Análisis de relaciones de vecindad y dependencia espacial en R

Rodrigo Tapia McClung

Agosto 4, 2017

Vecindad y dependencia espacial

Queremos estudiar el comportamiento espacial de un conjunto de polígonos. En particular, explorar si presentan autocorrelación espacial global y localmente. Para ello, recordemos cómo se calcula la I de Moran:

$$I = \frac{n}{\sum_i \sum_{j \neq i} w_{ij}} \frac{\sum_i \sum_{j \neq i} w_{ij}(x_i - \bar{x})(x_j - \bar{x})}{\sum_i (x_i - \bar{x})^2}$$

Y la versión local se puede escribir como:

$$I_i = \frac{(x_i - \bar{x}) \sum_{j \neq i} w_{ij}(x_j - \bar{x})}{\frac{1}{n} \sum_{j \neq i} (x_j - \bar{x})^2}$$

Los valores x son nuestras observaciones y los w_{ij} representan los elementos de una matriz de pesos (renglón i, columna j). Hay que construir esta matriz.

Requisitos: tener los SHPs en alguna carpeta y cambiar a ese directorio de trabajo. Por ejemplo:

```
# Cambiar directorio de trabajo
setwd("C:/Descargas/R/practica2")
```

Asegurarse de tener instaladas las librerías que vamos a usar:

```
# Lista de librerías:
list.of.packages <- c("rgdal", "sp", "GISTools", "RColorBrewer", "ggplot2",
    "reshape2", "grid", "gridExtra", "spdep")
# Ver qué no está instalado
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,
    "Package"])]
# Si falta algo, instalarlo
if (length(new.packages)) install.packages(new.packages)
```

Abrir SHPs en R

Usaremos `rgdal` para abrir dos shapefiles en R (`estados.shp`)

```
library(rgdal)
```

```
## Warning: package 'rgdal' was built under R version 3.3.3
```

```
## Loading required package: sp
```

```
## Warning: package 'sp' was built under R version 3.3.3
```

```
## rgdal: version: 1.2-7, (SVN revision 660)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.0.1, released 2015/09/15
##  Path to GDAL shared files: C:/Users/rodrigo.tapia/Documents/R/win-library/3.3/rgdal/gdal
##  Loaded PROJ.4 runtime: Rel. 4.9.2, 08 September 2015, [PJ_VERSION: 492]
##  Path to PROJ.4 shared files: C:/Users/rodrigo.tapia/Documents/R/win-library/3.3/rgdal/proj
##  Linking to sp version: 1.2-4
```

```
edos <- readOGR(".", "estados_sorted", stringsAsFactors = FALSE, GDAL1_integer64_policy = T)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "estados_sorted"
## with 32 features
## It has 4 fields
## Integer64 fields read as doubles:  POP_ADMIN CVE_ENT
```

Pegarle los atributos de interés a la tabla del SHP por medio de un `merge`

Nos interesa leer un archivo de tipo CSV que tiene información acerca de homicidios dolosos vinculados con la delincuencia organizada por año y por estado:

```
# La primera columna es de texto y las demás de enteros
homicidios <- read.csv("homicidios.csv", colClasses = c(rep("character", 1),
    rep("integer", 7)))
```

Si quieres, puedes ver las columnas de este archivo con:

```
colnames(homicidios)
```

Observa que en nuestro CSV los nombres de las columnas son los años. `R` les asigna una letra al inicio, de modo que se llaman `A2006`, `A2007`, etc.

Cargamos la librería `sp`:

```
library(sp)
```

Ahora usamos el método `merge` para pegar las tablas de atributos por medio de una llave única. En este caso, la clave del estado en cuestión, `CVE_ENT`.

```
# Pegarle los datos de homicidios a los estados
edos <- merge(edos, homicidios, by.x = "CVE_ENT")
```

Si ejecutas `edos@data` verás que cada estado tiene la nueva información que le acabamos de anexar. Puedes ver las nuevas columnas en la tabla de los estados con:

```
names(edos@data)
```

Exploración visual de los datos

Antes de entrar de lleno en el análisis de la autocorrelación y la dependencia espacial, es interesante primero mapear las variables que nos interesan. Tenemos 7 años de datos de homicidios y solo toma algo de tiempo hacer que la computadora haga todo el procesamiento de los datos por nosotros. Pero vale la pena ser un poco sensatos en lo que lo que vamos a pedir.

Podemos hacer un mapa de desviaciones estándar que nos muestre la variación espacial alrededor del promedio de homicidios por año por estado.

Hay dos maneras de hacer estos mapas, cada una con sus pros y contras. Primero vamos a trabajar con la librería `GISTools` y después con `ggplot2`.

Un mapa de desviaciones estándar - `GISTools`

Cargamos la librería y otra más para usar colores:

```
library(GISTools)
```

```
## Loading required package: maptools
```

```
## Warning: package 'maptools' was built under R version 3.3.3
```

```
## Checking rgeos availability: TRUE
```

```
## Loading required package: RColorBrewer
```

```
## Loading required package: MASS
```

```
## Loading required package: rgeos
```

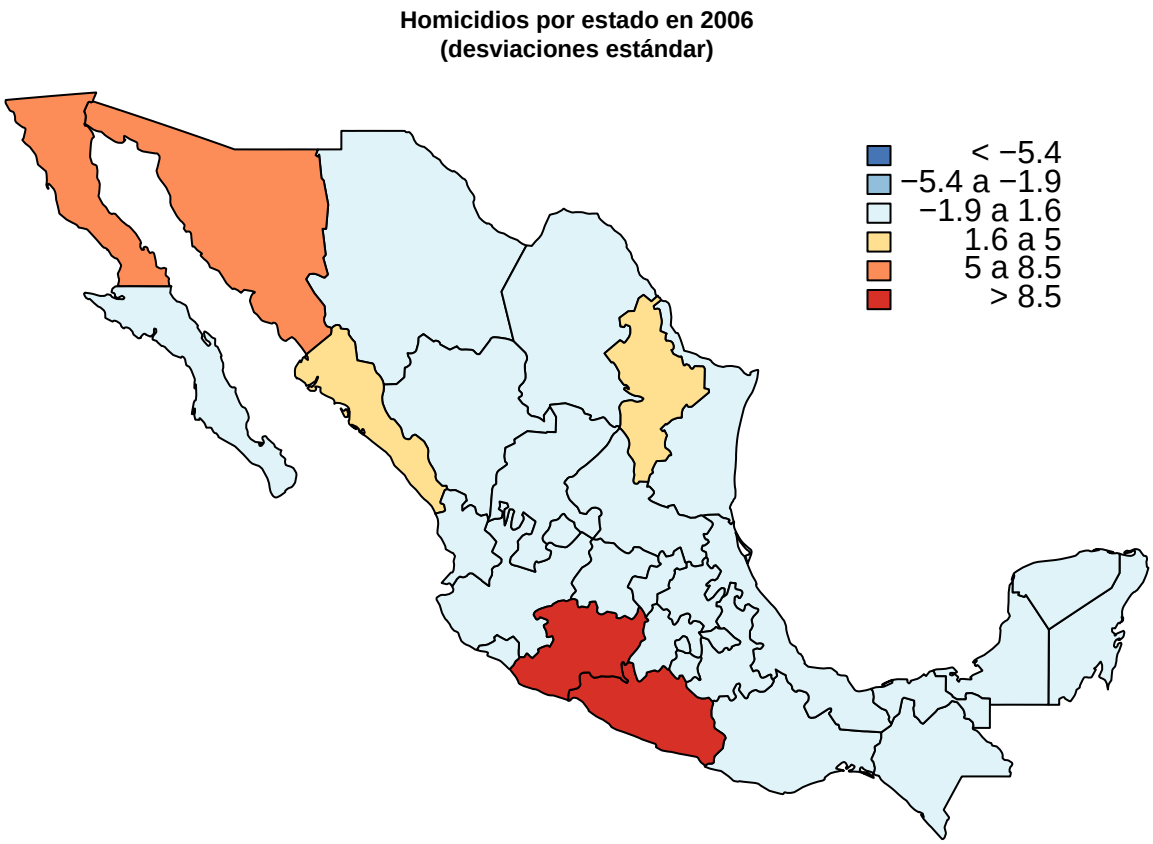
```
## Warning: package 'rgeos' was built under R version 3.3.3
```

```
## rgeos version: 0.3-23, (SVN revision 546)
##  GEOS runtime version: 3.5.0-CAPI-1.9.0 r4084
##  Linking to sp version: 1.2-4
##  Polygon checking: TRUE
```

```
library(RColorBrewer)
```

Definimos nuestro mapa temático:

```
# Definir márgenes para ocupar todo el espacio
par(mar = c(0, 0, 1.5, 0))
# Definir un esquema para colorear el mapa de acuerdo a desviaciones
# estándar
shades <- auto.shading(edos$A2006, cutter = sdCuts, n = 6, cols = rev(brewer.pal(6,
    "RdYlBu")))
# Definimos el mapa temático
choropleth(edos, edos$A2006, shades)
# Agregar una leyenda
choro.legend(-95, 32, shades, under = "<", over = ">", between = "a", box.lty = "blank",
    x.intersp = 0.5, y.intersp = 0.75)
# Agregar título
title(main = "Homicidios por estado en 2006\n(desviaciones estándar)", cex.main = 0.75)
```



Este mapa muestra que, para 2006, la mayoría de los estados tienen valores por debajo del promedio y hay uno con valores entre y 1 y 2 SD, y dos con valores por encima de 2 SD. ¿Cómo se compara esto año con año? Vamos a hacer mapas de este estilo para cada año y compararlos.

