1 A Front End For the Rcartogram Package

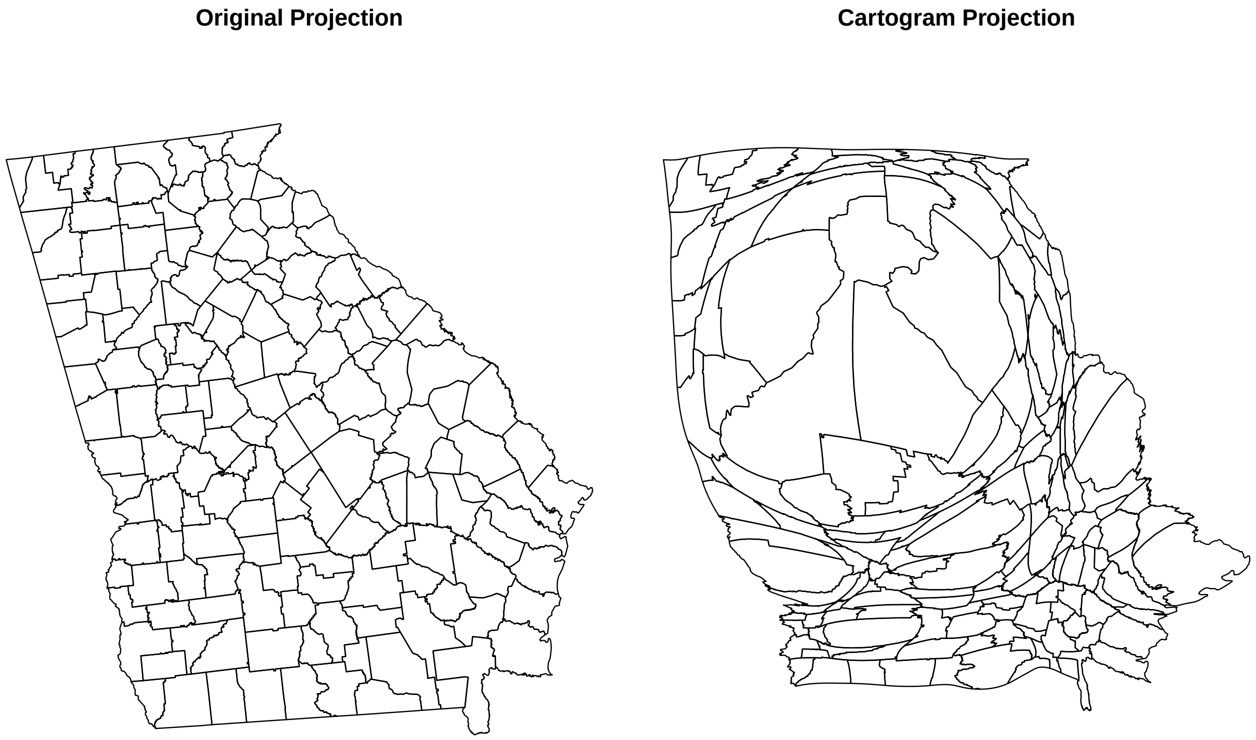
This demonstrates the *getcartr* package. Firstly, make sure you have the `Rcartogram` package installed. This can be downloaded from http://www.omegahat.org/Rcartogram/ . Next - if you haven't already done so, install this package from github - make sure `devtools` is installed in R, then enter `library(devtools);install_github('chrisbrunsdon/getcartr',subdir='getcartr')`. Now you are ready to use the package:

Now load the package:

```r
library(getcartr)
```

`quick.carto` gives you a cartogram very quickly:

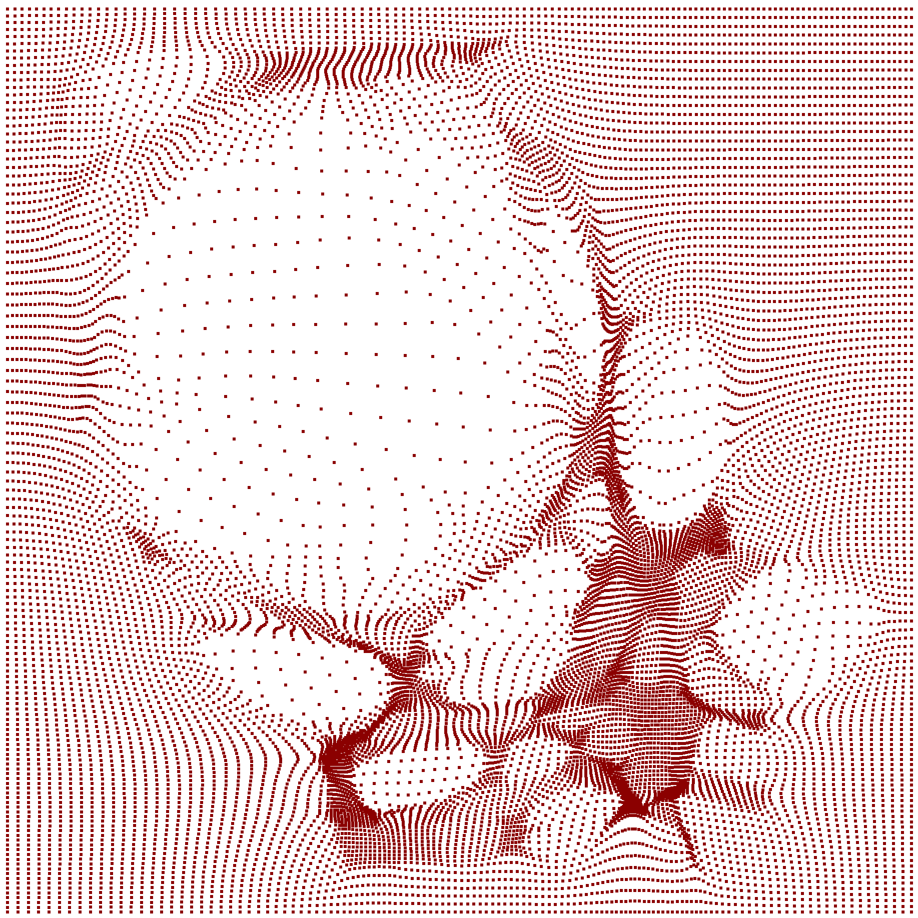
```r
data(georgia)
georgia.carto <- quick.carto(georgia,georgia2$TotPop90, blur=1)
par(mfrow=c(1,2),mar=c(0.5,0.5,3,0.5))
plot(georgia2)
title("Original Projection")
plot(georgia.carto)
title("Cartogram Projection")
```



and plot the transformation mesh:

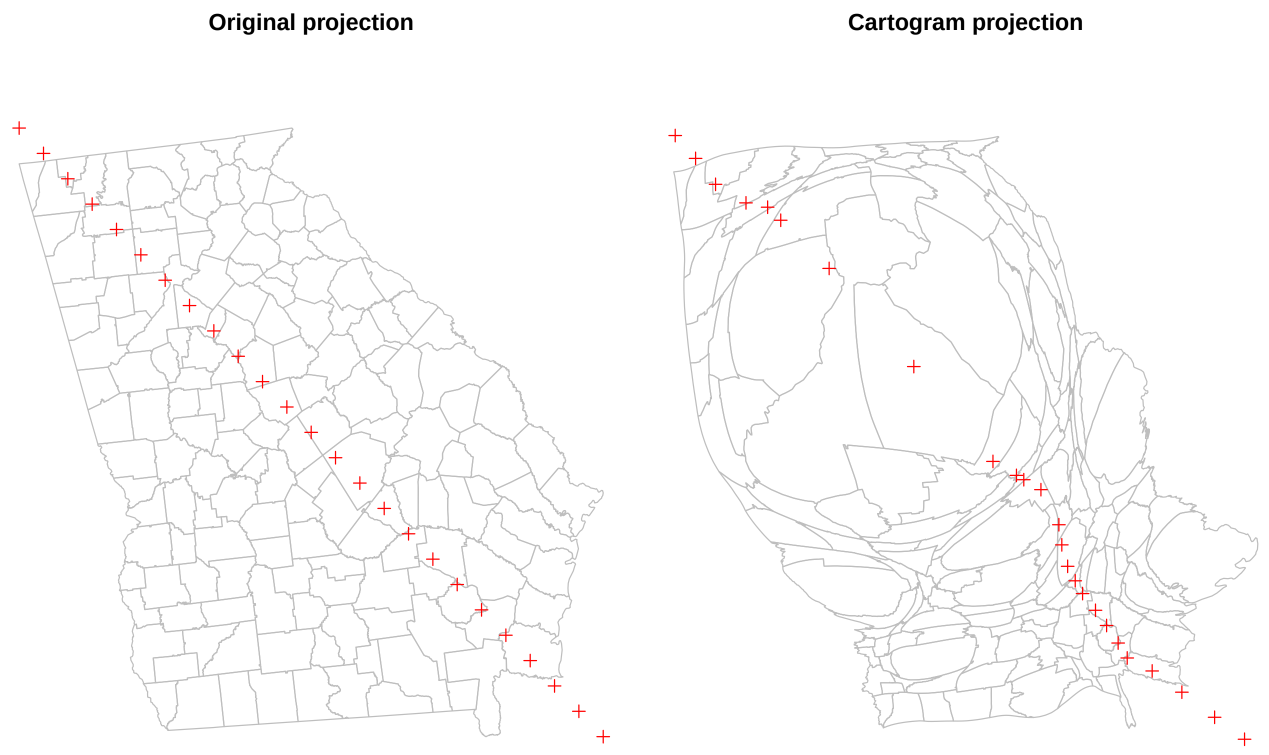
```r
# Load an example SpatialPolygonsDataFrame from GISTools
data(georgia)
# Create the cartogram.  TotPop90 contains 1990 county populations
# "georgia2" is used as it has a plane projection coordinate system
```

```
georgia.carto <- quick.carto(georgia2,georgia2$TotPop90)
# Draw the dispersion plot
dispersion(georgia.carto, col='darkred')
```



