Chapter 6 Analysing spatial patterns

6.1 Learning outcomes

By the end of this practical you should be able to:

1. Describe and evaluate methods for analysing spatial patterns
2. Execute data cleaning and manipulation appropairte for analysis
3. Determine the locations of spatial clusters using point pattern analysis methods
4. Investigate the degree to which values at spatial points are similar (or different) to each other

6.2 Recommended listening

Some of these practicals are long, take regular breaks and have a listen to some of our fav tunes each week.

[Andy](https://www.youtube.com/watch?v=hlVBg7_08n0) [Adam](<https://open.spotify.com/album/4OCAtcMneSzQKZGJK95ayo?si=YRtNrsspTF6XpTPRjv2XFg> This week it’s the head honcho himself, the man, the legend that is Tony Colman, CEO and founder of Hospital Records — his new album Building Better Worlds is a masterpiece! Enjoy!

6.3 Introduction

In this practical you will learn how to begin to analyse patterns in spatial data. Using data you are already familiar with, in the first part of the practical, you will explore some techniques for analysing patterns of point data in R. Then, in the second part of the practial, you will explore spatial autocorrelation using R or ArcGIS…or both if you wish.

In this analysis we will analyse the patterns of Blue Plaques — you will see these placed on around the UK linking the buildings of the present to people of the past.

The question we want to answer is: **“For any given London Borough, are the Blue Plaques within that borough distributed randomly or do they exhibit some kind of dispersed or clustered pattern?”**

To answer this question, we will make use of some of the Point Pattern Analysis functions found in the spatstat package.

*#first library a few packages that we will use during the practical*

*#note you may need to install them first...*

**library**(spatstat)

**library**(sp)

**library**(rgeos)

**library**(maptools)

**library**(GISTools)

**library**(tmap)

**library**(sf)

**library**(geojson)

**library**(geojsonio)

**library**(tmaptools)

6.4 Setting up your data

Now, assuming that you’ve got a copy of your London Boroughs shapefile (from week 1) in your new week 6 folder, along with a shapefile of your Blue Plaques. If not.. read in the data from [here](http://geoportal.statistics.gov.uk/datasets/8edafbe3276d4b56aec60991cbddda50_2)

*##First, get the London Borough Boundaries*

EW <- **geojson\_read**(

"https://opendata.arcgis.com/datasets/8edafbe3276d4b56aec60991cbddda50\_4.geojson",

what = "sp")

Pull out london using grep and the regex wildcard for’start of the string’ (^) to to look for the bit of the district code that relates to London (E09) from the ‘lad15cd’ column in the data slot of our spatial polygons dataframe

BoroughMap <- EW[**grep**("^E09",EW@data$lad15cd),]

*#plot it using the base plot function*

**qtm**(BoroughMap)

**summary**(BoroughMap)

## Object of class SpatialPolygonsDataFrame

## Coordinates:

## min max

## x -0.5097014 0.3340243

## y 51.2867587 51.6918756

## Is projected: FALSE

## proj4string :

## [+proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0]

## Data attributes:

## objectid lad15cd lad15nm lad15nmw

## Min. :294 E09000001: 1 Barking and Dagenham: 1 :33

## 1st Qu.:302 E09000002: 1 Barnet : 1 Abertawe : 0

## Median :310 E09000003: 1 Bexley : 1 Blaenau Gwent: 0

## Mean :310 E09000004: 1 Brent : 1 Bro Morgannwg: 0

## 3rd Qu.:318 E09000005: 1 Bromley : 1 Caerdydd : 0

## Max. :326 E09000006: 1 Camden : 1 Caerffili : 0

## (Other) :27 (Other) :27 (Other) : 0

## st\_areashape st\_lengthshape

## Min. : 2680711 Min. : 6911

## 1st Qu.: 27179391 1st Qu.:25477

## Median : 38007465 Median :31190

## Mean : 48044245 Mean :32832

## 3rd Qu.: 57363951 3rd Qu.:37693

## Max. :150198167 Max. :63033

##

BNG = "+init=epsg:27700"

BoroughMapBNG <- **spTransform**(BoroughMap,BNG)

Now get the location of all Blue Plaques in the City

*##Now get the location of all Blue Plaques in the City*

BluePlaques <- **geojson\_read**(

"https://s3.eu-west-2.amazonaws.com/openplaques/open-plaques-london-2018-04-08.geojson",

what = "sp")

**summary**(BluePlaques)

## Object of class SpatialPointsDataFrame

## Coordinates:

## min max

## coords.x1 -0.477 0.21903

## coords.x2 0.000 51.67830

## Is projected: FALSE

## proj4string :

## [+proj=longlat +datum=WGS84 +no\_defs +ellps=WGS84 +towgs84=0,0,0]

## Number of points: 2812

## Data attributes:

## id

## Min. : 1.0

## 1st Qu.: 711.8

## Median : 6089.0

## Mean :10622.0

## 3rd Qu.:10358.2

## Max. :49190.0

##

## inscription

## Frank Matcham (1854-1920) theatre architect designed this theatre : 3

## Charlie Chester MBE : 2

## Grimâ\200\231s Dyke : 2

## Lillie Langtry 1852-1929 actress lived here : 2

## 'Canons' gate pillars\r\nEntrance gate piers to 'Canons' from Old Turnpike Road\r\nBuilt in 1712 for James Brydges 1673-1744, 1st Duke of Chandos. House demolished 1740 and existing buildings constructed from salvaged materials. Piers refurbished 1998.: 1