Varios mapas de desviaciones estándar

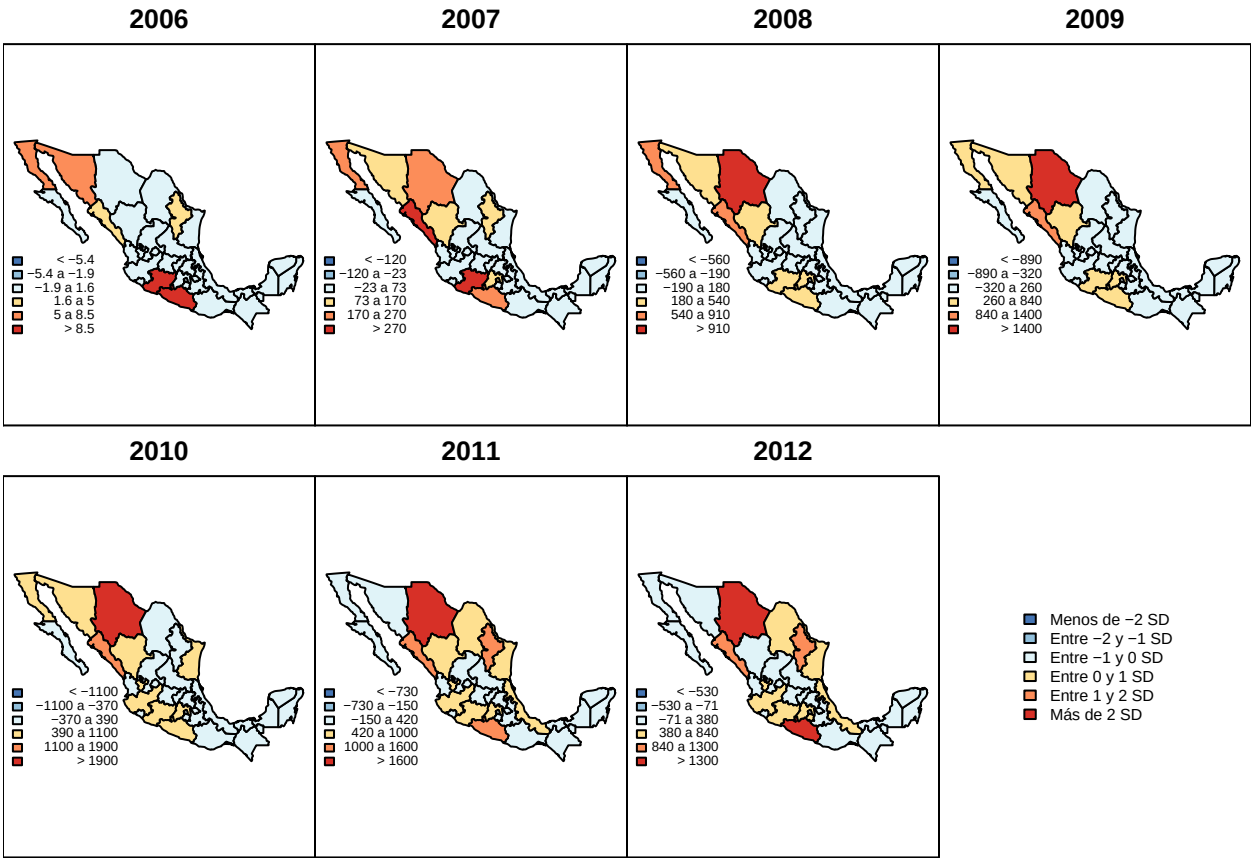
Podríamos repetir las mismas instrucciones de arriba tantas veces como mapas queramos producir... Pero eso es ineficiente. Mejor vamos a hacer una función:

```
# Definir una función que haga un mapa como el anterior
makeChoro <- function(var, title) {
    shades <- auto.shading(var, cutter = sdCuts, n = 6, cols = rev(brewer.pal(6,
        "RdYlBu")))
    choropleth(edos, var, shades)
    choro.legend(-118.5, 22.5, shades, under = "<", over = ">", between = "a",
        x.intersp = -2.5, y.intersp = 0.9, box.lty = "blank", cex = 0.6)
    title(title)
    box(col = "black")
}
# Modificar parámetros del plot para poder acomodar bien: sin margen arriba,
# hacer un grid de 2 x 4 y un poco de margen entre cada plot
op <- par(oma = c(0, 0, 0, 0), mfrow = c(2, 4), mar = c(0, 0, 2, 0))

# Hacer mapa para cada año
p2006 <- makeChoro(edos$A2006, "2006")
p2007 <- makeChoro(edos$A2007, "2007")
```

```
p2008 <- makeChoro(edos$A2008, "2008")
p2009 <- makeChoro(edos$A2009, "2009")
p2010 <- makeChoro(edos$A2010, "2010")
p2011 <- makeChoro(edos$A2011, "2011")
p2012 <- makeChoro(edos$A2012, "2012")
plot.new()  # Hacer como que hacemos un nuevo plot para avanzar en el grid

legend("center", legend = c("Menos de -2 SD", "Entre -2 y -1 SD", "Entre -1 y 0 SD",
    "Entre 0 y 1 SD", "Entre 1 y 2 SD", "Más de 2 SD"), fill = shades$cols,
    bty = "n", cex = 0.75, y.intersp = 1, x.intersp = 1, ncol = 1)
```



A partir de estos mapas, podemos ver que hay algunos estados que tienen un número de homicidios por arriba del promedio anual en distintos años (por ejemplo, Chihuahua). Sin embargo, es un poco difícil comparar tanto los valores reales del número de homicidios como de las desviaciones estándar para distintos años. No obstante, esto nos da una indicación de que en Chihuahua algo pasa. Esto lo retomaremos más adelante.

Por lo pronto, vamos a hacer otros mapas de desviaciones estándar con otra librería.

Mapas de desviaciones estándar - `ggplot2`

Primero vamos a definir una función para asignarle una etiqueta a cada estado para ver en qué intervalo de desviaciones estándar se encuentra en cada año:

```
sdClass <- function(var, year){
  mean <- mean(var)
  sd <- sd(var)
```

```
  sd.vec <- vector(mode = "character",length = length(var))

  sd.vec[var <= mean-2*sd] <- '<2' # var <= -2 SD
  sd.vec[var > mean-2*sd & var < mean-sd] <- '-2-1' # -2 SD < var <= -1 SD
  sd.vec[var > mean-sd & var <= mean] <- '-1-0' # -1 SD < var <= 0 SD
  sd.vec[var > mean & var <= mean+sd] <- '0-1' # 0 SD < var <= 1 SD
  sd.vec[var > mean+sd & var < mean+2*sd] <- '1-2' # 1 SD < var < 2 SD
  sd.vec[var >= mean+2*sd] <- '>2' # var => 2 SD

  # Lo hacemos un factor para que, aunque no haya elementos
  # en el intervalo de -2SD, exista y lo podamos usar
  sd.vec <- factor(sd.vec, levels = c('<-2', '-2-1', '-1-0', '0-1', '1-2', '>2'))
  return(sd.vec)
}
```

Pero para no confundirnos con los datos que trabajamos anteriormente, volvemos a leer los archivos y los asignamos a nuevas variables:

```
edos.gg <- readOGR(".", "estados_sorted", stringsAsFactors = FALSE, GDAL1_integer64_policy = T)
```

```
## OGR data source with driver: ESRI Shapefile
## Source: ".", layer: "estados_sorted"
## with 32 features
## It has 4 fields
## Integer64 fields read as doubles:  POP_ADMIN CVE_ENT
```

```
homicidios.gg <- read.csv("homicidios.csv", colClasses = c(rep("character",
    1), rep("integer", 7)))
```

