Spatial Data Analysis with R

Release 1.0

Robert J. Hijmans and Aniruddha Ghosh

Oct 08, 2018

CONTENTS

CHAPTER

ONE

1. INTRODUCTION

In this section we introduce a number of approaches and techniques that are commonly used in spatial data analysis and modelling.

Spatial data are mostly like other data. The same general principles apply. But there are few things that are rather important to consider when using spatial data that are not common with other data types. These are discussed in Chapters 2 and 3 and include issues of scale and zonation (the modifiable areal unit problem), distance and spatial autocorrelation.

The other chapters, introduce methods in different areas of spatial data analysis. These include the three classical area of spatial statistics (point pattern analysis, regression and inference with spatial data, geostatistics (interpolation using Kriging), as well some other methods (local and global regression and classification with spatial data).

Some of the material presented here is based on examples in the book "Geographic Information Analysis" by David O'Sullivan and David J. Unwin. This book provides an excellent and very accessible introduction to spatial data analysis. It has much more depth than what we present here. But the book does not show how to practically implement the approaches that are discussed — which is the main purpose of this website.

The spatial statistical methods are treated in much more detail in "Applied Spatial Data Analysis with R" by Bivand, Pebesma and Gómez-Rubio.

This section builds on our Introduction to Spatial Data Manipulation R, that you should read first.

CHAPTER

TWO

2. SCALE AND DISTANCE

2.1 Introduction

Scale, aggregations, and distance are two key concepts in spatial data analysis that can be tricky to come to grips with. This chapter first discusses scale and related concepts resolution, aggregation and zonation. The second part of the chapter discusses distance and adjacency.

2.2 Scale and resolution

The term "scale" is tricky. In its narrow geographic sense, it is the the ratio of a distance on a (paper) map to the actual distance. So if a distance of 1 cm on map "A" represents 100 m in the real world, the map scale is 1/10,000 (1:10,000 or 10-4). If 1 cm on map "B" represents 10 km in the real world, the scale of that map is 1/1,000,000. The first map "A" would have relatively large scale (and high resolution) as compared to the second map "B", that would have a small scale (and low resolution). It follows that if the size maps "A" and "B" were the same, map "B" would represent a much larger area (would have a much larger "spatial extent"). For that reason, most people would refer to map "B" having a "larger scale". That is technically wrong, but there is not much point in fighting that, and it is simply best to avoid the term "scale", and certainly "small scale" and "large scale", because that technically means the opposite of what most people think. *If* you want to use these terms, you should probably use them how they are commonly understood; unless you are among cartographers, of course.

Now that mapping has become a computer based activity, scale is even more treacherous. You can use the same data to make maps of different sizes. These would all have a different scale. With digital data, we are more interested in the "inherent" or "measurement" scale of the data. This is sometimes referred to as "grain" but I use "(spatial) resolution". In the case of raster data the notion of resolution is straightforward: it is the size of the cells. For vector data resolution is not as well defined, and it can vary largely within a data set, but you can think of it as the average distance between the nodes (coordinate pairs) of the lines or polygons. Point data do not have a resolution, unless cases that are within a certain distance of each other are merged into a single point (the actual geographic objects represented by points, actually do cover some area; so the actual average size of those areas could also be a measure of interest, but it typically is not).

In the digital world it is easy to create a "false resolution", either by dividing raster cells into 4 or more smaller cells, or by adding nodes in-between nodes of polygons. Imagine having polygons with soils data for a country. Let's say that these polygons cover, on average, an area of 100 * 100 = 10,000 km^2. You can transfer the soil properties associated with each polygon, e.g. pH, to a raster with 1 km^2 spatial resolution; and now might (incorrectly) say that you have a 1 km^2 spatial resolution soils map. So we need to distinguish the resolution of the representation (data) and the resolution of the measurements or estimates. The lowest of the two is the one that matters.

Why does scale/resolution matter?

First of all, different processes have different spatial and temporal scales at which they operate Levin, 1992 — in this context, scale refers both to "extent" and "resolution". Processes that operate over a larger extent (e.g., a forest) can be studied at a larger resolution (trees) whereas processes that operate over a smaller extent (e.g. a tree) may need to be studied at the level of leaves.

From a practical perspective: it affects our estimates of length and size. For example if you wanted to know the length of the coastline of Britain, you could use the length of spatial dataset representing that coastline. You could get rather different numbers depending on the data set used. The higher the resolution of the spatial data, the longer the coastline would appear to be. This is not just a problem of the representation (the data), also at a theoretical level, one can argue that the length of the coastline is not defined, as it becomes infinite if your resolution approaches zero. This is illustrated here

Resolution also affects our understanding of relationships between variables of interest. In terms of data collection this means that we want data to be at the highest spatial (and temporal) resolution possible (affordable). We can *aggregate* our data to lower resolutions, but it is not nearly as easy, or even impossible to correctly *disaggregate* ("downscale") data to a higher resolution.

2.3 Zonation

Geographic data are often aggregated by zones. While we would like to have data at the most granular level that is possible or meanigful (individuals, households, plots, sites), reality is that we often can only get data that is aggregated. Rather than having data for individuals, we may have mean values for all inhabitants of a census district. Data on population, disease, income, or crop yield, is typically available for entire countries, for a number of sub-national units (e.g. provinces), or a set of raster cells.

The areas used to aggregate data are arbitrary (at least relative to the data of interest). The way the borders of the areas are drawn (how large, what shape, where) can strongly affect the patterns we see and the outcome of data analysis. This is sometimes referred to as the "Modifiable Areal Unit Problem" (MAUP). The problem of analyzing aggregated data is referred to as "Ecological Inference".

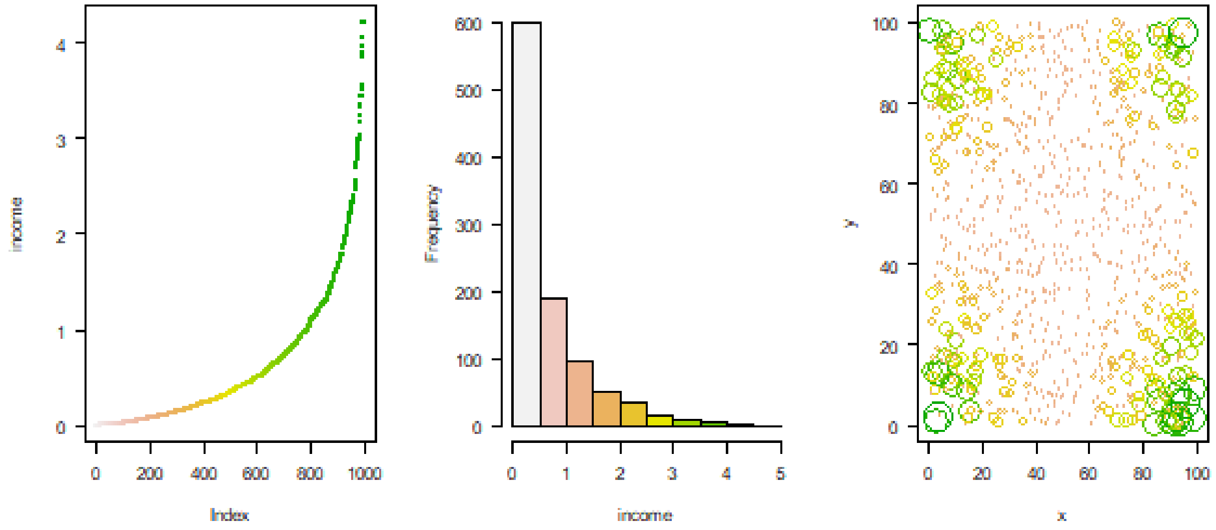
To illustrate the effect of zonation and aggregation, I create a region with 1000 households. For each household we know where they live and what their annual income is. I then aggregate the data to a set of zones.

The income distribution data

```r
set.seed(0)
xy <- cbind(x=runif(1000, 0, 100), y=runif(1000, 0, 100))
income <- (runif(1000) * abs((xy[,1] - 50) * (xy[,2] - 50))) / 500
```

Inspect the data, both spatially and non-spatially. The first two plots show that there are many poor people and a few rich people. The thrird that there is a clear spatial pattern in where the rich and the poor live.

```r
par(mfrow=c(1,3), las=1)
plot(sort(income), col=rev(terrain.colors(1000)), pch=20, cex=.75, ylab='income')
hist(income, main='', col=rev(terrain.colors(10)),  xlim=c(0,5), breaks=seq(0,5,0.5))
plot(xy, xlim=c(0,100), ylim=c(0,100), cex=income, col=rev(terrain.
↪colors(50))[10*(income+1)])
```



Income inequality is often expressed with the Gini coefficient.

```
    n <- length(income)
    G <- (2 * sum(sort(income) * 1:n)/sum(income) - (n + 1)) / n
    G
## [1] 0.5814548
```

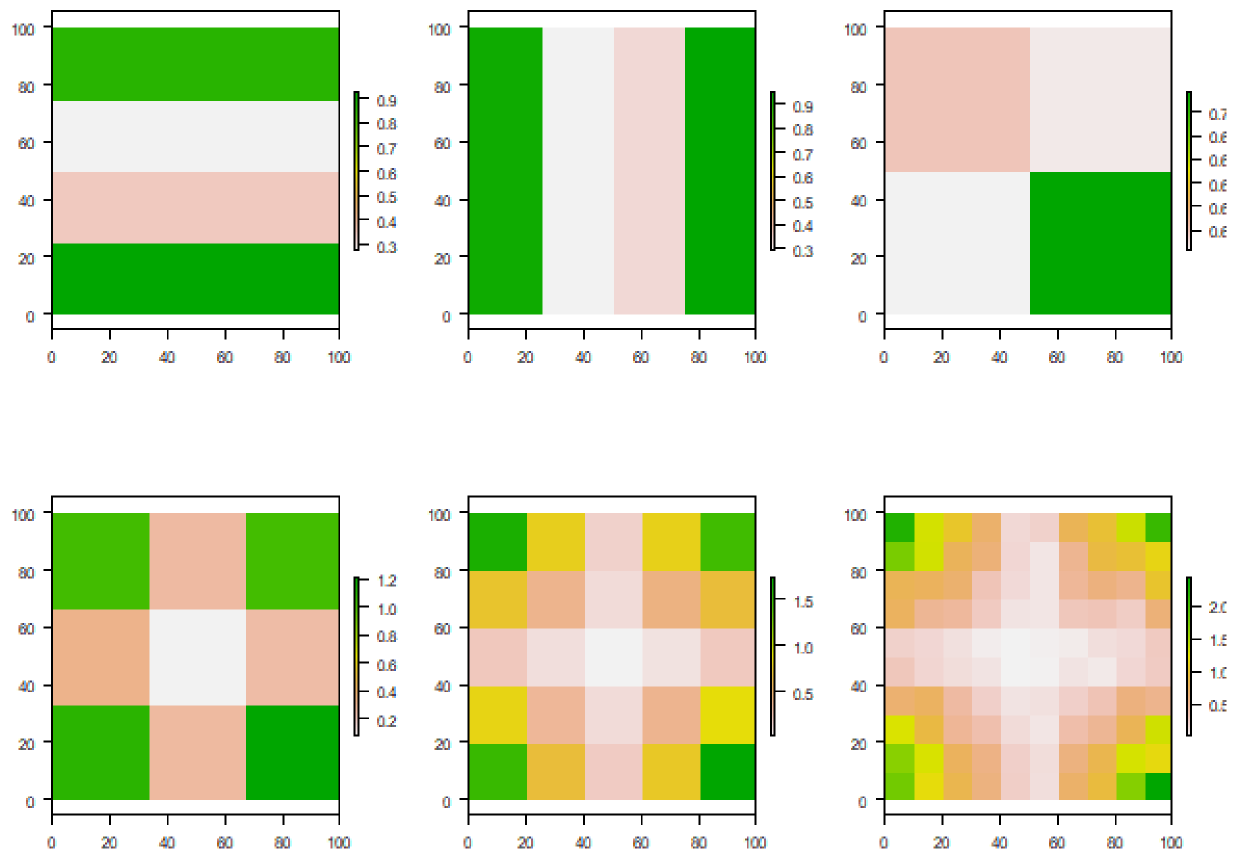
For our data set the Gini coefficient is 0.581.

Now assume that the household data was grouped by some kind of census districts. I create different districts, in our case rectangular raster cells, and compute mean income for each district.

```
library(raster)
r1 <- raster(ncol=1, nrow=4, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r1 <- rasterize(xy, r1, income, mean)

r2 <- raster(ncol=4, nrow=1, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r2 <- rasterize(xy, r2, income, mean)

r3 <- raster(ncol=2, nrow=2, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r3 <- rasterize(xy, r3, income, mean)

r4 <- raster(ncol=3, nrow=3, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r4 <- rasterize(xy, r4, income, mean)

r5 <- raster(ncol=5, nrow=5, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r5 <- rasterize(xy, r5, income, mean)

r6 <- raster(ncol=10, nrow=10, xmn=0, xmx=100, ymn=0, ymx=100, crs=NA)
r6 <- rasterize(xy, r6, income, mean)
```

Have a look at the plots of the income distribution and the sub-regional averages.

```
par(mfrow=c(2,3), las=1)
plot(r1); plot(r2); plot(r3); plot(r4); plot(r5); plot(r6)
```



It is not surprising to see that the smaller the regions get, the better the real pattern is captured. But in all cases, the histograms show that we do not capture the full income distribution (compare to the histogram with the data for individuals).

```
par(mfrow=c(1,3), las=1)
hist(r4, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.5))
hist(r5, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.5))plo
hist(r6, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.5))
## Error: <text>:3:83: unexpected symbol
## 2: hist(r4, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.
↪5))
## 3: hist(r5, main='', col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.
↪5))plo
##                                                                                   ␣
↪  ^
```

2.4 Distance

Distance is a numerical description of how far apart things are. It is the most fundamental concept in geography. After all, Waldo Tobler's First Law of Geography states that "everything is related to everything else, but near things are more related than distant things". But how far away are things? That is not always as easy a question as it seems. Of course we can compute distance "as the crow flies" but that is often not relevant. Perhaps you need to also consider national borders, mountains, or other barriers. The distance between A and B may even by asymetric, meaning that it the distance from A to B is not the same as from B to A (for example, the President of the United States can call me, but I cannot call him (or her)); or because you go faster when walking downhill than when waling uphill.

2.4.1 Distance matrix

Distances are often described in a "distance matrix". In a distance matrix we have a number for the distance between all objects of interest. If the distance is symmetric, we only need to fill half the matrix.

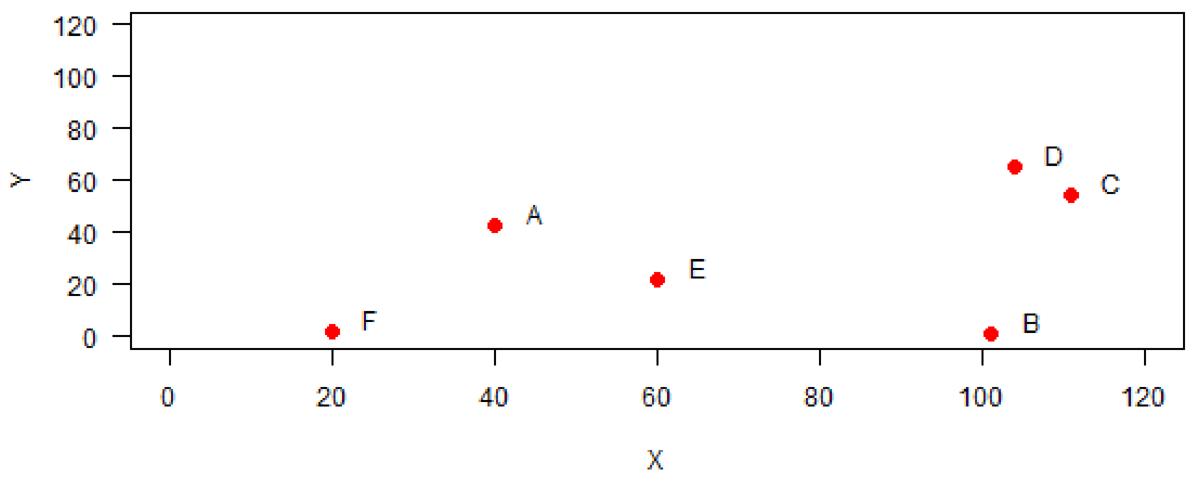
Let's create a distance matrix from a set of points. We start with a set of points

Set up the data, using x-y coordinates for each point:

```
A <- c(40, 43)
B <- c(101, 1)
C <- c(111, 54)
D <- c(104, 65)
E <- c(60, 22)
F <- c(20, 2)
pts <- rbind(A, B, C, D, E, F)
pts
##   [,1] [,2]
## A   40   43
## B  101    1
## C  111   54
## D  104   65
## E   60   22
## F   20    2
```

Plot the points and labels:

```
plot(pts, xlim=c(0,120), ylim=c(0,120), pch=20, cex=2, col='red', xlab='X', ylab='Y',␣
↪las=1)
text(pts+5, LETTERS[1:6])
```



You can use the `dist` function to make a distance matrix with a data set of any dimension.

```
dis <- dist(pts)
dis
##             A           B           C           D           E
## B   74.06079
## C   71.84706   53.93515
## D   67.67570   64.07027   13.03840
## E   29.00000   46.06517   60.20797   61.52235
## F   45.61798   81.00617 104.80935 105.00000   44.72136
```

We can check that for the first point using Pythagoras' theorem.

```
sqrt((40-101)^2 + (43-1)^2)
## [1] 74.06079
```

We can transform a distance matrix into a normal matrix.

```
D <- as.matrix(dis)
round(D)
##     A  B   C   D  E   F
## A   0 74  72  68 29  46
## B  74  0  54  64 46  81
## C  72 54   0  13 60 105
## D  68 64  13   0 62 105
## E  29 46  60  62  0  45
## F  46 81 105 105 45   0
```

Distance matrices are used in all kinds of non-geographical applications. For example, they are often used to create cluster diagrams (dendograms).

Question 4: *Show R code to make a cluster dendogram summarizing the distances between these six sites, and plot it. See* `?hclust`.

2.4.2 Distance for longitude/latitude coordinates

Now consider that the values in `pts` were coordinates in degrees (longitude/latitude). Then the cartesian distance as computed by the dist function would be incorrect. In that case we can use the pointDistance function from the `raster` package.

```
library(raster)
gdis <- pointDistance(pts, lonlat=TRUE)
gdis
##          [,1]    [,2]    [,3]    [,4]    [,5] [,6]
## [1,]        0      NA      NA      NA      NA   NA
## [2,] 7614198       0      NA      NA      NA   NA
## [3,] 5155577 5946748       0      NA      NA   NA
## [4,] 4581656 7104895 1286094       0      NA   NA
## [5,] 2976166 5011592 5536367 5737063       0   NA
## [6,] 4957298 9013726 9894640 9521864 4859627    0
```

Question 5: *What is the unit of the values in ‘‘gdis‘‘?*

2.5 Spatial influence

An important step in spatial statistics and modelling is to get a measure of the spatial influence between geographic objects. This can be expressed as a function of adjacency or (inverse) distance, and is often expressed as a spatial weights matrix. Influence is of course very complex and cannot really be measured and it can be estimated in many ways. For example the influence between a set of polyongs (countries) can be expressed as having a shared border or not (being ajacent); as the "crow-fly" distance between their centroids;or as the lengths of a shared border, and in other ways.

2.5.1 Adjacency

Adjacency is an important concept in some spatial analysis. In some cases objects are considered ajacent when they "touch", e.g. neighboring countries. In can also be based on distance. This is the most common approach when analyzing point data.

