification", palette = pal) +
  tm_borders() +
```

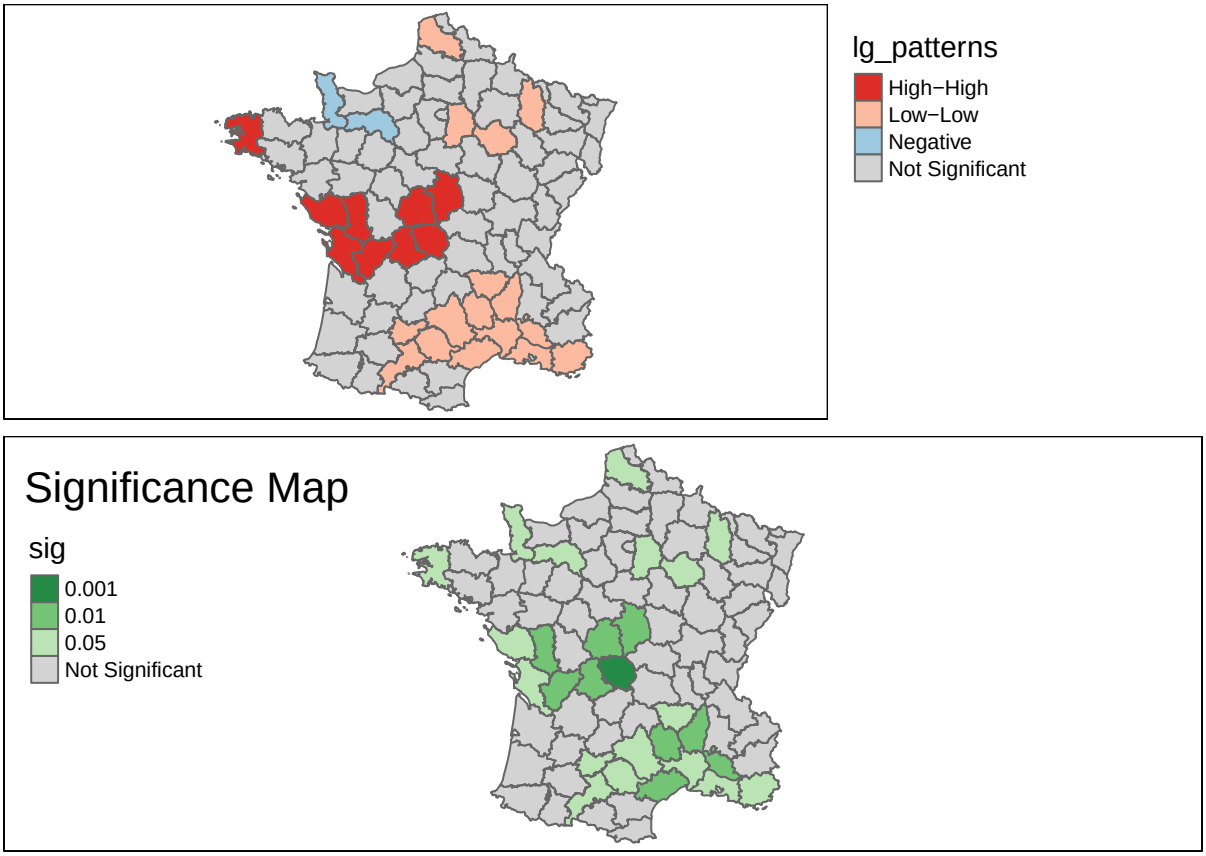
```
  tm_layout(legend.outside = TRUE)
```




```
local_geary_map <- function(polys,x,weights,permutations,alpha){

  W <- convert_matrix(weights)

  n <- nrow(W)
  lg <- local_geary(x,W)
  sims <- local_geary_sims(x,W,permutations)

  p_value <- get_p_value(sims,lg,type = "two-sided")
  mean_lg <- rep(NA, n)
  for (i in 1:n){
    mean_lg[i] <- mean(sims[i,])
  }


  xs <- standardize(x)
  lg_patterns <- rep(NA,n)