Ahora creamos un nuevo `data.frame` para esta nueva clasificación, le copiamos los valores de las claves de los estados y ejecutamos la función que acabamos de definir para cada año:

```
# Definir data frame y copiar datos de clave de entidad
sd.homicidios <- as.data.frame(homicidios.gg[,1])
# Renombrar la columna del nuevo data frame
colnames(sd.homicidios) = c('CVE_ENT')

# Ejecutar la función y clasificar cada año
sd.homicidios$'2006' <- sdClass(homicidios.gg$A2006, "2006")
sd.homicidios$'2007' <- sdClass(homicidios.gg$A2007, "2007")
sd.homicidios$'2008' <- sdClass(homicidios.gg$A2008, "2008")
sd.homicidios$'2009' <- sdClass(homicidios.gg$A2009, "2009")
sd.homicidios$'2010' <- sdClass(homicidios.gg$A2010, "2010")
sd.homicidios$'2011' <- sdClass(homicidios.gg$A2011, "2011")
sd.homicidios$'2012' <- sdClass(homicidios.gg$A2012, "2012")
```

Cargamos la librería:

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.3.3
```

Y le pegamos la clasificación que acabamos de hacer a los polígonos de los estados:

```
# Agregar clases de SD a los estados
edos.gg <- merge(edos.gg, sd.homicidios, by.x = "CVE_ENT")
```

Ahora queremos cambiarle el orden a los datos para poder tener una lista *hacia abajo* en vez de *hacia la*

derecha. Para esto, usamos el método `melt` de la librería `reshape2`:

```
# Cargar reshape2
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 3.3.3
```

```
# Hacer el melt de las clases de SD
sd.homicidios.melt <- melt(sd.homicidios, id = c("CVE_ENT"))
```

Toma un momento para ver la diferencia entre `sd.homicidios` y `sd.homicidios.melt`.

Lo malo de usar `ggplot` es que **necesita** usar un *data frame* de `R`. Si te fijas, nuestros estados son un `SpatialPolygonsDataFrame`. Hay que hacer algo al respecto. Usamos `fortify` para crear un data frame que contenga la geometría de los estados. Esto hace que perdamos muchos atributos, pero se los volvemos a pegar...

```
# Hacer un data frame de R
edos_geom.gg <- fortify(edos.gg, region = "CVE_ENT")
# Volverle a pegar los datos que ya tenía
edos_geom.gg <- merge(edos_geom.gg, edos.gg@data, by.x = "id", by.y = "CVE_ENT")
```

Esto provoca que cada estado se separe en segmentos y que tenga sus atributos. Ahora le pegamos los datos de la clasificación de los intervalos de desviaciones estándar:

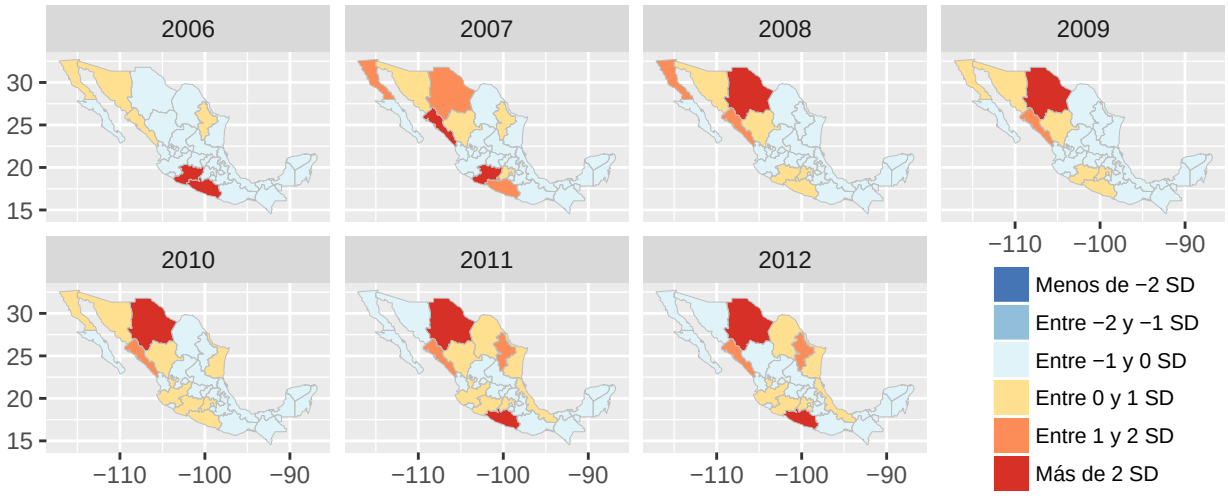
```
# Pegarle los datos de las clasificaciones de SD a cada segmento
plot.data.gg <- merge(edos_geom.gg, sd.homicidios.melt, by.x = "id", by.y = "CVE_ENT")
```

Toda esta vuelta que parece inútil es para poder usar algo super poderoso de `ggplot` que nos permita generar todos los mapas con una sola instrucción:

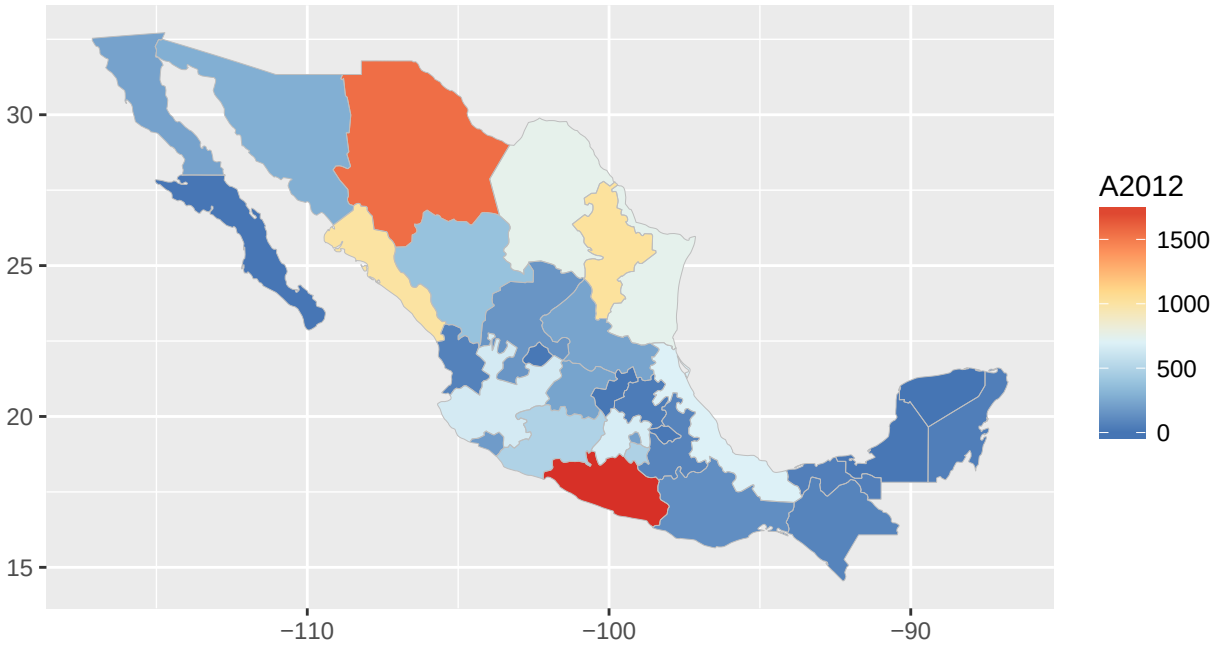
```
# Volvemos a hacer que la clasificacón de las SD sea un factor, porque se
# perdió al hacer el melt...
plot.data.gg$value <- factor(plot.data.gg$value, levels = c('<-2', '-2-1', '-1-0',
  '0-1', '1-2', '>2'))

# Definir el objeto de ggplot
ggplot(data = plot.data.gg, aes(x = long, y = lat, fill = value, group = group)) +
# Agregarle la geometría de los polígonos, colorear los bordes y aspecto 1:1
  geom_polygon() + geom_path(colour = "grey", lwd = 0.1) + coord_equal() +
# Hacer el facet wrap con 4 columnas
  facet_wrap(~variable, ncol = 4) +
# Agregar colores, leyenda y quitar nombres de los ejes
  scale_fill_brewer(palette = "RdYlBu", direction = -1, name = "",
    breaks = c('<-2', '-2-1', '-1-0', '0-1', '1-2', '>2'),
    labels = c('Menos de -2 SD', 'Entre -2 y -1 SD', "Entre -1 y 0 SD", "Entre 0 y 1 SD",
    "Entre 1 y 2 SD", "Más de 2 SD"), drop = FALSE, guide = "legend") +
  labs(x = NULL, y = NULL) +
# Modificar en donde aparece la leyenda
  theme(legend.position = c(0.89, 0.13), legend.key.size = unit(5, "mm"),
    legend.text = element_text(size = 8), legend.margin = margin(t = -1, unit = 'cm'))
```