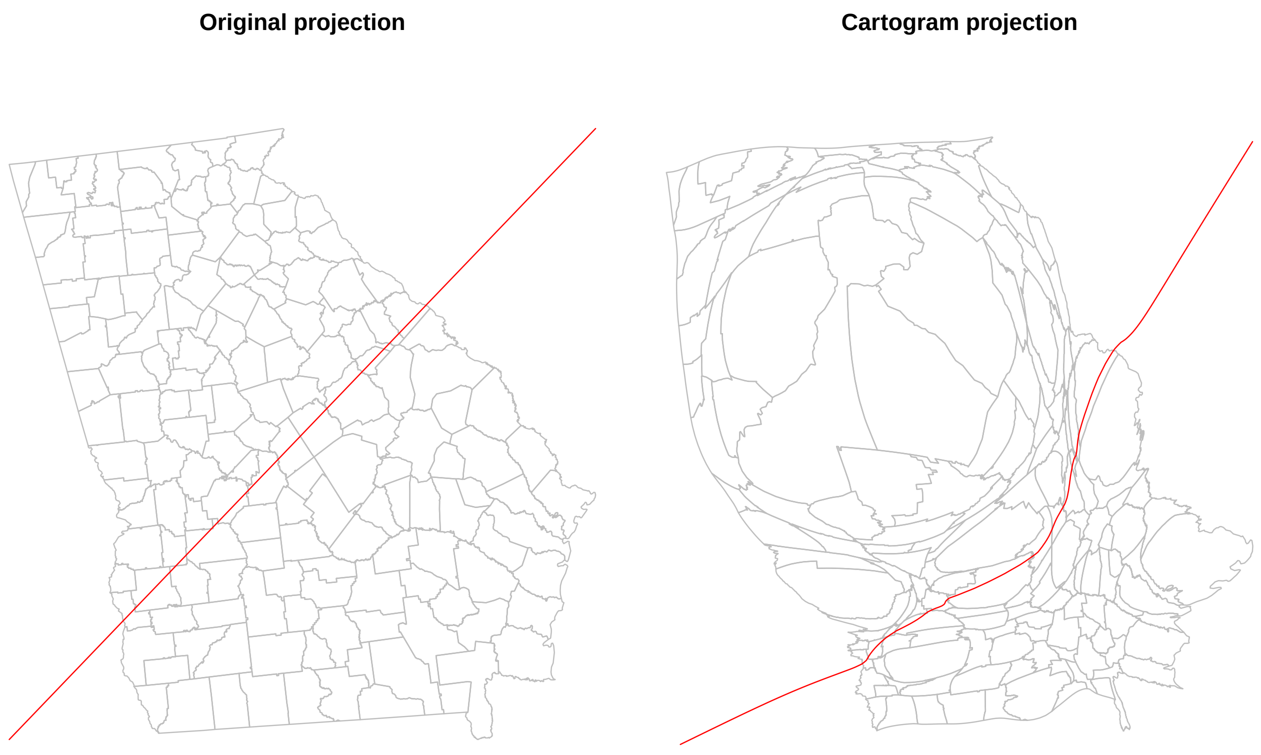
and apply the cartogram transform to new sets of points:

```
# Create a set of 25 points from northwest to southeast corners
xl <- seq(1419424,939220.7,l=25)
yl <- seq(905508,1405900,l=25)
pset <- readWKT(paste("MULTIPOINT(",paste(apply(cbind(xl,yl),1,function(x) paste("(",x[1],x[2],")")),col
#' #  Transform it
pset.carto <- warp.points(pset,georgia.carto)

# Plot the original points
par(mfrow=c(1,2),mar=c(0.5,0.5,3,0.5))
plot(georgia2,border='grey'); plot(pset, add=T, col='red')
title('Original projection')

# Plot in cartogram space
plot(georgia.carto,border='grey'); plot(pset.carto, add=T, col='red')
title('Cartogram projection')
```