We create an adjacency matrix for the point data analysed above. We define points as "ajacent" if they are within a distance of 50 from each other. Given that we have the distance matrix `D` this is easy to do.

```
a <-  D < 50
a
##       A     B     C     D     E     F
## A  TRUE FALSE FALSE FALSE  TRUE  TRUE
## B FALSE  TRUE FALSE FALSE  TRUE FALSE
## C FALSE FALSE  TRUE  TRUE FALSE FALSE
## D FALSE FALSE  TRUE  TRUE FALSE FALSE
## E  TRUE  TRUE FALSE FALSE  TRUE  TRUE
## F  TRUE FALSE FALSE FALSE  TRUE  TRUE
```

In adjacency matrices the diagonal values are often set to `NA` (we do not consider a point to be adjacent to itself). And `TRUE/FALSE` values are commonly stored as `1/0` (this is equivalent, and we can make this change with a simple trick: multiplication with 1)

```
diag(a) <- NA
Adj50 <- a * 1
Adj50
##    A  B  C  D  E  F
## A NA  0  0  0  1  1
## B  0 NA  0  0  1  0
## C  0  0 NA  1  0  0
## D  0  0  1 NA  0  0
## E  1  1  0  0 NA  1
## F  1  0  0  0  1 NA
```

2.5.2 Two nearest neighbours

What if you wanted to compute the "two nearest neighbours" (or three, or four) adjacency-matrix? Here is how you can do that. For each row, we first get the column numbers in order of the values in that row (that is, the numbers indicate how the values are ordered).

```
cols <- apply(D, 1, order)
# we need to transpose the result
cols <- t(cols)
```

And then get columns 2 to 3 (why not column 1?)

```
cols <- cols[, 2:3]
cols
##   [,1] [,2]
```

(continues on next page)

(continued from previous page)

```
## A     5     6
## B     5     3
## C     4     2
## D     3     5
## E     1     6
## F     5     1
```

As we now have the column numbers, we can make the row-column pairs that we want (`rowcols`).

```
rowcols <- cbind(rep(1:6, each=2), as.vector(t(cols)))
head(rowcols)
##      [,1] [,2]
## [1,]    1    5
## [2,]    1    6
## [3,]    2    5
## [4,]    2    3
## [5,]    3    4
## [6,]    3    2
```

We use these pairs as indices to change the values in matrix `Ak3`.

```
Ak3 <- Adj50 * 0
Ak3[rowcols] <- 1
Ak3
##    A  B  C  D  E  F
## A NA  0  0  0  1  1
## B  0 NA  1  0  1  0
## C  0  1 NA  1  0  0
## D  0  0  1 NA  1  0
## E  1  0  0  0 NA  1
## F  1  0  0  0  1 NA
```

2.5.3 Weights matrix

Rather than expressing spatial influence as a binary value (adjacent or not), it is often expressed as a continuous value. The simplest approach is to use inverse distance (the further away, the lower the value).

```
W <- 1 / D
round(W, 4)
##        A      B      C      D      E      F
## A    Inf 0.0135 0.0139 0.0148 0.0345 0.0219
## B 0.0135    Inf 0.0185 0.0156 0.0217 0.0123
## C 0.0139 0.0185    Inf 0.0767 0.0166 0.0095
## D 0.0148 0.0156 0.0767    Inf 0.0163 0.0095
## E 0.0345 0.0217 0.0166 0.0163    Inf 0.0224
## F 0.0219 0.0123 0.0095 0.0095 0.0224    Inf
```

Such as "spatial weights" matrix is often "row-normalized", such that the sum of weights for each row in the matrix is the same. First we get rid if the `Inf` values by changing them to `NA`. (Where did the `Inf` values come from?)

```
W[!is.finite(W)] <- NA
```

Then compute the row sums.

```
rtot <- rowSums(W, na.rm=TRUE)
# this is equivalent to
# rtot <- apply(W, 1, sum, na.rm=TRUE)
rtot
##          A          B          C          D          E          F
## 0.09860117 0.08170418 0.13530597 0.13285878 0.11141516 0.07569154
```

And divide the rows by their totals and check if they row sums add up to 1.

```
W <- W / rtot
rowSums(W, na.rm=TRUE)
## A B C D E F
## 1 1 1 1 1 1
```

The values in the columns do not add up to 1.

```
colSums(W, na.rm=TRUE)
##         A         B         C         D         E         F
## 0.9784548 0.7493803 1.2204900 1.1794393 1.1559273 0.7163082
```

2.5.4 Spatial influence for polygons

Above we looked at adjacency for a set of points. Here we look at it for polygons. The difference is that

```
library(raster)
p <- shapefile(system.file("external/lux.shp", package="raster"))
```

To find adjacent polygons, we can use the `spdep` package.

```
library(spdep)
```

We use `poly2nb` to create a "rook's case" neighbors-list. And from that a neighbors matrix.

```
wr <- poly2nb(p, row.names=p$ID_2, queen=FALSE)
wr
## Neighbour list object:
## Number of regions: 12
## Number of nonzero links: 46
## Percentage nonzero weights: 31.94444
## Average number of links: 3.833333
wm <- nb2mat(wr, style='B', zero.policy = TRUE)
dim(wm)
## [1] 12 12
```

Inspect the content or `wr` and `wm`

```
wr[1:6]
## [[1]]
## [1] 2 4 5
##
## [[2]]
## [1]  1  3  4  5  6 12
##
## [[3]]
## [1]  2  5  9 12
```

(continues on next page)

(continued from previous page)

```
##
## [[4]]
## [1] 1 2
##
## [[5]]
## [1] 1 2 3
##
## [[6]]
## [1]  2  8 12
wm[1:6,1:11]
##   [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
## 1    0    1    0    1    1    0    0    0    0     0     0
## 2    1    0    1    1    1    1    0    0    0     0     0
## 3    0    1    0    0    1    0    0    0    1     0     0
## 4    1    1    0    0    0    0    0    0    0     0     0
## 5    1    1    1    0    0    0    0    0    0     0     0
## 6    0    1    0    0    0    0    0    1    0     0     0
```

Compute the number of neighbors for each area.

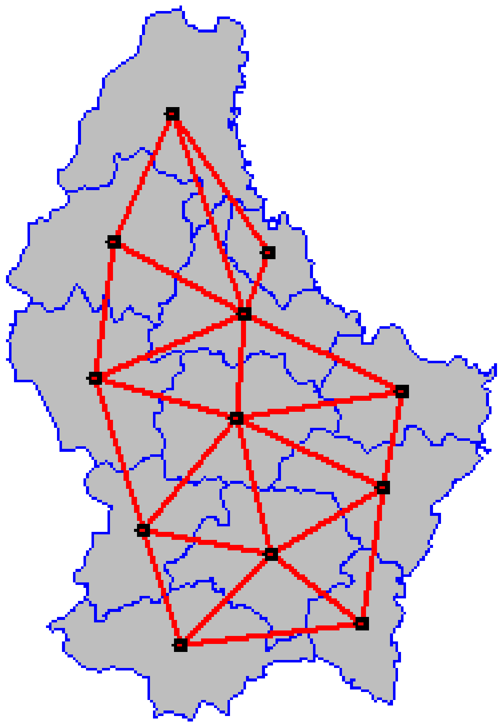
```
i <- rowSums(wm)
i
##  1  2  3  4  5  6  7 12  8  9 10 11
##  3  6  4  2  3  3  3  4  4  3  5  6
```

Expresses as percentage

```
round(100 * table(i) / length(i), 1)
## i
##    2    3    4    5    6
##  8.3 41.7 25.0  8.3 16.7
```

Plot the links between the polygons.

```
par(mai=c(0,0,0,0))
plot(p, col='gray', border='blue')
xy <- coordinates(p)
plot(wr, xy, col='red', lwd=2, add=TRUE)
```



Now some alternative approaches to compute "spatial influence".

Distance based:

```
wd10 <- dnearneigh(xy, 0, 10)
wd25 <- dnearneigh(xy, 0, 25, longlat=TRUE)
```

Nearest neighbors:

```
k3 <- knn2nb(knearneigh(xy, k=3, RANN=FALSE))
k6 <- knn2nb(knearneigh(xy, k=6, RANN=FALSE))
## Warning in knearneigh(xy, k = 6, RANN = FALSE): k greater than one-third of
## the number of data points
```

Lag-two Rook:

```
wr2 <- wr
for (i in 1:length(wr)) {
    lag1 <- wr[[i]]
    lag2 <- wr[lag1]
    lag2 <- sort(unique(unlist(lag2)))
    lag2 <- lag2[!(lag2 %in% c(wr[[i]], i))]
    wr2[[i]] <- lag2
}
```

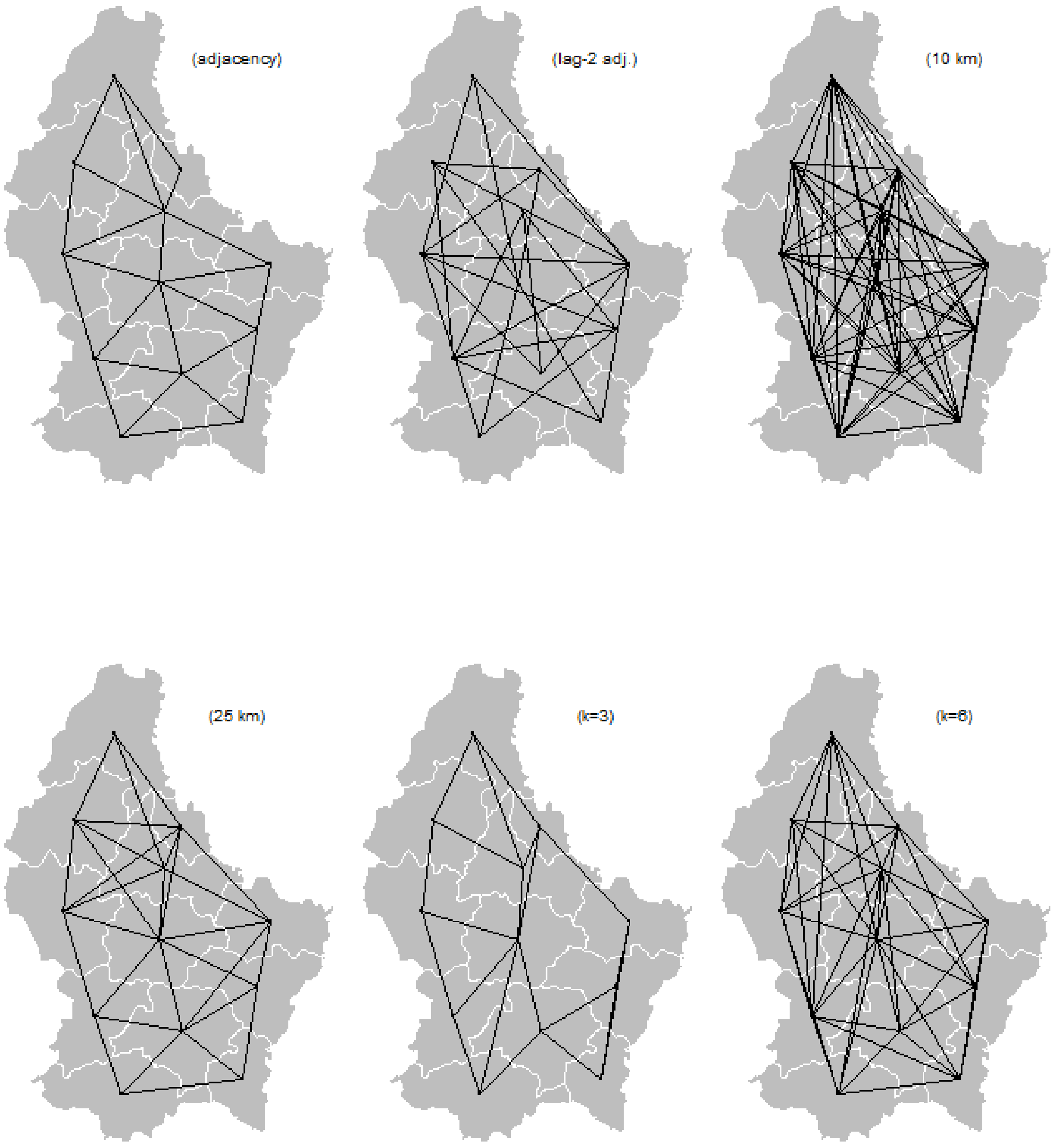
And now we plot them all using the `plotit` function.

```
  plotit <- function(nb, lab='') {
  plot(p, col='gray', border='white')
  plot(nb, xy, add=TRUE, pch=20)
  text(6.3, 50.1, paste0('(', lab, ')'), cex=1.25)
```

(continues on next page)

(continued from previous page)

```
}

par(mfrow=c(2, 3), mai=c(0,0,0,0))
plotit(wr, 'adjacency')
plotit(wr2, 'lag-2 adj.')
plotit(wd10, '10 km')
plotit(wd25, '25 km')
plotit(k3, 'k=3')
plotit(k6, 'k=6')
```


(adjacency)
(lag-2 adj.)
(10 km)
(25 km)
(k=3)
(k=6)

2.6 Raster based distance metrics

2.6.1 distance

2.6.2 cost distance

2.6.3 resistance distance

CHAPTER

THREE

3. SPATIAL AUTOCORRELATION

3.1 Introduction

Spatial autocorrelation is an important concept in spatial statistics. It is a both a nuisance, as it complicates statistical tests, and a feature, as it allows for spatial interpolation. Its computation and properties are often misunderstood. This chapter discusses what it is, and how statistics describing it can be computed.

Autocorrelation (whether spatial or not) is a measure of similarity (correlation) between nearby observations. To understand spatial autocorrelation, it helps to first consider temporal autocorrelation.

3.1.1 Temporal autocorrelation

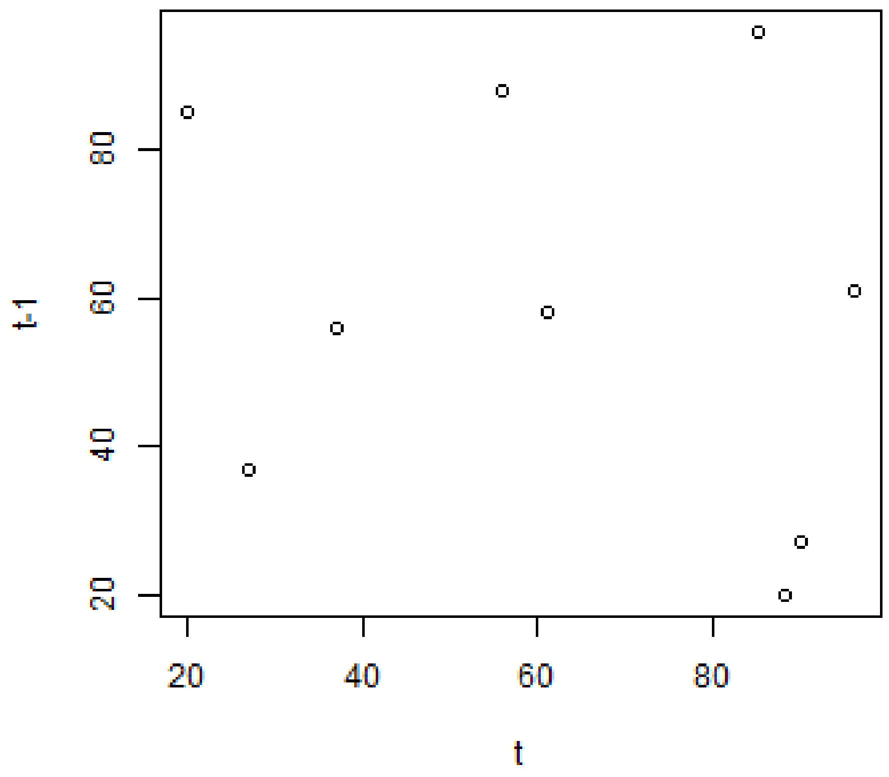
If you measure something about the same object over time, for example a persons weight or wealth, it is likely that two observations that are close to each other in time are also similar in measurement. Say that over a couple of years your weight went from 50 to 80 kg. It is unlikely that it was 60 kg one day, 50 kg the next and 80 the day after that. Rather it probably went up gradually, with the occasional tapering off, or even reverse in direction. The same may be true with your bank account, but that may also have a marked monthly trend. To measure the degree of association over time, we can compute the correlation of each observation with the next observation.

Let `d` be a vector of daily observations.

```r
set.seed(0)
d <- sample(100, 10)
d
##  [1] 90 27 37 56 88 20 85 96 61 58
```

Compute auto-correlation.

```r
a <- d[-length(d)]
b <- d[-1]
plot(a, b, xlab='t', ylab='t-1')
```

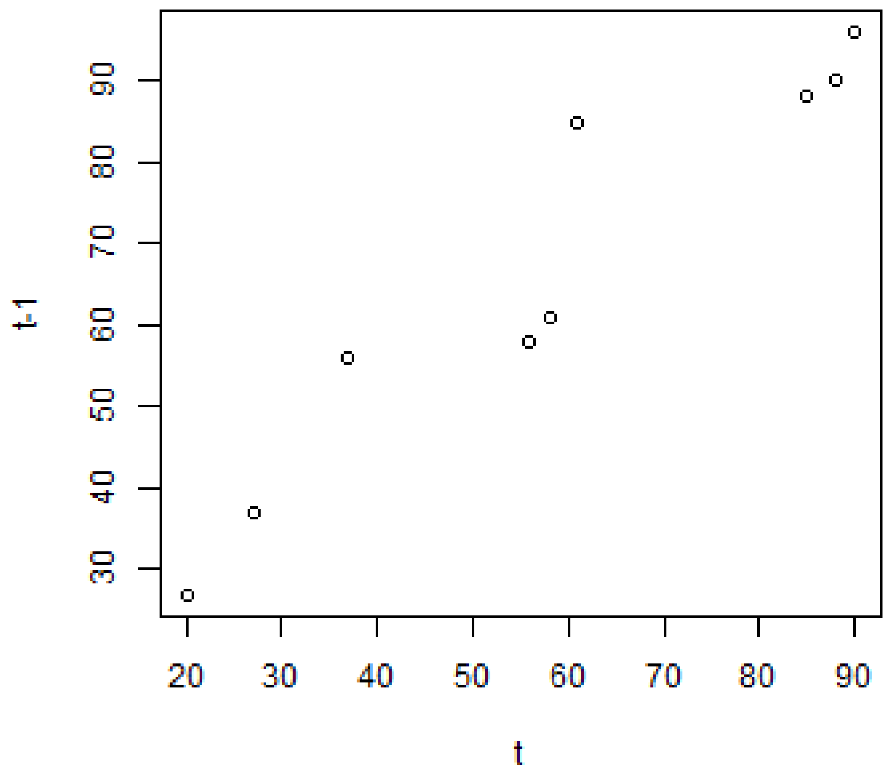


```
cor(a, b)
## [1] -0.2222057
```

The autocorrelation computed above is very small. Even though this is a random sample, you (almost) never get a value of zero. We computed the "one-lag" autocorrelation, that is, we compare each value to its immediate neighbour, and not to other nearby values.

After sorting the numbers in `d` autocorrelation becomes very strong (unsurprisingly).