## 'Father' Henry Willis 1821-1901 organ builder lived here : 1

## (Other) :2801

*#now set up an EPSG string to help set the projection*

BNG = "+init=epsg:27700"

WGS = "+init=epsg:4326"

BluePlaquesBNG <- **spTransform**(BluePlaques, BNG)

**summary**(BluePlaquesBNG)

## Object of class SpatialPointsDataFrame

## Coordinates:

## min max

## coords.x1 505575.2 622211.7

## coords.x2 -5527598.3 199509.2

## Is projected: TRUE

## proj4string :

## [+init=epsg:27700 +proj=tmerc +lat\_0=49 +lon\_0=-2 +k=0.9996012717

## +x\_0=400000 +y\_0=-100000 +datum=OSGB36 +units=m +no\_defs

## +ellps=airy

## +towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894]

## Number of points: 2812

## Data attributes:

## id

## Min. : 1.0

## 1st Qu.: 711.8

## Median : 6089.0

## Mean :10622.0

## 3rd Qu.:10358.2

## Max. :49190.0

##

## inscription

## Frank Matcham (1854-1920) theatre architect designed this theatre : 3

## Charlie Chester MBE : 2

## Grimâ\200\231s Dyke : 2

## Lillie Langtry 1852-1929 actress lived here : 2

## 'Canons' gate pillars\r\nEntrance gate piers to 'Canons' from Old Turnpike Road\r\nBuilt in 1712 for James Brydges 1673-1744, 1st Duke of Chandos. House demolished 1740 and existing buildings constructed from salvaged materials. Piers refurbished 1998.: 1

## 'Father' Henry Willis 1821-1901 organ builder lived here : 1

## (Other) :2801

*#plot the blue plaques in the city*

**tmap\_mode**("view")

**tm\_shape**(BoroughMapBNG) +

**tm\_polygons**(col = NA, alpha = 0.5) +

**tm\_shape**(BluePlaquesBNG) +

**tm\_dots**(col = "blue")

6.4.1 Data cleaning

Now, you might have noticed that there is at least one Blue Plaque that falls outside of the Borough boundaries. Errant plaques will cause problems with our analysis, so we need to clip the plaques to the boundaries…First we’ll remove any Plaques with the same grid reference as this will cause problems later on in the analysis..

*#remove duplicates*

BluePlaquesBNG <- **remove.duplicates**(BluePlaquesBNG)

Now just select the points inside London - thanks to Robin Lovelace for posting how to do this one, very useful!

BluePlaquesSub <- BluePlaquesBNG[BoroughMapBNG,]

*#check to see that they've been removed*

**tmap\_mode**("view")

**tm\_shape**(BoroughMapBNG) +

**tm\_polygons**(col = NA, alpha = 0.5) +

**tm\_shape**(BluePlaquesSub) +

**tm\_dots**(col = "blue")

6.4.2 Study area

From this point, we could try and carry out our analysis on the whole of London, but you might be waiting until next week for Ripley’s K to be calculated for this many points. Therefore to speed things up and to enable us to compare areas within London, we will select some individual boroughs. First we need to subset our SpatialPolygonsDataFrame to pull out a borough we are interested in. I’m going to choose Harrow as I know there are few enough points for the analysis to definitely work. If you wish, feel free to choose another borough in London and run the same analysis, but beware that if it happens that there are a lot of blue plaques in your borough, the analysis could fall over!!

*#extract the borough*

Borough <- BoroughMapBNG[BoroughMapBNG@data$lad15nm=="Harrow",]

*#or as an sf object:*

BoroughMapBNGSF <- **st\_as\_sf**(BoroughMapBNG)

BoroughSF <- BoroughMapBNGSF[BoroughMapBNGSF$lad15nm=="Harrow",]

*#Check to see that the correct borough has been pulled out*

**tm\_shape**(Borough) +

**tm\_polygons**(col = NA, alpha = 0.5)

Next we need to clip our Blue Plaques so that we have a subset of just those that fall within the borough or interest

*#clip the data to our single borough*

BluePlaquesSub <- BluePlaquesBNG[Borough,]