```
## Warning in grid.Call(L_stringMetric, as.graphicsAnnot(x$label)): font
## metrics unknown for character 0x1f
```



Claro, puedes hacer un solo mapa de desviaciones estándar con `ggplot`. Pero eso se queda de tarea...

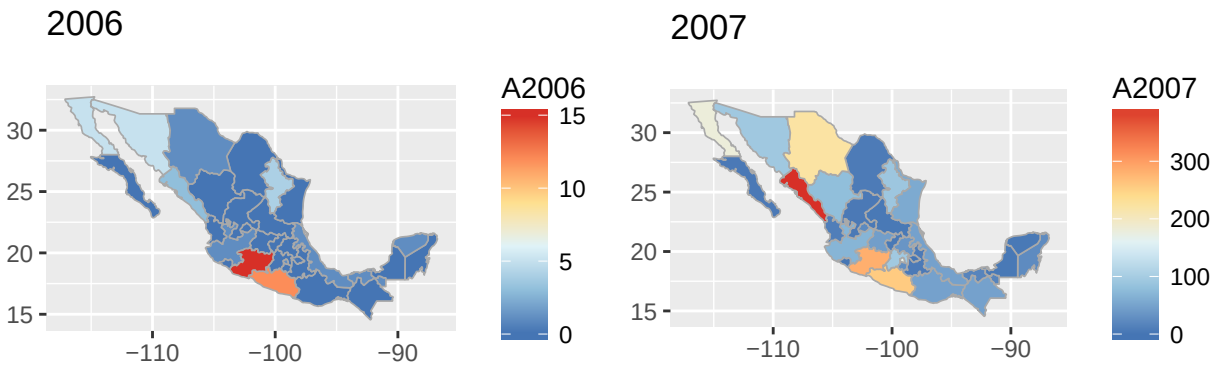


Si es muy enredado trabajar los datos para hacer un facet wrap, hay otra forma de hacer los plots en un grid con ggplot...

```
# Definir plot para un año
p2006 <- ggplot(edos_geom.gg2, aes(x = long, y = lat, group = group)) +
  geom_polygon(aes(x = long, y = lat, group = group, fill = A2006), color = "dark grey",
    size = 0.3) + coord_equal() +
  scale_fill_gradientn(colors = rev(brewer.pal(6, 'RdYlBu'))) +
  theme(legend.position = "right") +
  labs(x = NULL, y = NULL, title = "2006", subtitle = "", caption = "")
# Definir plot para otro año
p2007 <- ggplot(edos_geom.gg2, aes(x = long, y = lat, group = group)) +
  geom_polygon(aes(x = long, y = lat, group = group, fill = A2007), color = "dark grey",
    size = 0.3) + coord_equal() +
  scale_fill_gradientn(colors = rev(brewer.pal(6, 'RdYlBu'))) +
  theme(legend.position = "right") +
  labs(x = NULL, y = NULL, title = "2007", subtitle = "", caption = "")
# Cargar un par de librerías
library(grid)
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 3.3.3
```

```
# Hacer el grid...
grid.arrange(p2006, p2007, ncol = 2)
```



Un incoveniente es que se repiten las leyendas y `ggplot` no tiene un método específico para hacer que todas las gráficas compartan la misma leyenda. Se puede hacer pero es un poco enredado... Eso queda para otro taller... Del mismo modo, podrías hacer un *facet wrap* para histogramas de cada año que muestren la distribución de los homicidios por estado.

Por ahora, regresemos al objetivo que teníamos antes de analizar la información de manera visual.

Adyacencia y estadística espacial

Vamos a definir la adyacencia de polígonos. Para ello, primero veremos un ejemplo sencillo al usar puntos y después extenderemos el concepto a polígonos. Después seguiremos con el análisis de la estadística espacial de los homicidios en los estados.

Cargamos la librería `spdep`:

```
library(spdep)
```

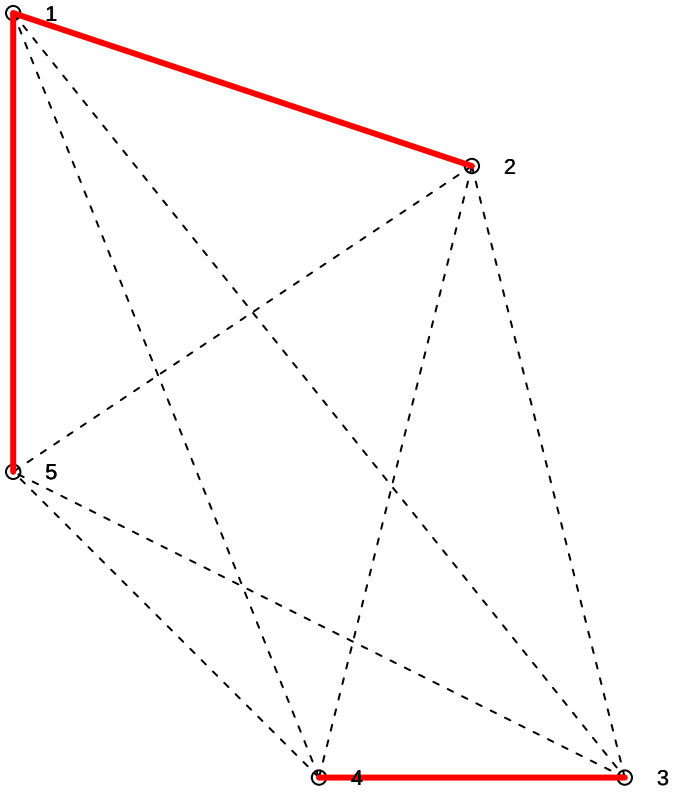
```
## Warning: package 'spdep' was built under R version 3.3.3
```

```
## Loading required package: Matrix
```

Matriz de adyacencia

Una matriz de adyacencia es una manera de expresar si dos entidades son contiguas o no. La manera más fácil de codificar esto es con una matriz de entradas binarias: 0 o 1. Muchas veces estas matrices son simétricas (¡pero no siempre!) pues si A es vecino de B, B suele ser vecino de A. Por ejemplo, considera el siguiente arreglo de puntos:

```
## NULL
```




```
## NULL
```

Los puntos 1 y 2 son vecinos, al igual que el 3 y el 4 y el 5 y 1. En forma de matriz esto lo podemos expresar como:

$w_{ij} \rightarrow$

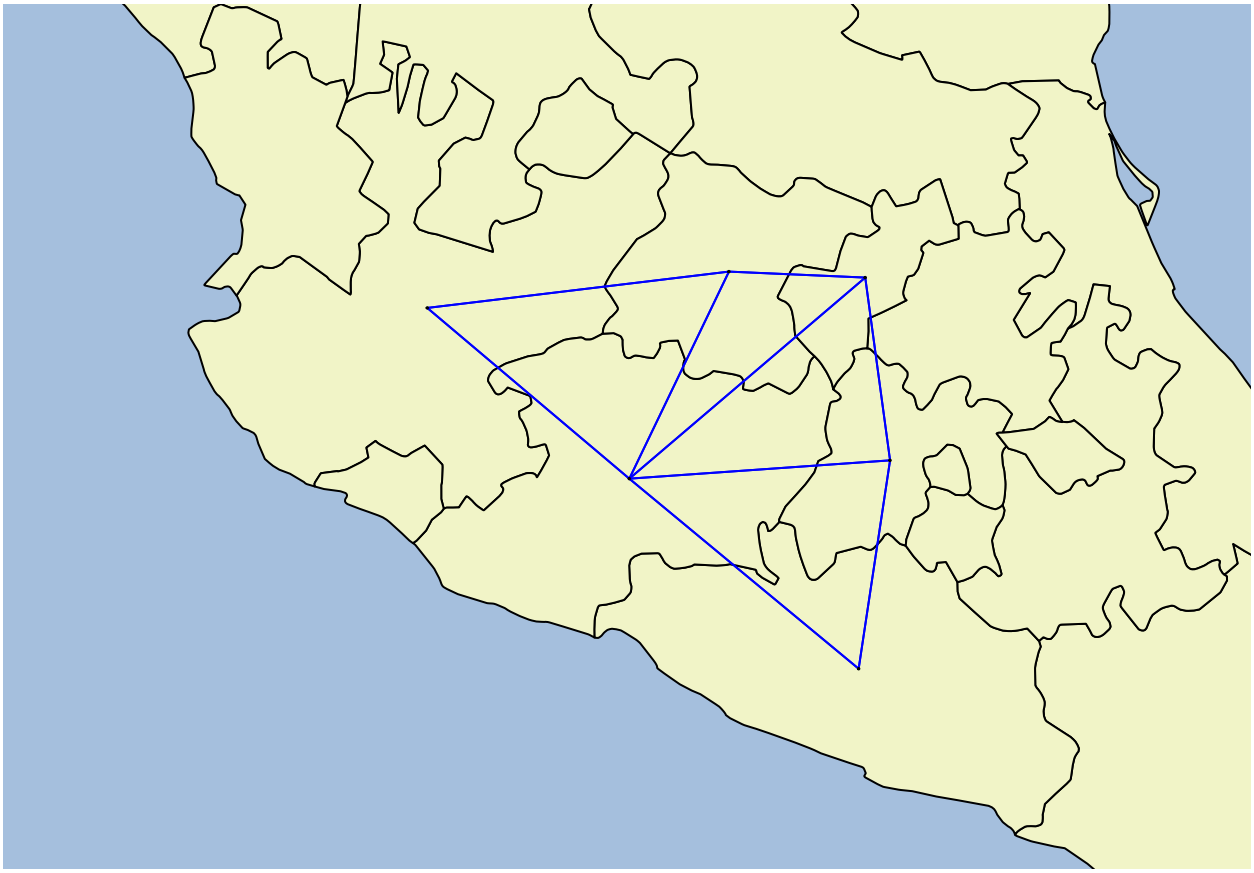
	1	2	3	4	5
1	0	**1**	0	0	**1**
2	**1**	0	0	0	0
3	0	0	0	**1**	0
4	0	0	**1**	0	0
5	**1**	0	0	0	0

Los 0s y **1**s son los **pesos**, que en este caso son binarios e indican si hay adyacencia o no. Esta matriz se puede transformar a una que se llama *estandarizada por renglones*. Esto significa sumar los pesos y dividirlos entre el total de la columna. Nuestra matriz anterior estandarizada por renglón se ve así:

$w_{ij} \rightarrow$

	1	2	3	4	5
1	0	**0.5**	0	0	**0.5**
2	**1**	0	0	0	0
3	0	0	0	**1**	0
4	0	0	**1**	0	0
5	**1**	0	0	0	0

Si definimos la adyacencia en base a los vecinos de orden mayor, las matrices pueden no ser simétricas. Por ahora, nos vamos a concentrar en adyacencia de primer orden de polígonos. Podemos pensar varias formas en que los polígonos podrían ser vecinos, pero las más comunes son: adyacencia tipo **reina** y tipo **torre**. Por ejemplo, consideremos solo los vecinos de Michoacán:



Aquí podemos ver que **todos** los polígonos que están *pegados* a Michoacán están catalogados como sus vecinos. Ahora tenemos que hacer lo mismo pero para los 32 estados.

Definir vecinos de contigüidad para los estados como una lista