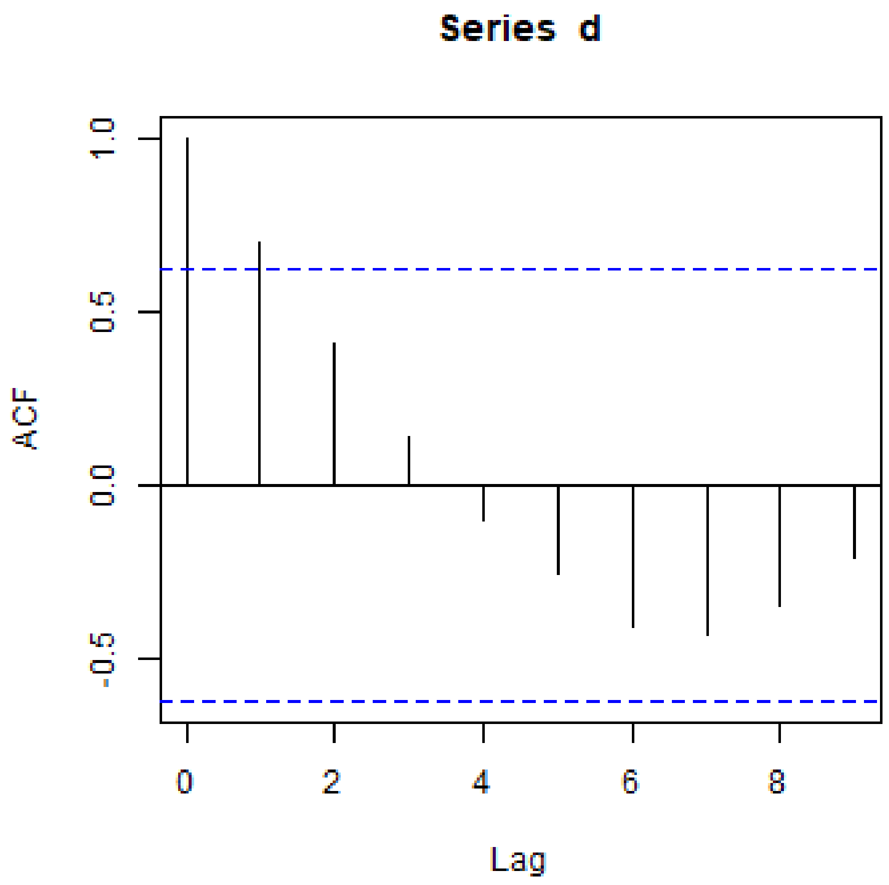
```
d <- sort(d)
d
##  [1] 20 27 37 56 58 61 85 88 90 96
a <- d[-length(d)]
b <- d[-1]
plot(a, b, xlab='t', ylab='t-1')
```



```
cor(a, b)
## [1] 0.9530641
```

The `acf` function shows autocorrelation computed in a slightly different way for several lags (it is 1 to each point it self, very high when comparing with the nearest neighbour, and than tapering off).

```
acf(d)
```



3.1.2 Spatial autocorrelation

The concept of *spatial* autocorrelation is an extension of temporal autocorrelation. It is a bit more complicated though. Time is one-dimensional, and only goes in one direction, ever forward. Spatial objects have (at least) two dimensions and complex shapes, and it may not be obvious how to determine what is "near".

Measures of spatial autocorrelation describe the degree two which observations (values) at spatial locations (whether they are points, areas, or raster cells), are similar to each other. So we need two things: observations and locations.

Spatial autocorrelation in a variable can be exogenous (it is caused by another spatially autocorrelated variable, e.g. rainfall) or endogenous (it is caused by the process at play, e.g. the spread of a disease).

A commonly used statistic that describes spatial autocorrelation is Moran's *I*, and we'll discuss that here in detail. Other indices include Geary's *C* and, for binary data, the join-count index. The semi-variogram also expresses the amount of spatial autocorrelation in a data set (see the chapter on interpolation).

3.2 Example data

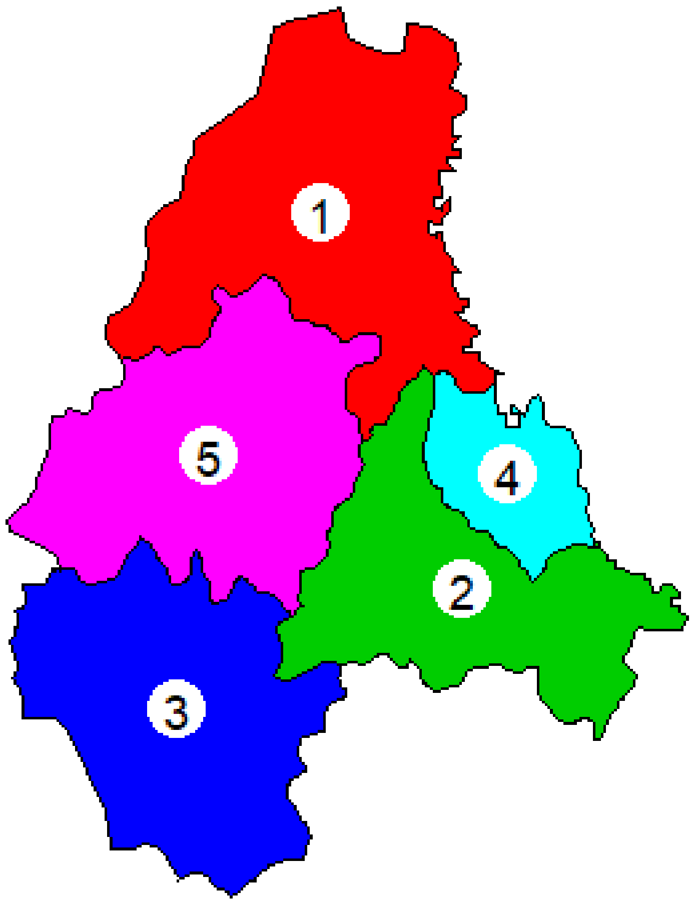
Read the example data

```
library(raster)
p <- shapefile(system.file("external/lux.shp", package="raster"))
p <- p[p$NAME_1=="Diekirch", ]
p$value <- c(10, 6, 4, 11, 6)
data.frame(p)
##   ID_1   NAME_1 ID_2   NAME_2 AREA value
## 0    1 Diekirch    1 Clervaux  312    10
## 1    1 Diekirch    2 Diekirch  218     6
## 2    1 Diekirch    3  Redange  259     4
## 3    1 Diekirch    4  Vianden   76    11
## 4    1 Diekirch    5    Wiltz  263     6
```

Let’s say we are interested in spatial autocorrelation in variable “AREA”. If there were spatial autocorrelation, regions of a similar size would be spatially clustered.

Here is a plot of the polygons. I use the `coordinates` function to get the centroids of the polygons to place the labels.

```
par(mai=c(0,0,0,0))
plot(p, col=2:7)
xy <- coordinates(p)
points(xy, cex=6, pch=20, col='white')
text(p, 'ID_2', cex=1.5)
```



3.3 Adjacent polygons

Now we need to determine which polygons are "near", and how to quantify that. Here we'll use adjacency as criterion. To find adjacent polygons, we can use package 'spdep'.

```
library(spdep)
w <- poly2nb(p, row.names=p$Id)
class(w)
## [1] "nb"
summary(w)
## Neighbour list object:
## Number of regions: 5
## Number of nonzero links: 14
## Percentage nonzero weights: 56
## Average number of links: 2.8
## Link number distribution:
##
## 2 3 4
## 2 2 1
## 2 least connected regions:
## 2 3 with 2 links
## 1 most connected region:
## 1 with 4 links
```

`summary(w)` tells us something about the neighborhood. The average number of neighbors (adjacent polygons) is 2.8, 3 polygons have 2 neighbors and 1 has 4 neighbors (which one is that?).

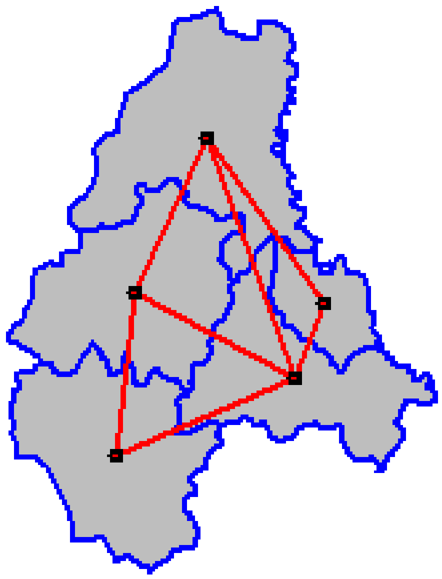
For more details we can look at the `str`ucture of `w`.

```
str(w)
## List of 5
##  $ : int [1:3] 2 4 5
##  $ : int [1:4] 1 3 4 5
##  $ : int [1:2] 2 5
##  $ : int [1:2] 1 2
##  $ : int [1:3] 1 2 3
##  - attr(*, "class")= chr "nb"
##  - attr(*, "region.id")= chr [1:5] "0" "1" "2" "3" ...
##  - attr(*, "call")= language poly2nb(pl = p, row.names = p$Id)
##  - attr(*, "type")= chr "queen"
##  - attr(*, "sym")= logi TRUE
```

Question 1:*Explain the meaning of the first 5 lines returned by str(w)*

Plot the links between the polygons.

```
plot(p, col='gray', border='blue', lwd=2)
plot(w, xy, col='red', lwd=2, add=TRUE)
```



We can transform `w` into a spatial weights matrix. A spatial weights matrix reflects the intensity of the geographic relationship between observations (see previous chapter).

```
wm <- nb2mat(w, style='B')
wm
##   [,1] [,2] [,3] [,4] [,5]
## 0    0    1    0    1    1
## 1    1    0    1    1    1
## 2    0    1    0    0    1
## 3    1    1    0    0    0
## 4    1    1    1    0    0
## attr(,"call")
## nb2mat(neighbours = w, style = "B")
```

3.4 Compute Moran's *I*

Now let's compute Moran's index of spatial autocorrelation

$$I = \frac{n}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(y_i - \bar{y})(y_j - \bar{y})}{\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}}$$

Yes, that looks impressive. But it is not much more than an expanded version of the formula to compute the correlation coefficient. The main thing that was added is the spatial weights matrix.

The number of observations

```
n <- length(p)
```

Get 'y' and 'ybar' (the mean value of y)

```
y <- p$value
ybar <- mean(y)
```

Now we need

$$(y_i - \bar{y})(y_j - \bar{y})$$

That is, (yi-ybar)(yj-ybar) for all pairs. I show two methods to get that.

Method 1:

```
dy <- y - ybar
g <- expand.grid(dy, dy)
yiyj <- g[,1] * g[,2]
```

Method 2:

```
yi <- rep(dy, each=n)
yj <- rep(dy)
yiyj <- yi * yj
```

Make a matrix of the multiplied pairs

```
pm <- matrix(yiyj, ncol=n)
```

And multiply this matrix with the weights to set to zero the value for the pairs that are not adjacent.

```
pmw <- pm * wm
pmw
##     [,1]  [,2] [,3]  [,4]  [,5]
## 0  0.00 -3.64 0.00  9.36 -3.64
## 1 -3.64  0.00 4.76 -5.04  1.96
## 2  0.00  4.76 0.00  0.00  4.76
## 3  9.36 -5.04 0.00  0.00  0.00
## 4 -3.64  1.96 4.76  0.00  0.00
## attr(,"call")
## nb2mat(neighbours = w, style = "B")
```

We now sum the values, to get this bit of Moran's *I*:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}(y_i - \bar{y})(y_j - \bar{y})$$

```
spmw <- sum(pmw)
spmw
## [1] 17.04
```

The next step is to divide this value by the sum of weights. That is easy.

```
smw <- sum(wm)
sw  <- spmw / smw
```

And compute the inverse variance of y

```
vr <- n / sum(dy^2)
```

The final step to compute Moran's *I*

```
MI <- vr * sw
MI
## [1] 0.1728896
```

This is a simple (but crude) way to estimate the expected value of Moran's *I*. That is, the value you would get in the absence of spatial autocorelation (if the data were spatially random). Of course you never really expect that, but that is how we do it in statistics. Note that the expected value approaches zero if *n* becomes large, but that it is not quite zero for small values of *n*.

```
EI <- -1/(n-1)
EI
## [1] -0.25
```

After doing this 'by hand', now let's use the spdep package to compute Moran's *I* and do a significance test. To do this we need to create a 'listw' type spatial weights object (instead of the matrix we used above). To get the same value as above we use "style='B'" to use binary (TRUE/FALSE) distance weights.

```
ww <-  nb2listw(w, style='B')
ww
## Characteristics of weights list object:
## Neighbour list object:
## Number of regions: 5
## Number of nonzero links: 14
## Percentage nonzero weights: 56
## Average number of links: 2.8
##
## Weights style: B
## Weights constants summary:
##   n nn S0 S1  S2
## B 5 25 14 28 168
```

Now we can use the `moran` function. Have a look at ?moran. The function is defined as 'moran(y, ww, n, Szero(ww))'. Note the odd arguments n and S0. I think they are odd, because "ww" has that information. Anyway, we supply them and it works. There probably are cases where it makes sense to use other values.

```
moran(p$value, ww, n=length(ww$neighbours), S0=Szero(ww))
## $I
## [1] 0.1728896
##
## $K
## [1] 1.432464

#Note that
Szero(ww)
## [1] 14
# is the same as
pmw
##      [,1]  [,2] [,3]  [,4]  [,5]
## 0   0.00 -3.64 0.00  9.36 -3.64
## 1  -3.64  0.00 4.76 -5.04  1.96
## 2   0.00  4.76 0.00  0.00  4.76
```

(continues on next page)

(continued from previous page)

```
## 3  9.36 -5.04 0.00  0.00  0.00
## 4 -3.64  1.96 4.76  0.00  0.00
## attr(,"call")
## nb2mat(neighbours = w, style = "B")
sum(pmw==0)
## [1] 11
```

Now we can test for significance. First analytically, using linear regression based logic and assumptions.

```
moran.test(p$value, ww, randomisation=FALSE)
##
##  Moran I test under normality
##
## data:  p$value
## weights: ww
##
## Moran I statistic standard deviate = 2.3372, p-value = 0.009714
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##         0.1728896        -0.2500000         0.0327381
```

Instead of the approach above you should use Monte Carlo simulation. That is the preferred method (in fact, the only good method). The oay it works that the values are randomly assigned to the polygons, and the Moran's *I* is computed. This is repeated several times to establish a distribution of expected values. The observed value of Moran's *I* is then compared with the simulated distribution to see how likely it is that the observed values could be considered a random draw.

```
moran.mc(p$value, ww, nsim=99)
##
##  Monte-Carlo simulation of Moran I
##
## data:  p$value
## weights: ww
## number of simulations + 1: 100
##
## statistic = 0.17289, observed rank = 96, p-value = 0.04
## alternative hypothesis: greater
```

Question 2: *How do you interpret these results (the significance tests)?*

Also try this code, it gives an error: `moran.mc(p$value, ww, nsim=999)`

Question 3: *What is the maximum value we can use for nsim?*

We can make a "Moran scatter plot" to visualize spatial autocorrelation. We first get the neighbouring values for each value.

```
n <- length(p)
ms <- cbind(id=rep(1:n, each=n), y=rep(y, each=n), value=as.vector(wm * y))
```

Remove the zeros

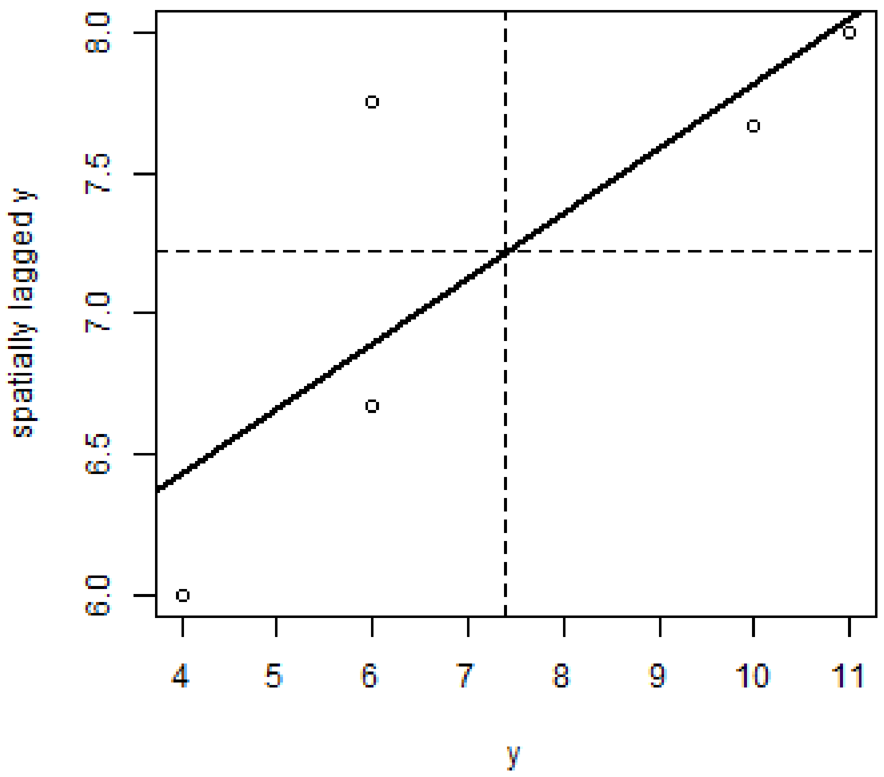
```
ms <- ms[ms[,3] > 0, ]
```

And compute the average neighbour value

```
ams <- aggregate(ms[,2:3], list(ms[,1]), FUN=mean)
ams <- ams[,-1]
colnames(ams) <- c('y', 'spatially lagged y')
head(ams)
##    y spatially lagged y
## 1 10           7.666667
## 2  6           7.750000
## 3  4           6.000000
## 4 11           8.000000
## 5  6           6.666667
```

Finally, the plot.

```
plot(ams)
reg <- lm(ams[,2] ~ ams[,1])
abline(reg, lwd=2)
abline(h=mean(ams[,2]), lt=2)
abline(v=ybar, lt=2)
```



Note that the slope of the regression line:

```
coefficients(reg)[2]
##  ams[, 1]
## 0.2315341
```

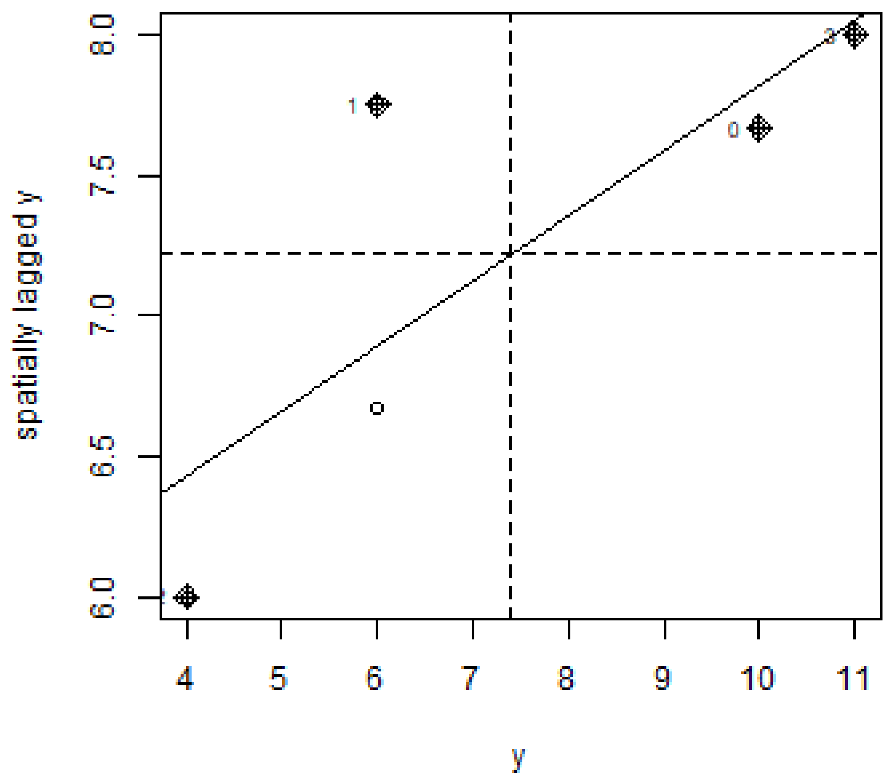
is almost the same as Moran's *I*.