  lg_patterns[xs >= 0 & W%*%xs >=0] <- "High-High"
  lg_patterns[xs <= 0 & W%*%xs <=0] <- "Low-Low"
  lg_patterns[lg > mean_lg] <- "Negative"
  lg_patterns[lg < mean_lg & xs*W%*%xs < 0] <- "Other-Positive"
  lg_patterns[p_value > alpha] <- "Not Significant"

  polys <- polys %>% mutate(lg_patterns)
```

```
  classes <- unique(lg_patterns)
  patts <- c("High-High", "Low-Low", "Negative", "Other-Positive", "Not Significant")
  colors <- c("#DE2D26","#FCBBA1","#9ECAE1", "#FEE5D9","#D3D3D3")
  mat <- matrix(c(patts,colors), ncol = 2)
  logi <- patts %in% classes
  pre_col <- matrix(mat[logi], ncol = 2)
  pal <- pre_col[,2]

  p1 <- tm_shape(polys) +
  tm_fill("lg_patterns", palette = pal) +
  tm_borders() +
  tm_layout(legend.outside = TRUE)

  p2 <- significance_map(polys,p_value,permutations,alpha)
  tmap_arrange(p1,p2,ncol = 1)
}
```

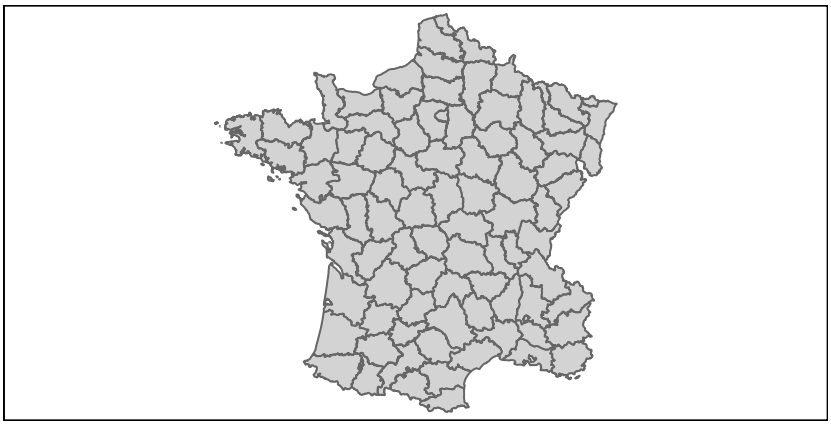
```
local_geary_map(guerry,guerry$Donatns,queen.weights,999,.05)
```

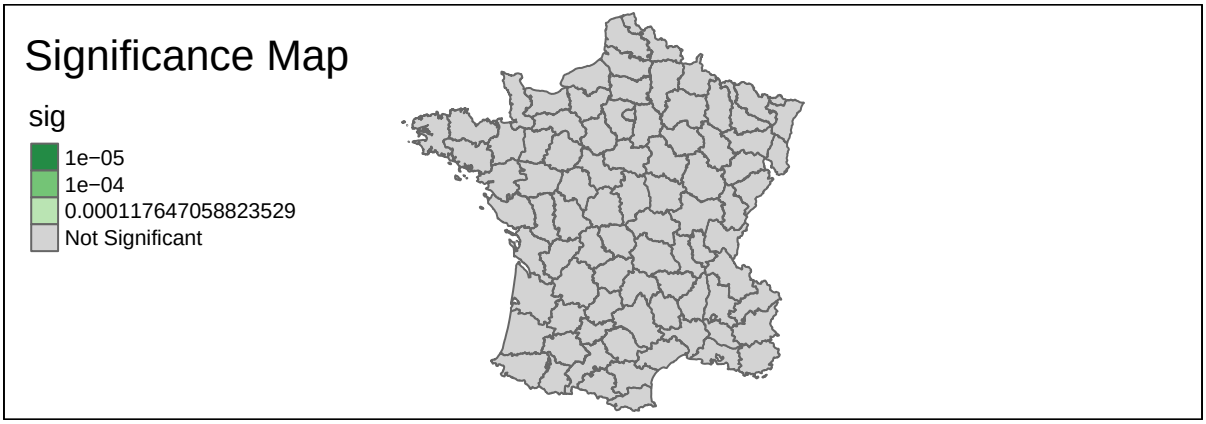



### Interpretation and significance

### Changing the significance threshold