*#check that it's worked*

**tmap\_mode**("view")

## tmap mode set to interactive viewing

**tm\_shape**(Borough) +

**tm\_polygons**(col = NA, alpha = 0.5) +

**tm\_shape**(BluePlaquesSub) +

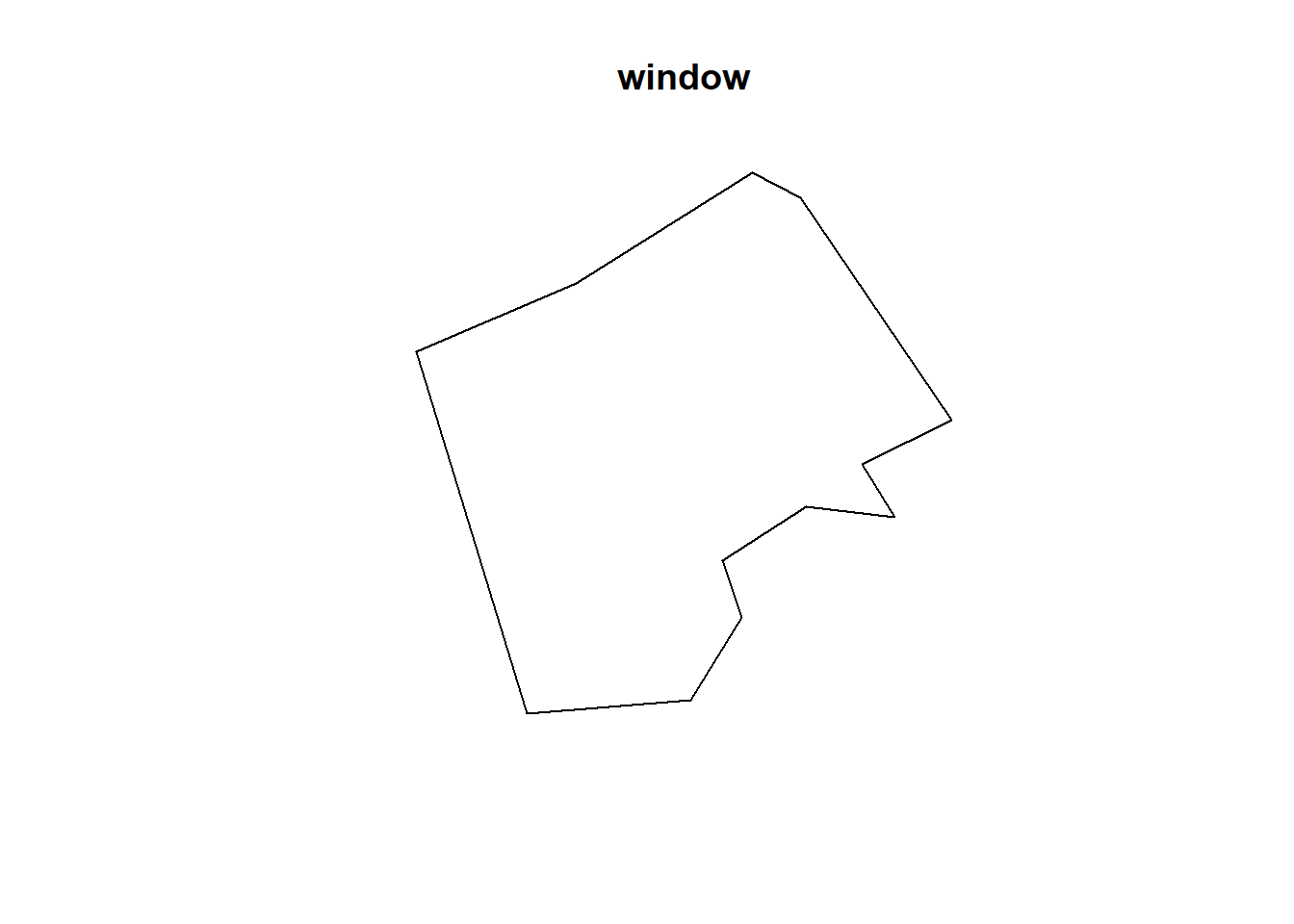
**tm\_dots**(col = "blue")

We now have all of our data set up so that we can start the analysis using spatstat. The first thing we need to do is create an observation window for spatstat to carry out its analysis within — we’ll set this to the extent of the Harrow boundary

*#now set a window as the borough boundary*

window <- **as.owin**(Borough)

**plot**(window)



spatstat has its own set of spatial objects that it works with (one of the delights of R is that different packages are written by different people and many have developed their own data types) — it does not work directly with the SpatialPolygonsDataFrames, SpatialPointsDataFrames or sf objects that we are used to. For point pattern analysis, we need to create a point pattern (ppp) object.