```
# Construir la lista de vecinos
edos.nbq <- poly2nb(edos, queen = T)  # TRUE: tipo reina
# Convertir la lista de vecinos en una lista de pesos estandarizados por renglón
edos.nbq.w <- nb2listw(edos.nbq)
```

Si quieres ver información acerca de la lista de pesos, puedes usar:

```
summary.nb(edos.nbq)
```

Autocorrelación espacial global y local

Queremos ver si hay alguna indicación de que las observaciones se acumulan en cierta región, o si estamos ante la presencia de un proceso aleatorio. Podemos calcular la `I` de Moran para algún año en particular, digamos 2012.

```
# Definimos una variable para no repetir todo el tiempo
var <- edos$A2012
moran.test(var, edos.nbq.w)
```

```
##
```

```
##  Moran I test under randomisation
##
## data:  var
## weights: edos.nbq.w
##
## Moran I statistic standard deviate = 1.4612, p-value = 0.07199
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##        0.14045922       -0.03225806        0.01397264
```

```
##          Moran I test under randomisation

## data:  var
## weights: edos.nbq.w

## Moran I statistic standard deviate = 1.5111, p-value = 0.06538
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##        0.14808294       -0.03225806        0.01424226
```

Como tiene un valor de `0.14` bastante más alto que el valor esperado, es un indicador de que estamos ante la presencia de autocorrelación espacial global. Ahora calculamos la `I` de Moran local:

```
# Calcular la I de Moran local
lmoran <- localmoran(var, edos.nbq.w)
```

Puedes ver información acerca de la `I` local con `summary(lmoran)`.

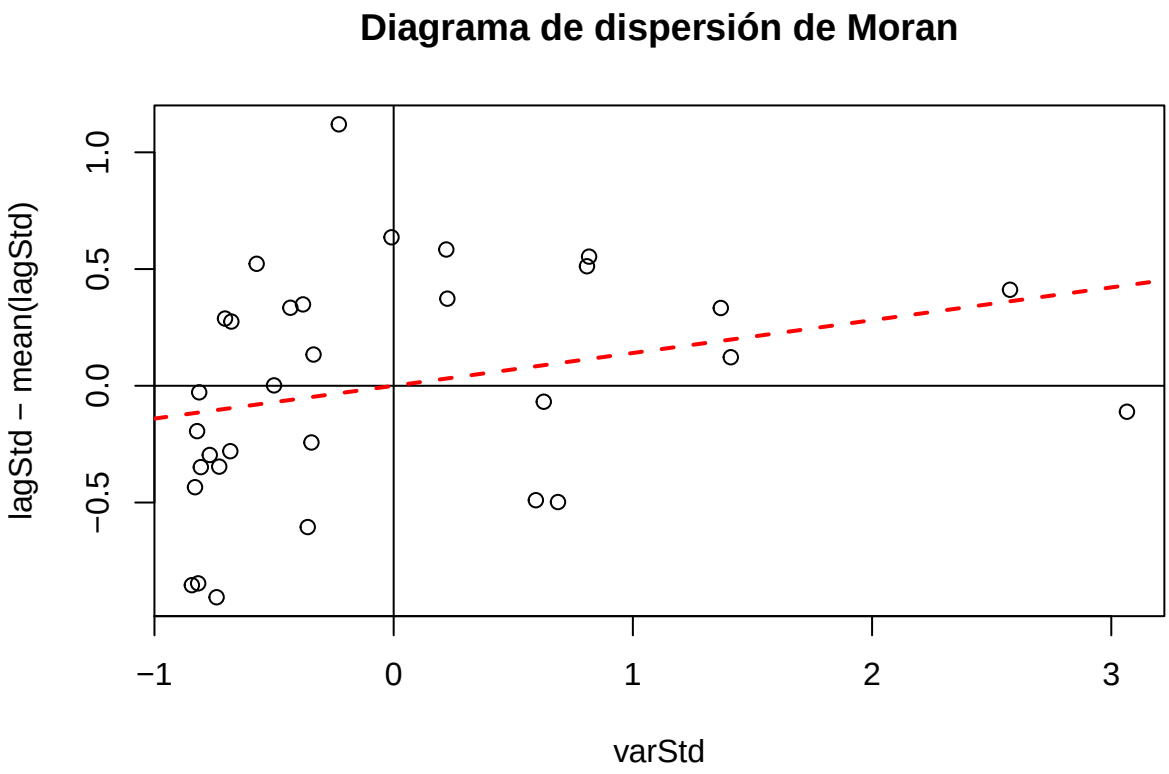
Diagrama de dispersión de Moran

Estandarizamos las variables:

```
lagStd <- lag.listw(edos.nbq.w, scale(var)) # usamos la variable estandarizada
varStd <- (var - mean(var))/sd(var)
# Es lo mismo que as.vector(scale(var))
```

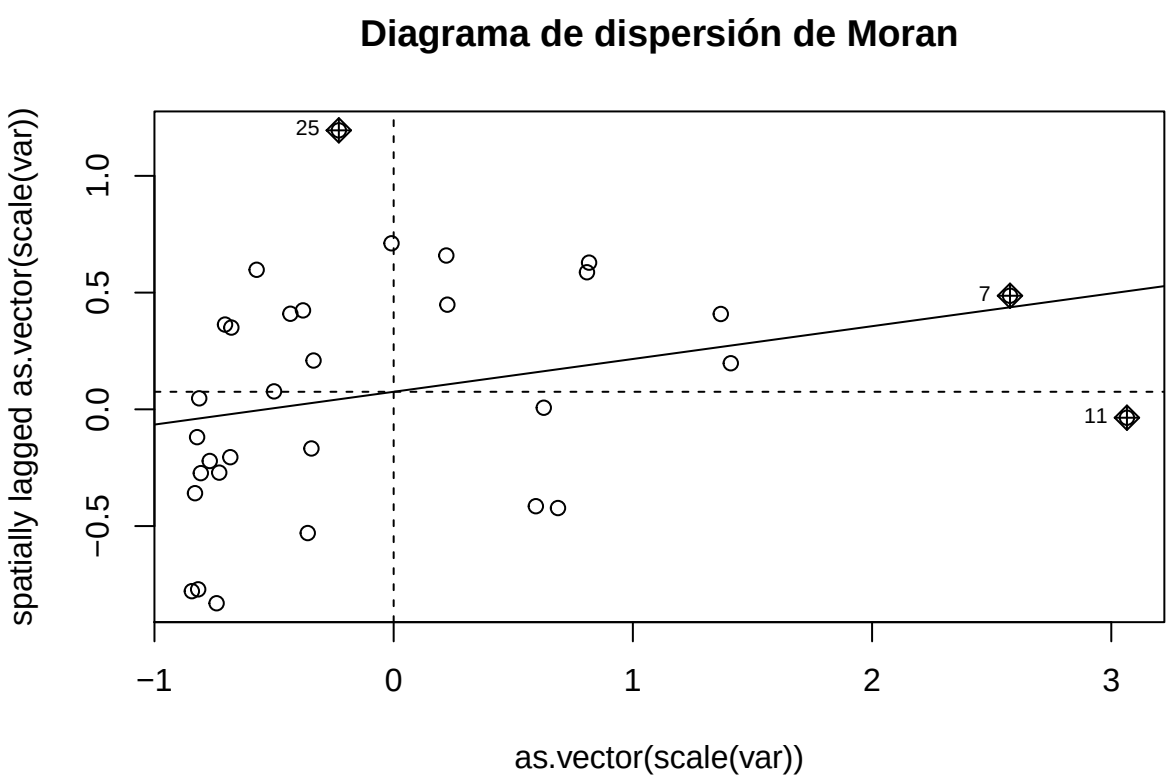
Y hacemos la gráfica:

```
plot(varStd,lagStd-mean(lagStd))
# Ejes que pasan por el origen
abline(h = 0, v = 0)
# Recta de ajuste lineal entre las dos variables
#abline(lm(lagStd ~ varStd), lty = 2, lwd = 2, col = "red")
abline(lm(lagStd-mean(lagStd) ~ varStd), lty = 2, lwd = 2, col = "red")
title("Diagrama de dispersión de Moran")
```



O más rápido, usamos la función `moran.plot()`

```
# Diagrama de dispersión
moran.plot(as.vector(scale(var)), edos.nbq.w)
title("Diagrama de dispersión de Moran")
```



Vemos que la pendiente es positiva, pues coincide con el valor de la `I` global.

Mapa de cúmulos

Definimos una significancia para nuestros mapas:

```
# Definir significancia
significancia <- 0.05
```

y un vector para almacenar el cuadrante en el que se encuentra cada punto en el diagrama de Moran:

```
# Definir vector de cuadrante del plot de Moran
cuadrante <- vector(mode = "numeric",length = length(var))
```

En el diagrama de dispersión de Moran, el eje `x` es la variable `varStd` y el eje `y` es `lagStd`. Así que encontramos en cuál cuadrante está cada punto:

```
# Definición de cúmulos
cuadrante[varStd > 0 & lagStd > 0] <- 1 # High-High
cuadrante[varStd < 0 & lagStd < 0] <- 2 # Low-Low
cuadrante[varStd < 0 & lagStd > 0] <- 3 # Low-High
cuadrante[varStd > 0 & lagStd < 0] <- 4 # High-Low
cuadrante[lmoran[,5] > significancia] <- 0 # Not significant
```

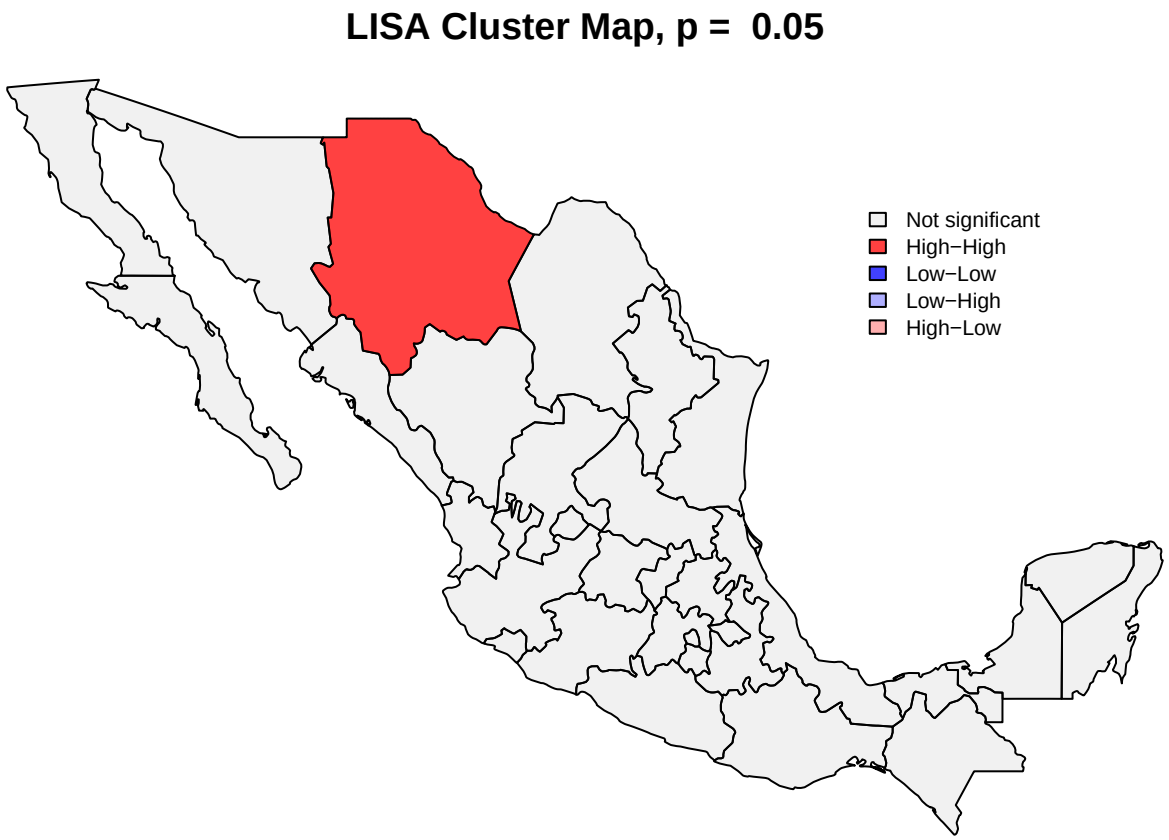
Y definimos los colores que vamos a usar:

```
# Colores para los cúmulos
cColors <- c(rgb(0.74, 0.74, 0.74, alpha = 0.2), rgb(1, 0, 0, alpha = 0.75),
```

```
    rgb(0, 0, 1, alpha = 0.75), rgb(0.58, 0.58, 1, alpha = 0.75), rgb(1, 0.58,
        0.58, alpha = 0.75))
# gris, rojo, azul, azul claro y rojo claro not significant, high-high,
# low-low, low-high, high-low
```

El mapa de cúmulos:

```
# Definir márgenes para ocupar todo el espacio
par(mar = c(0, 0, 1, 0))
# Primer mapa plot not significant
plot(edos[cuadrante == 0, ], col = cColors[1], pch = 16, cex = 0.75)
# plot high-highs
plot(edos[cuadrante == 1, ], col = cColors[2], add = T, pch = 16, cex = 0.75)
# plot low-lows
plot(edos[cuadrante == 2, ], col = cColors[3], add = T, pch = 16, cex = 0.75)
# plot low-highs
plot(edos[cuadrante == 3, ], col = cColors[4], add = T, pch = 16, cex = 0.75)
# plot high-lows
plot(edos[cuadrante == 4, ], col = cColors[5], add = T, pch = 16, cex = 0.75)
legend(-95, 30, legend = c("Not significant", "High-High", "Low-Low", "Low-High",
    "High-Low"), fill = cColors, bty = "n", cex = 0.7, y.intersp = 1, x.intersp = 1)
title(paste("LISA Cluster Map, p = ", significancia))
```



O de una forma más rápida, pintamos todos con una sola instrucción:

```
# Definir márgenes para ocupar todo el espacio
par(mar = c(0, 0, 1, 0))
```

```
intervalos <- c(0, 1, 2, 3, 4)
plot(edos, col = cColors[findInterval(cuadrante, intervalos)])
legend(-95, 30, legend = c("Not significant", "High-High", "Low-Low", "Low-High",
    "High-Low"), fill = cColors, bty = "n", cex = 0.7, y.intersp = 1, x.intersp = 1)
title(paste("LISA Cluster Map, p = ", significancia))
```

Pseudo-significancia y permutaciones

Veamos qué tan confiable es el resultado de la `I` local que obtuvimos. Para esto, podemos hacer permutaciones de las cuentas observadas en cada estado para ver qué tan diferente es de una distribución normal, pero manteniendo las ubicaciones espaciales. Definimos un número de permutaciones a realizar:

```
# Definir el número de simulaciones
sims <- 999
```

Definimos una matriz y un vector para guardar los resultados de las simulaciones.