```
local_geary_map(guerry,guerry$Donatns,queen.weights,99999,bonferroni)
```




## Getis-Ord Statistics

### Principle

A third class of statistics for local spatial autocorrelation was suggested by Getis and Ord (1992), and further elaborated upon in Ord and Getis (1995). It is derived from a point pattern analysis logic. In its earliest formulation the statistic consisted of a ratio of the number of observations within a given range of a point to the total count of points. In a more general form, the statistic is applied to the values at neighboring locations (as defined by the spatial weights). There are two versions of the statistic. They differ in that one takes the value at the given location into account, and the other does not.

The $G_i$ statistic consists of a ratio of the weighted average of the values in the neighboring locations, to the sum of all values, not including the value at the location $x_i$

$$G_i = \frac{\Sigma_{j \neq i} w_{ij} x_j}{\Sigma_{j \neq i} x_j}$$

In contrast, the $G_i^*$ statistic includes the value $x_i$ in numerator and denominator:

$$G_i^* = \frac{\Sigma_j w_{ij} x_j}{\Sigma_j x_j}$$

Note that in this case, the denominator is constant across all observations and simply consists of the total sum of all values in the data set.

The interpretation of the Getis-Ord statistics is very straightforward: a value larger than the mean (or, a positive value for a standardized z-value) suggests a high-high cluster or hot spot, a value smaller than the mean (or, negative for a z-value) indicates a low-low cluster or cold spot. In contrast to the Local Moran and Local Geary statistics, the Getis-Ord approach does not consider spatial outliers.

Inference is based on conditional permutation, using an identical procedure as for the other statistics.

### Implementation

```
local_g <- function(x,W){
  lag <- W%*%x
  sum <- sum(x)
  g <- lag/sum
}
g <- local_g(guerry$Donatns,W)
g
```

```
##           [,1]
## 1  0.005591602
## 2  0.009397496
## 3  0.012869755
## 4  0.005899573
## 5  0.005622224
## 6  0.004857546
## 7  0.009950736
## 8  0.009631099
## 9  0.007032592
## 10 0.007909610
## 11 0.006014938
## 12 0.003933049
## 13 0.015242825
## 14 0.009958611
## 15 0.020170714
## 16 0.019022821
## 17 0.015782358
## 18 0.013051155
## 19 0.010192047
## 20 0.027046638
## 21 0.019429600
## 22 0.017590355
## 23 0.012872671
## 24 0.006371504
## 25 0.017025325
## 26 0.011285570
## 27 0.021983195
## 28 0.003989335
## 29 0.006701873
## 30 0.009517652
## 31 0.015062592
## 32 0.005376372
## 33 0.013162269
## 34 0.016383426
## 35 0.011769400
```

```
## 36 0.006855276
## 37 0.009268883
## 38 0.006848422
## 39 0.011896846
## 40 0.008159237
## 41 0.006554887
## 42 0.017878953
## 43 0.010943477
## 44 0.010491436
## 45 0.010038811
## 46 0.005699567
## 47 0.015010753
## 48 0.020501521
## 49 0.009455241
## 50 0.008763181
## 51 0.010454922
## 52 0.014096245
## 53 0.007814069
## 54 0.022016441
## 55 0.016084496
## 56 0.010444482
## 57 0.011435181
## 58 0.012287351
## 59 0.015963757
## 60 0.009673096
## 61 0.013779378
## 62 0.012195193
## 63 0.004918790
## 64 0.005357999
## 65 0.010266290
## 66 0.016196486
## 67 0.007144145
## 68 0.005963792
## 69 0.009375123
## 70 0.009680095
## 71 0.008145488
## 72 0.012935374
## 73 0.008577042
## 74 0.010506601
## 75 0.018976276
## 76 0.011716555
## 77 0.005010482
## 78 0.006246916
## 79 0.003670573
## 80 0.004829840
## 81 0.018780119
## 82 0.019643080
## 83 0.018815407
## 84 0.012918167
## 85 0.009328728
```