*#create a ppp object*

BluePlaquesSub.ppp <- **ppp**(x=BluePlaquesSub@coords[,1],y=BluePlaquesSub@coords[,2],window=window)

Try to understand what the different elements in command above is doing. If you are unsure, you can run elements of the code, for example:

BluePlaquesSub@coords[,1]

## [1] 514971.1 512466.7 514966.0 517339.4 512215.1 515694.1 512269.5

## [8] 511792.6 515333.7 518598.3 515370.2 512335.7 511539.4 513371.5

## [15] 516746.3 515210.7 515093.2 515561.8 514805.8 513053.6 515166.2

## [22] 513750.4 512508.2 516451.0 514022.4 518187.5 516725.5 513392.3

## [29] 513008.4 517770.4 514177.1 519585.7 515300.0 514183.4 518560.0

## [36] 512639.2 515491.4 514789.8 519099.9 512346.3 512343.2

Have a look at the new ppp object

**plot**(BluePlaquesSub.ppp,pch=16,cex=0.5, main="Blue Plaques Harrow")

A close up of a logo

Description automatically generated

6.5 Point pattern analysis

6.5.1 Kernel Density Estimation

One way to summarise your point data is to plot the density of your points under a window called a ‘Kernel’. The size and shape of the Kernel affects the density pattern produced, but it is very easy to produce a Kernel Density Estimation (KDE) map from a ppp object using the density function.

**plot**(**density**(BluePlaquesSub.ppp, sigma = 500))

A close up of a logo

Description automatically generated

The sigma value sets the diameter of the Kernel (in the units your map is in — in this case, as we are in British National Grid the units are in metres). Try experimenting with different values of sigma to see how that affects the density estimate.

**plot**(**density**(BluePlaquesSub.ppp, sigma = 1000))

A close up of a logo

Description automatically generated

6.5.2 Quadrat Analysis

So as you saw in the lecture, we are interesting in knowing whether the distribution of points in our study area differs from ‘complete spatial randomness’ — CSR. That’s different from a CRS! Be careful!

The most basic test of CSR is a quadrat analysis. We can carry out a simple quadrat analysis on our data using the quadrat count function in spatstat. *Note, I wouldn’t recommend doing a quadrat analysis in any real piece of analysis you conduct, but it is useful for starting to understand the Poisson distribution…*

*#First plot the points*

**plot**(BluePlaquesSub.ppp,pch=16,cex=0.5, main="Blue Plaques in Harrow")

*#now count the points in that fall in a 6 x 6 grid overlaid across the window*

**plot**(**quadratcount**(BluePlaquesSub.ppp, nx = 6, ny = 6),add=T,col="red")

A close up of a map

Description automatically generated

In our case here, want to know whether or not there is any kind of spatial patterning associated with the Blue Plaques in areas of London. If you recall from the lecture, this means comparing our observed distribution of points with a statistically likely (Complete Spatial Random) distibution, based on the Poisson distribution.

Using the same quadratcount function again (for the same sized grid) we can save the results into a table:

*#run the quadrat count*

Qcount<-**data.frame**(**quadratcount**(BluePlaquesSub.ppp, nx = 6, ny = 6))

*#put the results into a data frame*

QCountTable <- **data.frame**(**table**(Qcount$Freq, exclude=NULL))

*#view the data frame*

QCountTable

## Var1 Freq

## 1 0 11

## 2 1 9

## 3 2 3

## 4 3 2

## 5 4 2

## 6 5 1

## 7 7 1

*#we don't need the last row, so remove it*

QCountTable <- QCountTable[-**nrow**(QCountTable),]

Check the data type in the first column - if it is factor, we will need to convert it to numeric

**class**(QCountTable[,1])

## [1] "factor"

*#oops, looks like it's a factor, so we need to convert it to numeric*

vect<- **as.numeric**(**levels**(QCountTable[,1]))

vect <- vect[1:6]

QCountTable[,1] <- vect

OK, so we now have a frequency table — next we need to calculate our expected values. The formula for calculating expected probabilities based on the Poisson distribution is:

\[Pr= (X =k) = \frac{\lambda^{k}e^{-\lambda}}{k!}\]

*#calculate the total blue plaques (Var \* Freq)*

QCountTable$total <- QCountTable[,1]\*QCountTable[,2]

*#calculate mean*

sums <- **colSums**(QCountTable[,-1])

sums

## Freq total

## 28 34

*#and now calculate our mean Poisson parameter (lambda)*

lambda <- sums[2]/sums[1]

Calculate expected using the Poisson formula from above \(k\) is the number of blue plaques counted in a square and is found in the first column of our table…

QCountTable$Pr <- ((lambda^QCountTable[,1])\***exp**(-lambda))/**factorial**(QCountTable[,1])

*#now calculate the expected counts and save them to the table*

QCountTable$Expected <- **round**(QCountTable$Pr \* sums[1],0)

QCountTable

## Var1 Freq total Pr Expected

## 1 0 11 0 0.296922026 8

## 2 1 9 9 0.360548174 10

## 3 2 3 6 0.218904249 6

## 4 3 2 6 0.088604101 2

## 5 4 2 8 0.026897673 1

## 6 5 1 5 0.006532292 0

*#Compare the frequency distributions of the observed and expected point patterns*

**plot**(**c**(1,5),**c**(0,14), type="n", xlab="Number of Blue Plaques (Red=Observed,

Blue=Expected)", ylab="Frequency of Occurances")

**points**(QCountTable$Freq, col="Red", type="o", lwd=3)

**points**(QCountTable$Expected, col="Blue", type="o", lwd=3)

A close up of a map

Description automatically generated

Comparing the observed and expected frequencies for our quadrat counts, we can observe that they both have higher frequency counts at the lower end — something reminiscent of a Poisson distribution. This could indicate that for this particular set of quadrats, our pattern is close to Complete Spatial Randomness (i.e. no clustering or dispersal of points). But how do we confirm this?