Here is a more direct approach to accomplish the same thing (but hopefully the above makes it clearer how this is actually computed). Note the row standardisation of the weights matrix:

```
rwm <- mat2listw(wm, style='W')
# Checking if rows add up to 1
mat <- listw2mat(rwm)
apply(mat, 1, sum)[1:15]
##    0    1    2    3    4 <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA> <NA>
##    1    1    1    1    1   NA   NA   NA   NA   NA   NA   NA   NA   NA   NA
```

Now we can plot

```
moran.plot(y, rwm)
```



Question 4: *Show how to use the 'geary' function to compute Geary's C*

Question 5: *Write your own Monte Carlo simulation test to compute p-values for Moran's I, replicating the results we obtained with the function from spdep. Show a histogram of the simulated values.*

Question 6: *Write your own Geary C function, by completing the function below*

```
gearyC <- ((n-1)/sum(( "----")\^2)) * sum(wm * (" --- ")\^2) / (2 * sum(wm))
```

CHAPTER

FOUR

4. INTERPOLATION

4.1 Introduction

Almost any variable of interest has spatial autocorrelation. That can be a problem in statistical tests, but it is a very useful feature when we want to predict values at locations where no measurements have been made; as we can generally safely assume that values at nearby locations will be similar. There are several spatial interpolation techniques. We show some of them in this chapter.

4.1.1 Get the data

You can download the data used in the examples here: precipitation, counties, airqual, or with the script below.

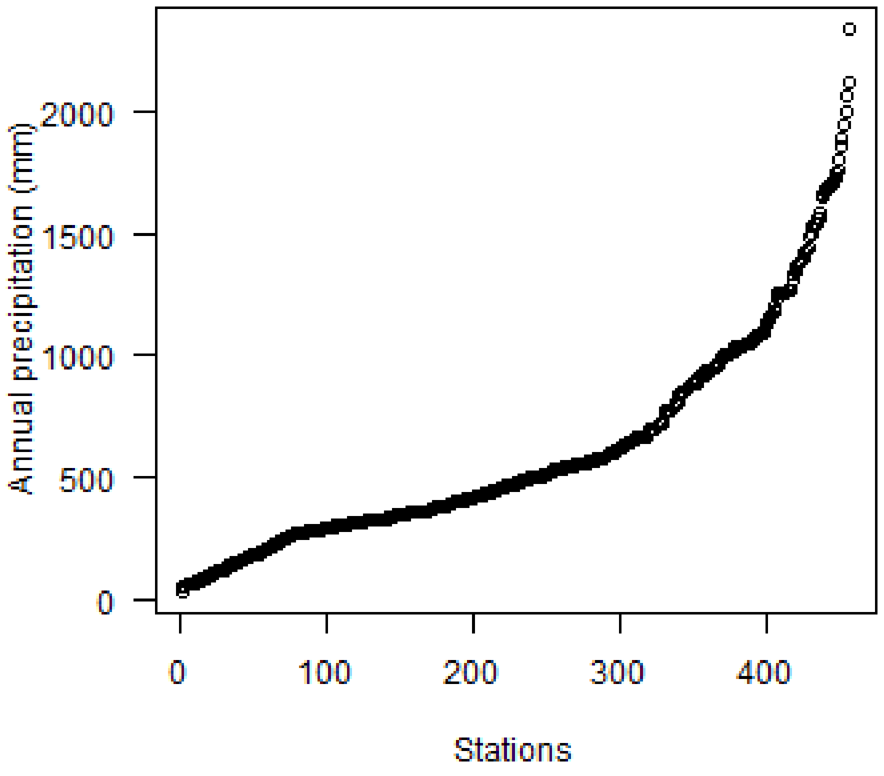
4.2 Temperature in California

We will be working with temperature data for California.

```
# If have not yet done so, first install the rspatial package (see the line below).␣
↪You may need to install the devtools package first
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
## Loading required package: rspatial

library(rspatial)
d <- sp_data('precipitation')
head(d)
##      ID                 NAME   LAT    LONG ALT  JAN FEB MAR APR MAY JUN
## 1 ID741         DEATH VALLEY 36.47 -116.87 -59  7.4 9.5 7.5 3.4 1.7 1.0
## 2 ID743  THERMAL/FAA AIRPORT 33.63 -116.17 -34  9.2 6.9 7.9 1.8 1.6 0.4
## 3 ID744          BRAWLEY 2SW 32.96 -115.55 -31 11.3 8.3 7.6 2.0 0.8 0.1
## 4 ID753 IMPERIAL/FAA AIRPORT 32.83 -115.57 -18 10.6 7.0 6.1 2.5 0.2 0.0
## 5 ID754               NILAND 33.28 -115.51 -18  9.0 8.0 9.0 3.0 0.0 1.0
## 6 ID758        EL CENTRO/NAF 32.82 -115.67 -13  9.8 1.6 3.7 3.0 0.4 0.0
##   JUL  AUG SEP OCT NOV DEC
## 1 3.7  2.8 4.3 2.2 4.7 3.9
## 2 1.9  3.4 5.3 2.0 6.3 5.5
## 3 1.9  9.2 6.5 5.0 4.8 9.7
## 4 2.4  2.6 8.3 5.4 7.7 7.3
## 5 8.0  9.0 7.0 8.0 7.0 9.0
## 6 3.0 10.8 0.2 0.0 3.3 1.4
```

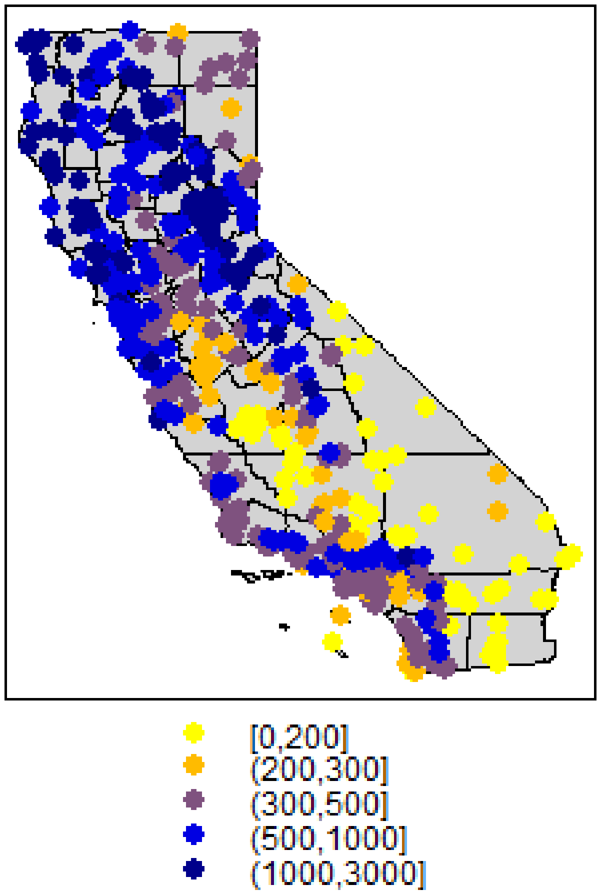
Compute annual precipitation

```
d$prec <- rowSums(d[, c(6:17)])
plot(sort(d$prec), ylab='Annual precipitation (mm)', las=1, xlab='Stations')
```



Now make a quick map.

```
library(sp)
dsp <- SpatialPoints(d[,4:3], proj4string=CRS("+proj=longlat +datum=NAD83"))
dsp <- SpatialPointsDataFrame(dsp, d)
CA <- sp_data("counties")

# define groups for mapping
cuts <- c(0,200,300,500,1000,3000)
# set up a palette of interpolated colors
blues <- colorRampPalette(c('yellow', 'orange', 'blue', 'dark blue'))
pols <- list("sp.polygons", CA, fill = "lightgray")
spplot(dsp, 'prec', cuts=cuts, col.regions=blues(5), sp.layout=pols, pch=20, cex=2)
```



Transform longitude/latitude to planar coordinates, using the commonly used coordinate reference system for California ("Teale Albers") to assure that our interpolation results will align with other data sets we have.

```
TA <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000 +datum=NAD83 +units=m +ellps=GRS80 +towgs84=0,0,0")
library(rgdal)
dta <- spTransform(dsp, TA)
cata <- spTransform(CA, TA)
```

4.2.1 9.2 NULL model

We are going to interpolate (estimate for unsampled locations) the precipitation values. The simplest way would be to take the mean of all observations. We can consider that a "Null-model" that we can compare other approaches to. We'll use the Root Mean Square Error (RMSE) as evaluation statistic.

```
RMSE <- function(observed, predicted) {
  sqrt(mean((predicted - observed)^2, na.rm=TRUE))
}
```

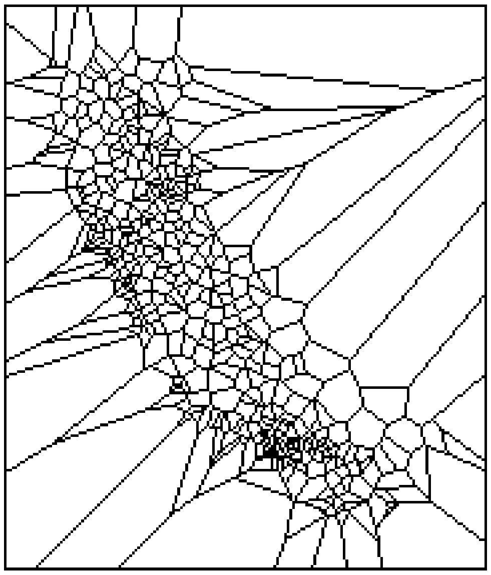
Get the RMSE for the Null-model

```
null <- RMSE(mean(dsp$prec), dsp$prec)
null
## [1] 435.3217
```

4.2.2 proximity polygons

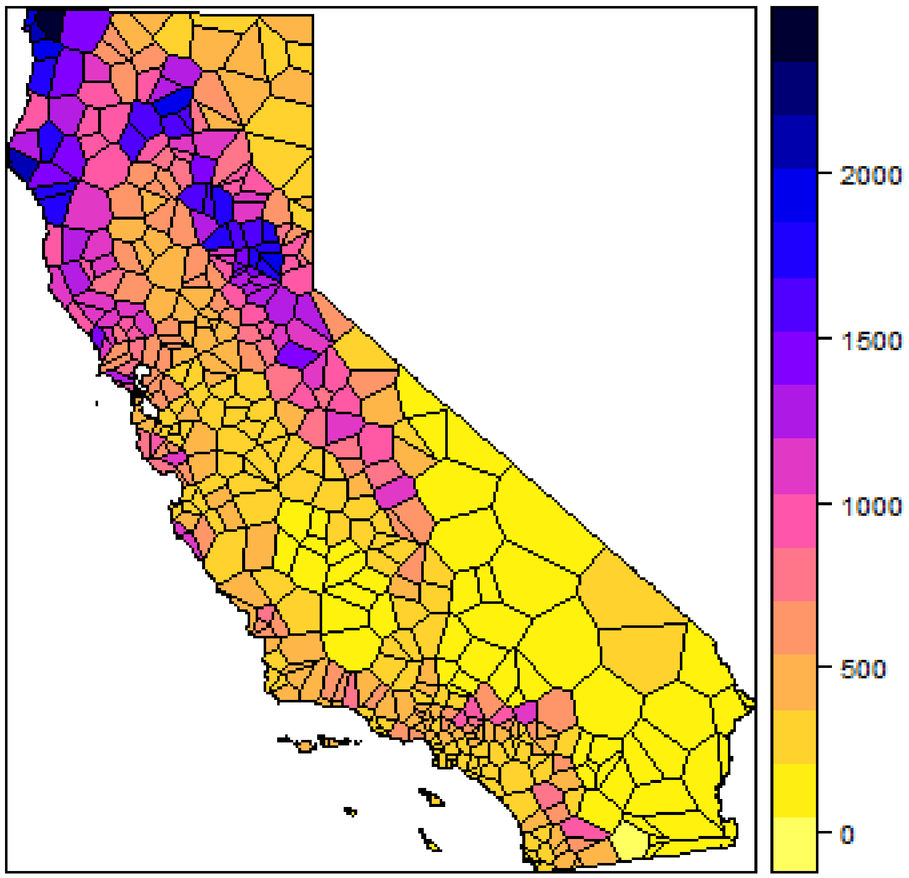
Proximity polygons can be used to interpolate categorical variables. Another term for this is "nearest neighbour" interpolation.

```
library(dismo)
v <- voronoi(dta)
## Loading required namespace: deldir
plot(v)
```



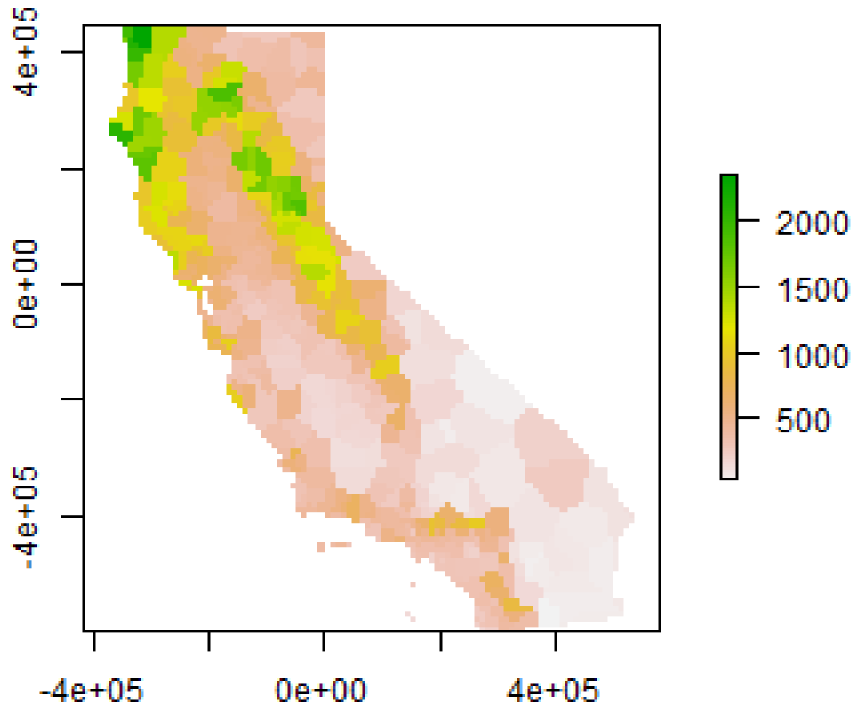
Looks weird. Let's confine this to California

```
ca <- aggregate(cata)
## Loading required namespace: rgeos
vca <- intersect(v, ca)
spplot(vca, 'prec', col.regions=rev(get_col_regions()))
```



Much better. These are polygons. We can 'rasterize' the results like this.

```
r <- raster(cata, res=10000)
vr <- rasterize(vca, r, 'prec')
plot(vr)
```



Now evaluate with 5-fold cross validation.

```
set.seed(5132015)
kf <- kfold(nrow(dta))

rmse <- rep(NA, 5)
for (k in 1:5) {
  test <- dta[kf == k, ]
  train <- dta[kf != k, ]
  v <- voronoi(train)
  p <- extract(v, test)
  rmse[k] <- RMSE(test$prec, p$prec)
}
rmse
## [1] 199.0686 187.8069 166.9153 191.0938 238.9696
mean(rmse)
## [1] 196.7708
1 - (mean(rmse) / null)
## [1] 0.5479875
```

Question 1: *Describe what each step in the code chunk above does*

Question 2: *How does the proximity-polygon approach compare to the NULL model?*

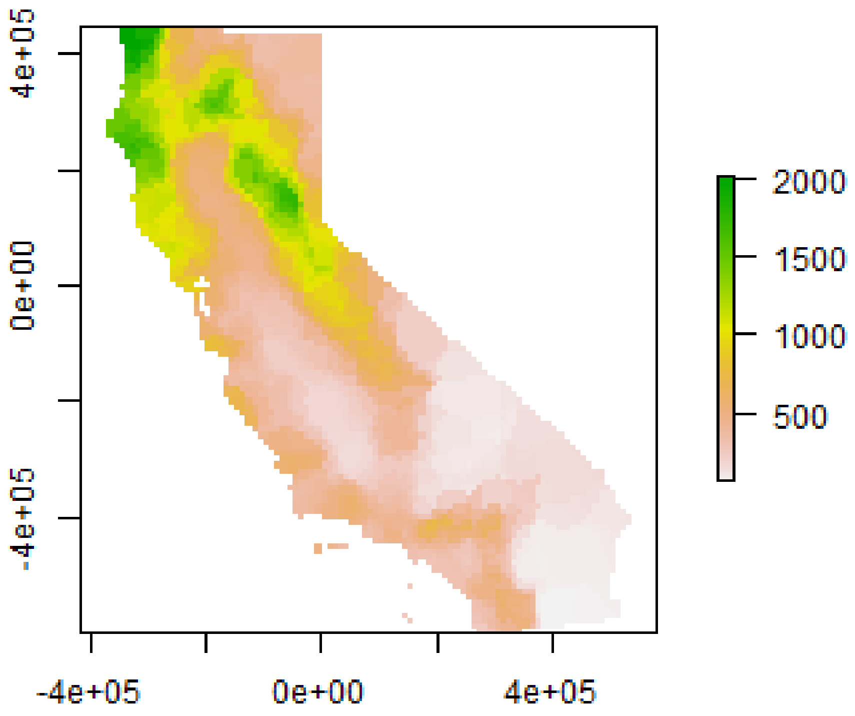
Question 3: *You would not typically use proximty polygons for rainfall data. For what kind of data would you use them?*

4.2.3 Nearest neighbour interpolation

Here we do nearest neighbour interpolation considering multiple (5) neighbours.

We can use the `gstat` package for this. First we fit a model. `~1` means "intercept only". In the case of spatial data, that would be only 'x' and 'y' coordinates are used. We set the maximum number of points to 5, and the "inverse distance power" `idp` to zero, such that all five neighbors are equally weighted

```
library(gstat)
gs <- gstat(formula=prec~1, locations=dta, nmax=5, set=list(idp = 0))
nn <- interpolate(r, gs)
## [inverse distance weighted interpolation]
nnmsk <- mask(nn, vr)
plot(nnmsk)
```



Cross validate the result. Note that we can use the `predict` method to get predictions for the locations of the test points.

```
rmsenn <- rep(NA, 5)
for (k in 1:5) {
  test <- dta[kf == k, ]
  train <- dta[kf != k, ]
  gscv <- gstat(formula=prec~1, locations=train, nmax=5, set=list(idp = 0))
  p <- predict(gscv, test)$var1.pred
```

(continues on next page)

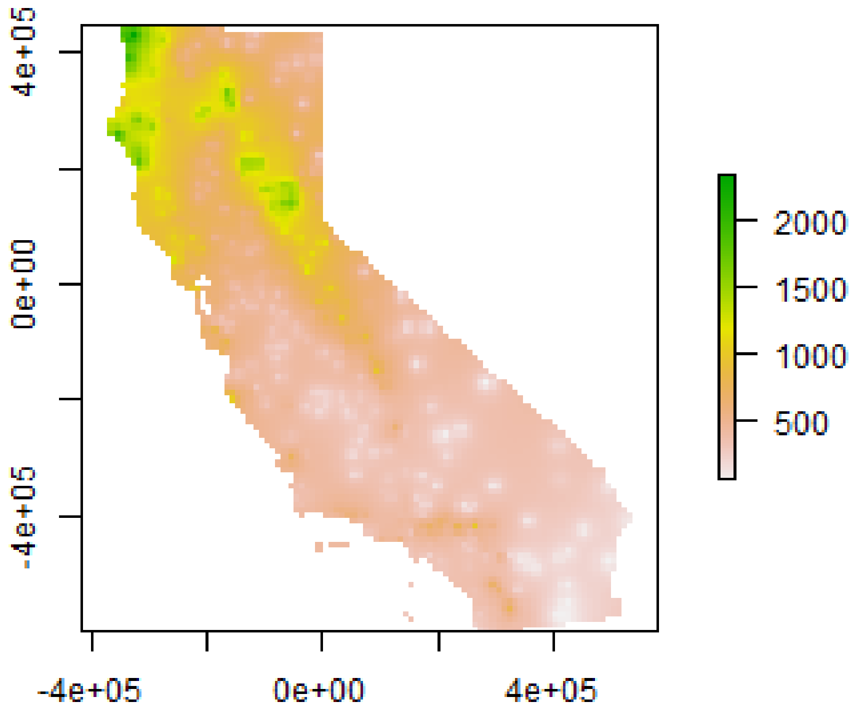
(continued from previous page)

```
  rmsenn[k] <- RMSE(test$prec, p)
}
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
rmsenn
## [1] 200.6222 190.8336 180.3833 169.9658 237.9067
mean(rmsenn)
## [1] 195.9423
1 - (mean(rmsenn) / null)
## [1] 0.5498908
```

4.2.4 Inverse distance weighted

A more commonly used method is "inverse distance weighted" interpolation. The only difference with the nearest neighbour approach is that points that are further away get less weight in predicting a value a location.

```
library(gstat)
gs <- gstat(formula=prec~1, locations=dta)
idw <- interpolate(r, gs)
## [inverse distance weighted interpolation]
idwr <- mask(idw, vr)
plot(idwr)
```



Question 4: *IDW generated rasters tend to have a noticeable artefact. What is that?*

Cross validate. We can predict to the locations of the test points

```
rmse <- rep(NA, 5)
for (k in 1:5) {
  test <- dta[kf == k, ]
  train <- dta[kf != k, ]
  gs <- gstat(formula=prec~1, locations=train)
  p <- predict(gs, test)
  rmse[k] <- RMSE(test$prec, p$var1.pred)
}
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
## [inverse distance weighted interpolation]
rmse
## [1] 215.3319 211.9383 190.0231 211.8308 230.1893
mean(rmse)
## [1] 211.8627
1 - (mean(rmse) / null)
## [1] 0.5133192
```

Question 5: *Inspect the arguments used for and make a map of the IDW model below. What other name could you* *give to this method (IDW with these parameters)? Why?*

```
gs2 <- gstat(formula=prec~1, locations=dta, nmax=1, set=list(idp=1))
```

4.3 Calfornia Air Pollution data

We use California Air Pollution data to illustrate geostatistcal (Kriging) interpolation.