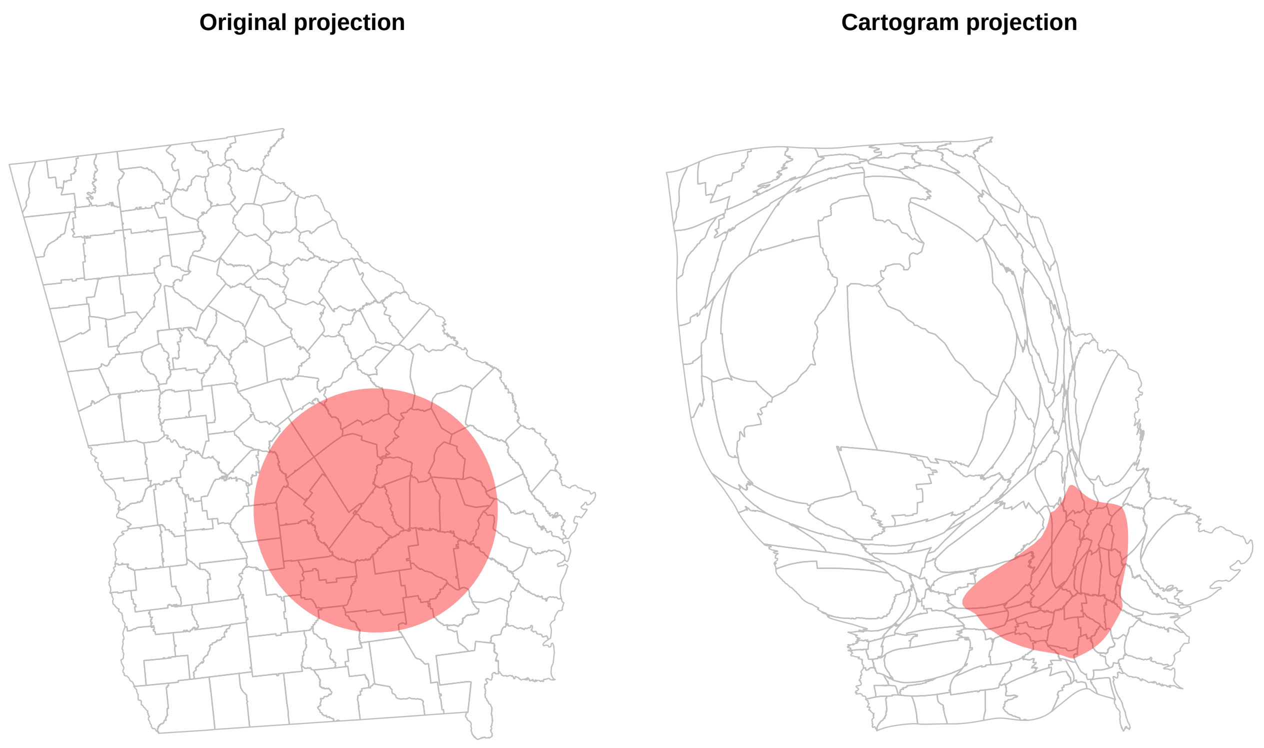
or lines:

```
# Create a line from southwest to northeast corners,  with 100 segments
xl <- seq(939220.7,1419424,l=100)
yl <- seq(905508,1405900,l=100)
aline <- readWKT(paste("LINESTRING(",paste(apply(cbind(xl,yl),1,function(x) paste(x[1],x[2])),collapse=

#  Transform it
aline.carto <- warp.lines(aline,georgia.carto)

# Plot the original line
par(mfrow=c(1,2),mar=c(0.5,0.5,3,0.5))
plot(georgia2,border='grey'); plot(aline, add=T, col='red')
title('Original projection')

# Plot in cartogram space
plot(georgia.carto,border='grey'); plot(aline.carto, add=T, col='red')
title('Cartogram projection')
```