To check for sure, we can use the quadrat.test function, built into spatstat. This uses a Chi Squared test to compare the observed and expected frequencies for each quadrat (rather than for quadrat bins, as we have just computed above). If the p-value of our Chi-Squared test is > 0.05, then we can reject a null hyphothesis that says “there is *no* complete spatial randomness in our data” (think of a null-hypothesis as the opposite of a hypothesis that says our data exhibit complete spatial randomness). What we need to look for is a value for p > 0.05. If our p-value is > 0.05 then this indicates that we have CSR and there is no pattern in our points. If it is < 0.05, this indicates that we do have clustering in our points.

teststats <- **quadrat.test**(BluePlaquesSub.ppp, nx = 6, ny = 6)

teststats

##

## Chi-squared test of CSR using quadrat counts

## Pearson X2 statistic

##

## data: BluePlaquesSub.ppp

## X2 = 37.999, df = 28, p-value = 0.1968

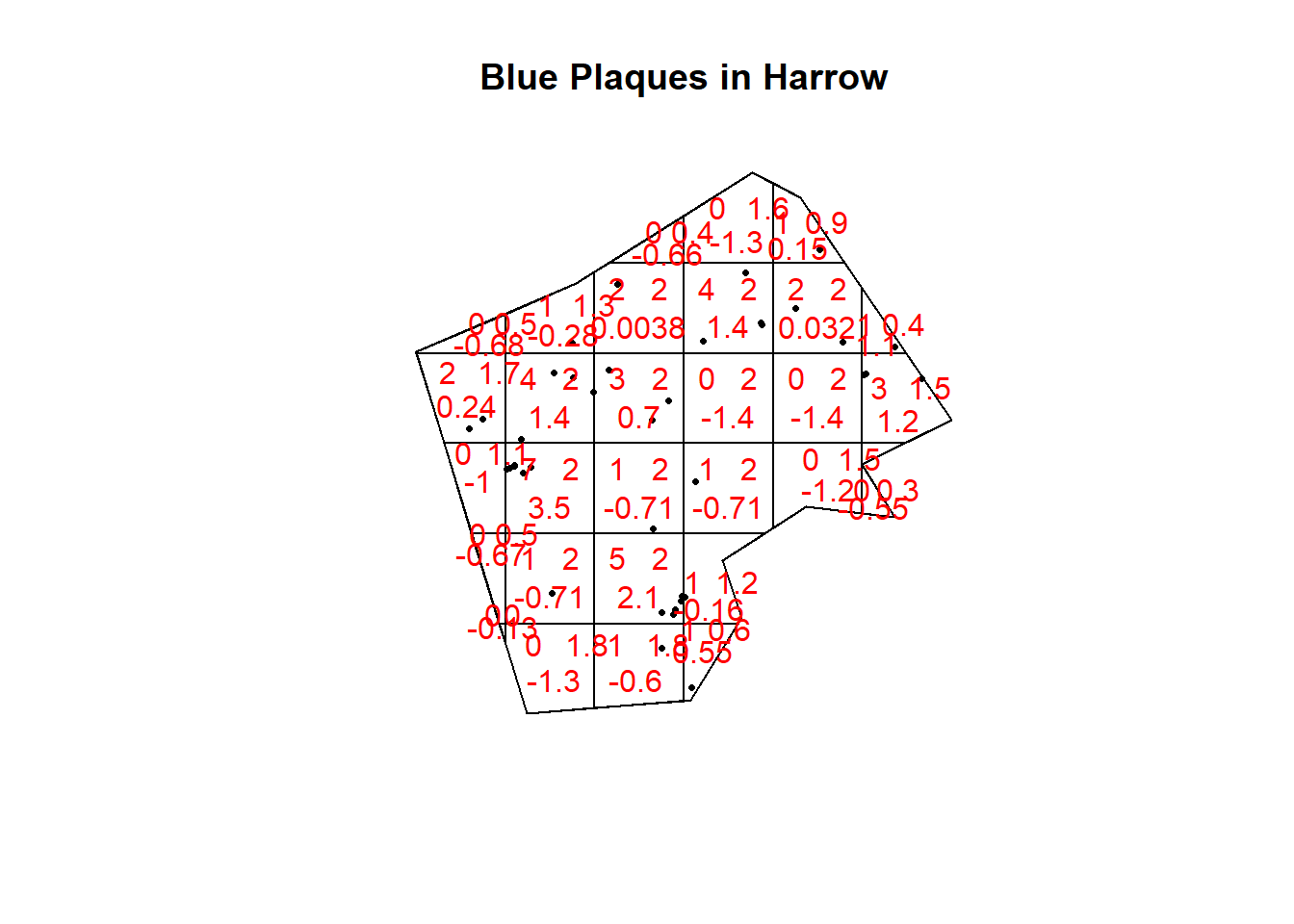
## alternative hypothesis: two.sided

##

## Quadrats: 29 tiles (irregular windows)

**plot**(BluePlaquesSub.ppp,pch=16,cex=0.5, main="Blue Plaques in Harrow")

**plot**(teststats, add=T, col = "red")



So we can see that the indications are there is no spatial patterning for Blue Plaques in Harrow — at least for this particular grid. Note the warning message — some of the observed counts are very small (0) and this may affect the accuracy of the quadrat test. Recall that the Poisson distribution only describes observed occurrances that are counted in integers — where our occurances = 0 (i.e. not observed), this can be an issue. We also know that there are various other problems that might affect our quadrat analysis, such as the [modifiable areal unit problem](https://www.youtube.com/watch?v=cZdlxDx4Nvs).

In the new plot, we can see three figures for each quadrat. The top-left figure is the observed count of points; the top-right is the Poisson expected number of points; the bottom value is the Pearson residual value, or (Observed - Expected) / Sqrt(Expected).

6.5.3 Try experimenting…

Try running a quadrat analysis for different grid arrangements (2 x 2, 3 x 3, 10 x 10 etc.) — how does this affect your results?

6.5.4 Ripley’s K

One way of getting around the limitations of quadrat analysis is to compare the observed distribution of points with the Poisson random model for a whole range of different distance radii. This is what Ripley’s K function computes.

We can conduct a Ripley’s K test on our data very simply with the spatstat package using the kestfunction.

K <- **Kest**(BluePlaquesSub.ppp, correction="border")

**plot**(K)

A close up of a map

Description automatically generated

The plot for K has a number of elements that are worth explaining. First, the *Kpois(r)* line in Red is the theoretical value of K for each distance window (r) under a Poisson assumption of Complete Spatial Randomness. The Black line is the estimated values of K accounting for the effects of the edge of the study area.

Where the value of K falls above the line, the data appear to be clustered at that distance. Where the value of K is below the line, the data are dispersed. From the graph, we can see that up until distances of around 1300 metres, Blue Plaques appear to be clustered in Harrow, however, at around 1500 m, the distribution appears random and then dispersed between about 1600 and 2100 metres.