```
local_g_sims <- function(x,W, permutations){
  n <- nrow(W)
  id <- 1:n
```

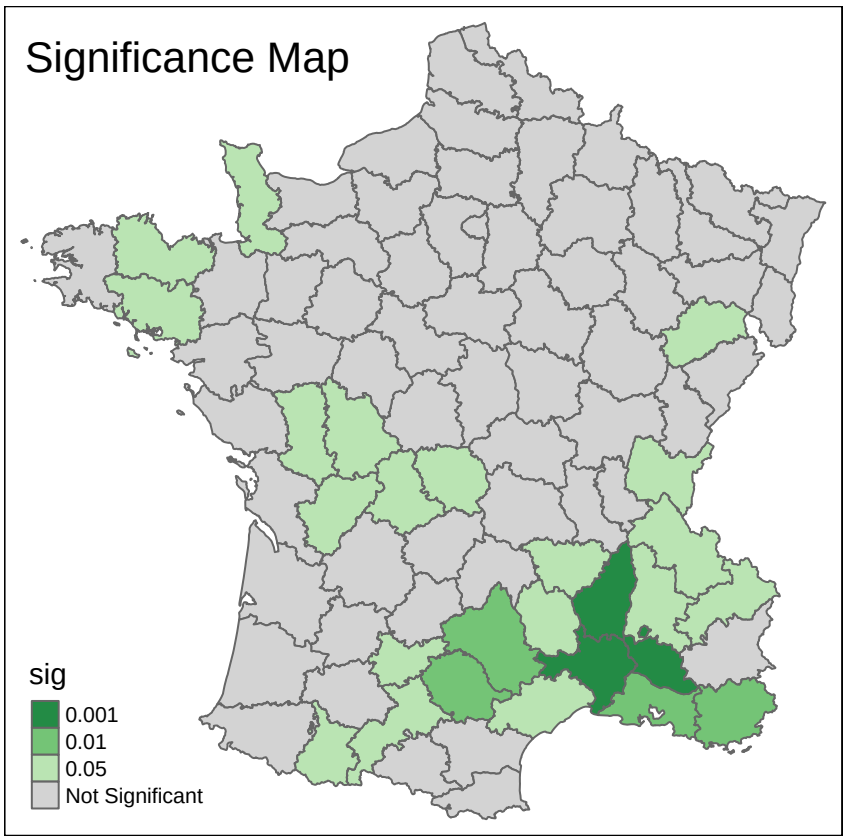
```
  local.sims  <- matrix(NA, nrow = n, ncol=permutations)
  x.sample = matrix(NA, nrow = n, ncol = permutations)
  for(i in 1:n){
    sample.indices <- sample(id[-i], permutations, replace = TRUE)
    x.sample[i,] <- x[sample.indices]
  }

  B <- binarize(W,threshold = .001)
  lag <- W%*%x.sample
  sum <- sum(x)
  local.sims <- lag / sum
  local.sims
}
```

```
sims <- local_g_sims(guerry$Donatns,W,999)
```

```
p_value <- get_p_value(sims,g,type = "two-sided")
p_value
```

```
##  [1] 0.019 0.235 0.327 0.052 0.044 0.000 0.420 0.377 0.089 0.143 0.007 0.001
## [13] 0.188 0.327 0.032 0.054 0.112 0.321 0.338 0.019 0.031 0.065 0.382 0.042
## [25] 0.076 0.482 0.079 0.000 0.031 0.272 0.209 0.020 0.319 0.106 0.432 0.047
## [37] 0.269 0.105 0.459 0.099 0.043 0.088 0.409 0.401 0.340 0.013 0.141 0.038
## [49] 0.233 0.190 0.419 0.277 0.139 0.031 0.194 0.379 0.478 0.406 0.115 0.433
## [61] 0.263 0.411 0.017 0.076 0.387 0.190 0.117 0.023 0.249 0.267 0.325 0.355
## [73] 0.144 0.370 0.047 0.445 0.003 0.017 0.002 0.000 0.075 0.024 0.046 0.324
## [85] 0.272
```