. . . or areas.

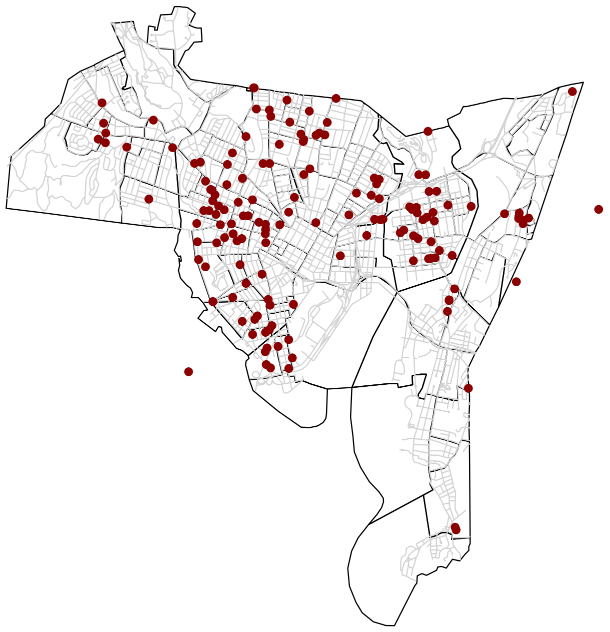
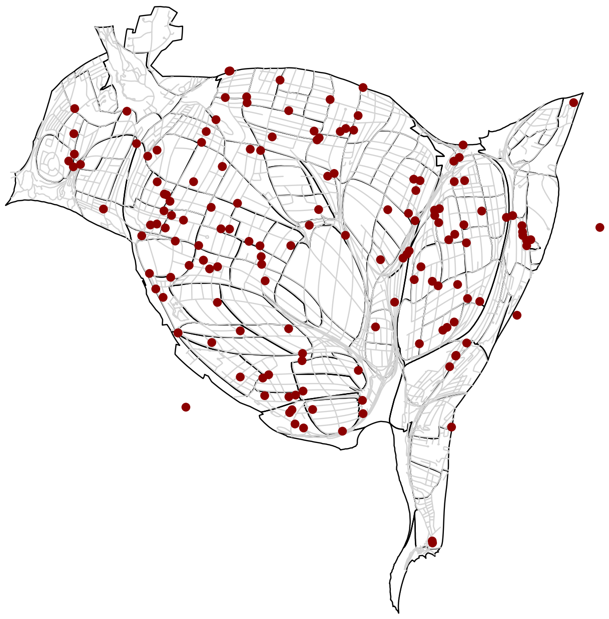
```
# Create a pseudocircle with 100 segments
xl <- cos(seq(0,2*pi,l=100))*100000 + 1239348
yl <- sin(seq(0,2*pi,l=100))*100000 + 1093155
circ <- readWKT(paste("MULTIPOLYGON(((",paste(apply(cbind(xl,yl),1,function(x) sprintf("%7.0f %7.0f",x[
#' #  Transform it
circ.carto <- warp.polys(circ,georgia.carto)

# Plot the original circle
par(mfrow=c(1,2),mar=c(0.5,0.5,3,0.5))
plot(georgia2,border='grey'); plot(circ, add=T, col=rgb(1,0,0,0.4), border=NA)
title('Original projection')

# Plot in cartogram space
plot(georgia.carto,border='grey'); plot(circ.carto, add=T, col=rgb(1,0,0,0.4), border=NA)
title('Cartogram projection')
```