6.5.5 Alternatives to Ripley’s K

There are a number of alternative measures of spatial clustering which can be computed in spatstatsuch as the G and the L functions — I won’t go into them now, but if you are interested, you should delve into the following references:

* Bivand, R. S., Pebesma, E. J., & Gómez-Rubio, V. (2008). “Applied spatial data analysis with R.” New York: Springer.
* Brundson, C., Comber, L., (2015) “An Introduction to R for Spatial Analysis & Mapping”. Sage.
* <https://research.csiro.au/software/wp-content/uploads/sites/6/2015/02/Rspatialcourse_CMIS_PDF-Standard.pdf>

6.6 Density-based spatial clustering of applications with noise: DBSCAN

Quadrat and Ripley’s K analysis are useful exploratory techniques for telling us if we have spatial clusters present in our point data, but they are not able to tell us WHERE in our area of interest the clusters are occurring. To discover this we need to use alternative techniques. One popular technique for discovering clusters in space (be this physical space or variable space) is DBSCAN. For the complete overview of the DBSCAN algorithm, read the original paper by [Ester et al. (1996)](http://www.aaai.org/Papers/KDD/1996/KDD96-037.pdf) or consult the [wikipedia page](https://en.wikipedia.org/wiki/DBSCAN)

**library**(raster)

**library**(fpc)

**library**(plyr)

**library**(OpenStreetMap)

We will now carry out a DBSCAN analysis of blue plaques in my borough to see if there are any clusters present.

*#first check the coordinate reference system of the Harrow spatial polygon:*

**crs**(Borough)

## CRS arguments:

## +init=epsg:27700 +proj=tmerc +lat\_0=49 +lon\_0=-2 +k=0.9996012717

## +x\_0=400000 +y\_0=-100000 +datum=OSGB36 +units=m +no\_defs

## +ellps=airy

## +towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894

DBSCAN requires you to input two parameters: 1. *Epsilon* - this is the radius within which the algorithm with search for clusters 2. *MinPts* - this is the minimum number of points that should be considered a cluster

Based on the results of the Ripley’s K analysis earlier, we can see that we are getting clustering up to a radius of around 1200m, with the largest bulge in the graph at around 700m. Therefore, 700m is probably a good place to start and we will begin by searching for clusters of at least 4 points…

*#first extract the points from the spatial points data frame*

BluePlaquesSubPoints <- **data.frame**(BluePlaquesSub@coords[,1:2])

*#now run the dbscan analysis*

db <- fpc::**dbscan**(BluePlaquesSubPoints, eps = 700, MinPts = 4)

*#now plot the results*

**plot**(db, BluePlaquesSubPoints, main = "DBSCAN Output", frame = F)

**plot**(Borough, add=T)

A close up of a map

Description automatically generated You could also use kNNdistplot() from the dbscan pacakge to find a suitable eps value based on the ‘knee’ in the plot…

*# used to find suitable eps value based on the knee in plot*

*# k is no of nearest neighbours used, use min points*

**library**(dbscan)

dbscan::**kNNdistplot**(BluePlaquesSubPoints, k = 4)

So the DBSCAN analysis shows that for these values of eps and MinPts there are three clusters in the area I am analysing. Try varying eps and MinPts to see what difference it makes to the output.