```
significance_map(guerry,p_value,999,.05)
```

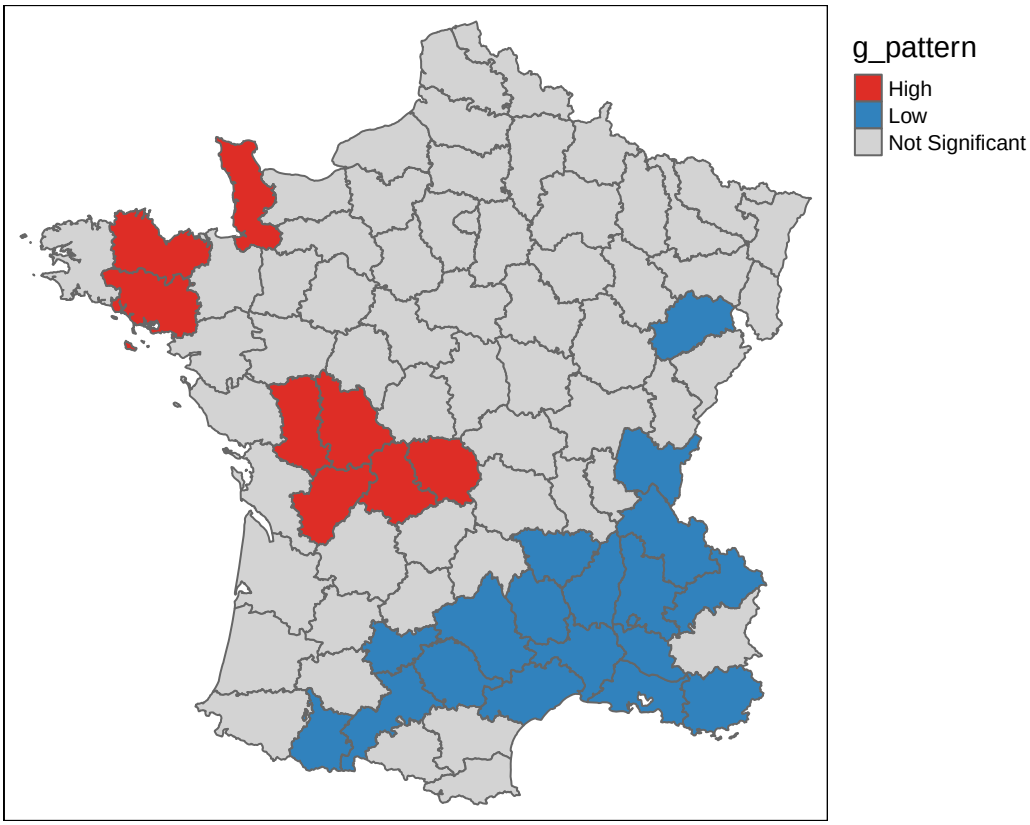



```
guerry$doub_sig <- NA
guerry$gpvalue <- p_value
guerry$g <- g
```

```
g_pattern <- rep(NA,85)
g_pattern[g > mean(g)] <- "High"
g_pattern[g < mean(g)] <- "Low"
g_pattern[p_value > .05] <- "Not Significant"
guerry <- guerry %>% mutate(g_pattern)
```