2 Alternative approach

Use the `carto.transform` function to create a new *function* to apply the cartogram transform to any shapefile. This function automatically determines whether the input object is `SpatialPolygons*`, `SpatialLines*` or `SpatialPoints*`. Here the Newhaven data in `GISTools` will be used:

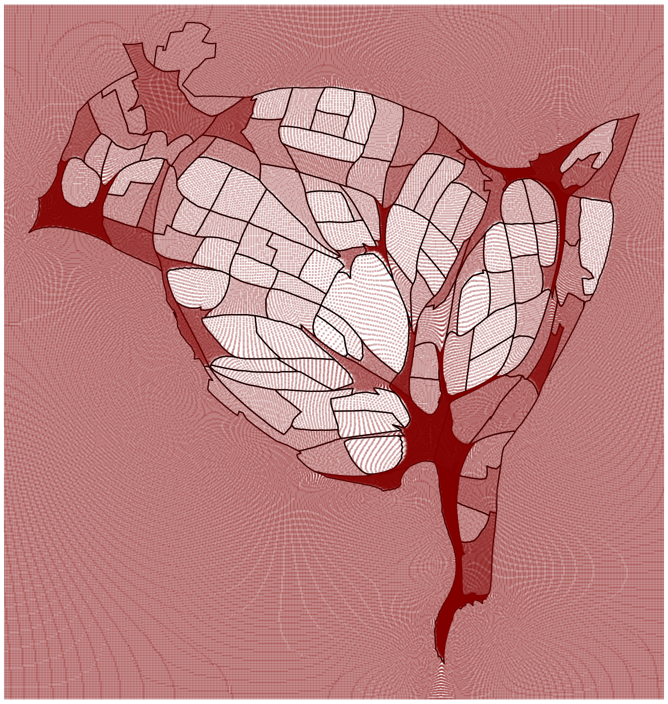
```
# Get the newhaven data
require(GISTools)
data(newhaven)
# Plot the untransformed data
par(mfrow=c(1,2),mar=c(0.5,0.5,3,0.5))
plot(blocks) # Census blocks
plot(roads,col='lightgrey',add=TRUE) # Roads
plot(burgres.f,col='darkred',pch=16,add=TRUE) # Forced entry burglaries

# Create the cartogram transform function
# The 'res' here gives the resolution of the cartogram interpolation grid
# - default is 128,  so here we have 4 times that resolution. Slower but
# more accurate...
to.carto <- carto.transform(blocks,blocks$POP1990,res=512)

# Now create a cartogram version of the map
# Plot the blocks carto
plot(to.carto(blocks))
# Add roads,  transformed to cartogram space
plot(to.carto(roads),add=TRUE,col='lightgrey')
# Add forced entry residential burglaries, transformed to cartogram space
plot(to.carto(burgres.f),add=TRUE,col='darkred',pch=16)
```