Now of course the plot above is a little basic and doesn’t look very aesthetically pleasing. As this is R and R is brilliant, we can always produce a much nicer plot by extracting the useful information from the DBSCAN output and use ggplot2 to produce a much cooler map…

**library**(ggplot2)

Our new db object contains lots of info including the cluster each set of point coordinates belongs to, whether the point is a seed point or a border point etc. We can get a summary by just calling the object

db

## dbscan Pts=41 MinPts=4 eps=700

## 0 1 2 3 4

## border 17 0 1 3 4

## seed 0 8 6 1 1

## total 17 8 7 4 5

If you open up the object in the environment window in RStudio, you will also see the various slots in the object, including cluster

db$cluster

## [1] 2 1 2 0 1 0 1 0 2 3 2 1 0 4 0 2 0 0 0 4 2 4 1 0 4 3 0 4 0 0 0 0 2 0 3

## [36] 1 0 0 3 1 1

We can now add this cluster membership info back into our dataframe

BluePlaquesSubPoints$cluster <- db$cluster

Next we are going to create some convex hull polygons to wrap around the points in our clusters. Use the ddply function in the plyr package to get the convex hull coordinates from the cluster groups in our dataframe

chulls <- **ddply**(BluePlaquesSubPoints, .(cluster),

**function**(df) df[**chull**(df$coords.x1, df$coords.x2), ])

As 0 isn’t actually a cluster (it’s all points that aren’t in a cluster) drop it from the dataframe

chulls <- **subset**(chulls, cluster>=1)

Now create a ggplot2 object from our data

dbplot <- **ggplot**(data=BluePlaquesSubPoints,

**aes**(coords.x1,coords.x2, colour=cluster, fill=cluster))

*#add the points in*

dbplot <- dbplot + **geom\_point**()

*#now the convex hulls*

dbplot <- dbplot + **geom\_polygon**(data = chulls,

**aes**(coords.x1,coords.x2, group=cluster),

alpha = 0.5)

*#now plot, setting the coordinates to scale correctly and as a black and white plot*

*#(just for the hell of it)...*

dbplot + **theme\_bw**() + **coord\_equal**()

A close up of text on a white surface

Description automatically generated

Now we are getting there, but wouldn’t it be better to add a basemap?!

*###add a basemap*

*##First get the bbox in lat long for Harrow*

latlong <- "+init=epsg:4326"

BoroughWGS <-**spTransform**(Borough, **CRS**(latlong))

BoroughWGS@bbox

## min max

## x -0.4040502 -0.2671315

## y 51.5549876 51.6405356

Now convert the basemap to British National Grid

basemap<-**openmap**(**c**(51.5530613,-0.4040719),**c**(51.6405318,-0.2671556), zoom=NULL,"stamen-toner")

*# convert the basemap to British National Grid - remember we created the*

*# BNG object right at the beginning of the practical - it's an epsg string...*

basemap\_bng<-**openproj**(basemap, projection=BNG)

Now we can plot our fancy map with the clusters on…

**autoplot**(basemap\_bng) + **geom\_point**(data=BluePlaquesSubPoints,

**aes**(coords.x1,coords.x2,

colour=cluster, fill=cluster)) +

**geom\_polygon**(data = chulls, **aes**(coords.x1,coords.x2, group=cluster, fill=cluster),

alpha = 0.5)

A close up of a map

Description automatically generated

6.7 Point pattern analysis summary

This is end of the point pattern analysis section of the practical. You have been introduced to the basics of Point Pattern Analysis examining the distribution of Blue Plaques in a London Borough. At this point, you may wish to try running similar analyses on different boroughs (or indeed the whole city) and playing with some of the outputs — although you will find that Ripley’s K will fall over very quickly if you try to run the analysis on that many points)

This how you might make use of these techniques in another context or with different point data…

6.8 Analysing Spatial Autocorrelation with Moran’s I, LISA and friends

Now, at this point you have a choice and at bit like in those [Fighting Fantasy](https://en.wikipedia.org/wiki/Fighting_Fantasy) books that I used to read as a kid, you can select either **Option 1** (which may lead to firey death by dragon) or **Option 2** (which could lead to a pot of gold)…

**Option 1**. If you’ve had enough of coding and you think you might like to do your coursework in ArcGIS and have a bit more practice with model builder, then go to [ArcMap part 2](https://andrewmaclachlan.github.io/CASA0005repo/analysing-spatial-patterns.html#arcmap-part-2). This will show you how to analyse spatial autocorrelation in ArcGIS and will give you more practice will model builder.