4.3.1 Data preparation

We use the airqual dataset to interpolate ozone levels for California (averages for 1980-2009). Use the variable `OZDLYAV` (unit is parts per billion). Original data source.

Read the data file. To get easier numbers to read, I multiply OZDLYAV with 1000

```
x <- sp_data("airqual.csv")
x$OZDLYAV <- x$OZDLYAV * 1000
```

Create a SpatialPointsDataFrame and transform to Teale Albers. Note the `units=km` which was needed to fit the variogram.

```
library(sp)
coordinates(x) <- ~LONGITUDE + LATITUDE
proj4string(x) <- CRS('+proj=longlat +datum=NAD83')
TA <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000␣
↪+datum=NAD83 +units=km +ellps=GRS80")
library(rgdal)
aq <- spTransform(x, TA)
```

Create an template raster to interpolate to. E.g., given a SpatialPolygonsDataFrame of California, 'ca'. Coerce that to a 'SpatialGrid' object (a different representation of the same idea)

```
cageo <- sp_data('counties.rds')
ca <- spTransform(cageo, TA)
r <- raster(ca)
res(r) <- 10  # 10 km if your CRS's units are in km
g <- as(r, 'SpatialGrid')
```

4.3.2 Fit a variogram

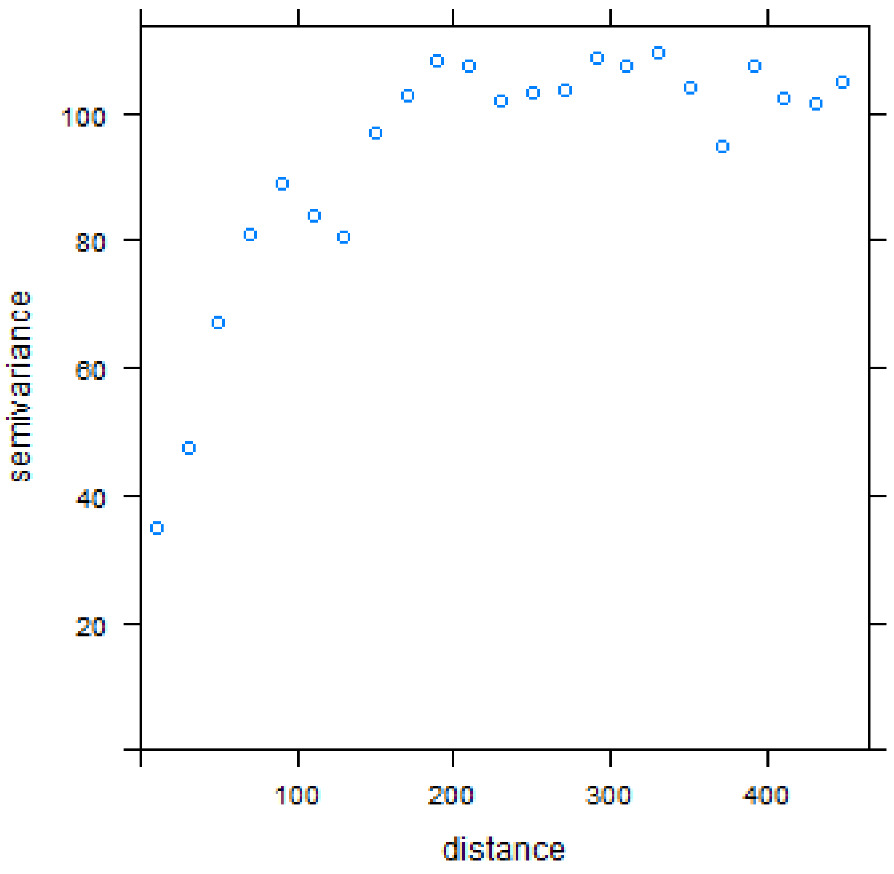
Use gstat to create an emperical variogram 'v'

```
library(gstat)
gs <- gstat(formula=OZDLYAV~1, locations=aq)
v <- variogram(gs, width=20)
head(v)
##     np      dist    gamma dir.hor dir.ver   id
## 1 1010  11.35040 34.80579       0       0 var1
## 2 1806  30.63737 47.52591       0       0 var1
## 3 2355  50.58656 67.26548       0       0 var1
## 4 2619  70.10411 80.92707       0       0 var1
## 5 2967  90.13917 88.93653       0       0 var1
```

(continues on next page)

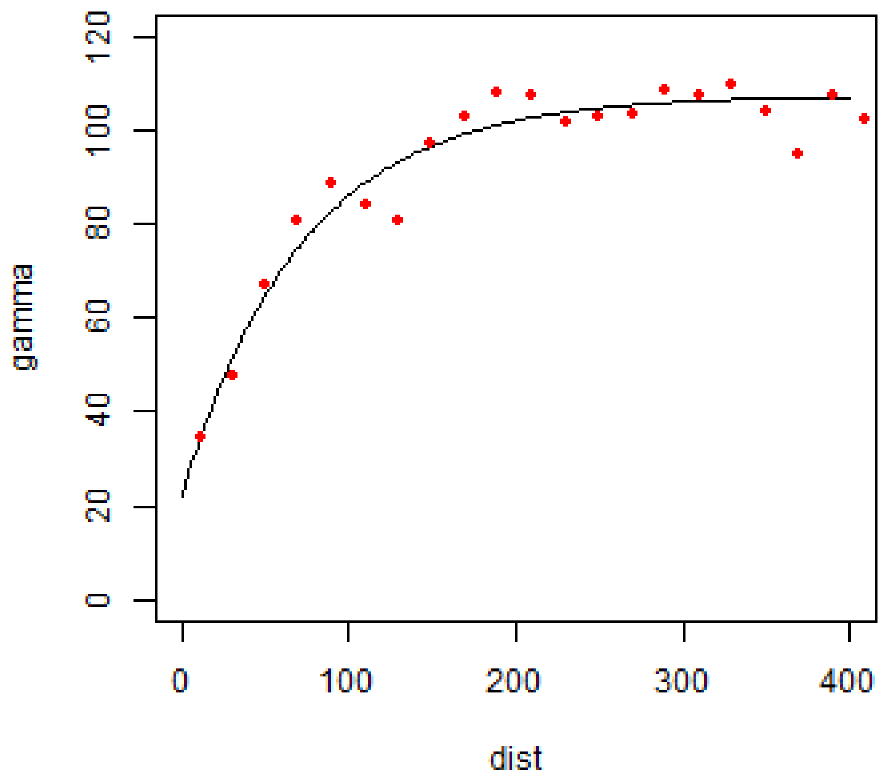
(continued from previous page)

```
## 6 3437 110.42302 84.13589       0        0 var1
plot(v)
```



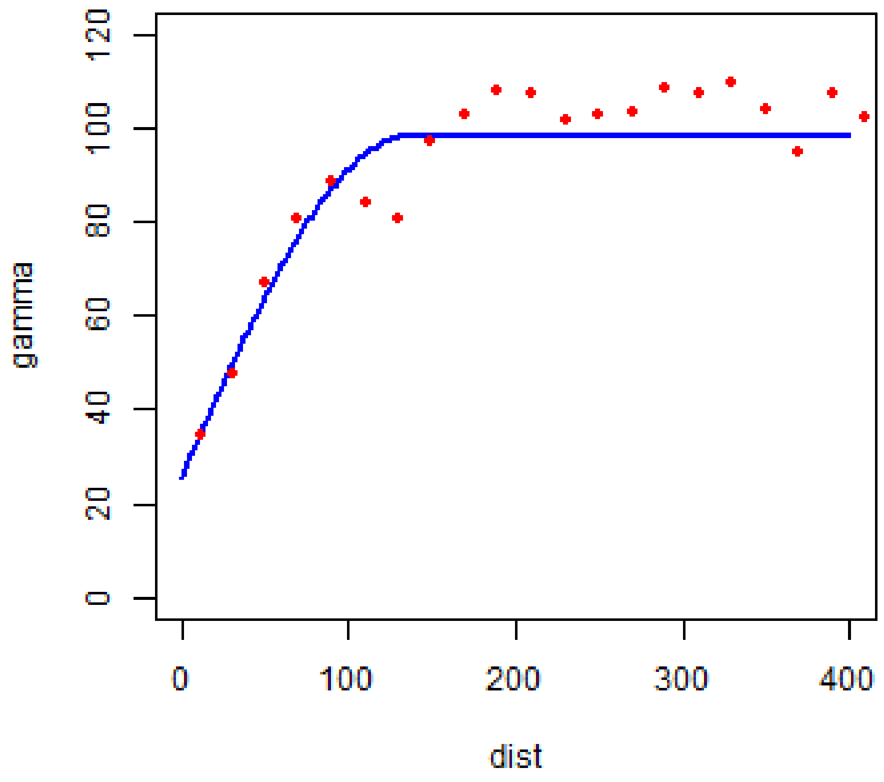
Now, fit a model variogram

```
fve <- fit.variogram(v, vgm(85, "Exp", 75, 20))
fve
##   model    psill    range
## 1   Nug 21.96600  0.00000
## 2   Exp 85.52957 72.31404
plot(variogramLine(fve, 400), type='l', ylim=c(0,120))
points(v[,2:3], pch=20, col='red')
```



Try a different type (spherical in stead of exponential)

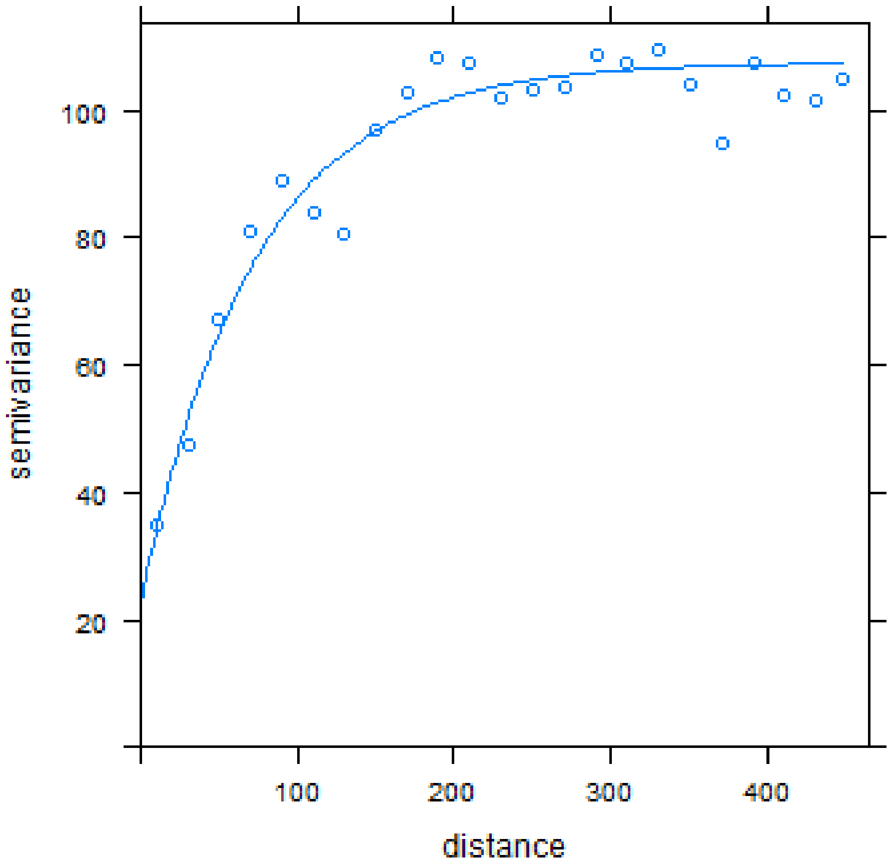
```
fvs <- fit.variogram(v, vgm(85, "Sph", 75, 20))
fvs
##   model     psill    range
## 1   Nug 25.57019   0.0000
## 2   Sph 72.65881 135.7744
plot(variogramLine(fvs, 400), type='l', ylim=c(0,120) ,col='blue', lwd=2)
points(v[,2:3], pch=20, col='red')
```



Both look pretty good in this case.

Another way to plot the variogram and the model

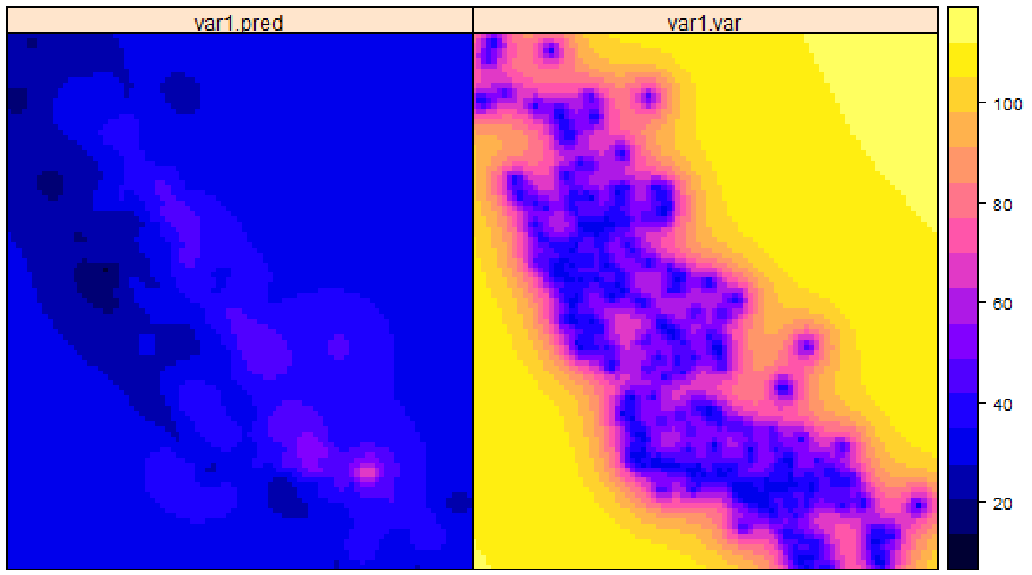
```
plot(v, fve)
```



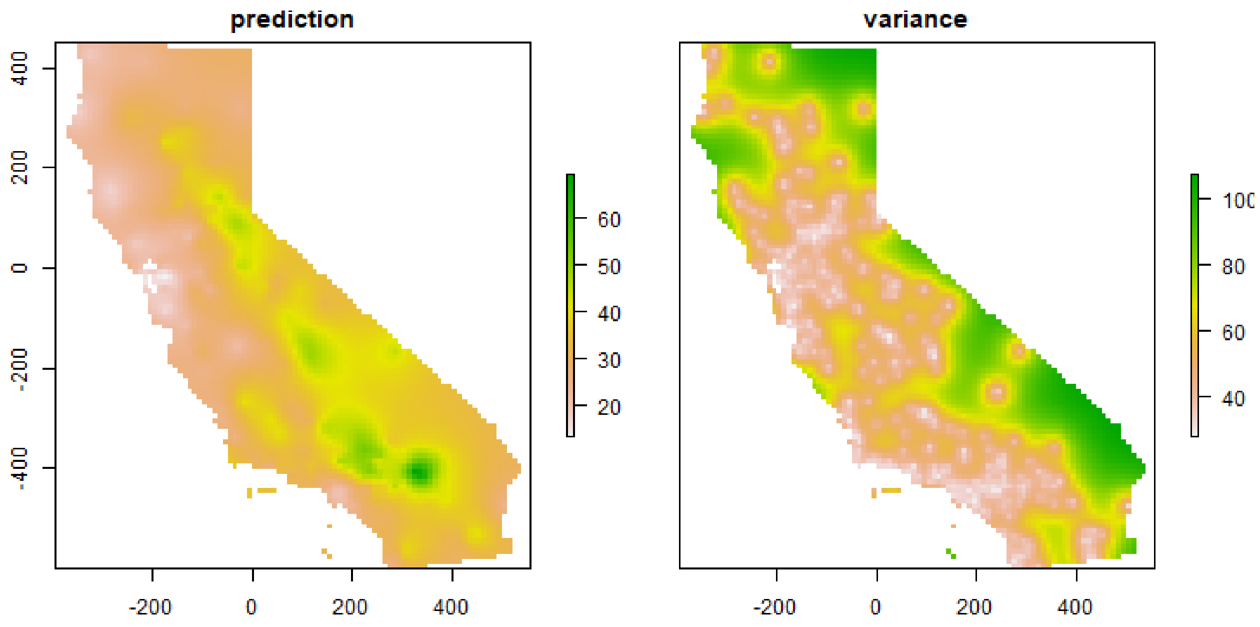
4.3.3 Ordinary kriging

Use variogram `fve` in a kriging interpolation

```
k <- gstat(formula=OZDLYAV~1, locations=aq, model=fve)
# predicted values
kp <- predict(k, g)
## [using ordinary kriging]
spplot(kp)
```



```
# variance
ok <- brick(kp)
ok <- mask(ok, ca)
names(ok) <- c('prediction', 'variance')
plot(ok)
```



4.3.4 Compare with other methods

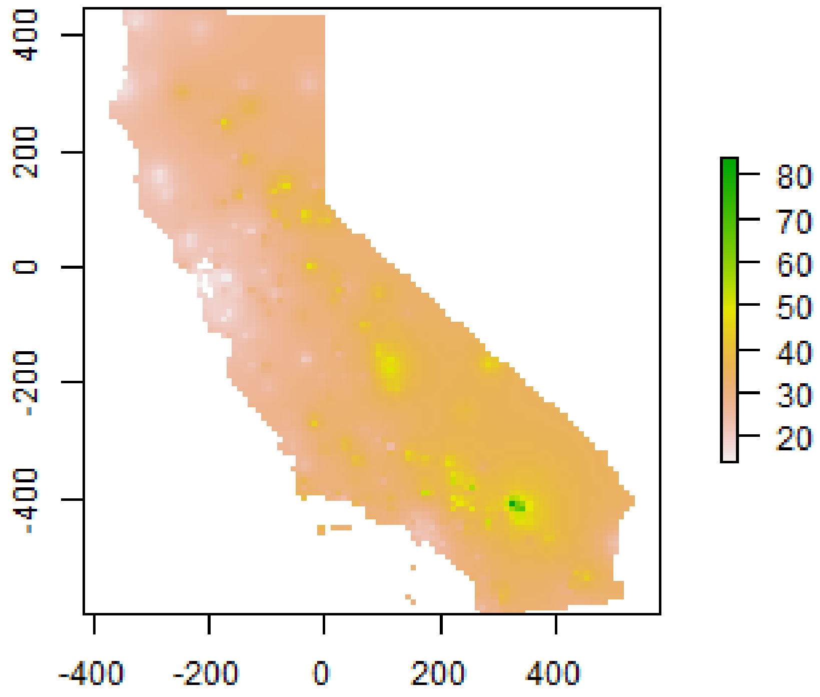
Let's use gstat again to do IDW interpolation. The basic approach first.

```
library(gstat)
idm <- gstat(formula=OZDLYAV~1, locations=aq)
```