```
pal <- c("#DE2D26", "#3182BD", "#D3D3D3")
```

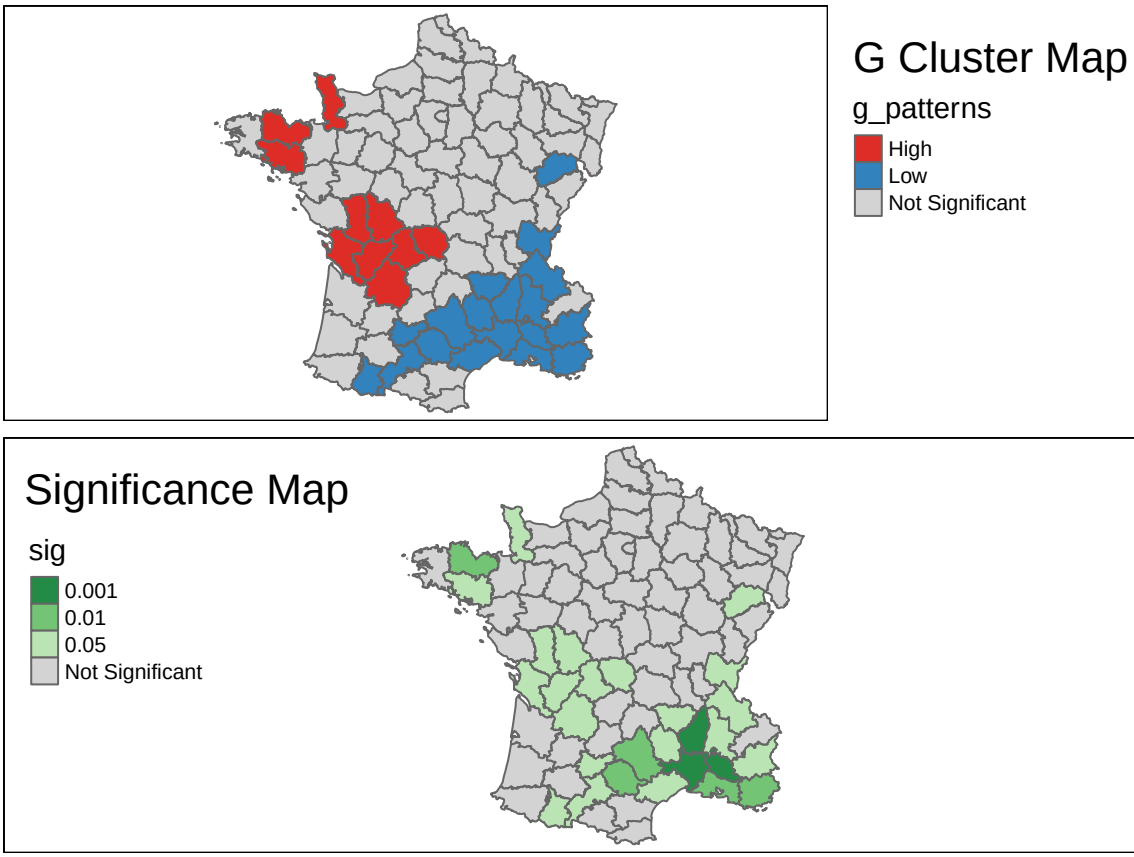
```
tm_shape(guerry) +
  tm_fill("g_pattern", palette = pal) +
  tm_borders() +
  tm_layout(legend.outside = TRUE)
```




```
local_g_map <- function(polys,x,weights,permutations,alpha,type = "G"){

  W <- convert_matrix(weights)

  if (type == "G*"){
    diag(W) <- 1
  }

  n <- nrow(W)
  g <- local_g(x,W)
  sims <- local_g_sims(x,W,permutations)
  p_value <- get_p_value(sims,g,type = "two-sided")

  g_patterns <- rep(NA,n)
  g_patterns[g > mean(g)] <- "High"
  g_patterns[g < mean(g)] <- "Low"
  g_patterns[p_value > alpha] <- "Not Significant"

  polys <- polys %>% mutate(g_patterns)

  classes <- unique(g_patterns)
  patts <- c("High", "Low", "Not Significant")
  colors <- c("#DE2D26", "#3182BD", "#D3D3D3")
  mat <- matrix(c(patts,colors), ncol = 2)
  logi <- patts %in% classes
  pre_col <- matrix(mat[logi], ncol = 2)
  pal <- pre_col[,2]
```

```
  if (type == "G*"){
    map_type <- "G*"
  } else {
    map_type <- "G"
  }

  map_title <- paste0(map_type, " Cluster Map")

  p1 <- tm_shape(polys) +
  tm_fill("g_patterns", palette = pal) +
  tm_borders() +
  tm_layout(title = map_title,legend.outside = TRUE)

  p2 <- significance_map(polys,p_value,permutations,alpha)
  tmap_arrange(p1,p2,ncol = 1)
}
```