```
# Definir una matriz y un vector para guardar dos resultados
# La I local simulada
sim.I <- matrix(0, sims, 32)
# Los p-values simulados
sim.p <- matrix(0, 1, 32)
```

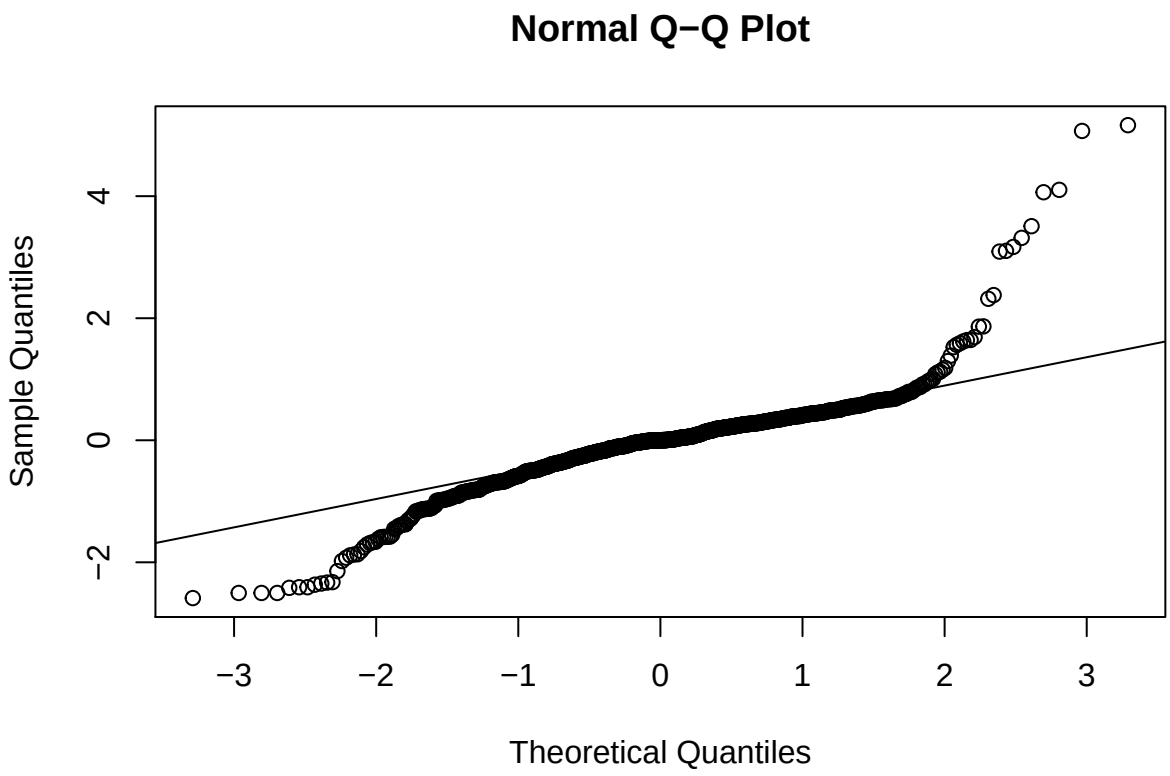
Y encontramos los valores de la `I` local para cada simulación:

```
# Encontrar la I de Moran local para cada simulación
for(i in 1:sims){
```

```
  sim.I[i,] <- localmoran(sample(var), edos.nbq.w)[,1]
}
```

Podemos hacer una gráfica QQ para ver si nuestras simulaciones se parecen, o no, a una distribución normal:

```
# QQ plots for polygon
qqnorm(sim.I[,1])
qqline(sim.I[,1])
```



Queremos comparar `M` simulaciones de los datos conel valor observado y contar cuántas son mayores o menores que el valor local observado. La pseudo-significancia la podemos definir como

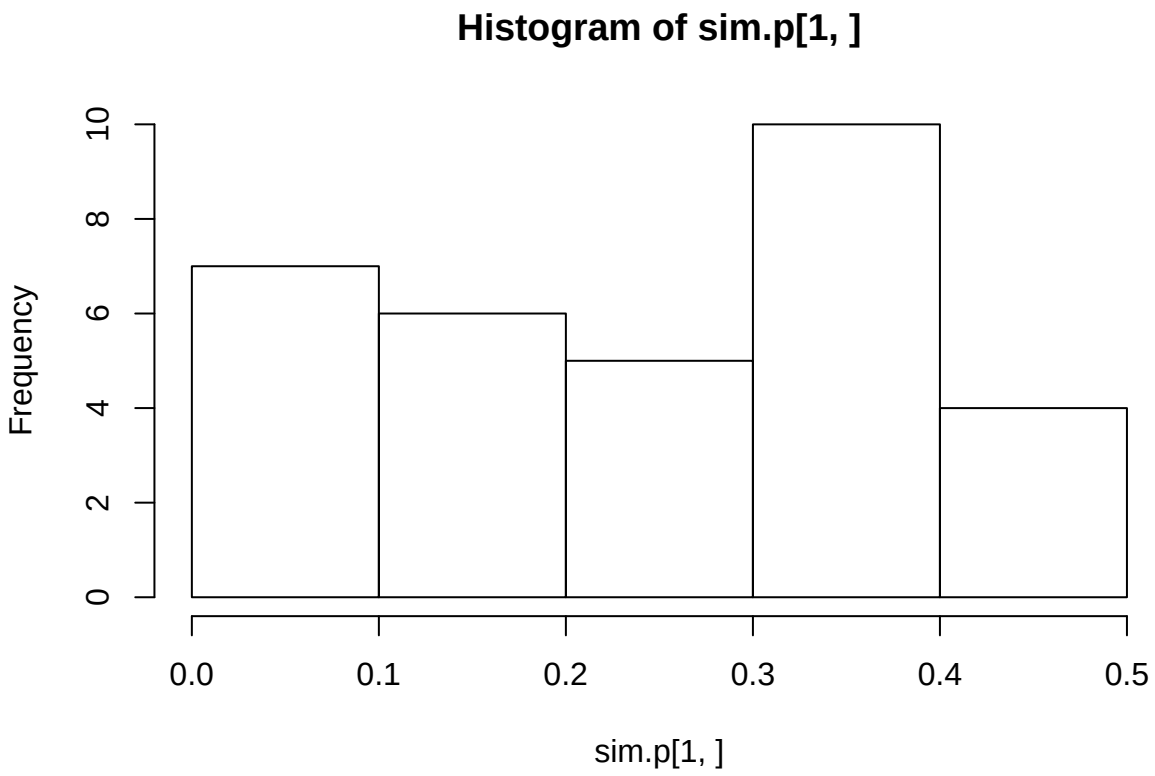
$$p = \frac{M+1}{R+1}$$

donde `M` es el número de veces que la estadística simulada es mayor (menor) o igual que la observada y `R` es el número de simulaciones.

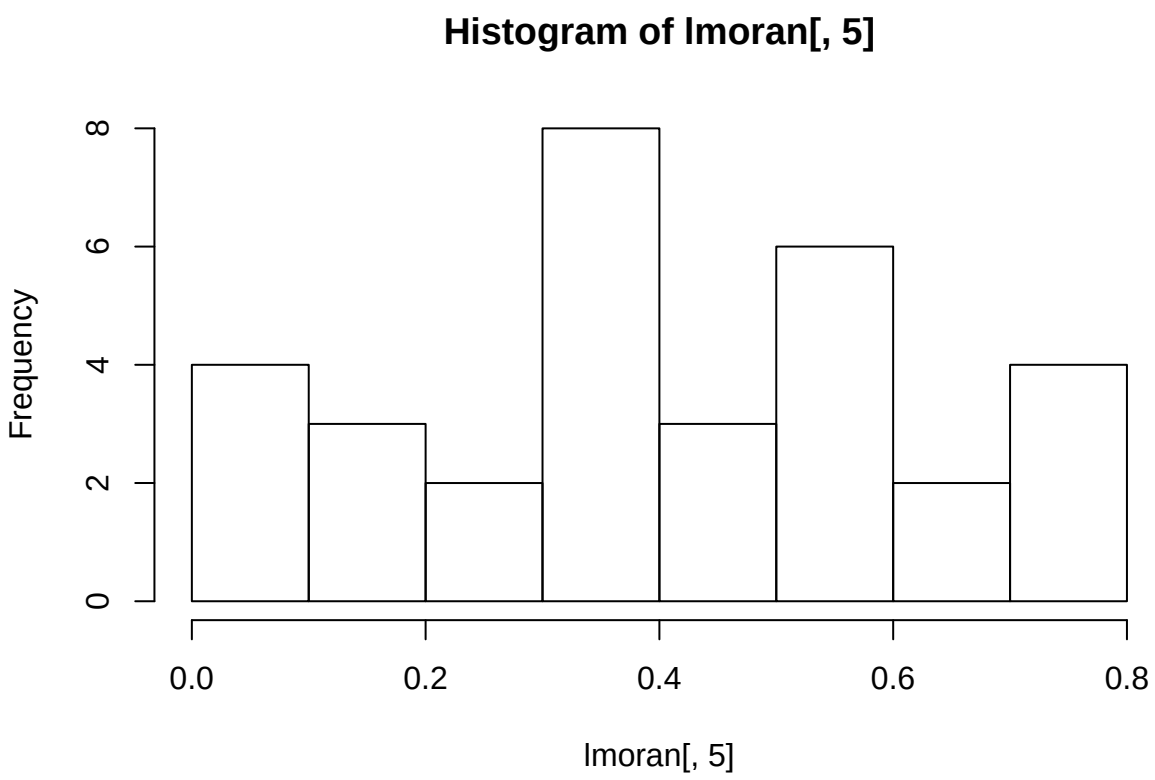
```
# Calcular el valor p definido arriba
for(i in 1:32){
  ifelse(lmoran[i,1]>0, larger <- sim.I[,i]>=lmoran[i,1], larger <- sim.I[,i]<=lmoran[i,1])
  sim.p[i] <- (sum(larger == T)+1)/(sims+1)
}
```

Con esto, podemos comparar la distribución de las significancias para la `I` de Moran observada y las simuladas:

```
# Comparar los histogramas de los p-values
hist(sim.p[1,])
```




```
hist(lmoran[,5])
```



Y por último, podemos hacer un mapa de cúmulos tomando en cuenta estas pseudo-significancias. Como antes, definimos un vector para los cuadrantes de los valores simulados:

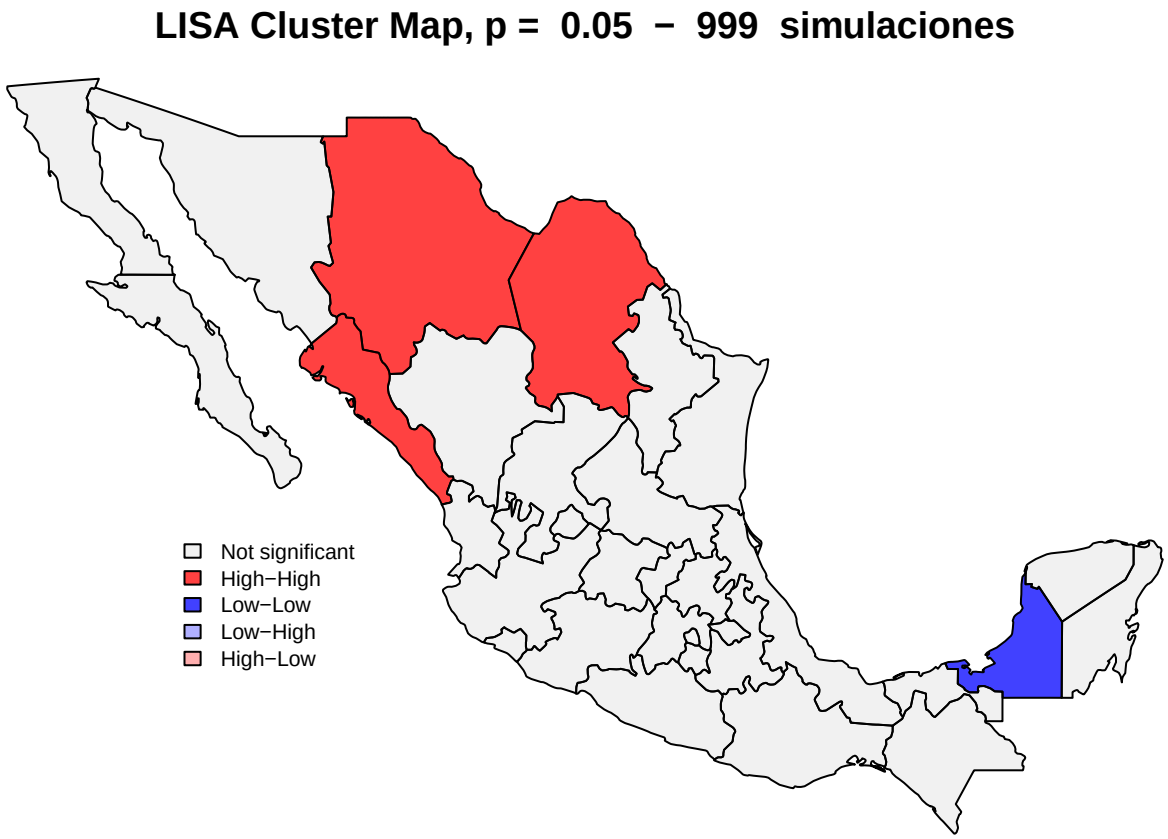
```
# Definir vector de cuadrante del plot de Moran de las simulaciones
cuadrante.sim <- vector(mode = "numeric", length = length(var))
```

Encontramos en cuál cuadrante está cada punto:

```
# Definición de cúmulos
cuadrante.sim[varStd > 0 & lagStd > 0] <- 1 # High-High
cuadrante.sim[varStd < 0 & lagStd < 0] <- 2 # Low-Low
cuadrante.sim[varStd < 0 & lagStd > 0] <- 3 # Low-High
cuadrante.sim[varStd > 0 & lagStd < 0] <- 4 # High-Low
cuadrante.sim[sim.p > significancia] <- 0 # Not significant
```

Y hacemos el mapa de las simulaciones:

```
# Definir márgenes para ocupar todo el espacio
par(mar = c(0, 0, 1, 0))
plot(edos, col = cColors[findInterval(cuadrante.sim, intervalos)])
legend(-113, 22, legend = c("Not significant", "High-High", "Low-Low", "Low-High",
    "High-Low"), fill = cColors, bty = "n", cex = 0.7, y.intersp = 1, x.intersp = 1)
title(paste("LISA Cluster Map, p = ", significancia, " - ", sims, " simulaciones"))
```

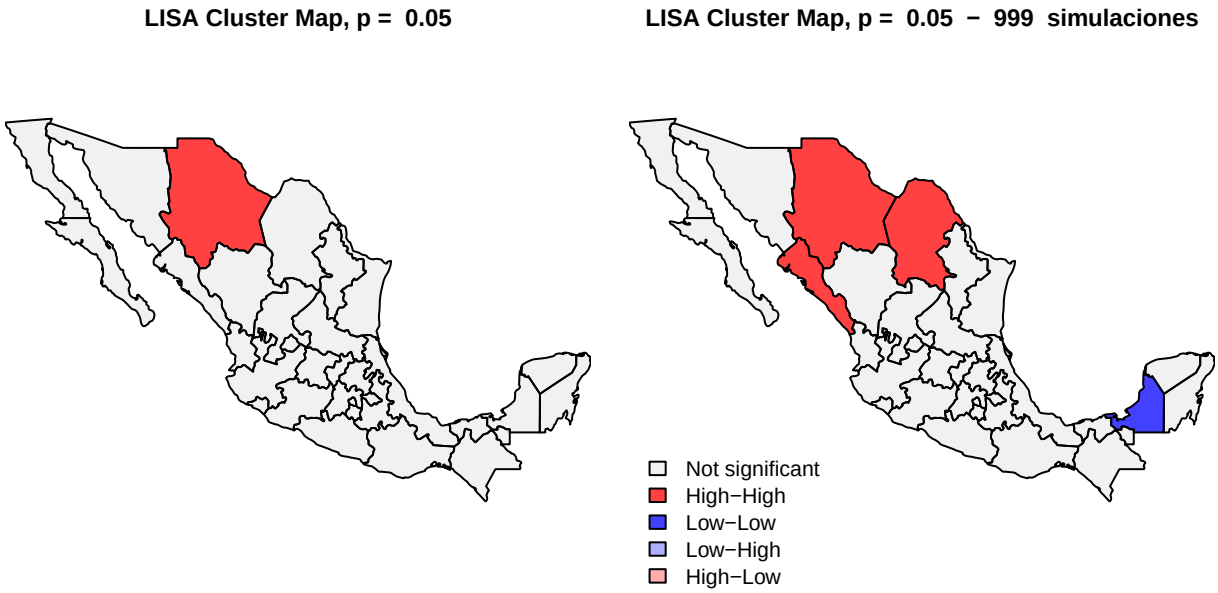


Para comparar los dos mapas de cúmulos, los podemos poner juntos:

```
# Definir que tendremos un gráfica de 1 renglón y dos columnas
par(mfrow = c(1, 2))
# Sin margen arriba, un poco de margen entre cada plot y definir tipo de
# región cuadrada
op <- par(oma = c(0, 0, 0, 0), mar = c(0, 1, 0, 0), pty = "s")

# Primer mapa
plot(edos, col = cColors[findInterval(cuadrante, intervalos)], xaxs = "i", yaxs = "i")
title(paste("LISA Cluster Map, p = ", significancia), cex.main = 0.75)

# Segundo mapa
plot(edos, col = cColors[findInterval(cuadrante.sim, intervalos)], xaxs = "i",
    yaxs = "i")
legend("bottomleft", legend = c("Not significant", "High-High", "Low-Low", "Low-High",
    "High-Low"), fill = cColors, bty = "n", cex = 0.7, y.intersp = 1, x.intersp = 1)
title(paste("LISA Cluster Map, p = ", significancia, " - ", sims, " simulaciones"),
    cex.main = 0.75)
```




```
# Regresar a una sola gráfica en toda la página
par(mfrow = c(1,1))
```