**Option 2**. If you are a total bad-ass and want to continue with R, then brilliant!! Woo hoo!! You can keep following the instuctions below.

…Of course there is also **Option 3** and that is to give both a go…

Have the Arc lot gone? OK great, here’s some more lovely R…

In this section we are going to explore patterns of spatially referenced continuous observations using various measures of spatial autocorrelation. Spatial autocorrelation is a measure of similarity between nearby data. Check out the various references in the reading list for more information about the methods we will explore today. There are also useful links in the helpfile of the spdep package which we will be using here.

6.8.1 Data download

Before we get any further, let’s get some ward boundaries read in to R — download LondonWardData from [Github](https://github.com/andrewmaclachlan/CASA0005repo/tree/master/prac6_data) and read it in…

**library**(rgdal)

*#read the ward data in*

LondonWards <- **readOGR**("prac6\_data/LondonWards.shp")

## OGR data source with driver: ESRI Shapefile

## Source: "C:\Users\ucfnmac\OneDrive - University College London\Teaching\CASA0005repo\prac6\_data\LondonWards.shp", layer: "LondonWards"

## with 625 features

## It has 77 fields

It’s probably projected correctly, but in case it isn’t give it a projection using the CRS() function in the raster package

**proj4string**(LondonWards) <- **CRS**("+init=epsg:27700")

*#have a look to check that everything looks OK..*

**tmap\_mode**("view")

**tm\_shape**(LondonWards) +

**tm\_polygons**(col = NA, alpha = 0.5) +

**tm\_shape**(BluePlaques) +

**tm\_dots**(col = "blue")

6.8.2 Data cleaning

Ah yes, we might need to lose the blue plaques that fall outside of London

**summary**(BluePlaquesBNG)

## Object of class SpatialPointsDataFrame

## Coordinates:

## min max

## coords.x1 505575.2 622211.7

## coords.x2 -5527598.3 199509.2

## Is projected: TRUE

## proj4string :

## [+init=epsg:27700 +proj=tmerc +lat\_0=49 +lon\_0=-2 +k=0.9996012717

## +x\_0=400000 +y\_0=-100000 +datum=OSGB36 +units=m +no\_defs

## +ellps=airy

## +towgs84=446.448,-125.157,542.060,0.1502,0.2470,0.8421,-20.4894]

## Number of points: 2711

## Data attributes:

## id

## Min. : 1.0

## 1st Qu.: 698.5

## Median : 6154.0

## Mean :10724.9

## 3rd Qu.:10397.0

## Max. :49190.0

##

## inscription

## Frank Matcham (1854-1920) theatre architect designed this theatre : 3

## Grimâ\200\231s Dyke : 2

## Lillie Langtry 1852-1929 actress lived here : 2

## 'Canons' gate pillars\r\nEntrance gate piers to 'Canons' from Old Turnpike Road\r\nBuilt in 1712 for James Brydges 1673-1744, 1st Duke of Chandos. House demolished 1740 and existing buildings constructed from salvaged materials. Piers refurbished 1998.: 1

## 'Father' Henry Willis 1821-1901 organ builder lived here : 1

## J. H. Greathead Chief Engineer City and South London Railway. Inventor of the Travelling Shield that made possible the cutting of the tunnels of London's deep level tube system : 1

## (Other) :2701

BluePlaquesSub <- BluePlaquesBNG[LondonWards,]

**tm\_shape**(LondonWards) +

**tm\_polygons**(col = NA, alpha = 0.5) +

**tm\_shape**(BluePlaquesSub) +

**tm\_dots**(col = "blue")

6.8.3 Data manipulation

The measures of spatial autocorrelation that we will be using require continuous observations (counts of blue plaques, average GCSE scores, average incomes etc.) to be spatially referenced (i.e. attached to a spatial unit like a ward or a borough). The file you have already has the various obervations associated with the London Ward data file already attached to it, but let’s continue with our blue plaques example for now.

To create a continuous observation from the blue plaques data we need to count all of the blue plaques that fall within each Ward in the City. Luckily, we can do this using the poly.counts() function in Chris Brunsdon’s excellent gistools package… (we could also use the over() function in the sppackage if we wanted)

res <- **poly.counts**(BluePlaquesSub, LondonWards)

*#and add this as a column in our spatialPolygonsDataframe*

LondonWards@data$BluePlaqueCount<-res

*#as the wards are of different sizes, perhaps best that we calculate a density*

LondonWards@data$BlueDensity <- LondonWards$BluePlaqueCount/**poly.areas**(LondonWards)

*#let's just check the data to see if the calculations have worked*

LondonWards@data

How about a quick choropleth map to see how we are getting on…

**tm\_shape**(LondonWards) +

**tm\_polygons**("BlueDensity",

style="jenks",

palette="PuOr",

midpoint=NA,

title="Blue Plaque Density")