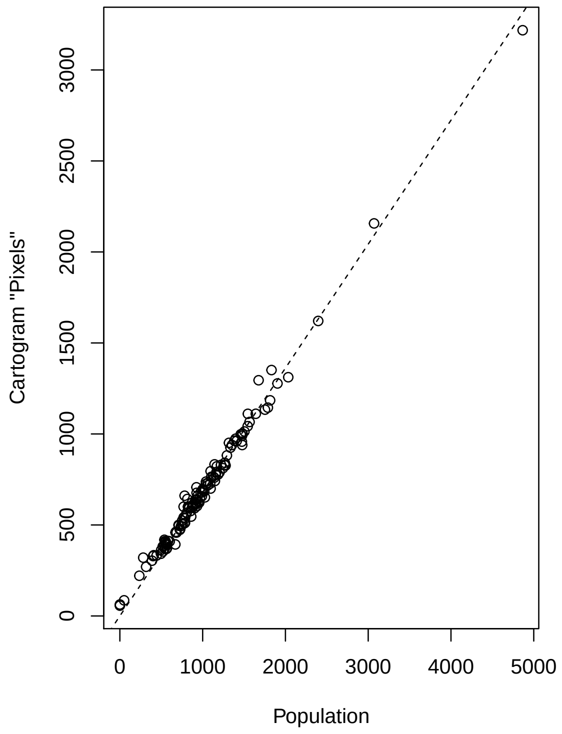



3 Diagnostics

Next, a few visual diagnostics. Firstly, superimpose the dispersion diagram on the cartogram itself. Secondly, plot the cartogram variable against the actual area of the corresponding polygon in the cartogram. Ideally these should lie on a straight line going through the origin.

```
# Superimpose dispersion diagram on cartogram:
par(mfrow=c(1,2),mar=c(0.5,0.5,3,0.5))
blocks.cart <- to.carto(blocks)
plot(blocks.cart)
dispersion(blocks.cart,col=rgb(0.5,0,0,0.3),add=TRUE)
# Scatterplot the cartogram variable against actual zone area
par(mar=c(5,5,5,5))
cartvar <- blocks$POP1990
cartarea <- poly.areas(blocks.cart)
plot(cartvar,cartarea,
     xlab='Population',ylab='Cartogram "Pixels"')
abline(lm(cartarea~0+cartvar),lty=2)
```





Finally, a numerical diagnostic also exists. Based on the idea that, from the definition, an ideal cartogram transform will create a set of zones whose area is proportional to the chosen statistical variable associated with each zone. Thus, as stated above, these two quantities, when plotted, should lie on a straight line passing through the origin. Thus, the R^2 statistic obtained when regressing one on the other **without an intercept** should be equal to one. If x_i is the area of zone i in cartogram space, and y_i is the associated value of the cartogram variable, then the estimated slope of the line, $\hat{s}$ is given by

$$\hat{s} = \frac{\sum_i x_i y_i}{\sum x_i^2}$$

and then

$$R^2 = \frac{\sum_i (\hat{s}x_i)^2}{\sum_i (\hat{s}x_i)^2 + \sum_i (y_i - \hat{s}x_i)^2}$$

The `cart.r2` function computes this quantity:

```
# Check out the r-squared for the cartogram just made...
cart.r2(blocks.cart,blocks$POP1990)
```

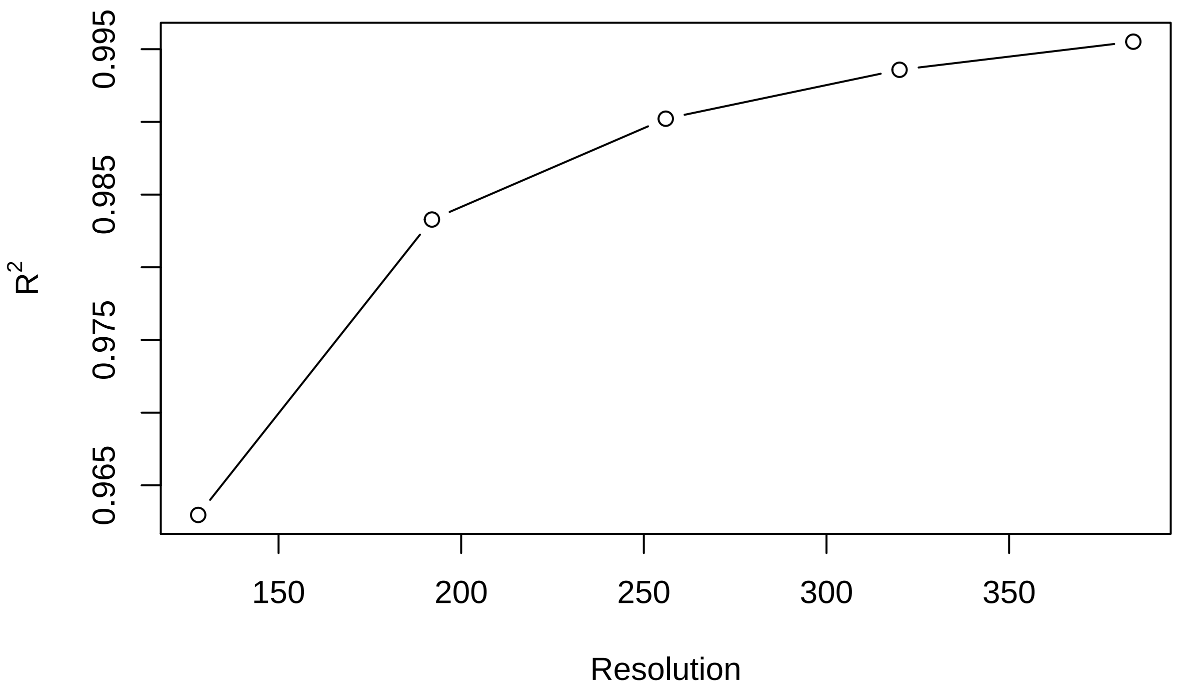
```
## [1] 0.9974218
```

```
# Now try it for different resolutions
result = NULL
res.list <- c(128,192,256,320,384)
for (r in res.list) result <- c(result,
    cart.r2(quick.carto(blocks,res=r,blocks$POP1990,thresh=0.1),blocks$POP1990))
result <- cbind(res.list,result)
result
```

```
##      res.list    result
## [1,]      128 0.9629652
## [2,]      192 0.9832846
## [3,]      256 0.9902184
## [4,]      320 0.9935837
## [5,]      384 0.9955160
```