(continues on next page)

(continued from previous page)

```
idp <- interpolate(r, idm)
## [inverse distance weighted interpolation]
idp <- mask(idp, ca)
plot(idp)
```



We can find good values for the idw parameters (distance decay and number of neighbours) through optimization. For simplicity's sake I do not do that *k* times here. The optim function may be a bit hard to grasp at first. But the essence is simple. You provide a function that returns a value that you want to minimize (or maximize) given a number of unknown parameters. Your provide initial values for these parameters, and optim then searches for the optimal values (for which the function returns the lowest number).

```
RMSE <- function(observed, predicted) {
  sqrt(mean((predicted - observed)^2, na.rm=TRUE))
}

f1 <- function(x, test, train) {
  nmx <- x[1]
  idp <- x[2]
  if (nmx < 1) return(Inf)
  if (idp < .001) return(Inf)
  m <- gstat(formula=OZDLYAV~1, locations=train, nmax=nmx, set=list(idp=idp))
  p <- predict(m, newdata=test, debug.level=0)$var1.pred
  RMSE(test$OZDLYAV, p)
```

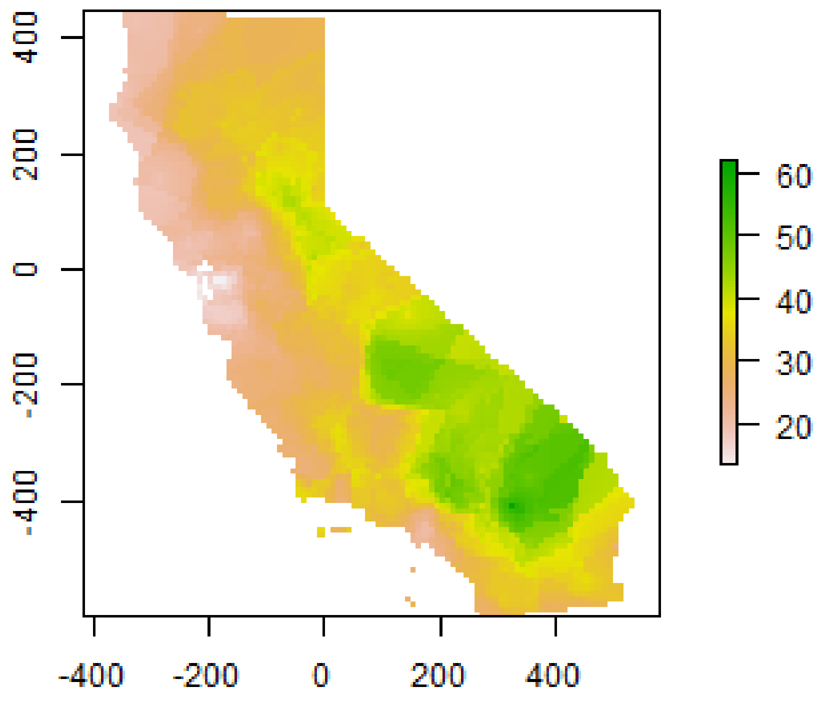
(continues on next page)

(continued from previous page)

```
}
set.seed(20150518)
i <- sample(nrow(aq), 0.2 * nrow(aq))
tst <- aq[i,]
trn <- aq[-i,]
opt <- optim(c(8, .5), f1, test=tst, train=trn)
opt
## $par
## [1] 9.1569933 0.4521968
##
## $value
## [1] 7.28701
##
## $counts
## function gradient
##       41       NA
##
## $convergence
## [1] 0
##
## $message
## NULL
```

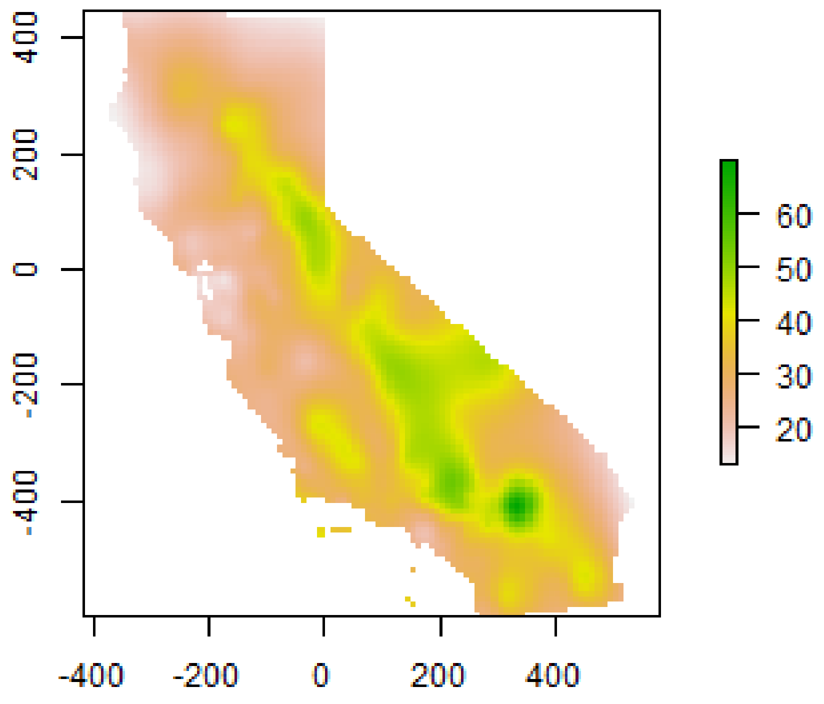
Our optimal IDW model

```
m <- gstat(formula=OZDLYAV~1, locations=aq, nmax=opt$par[1], set=list(idp=opt$par[2]))
idw <- interpolate(r, m)
## [inverse distance weighted interpolation]
idw <- mask(idw, ca)
plot(idw)
```



A thin plate spline model

```
library(fields)
m <- Tps(coordinates(aq), aq$OZDLYAV)
tps <- interpolate(r, m)
tps <- mask(tps, idw)
plot(tps)
```



4.3.5 Cross-validate

Cross-validate the three methods (IDW, Ordinary kriging, TPS) and add RMSE weighted ensemble model.

```
library(dismo)

nfolds <- 5
k <- kfold(aq, nfolds)

ensrmse <- tpsrmse <- krigrmse <- idwrmse <- rep(NA, 5)

for (i in 1:nfolds) {
  test <- aq[k!=i,]
  train <- aq[k==i,]
  m <- gstat(formula=OZDLYAV~1, locations=train, nmax=opt$par[1], set=list(idp=opt
↪$par[2]))
  p1 <- predict(m, newdata=test, debug.level=0)$var1.pred
  idwrmse[i] <-  RMSE(test$OZDLYAV, p1)

  m <- gstat(formula=OZDLYAV~1, locations=train, model=fve)
  p2 <- predict(m, newdata=test, debug.level=0)$var1.pred
  krigrmse[i] <-  RMSE(test$OZDLYAV, p2)
```

(continues on next page)

(continued from previous page)

```
  m <- Tps(coordinates(train), train$OZDLYAV)
  p3 <- predict(m, coordinates(test))
  tpsrmse[i] <-  RMSE(test$OZDLYAV, p3)

  w <- c(idwrmse[i], krigrmse[i], tpsrmse[i])
  weights <- w / sum(w)
  ensemble <- p1 * weights[1] + p2 * weights[2] + p3 * weights[3]
  ensrmse[i] <-  RMSE(test$OZDLYAV, ensemble)

}
rmi <- mean(idwrmse)
rmk <- mean(krigrmse)
rmt <- mean(tpsrmse)
rms <- c(rmi, rmt, rmk)
rms
## [1] 7.925989 8.816963 7.588549
rme <- mean(ensrmse)
rme
## [1] 7.718896
```

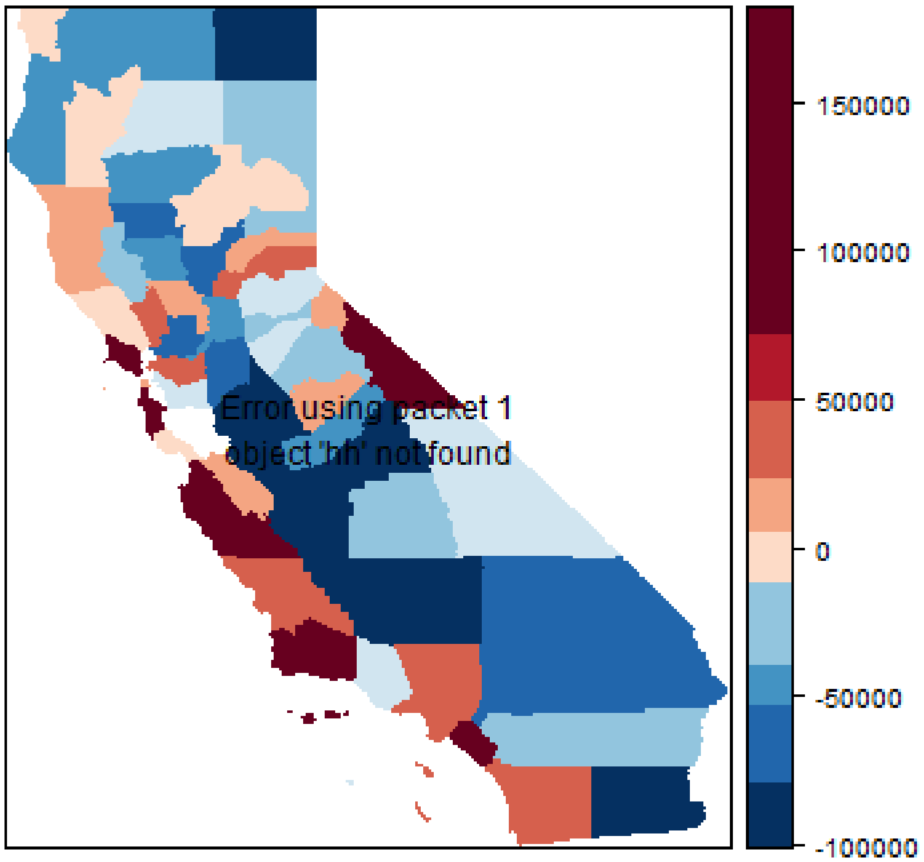
Question 6: *Which method performed best?*

We can use the rmse scores to make a weighted ensemble. Let's look at the maps

```
weights <- ( rms / sum(rms) )
s <- stack(idw, ok[[1]], tps)
ensemble <- sum(s * weights)
```

And compare maps.

```
s <- stack(idw, ok[[1]], tps, ensemble)
names(s) <- c('IDW', 'OK', 'TPS', 'Ensemble')
plot(s)
```



Question 7: *Show where the largest difference exist between IDW and OK.*

Question 8: *Show where the difference between IDW and OK is within the 95% confidence limit of the OK prediction.*

Question 9: *Can you describe the pattern we are seeing, and speculate about what is causing it?*

CHAPTER

FIVE

5. SPATIAL DISTRIBUTION MODELS

This page shows how you can use the Random Forest algorithm to make spatial predictions. This approach is widely used, for example to classify remote sensing data into different land cover classes. But here our objective is to predict the entire range of a species based on a set of locations where it has been observed. As an example, we use the hominid species *Imaginus magnapedum* (also known under the vernacular names of "bigfoot" and "sasquatch"). This species is so hard to find (at least by scientists) that its very existence is commonly denied by the mainstream media! For more information about this controversy, see the article by Lozier, Aniello and Hickerson: Predicting the distribution of Sasquatch in western North America: anything goes with ecological niche modelling.

We want to find out

1. What the complete range of the species might be.
2. How good (general) our model is by predicting the range of the Eastern sub-species, with data from the Western sub-species.
3. Predict where in Mexico the creature is likely to occur.
4. How climate change might affect its distribution.

In this context, this type of analysis is often referred to as 'species distribution modeling' or 'ecological niche modeling'. Here is a more in-depth discussion of this technique.

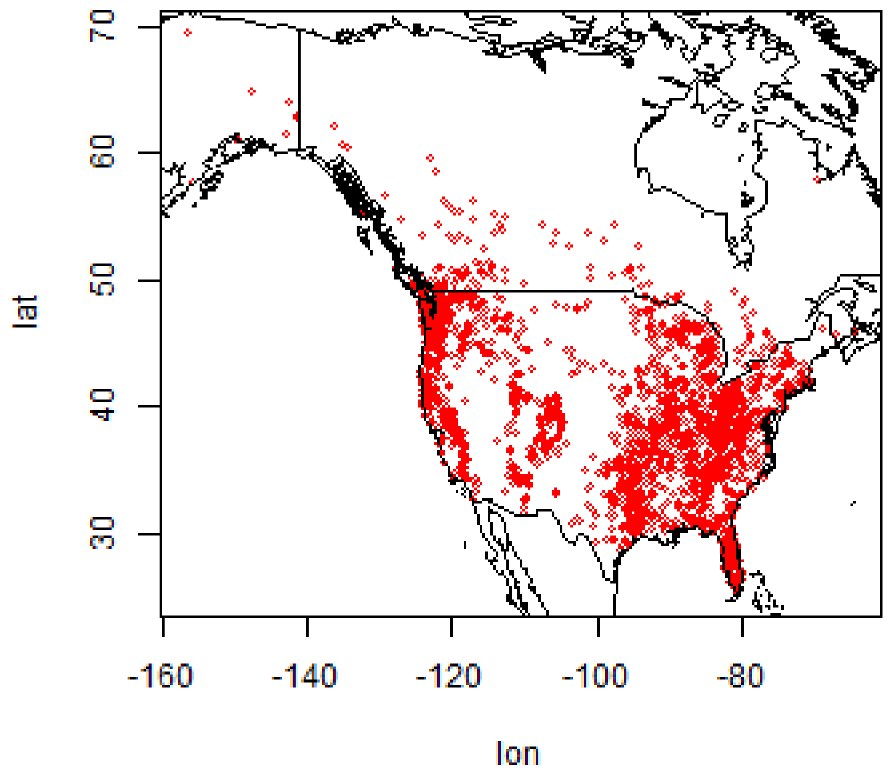
5.1 Data

5.1.1 Observations

```
# If have not yet done so, first install the rspatial package (see the line below).␣
↪You may need to install the devtools package first
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
library(rspatial)
bf <- sp_data('bigfoot')
dim(bf)
## [1] 3092    3
head(bf)
##         lon      lat Class
## 1 -142.9000 61.50000     A
## 2 -132.7982 55.18720     A
## 3 -132.8202 55.20350     A
## 4 -141.5667 62.93750     A
## 5 -149.7853 61.05950     A
## 6 -141.3165 62.77335     A
```

Plot the locations

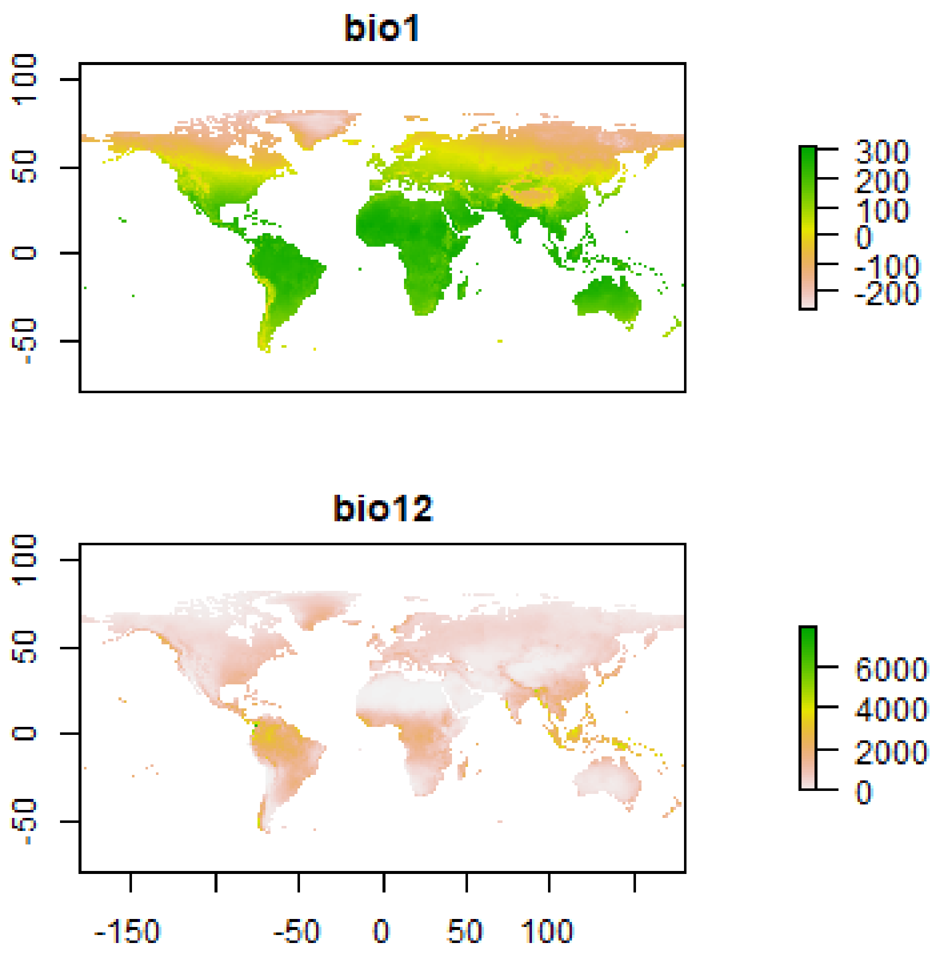
```
plot(bf[,1:2], cex=0.5, col='red')
library(maptools)
## Checking rgeos availability: TRUE
data(wrld_simpl)
plot(wrld_simpl, add=TRUE)
```



5.1.2 Predictors

Supervised classification often uses predictor data obtained from satellite remote sensing. But here, as is common in species distribution modeling, we use climate data. Specifically, we use 'bioclimatic variables', see: http://www.worldclim.org/bioclim

```
library(raster)
wc <- getData('worldclim', res=10, var='bio')
plot(wc[[c(1, 12)]], nr=2)
```