```
local_g_map(guerry,guerry$Donatns,queen.weights,999,.05)
```



### Interpretation and significance

## Local Join Count Statistic

### Principle

Recently, Anselin and Li (2019) showed how a constrained version of the $G_i^*$ statistic yields a local version of the well-known join count statistic for spatial autocorrelation of binary variables, popularized by Cliff and Ord (1973). Expressed as a LISA statistic, a local version of the so-called BB join count statistic is

$$BB_i = x_i \Sigma_j w_{ij} x_j$$

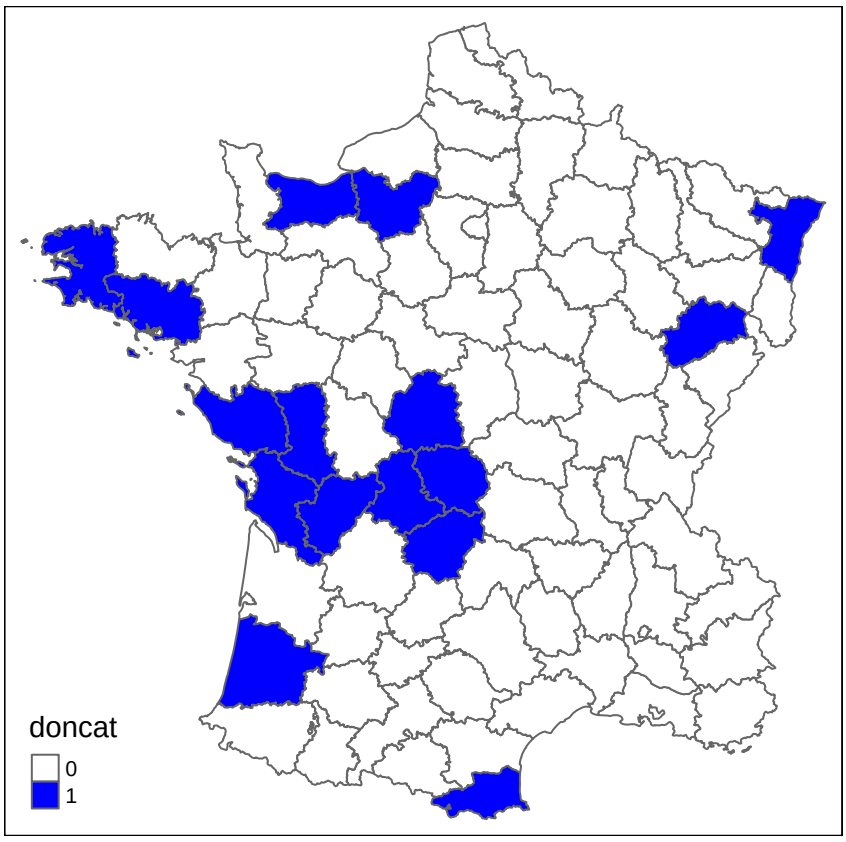
where $x_{i,j}$ can only take on the values of 1 and 0, and $w_{ij}$ are the elements of a binary spatial weights matrix (i.e., not row-standardized). For the most meaningful results, the value of 1 should be chosen for the case with the fewest observations (of course, the definition of what is 1 and 0 can easily be switched).

The statistic is only meaningful for those observations where $x_i = 1$, since for $x_i = 0$ the result will always equal zero. A pseudo p-value is obtained by means of a conditional permutation approach, in the same way as for the other local spatial autocorrelation statistics, but only for those observations with $x_i = 1$. The same caveats as before should be kept in mind when interpreting the results, which are subject to multiple comparisons and the sensitivity of the pseudo p-value to the actual simulation experiment (random seed, number of permutations). Technical details are provided in Anselin and Li (2019).

### Implementation

```
doncat <- rep(0, 85)
doncat[guerry$Donatns > 10996] <- 1
guerry$doncat <- doncat
```

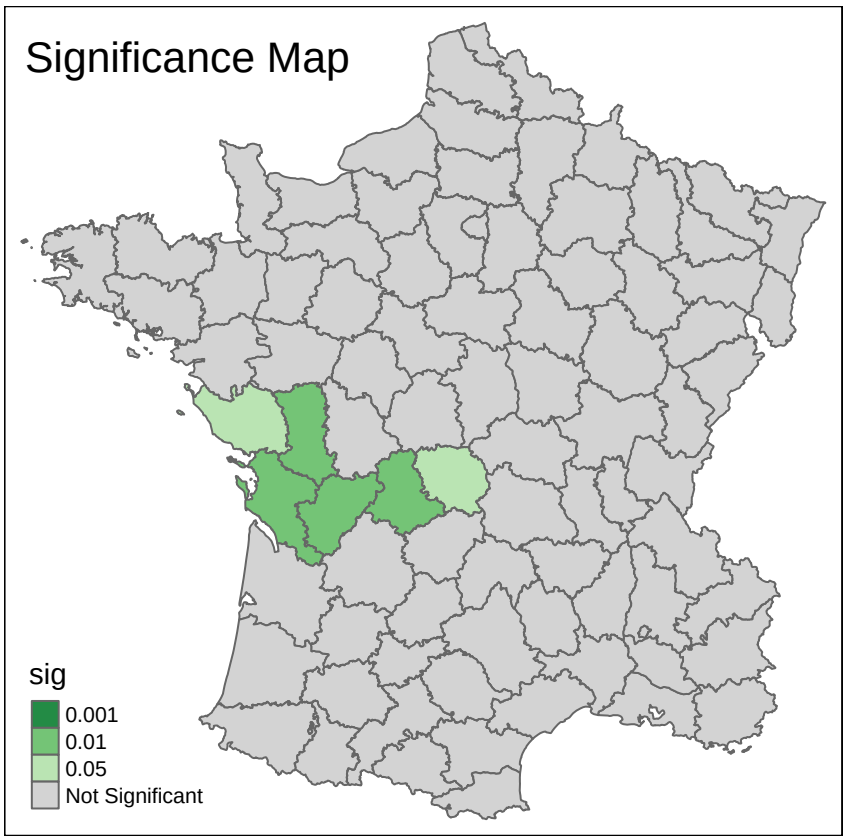
```
tm_shape(guerry) +
  tm_fill("doncat", style = "cat", palette = c("white", "blue")) +
  tm_borders()
```