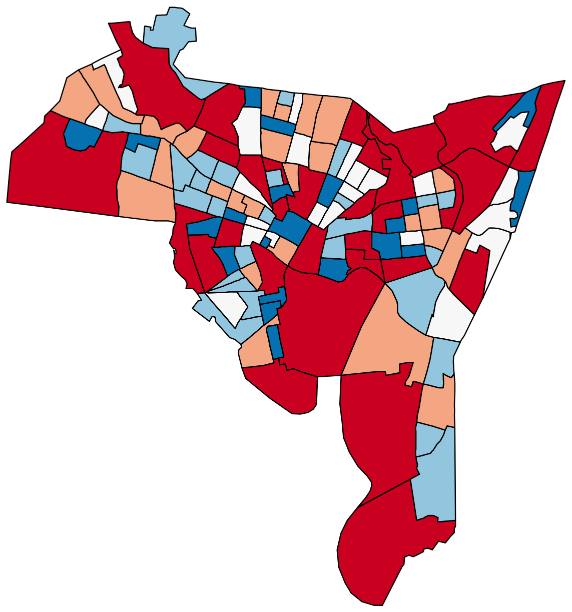
and show this as a graph

```
plot(result,xlab='Resolution',ylab=expression(R^2),type='b')
```

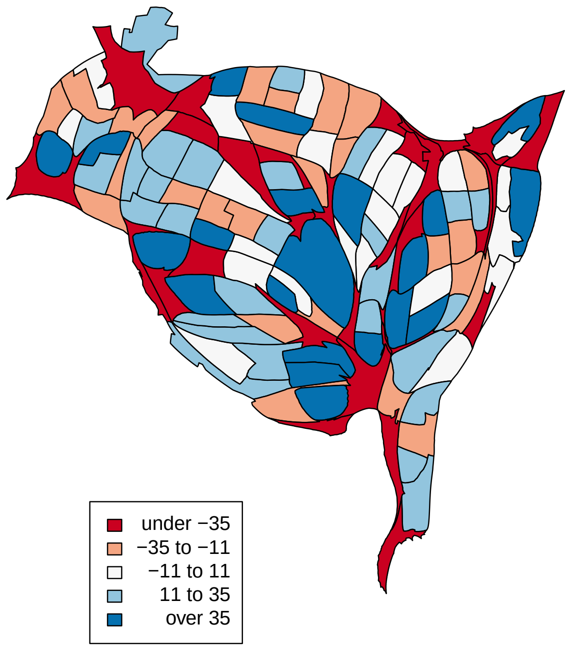


The straight line fitted through the origin can also be used as a basis to identify whether areas are under or over represented, by considering the residual value $y_i - \hat{s}x_i$ for each zone i. Positive values of this suggest under representation, and negative values suggest over representation (although the R^2 figures above suggest the degree of error is very small). These residuals can be computed using the `cart.resids` function, and then mapped. Here they are mapped in both cartogram space and the standard map projection.

```
par(mfrow=c(1,2),mar=c(1,1,1,1))
resids <- cart.resids(blocks.cart,blocks$POP1990)
shades <- auto.shading(c(resids,-resids),n=5,col=brewer.pal(5,'RdBu'))
choropleth(blocks,resids,shading=shades)
title('Discrepancy (Cartogram Pixels) - Standard Projection')
choropleth(blocks.cart,resids,shading=shades)
choro.legend(100,100,shades)
title('Discrepancy (Cartogram Pixels) - Cartogram Projection')
```



Discrepancy (Cartogram Pixels) − Standard Projection



Discrepancy (Cartogram Pixels) − Cartogram Projection