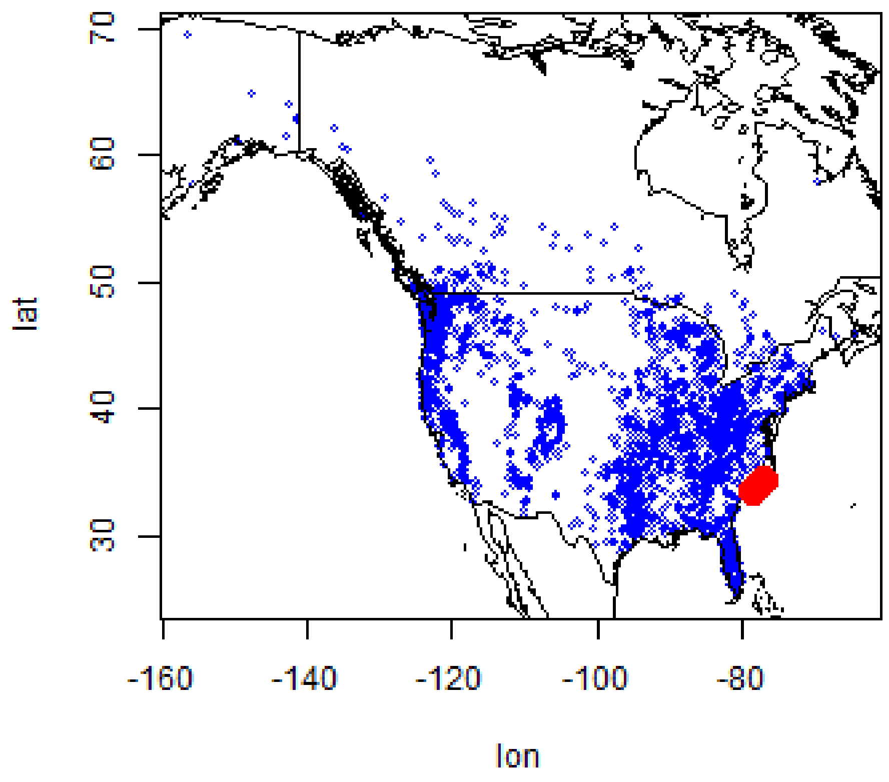
Now extract climate data for the locations of our observations. That is, get data about the climate that the species likes, apparently.

```
bfc <- extract(wc, bf[,1:2])
head(bfc)
##      bio1 bio2 bio3  bio4 bio5 bio6 bio7 bio8 bio9 bio10 bio11 bio12 bio13
## [1,]  -14  102   27  9672  174 -197  371   51  -11   108  -137   973   119
## [2,]   62   55   31  4136  157  -17  174   43   98   118    15  2602   385
## [3,]   62   55   31  4136  157  -17  174   43   98   118    15  2602   385
## [4,]  -57  125   23 15138  206 -332  538  127 -129   127  -256   282    67
## [5,]   10   80   25  8308  174 -140  314   66    5   119   -91   532    81
## [6,]  -59  128   23 14923  204 -334  538  122 -130   122  -255   322    75
##      bio14 bio15 bio16 bio17 bio18 bio19
## [1,]    43    30   332   156   290   210
## [2,]   128    33   953   407   556   721
## [3,]   128    33   953   407   556   721
## [4,]     6    81   163    22   163    27
## [5,]    22    41   215    72   159   117
## [6,]     8    79   183    28   183    32

# Any missing values?
i <- which(is.na(bfc[,1]))
i
## [1]  862 2667
```

(continues on next page)

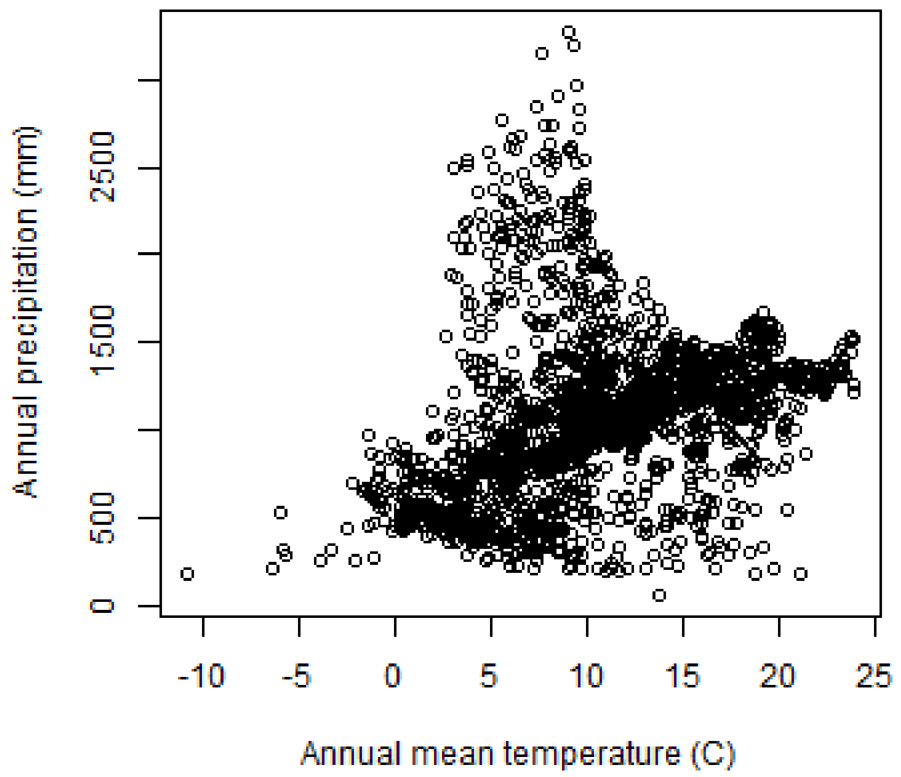
(continued from previous page)

```
plot(bf[,1:2], cex=0.5, col='blue')
plot(wrld_simpl, add=TRUE)
points(bf[i, ], pch=20, cex=3, col='red')
## Warning in data.matrix(x): NAs introduced by coercion
```



Here is a plot that illustrates a component of the ecological niche of our species of interest.

```
plot(bfc[ ,'bio1'] / 10, bfc[, 'bio12'], xlab='Annual mean temperature (C)',
         ylab='Annual precipitation (mm)')
```



5.1.3 Background data

Normally, one would build a model that would compare the values of the predictor variables as the locations where something was observed, with those values at the locations where it was not observed. But we do not have data from a systematic survey that determined presence and absence. We have presence-only data. (And, determining absence is not that simple. It is here now, it is gone tomorrow).

The common trick to deal with this is to not model presence versus absence, but presence versus a 'random expectation'. This random expectation (also referred to as 'background', or 'random-absence' data) is what you would get if the species had no preference for any of the predictor variables (or to other variables that are not in the model, but correlated with the predictor variables).

There is not much point in taking absence data from very far away (tropical Africa or Antarctica). Typically they are taken from more or less the entire study area for which we have presences data.

```
library(dismo)
# extent of all points
e <- extent(SpatialPoints(bf[, 1:2]))
e
## class       : Extent
## xmin        : -156.75
## xmax        : -64.4627
## ymin        : 25.141
```

(continues on next page)

(continued from previous page)

```
## ymax        : 69.5

# 5000 random samples (excluding NA cells) from extent e
set.seed(0)
bg <- sampleRandom(wc, 5000, ext=e)
dim(bg)
## [1] 5000   19
head(bg)
##      bio1 bio2 bio3  bio4 bio5 bio6 bio7 bio8 bio9 bio10 bio11 bio12 bio13
## [1,]  157  126   60  2935  262   55  207  124  191   197   122   379    88
## [2,]  -54  105   28  9244  142 -223  365   57  -62    68  -165   639    79
## [3,]  -57  104   20 14227  198 -317  515  106 -227   118  -247   473    71
## [4,]    1  119   24 12335  231 -251  482  138  -91   150  -168   844   104
## [5,]  208  169   44  7641  404   28  376  304  239   307   114   198    31
## [6,]  -89  111   23 12931  160 -316  476   78 -174    78  -248   476    76
##      bio14 bio15 bio16 bio17 bio18 bio19
## [1,]     0   100   225     2     4   222
## [2,]    28    30   226   101   219   138
## [3,]    17    46   197    55   194    59
## [4,]    34    33   301   128   291   137
## [5,]     2    50    73    11    52    62
## [6,]    25    40   193    79   193    82
```

5.1.4 Combine presence and background

```
d <- rbind(cbind(pa=1, bfc), cbind(pa=0, bg))
d <- data.frame(d)
dim(d)
## [1] 8092   20
```

5.2 Fit a model

Now we have the data to fit a model. But I am going to split the data into East and West. Let's say I believe these are actually are different, albeit related, sub-species (The Eastern Sasquatch is darker and less hairy). I am principally interested in the western sub-species.

```
de <- d[bf[,1] > -102, ]
dw <- d[bf[,1] <= -102, ]
```

5.2.1 CART

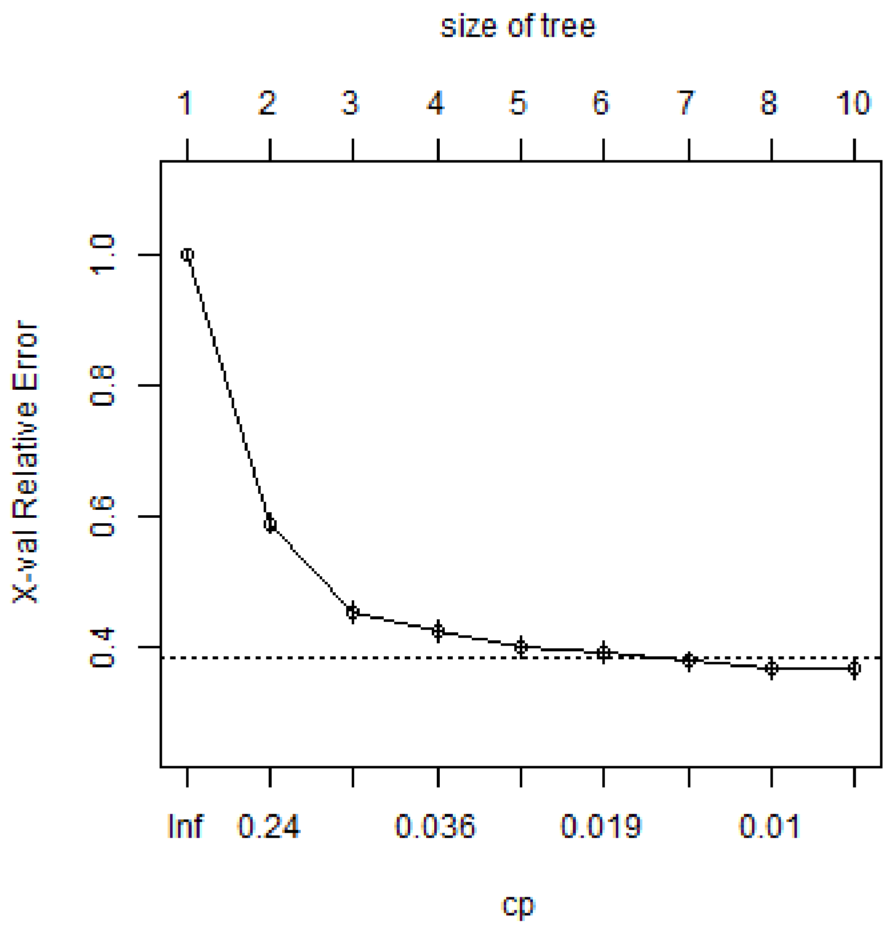
Let's first look at a Classification and Regression Trees (CART) model.

```
library(rpart)
cart <- rpart(pa~., data=dw)
printcp(cart)
##
## Regression tree:
## rpart(formula = pa ~ ., data = dw)
##
```

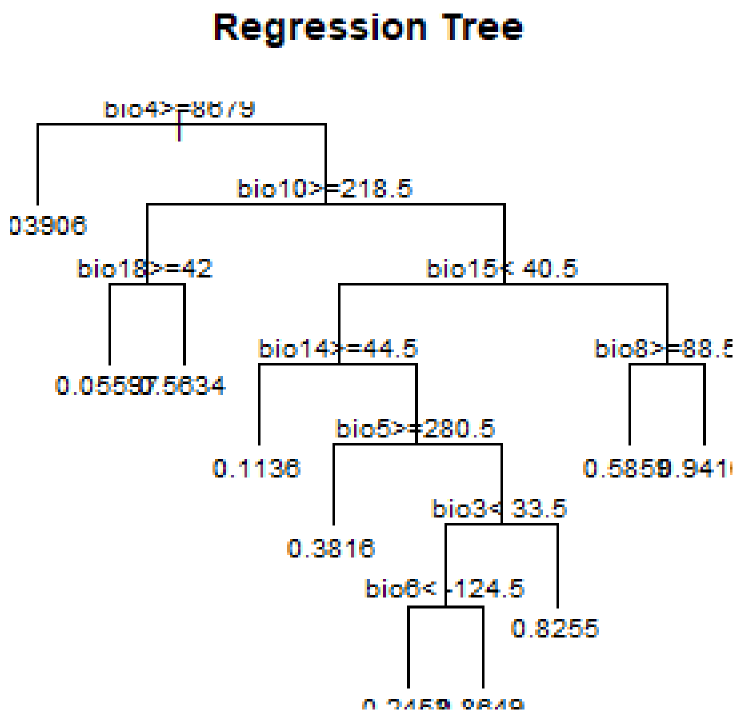
(continues on next page)

(continued from previous page)

```
## Variables actually used in tree construction:
## [1] bio10 bio14 bio15 bio18 bio3  bio4  bio5  bio6  bio8
##
## Root node error: 762.45/3246 = 0.23489
##
## n= 3246
##
##         CP nsplit rel error  xerror     xstd
## 1 0.410197      0   1.00000 1.00048 0.008909
## 2 0.137588      1   0.58980 0.59041 0.014191
## 3 0.044259      2   0.45222 0.45474 0.016586
## 4 0.029121      3   0.40796 0.42701 0.016572
## 5 0.018954      4   0.37884 0.40239 0.016586
## 6 0.018324      5   0.35988 0.39294 0.016337
## 7 0.010113      6   0.34156 0.37926 0.015777
## 8 0.010008      7   0.33144 0.36821 0.016014
## 9 0.010000      9   0.31143 0.36821 0.016014
plotcp(cart)
```



```
plot(cart, uniform=TRUE, main="Regression Tree")
text(cart, cex=.8)
```



```
# text(cart, use.n=TRUE, all=TRUE, cex=.8)
```

Question 1: *Describe the conditions under which you have the highest probability of finding our beloved species?*

5.2.2 Random Forest

CART gives us a nice result to look at that can be easily interpreted (as you just illustrated with your answer to Question 1). But the approach suffers from high variance (meaning that the model will be over-fit, it is different each time a somewhat different datasets are used). Random Forest does not have that problem as much. Above, with CART, we use regression, let's do both regression and classification here. First classification.

```
library(randomForest)
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
# create a factor to indicated that we want classification
fpa <- as.factor(dw[, 'pa'])
```

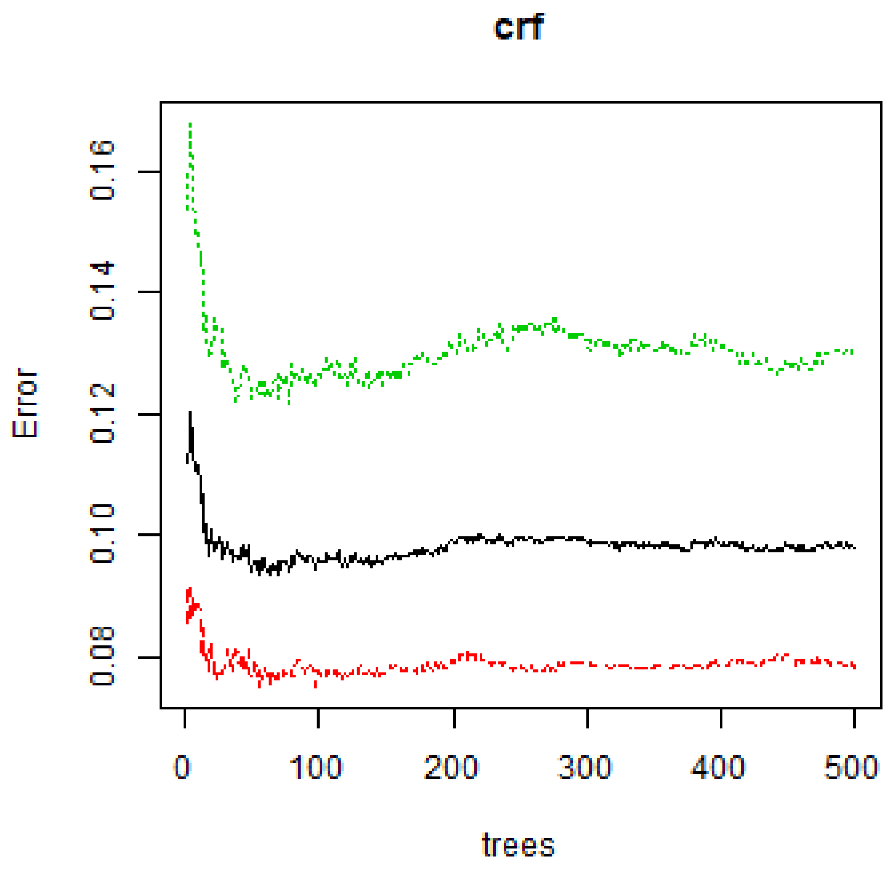
Now fit the RandomForest model

```
crf <- randomForest(dw[, 2:ncol(dw)], fpa)
crf
##
```

(continues on next page)

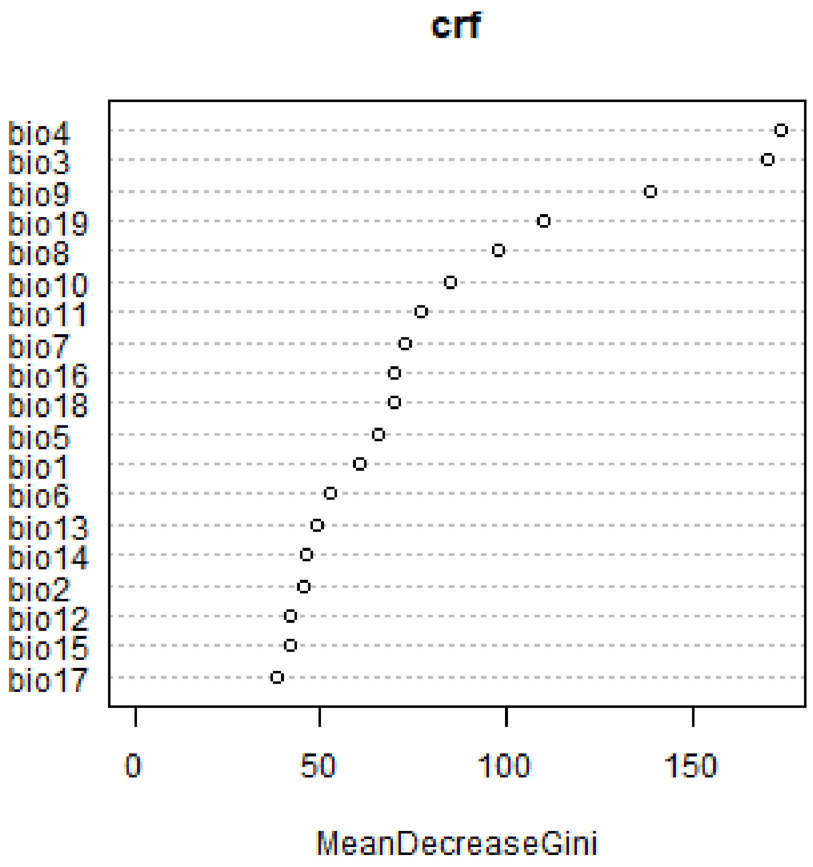
(continued from previous page)

```
## Call:
##  randomForest(x = dw[, 2:ncol(dw)], y = fpa)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 9.8%
## Confusion matrix:
##      0    1 class.error
## 0 1864  158  0.07814045
## 1  160 1064  0.13071895
plot(crf)
```