```
local_bb <- function(x,W){
  B <- binarize(W,threshold = .00001)
  bb <- x * B%*%x
}
bb <- local_bb(guerry$doncat,W)
```

```
local_bb_sims <- function(x,W,permutations){
  n <- nrow(W)
  id <- 1:n

  local.sims <- matrix(NA, nrow = n, ncol=permutations)
  x.sample = matrix(NA, nrow = n, ncol = permutations)
  for(i in 1:n){
    sample.indices <- sample(id[-i], permutations, replace = TRUE)
    x.sample[i,] <- x[sample.indices]
  }

  B <- binarize(W,threshold = .00001)
  local.sims <- x.sample * B%*%x.sample
  local.sims
}
```

```
sims <- local_bb_sims(guerry$doncat,W,999)
p_value <- get_p_value(sims,bb,type = "one-sided")
significance_map(guerry,p_value,999,.05)
```




**Function Appendix**

```
univariate_lisa_map <- function(area,x, weights, permutations, alpha){
  # function to create a bivariate LISA map
  # arguments:
  #    area: sf dataframe
  #    x: vectors of x values
  #    y: a vector of y values
  #    nperms: the number of permuations used to calculate the pvalues
  #    alpha: the alpha level required for significance
  # returns:
  #    a LISA map in GeoDa style


  # Converting the weights
  W <- convert_matrix(weights)

  # Computing the observed statistics and reference distributions
  observed <- univariate_moran(x,W)
  sims <- univariate_moran_sim(x,W, permutations = nperms)


  #computing p-value from observed statistics and reference distribution
  p_value <- get_p_value(sims,observed, type = "two-sided")
```

```
  # Standardizing the x and y vars
  xs <- standardize(x)

  #Assigning classifications
  lisa_patterns <- as.character( interaction(xs > 0, W%*%xs > 0) )
  lisa_patterns <- patterns %>%
        str_replace_all("TRUE","High") %>%
        str_replace_all("FALSE","Low")
  lisa_patterns[which(p_value > alpha)] <- "Not Significant"

  # Adding the classifications to the data frame
  area <- area %>% mutate(lisa_patterns)

  # Constructing the correct palette based on presense of classifications
  classes <- unique(lisa_patterns)
  patts <- c("High.High", "High.Low", "Low.High", "Low.Low", "Not Significant")
  colors <- c("#DE2D26","#FCBBA1","#C6DBEF", "#3182BD", "#D3D3D3")
  mat <- matrix(c(patts,colors), ncol = 2)
  logi <- patts %in% classes
  pre_col <- matrix(mat[logi], ncol = 2)
  pal <- pre_col[,2]

  # Making the LISA map
  p1 <- tm_shape(area) +
  tm_fill("lisa_patterns", palette = pal) +
  tm_borders() +
  tm_layout(title = "LISA cluster map")

  p2 <- significance_map(area, p_value, permutations = permutations,alpha = alpha)

  tmap_arrange(p1,p2, ncol = 1)
}
```