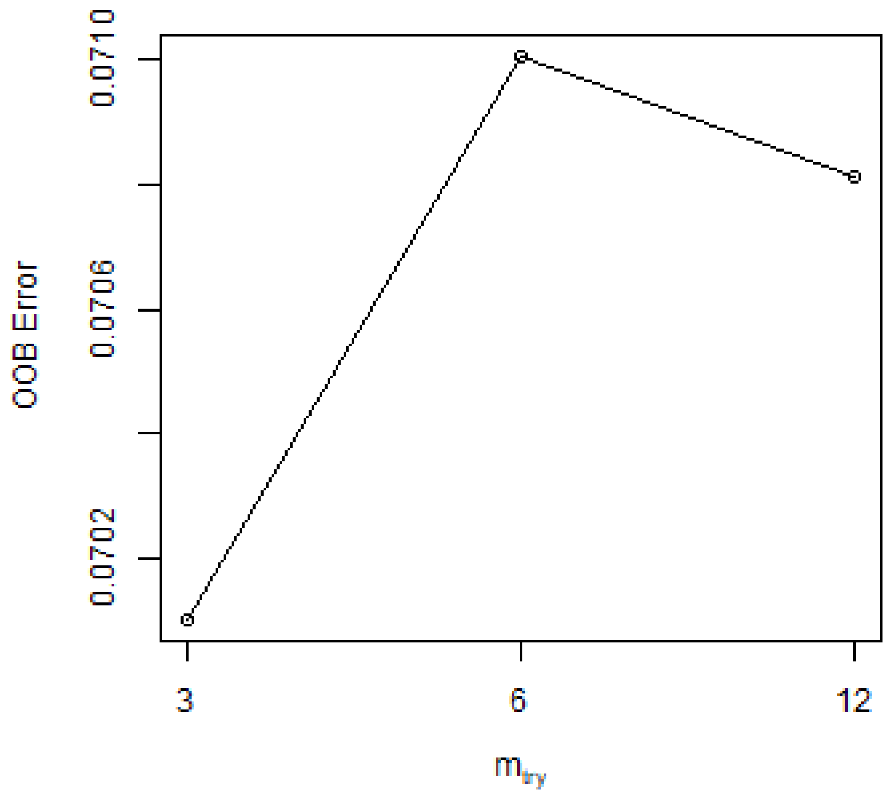
The variable importance plot shows which variables are most important in fitting the model. This is computing by randomizing each variable one by one and then computing the decline in model prediction.

```
varImpPlot(crf)
```



Now we use regression, rather than classification. First we tune a parameter.

```
trf <- tuneRF(dw[, 2:ncol(dw)], dw[, 'pa'])
## Warning in randomForest.default(x, y, mtry = mtryStart, ntree = ntreeTry, :
## The response has five or fewer unique values. Are you sure you want to do
## regression?
## mtry = 6  OOB error = 0.07100579
## Searching left ...
## Warning in randomForest.default(x, y, mtry = mtryCur, ntree = ntreeTry, :
## The response has five or fewer unique values. Are you sure you want to do
## regression?
## mtry = 3     OOB error = 0.07010267
## 0.01271897 0.05
## Searching right ...
## Warning in randomForest.default(x, y, mtry = mtryCur, ntree = ntreeTry, :
## The response has five or fewer unique values. Are you sure you want to do
## regression?
## mtry = 12    OOB error = 0.0708133
## 0.002710948 0.05
```



```
trf
##    mtry   OOBError
## 3     3 0.07010267
## 6     6 0.07100579
## 12   12 0.07081330
mt <- trf[which.min(trf[,2]), 1]
mt
## [1] 3
```

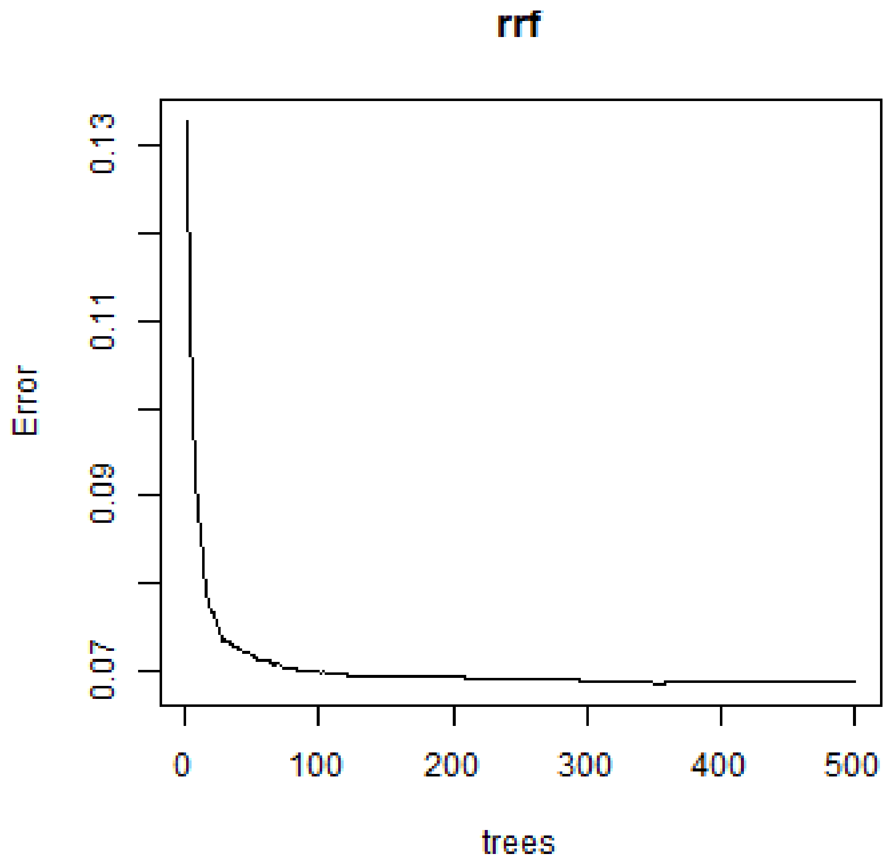
Question 2: *What did tuneRF help us find? What does the values of mt represent?*

```
rrf <- randomForest(dw[, 2:ncol(d)], dw[, 'pa'], mtry=mt)
## Warning in randomForest.default(dw[, 2:ncol(d)], dw[, "pa"], mtry = mt):
## The response has five or fewer unique values. Are you sure you want to do
## regression?
rrf
##
## Call:
##  randomForest(x = dw[, 2:ncol(d)], y = dw[, "pa"], mtry = mt)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 3
##
##           Mean of squared residuals: 0.06855845
```

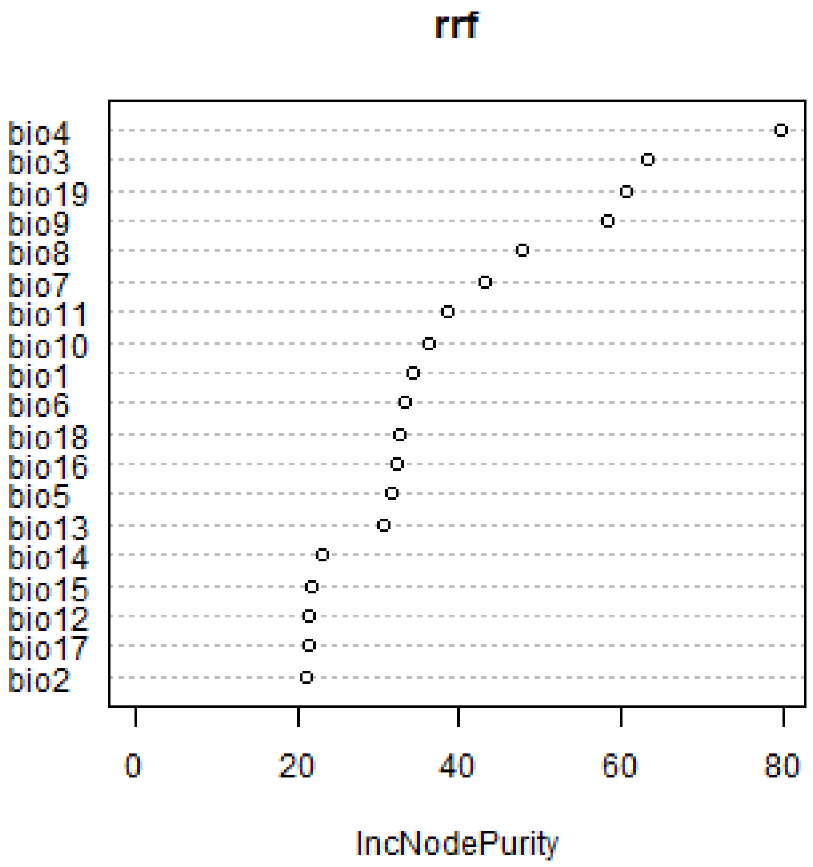
(continues on next page)

(continued from previous page)

```
##                      % Var explained: 70.81
plot(rrf)
```



```
varImpPlot(rrf)
```



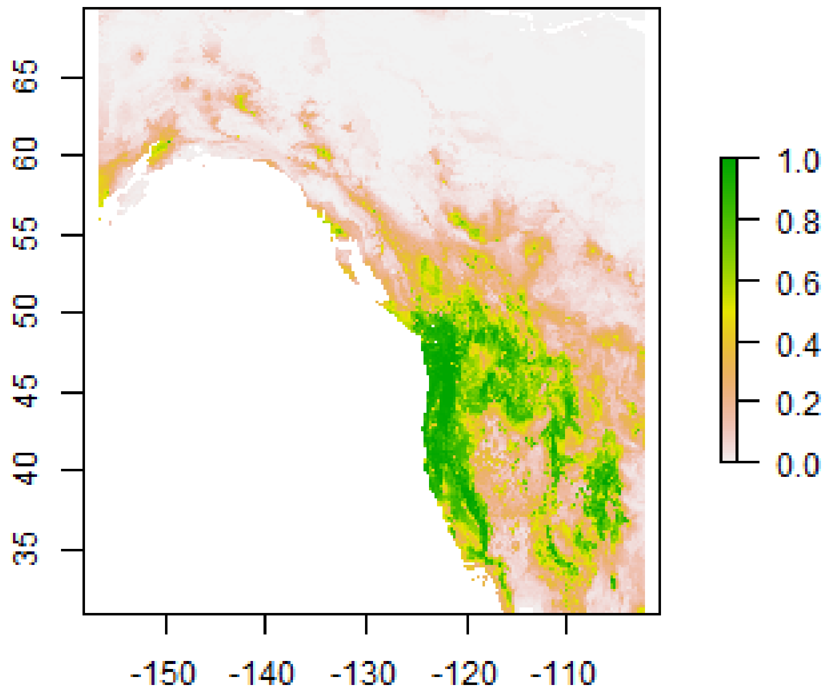
5.3 Predict

We can use the model to make predictions to any other place for which we have values for the predictor variables. Our climate data is global so we could find suitable places for bigfoot in Australia. At first I only want to predict to our study region, which I define as follows.

```
# Extent of the western points
ew <- extent(SpatialPoints(bf[bf[,1] <= -102, 1:2]))
ew
## class       : Extent
## xmin        : -156.75
## xmax        : -102.3881
## ymin        : 30.77722
## ymax        : 69.5
```

5.3.1 Regression

```
rp <- predict(wc, rrf, ext=ew)
plot(rp)
```

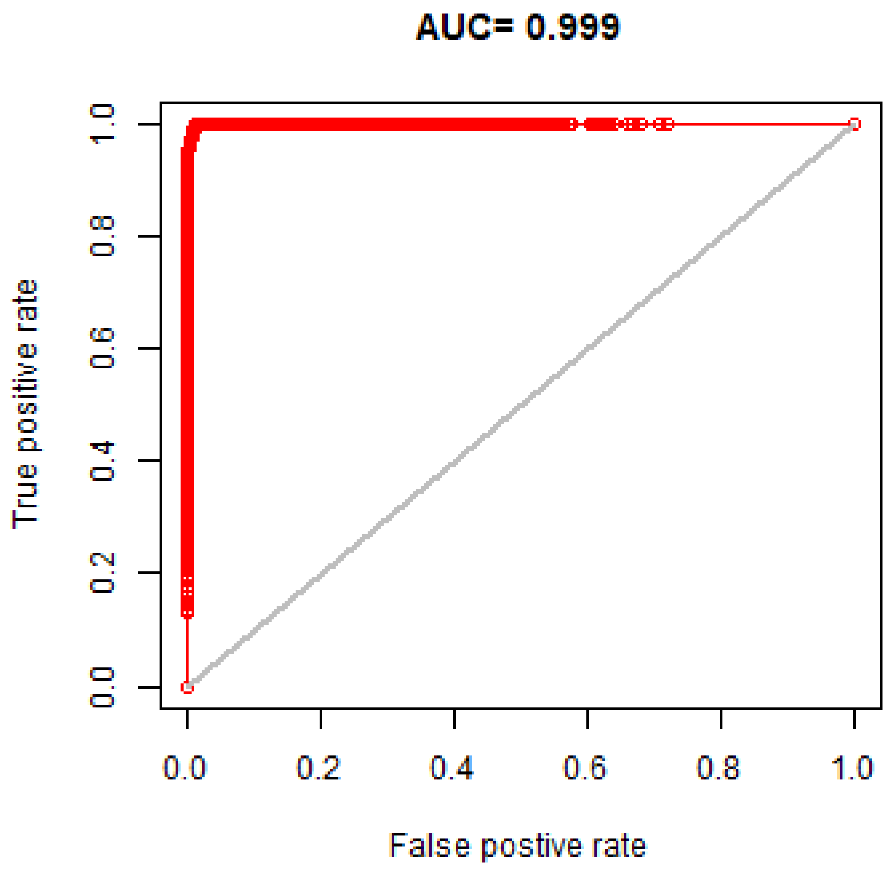


Note that the regression predictions are well-behaved, in the sense that they are between 0 and 1. However, they are continuous within that range, and if you wanted presence/absence, you would need a threshold. To get the optimal threshold, you would normally have a hold out data set, but here I used the training data for simplicity.

```
eva <- evaluate(dw[dw$pa==1, ], dw[dw$pa==0, ], rrf)
eva
## class          : ModelEvaluation
## n presences    : 1224
## n absences     : 2022
## AUC            : 0.9993919
## cor            : 0.9646471
## max TPR+TNR at : 0.4870895
```

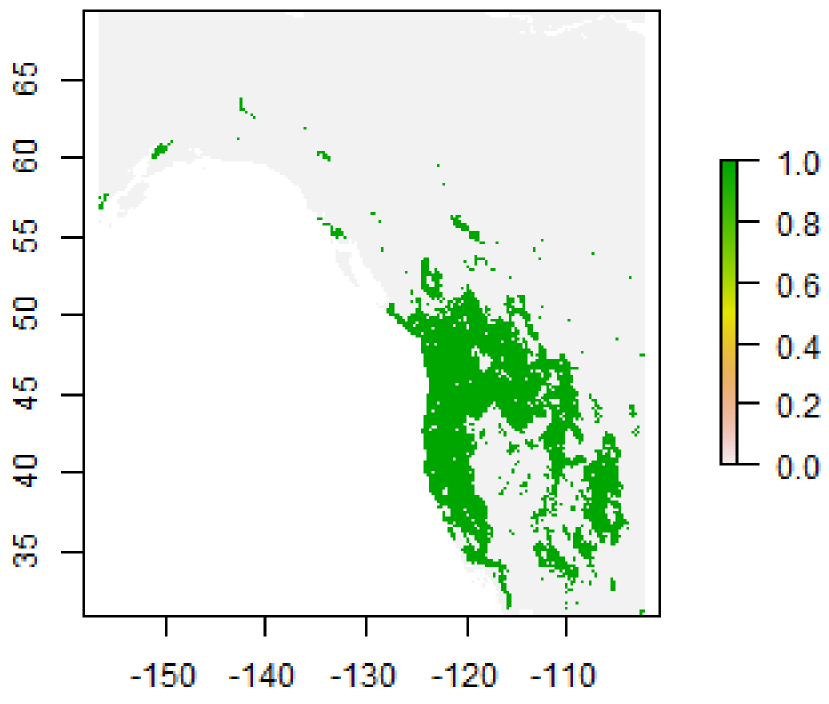
We can make a ROC plot

```
plot(eva, 'ROC')
```



Find a good threshold to determine presence/absence and plot the prediction.

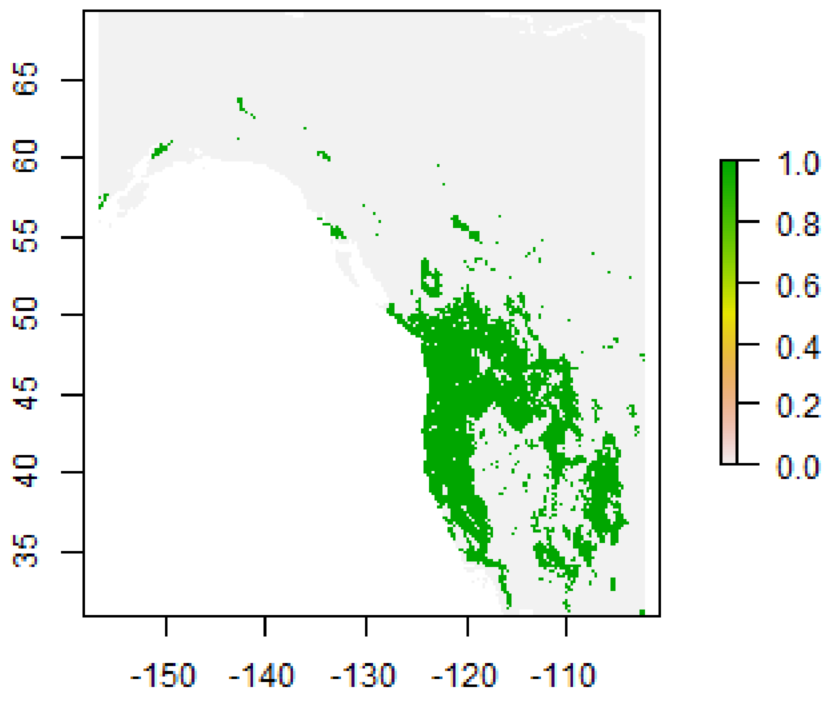
```
tr <- threshold(eva)
tr
##                 kappa spec_sens no_omission prevalence equal_sens_spec
## thresholds 0.4870895 0.4870895    0.393281  0.3814001        0.518784
##            sensitivity
## thresholds   0.7154833
plot(rp > tr[1, 'spec_sens'])
```



5.3.2 Classification

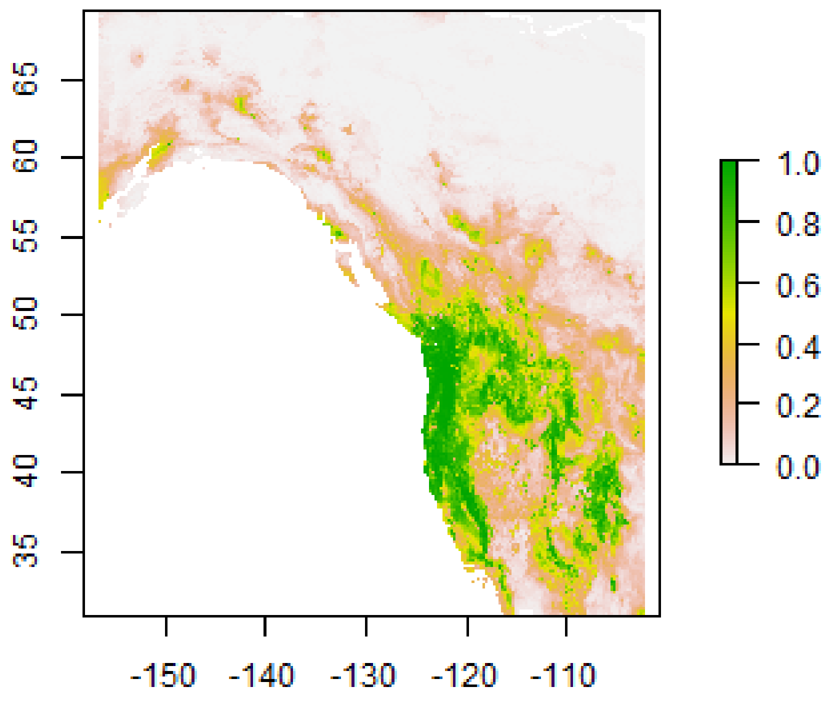
We can also use the classification Random Forest model to make a prediction.

```
rc <- predict(wc, crf, ext=ew)
plot(rc)
```



You can also get probabilities for the classes

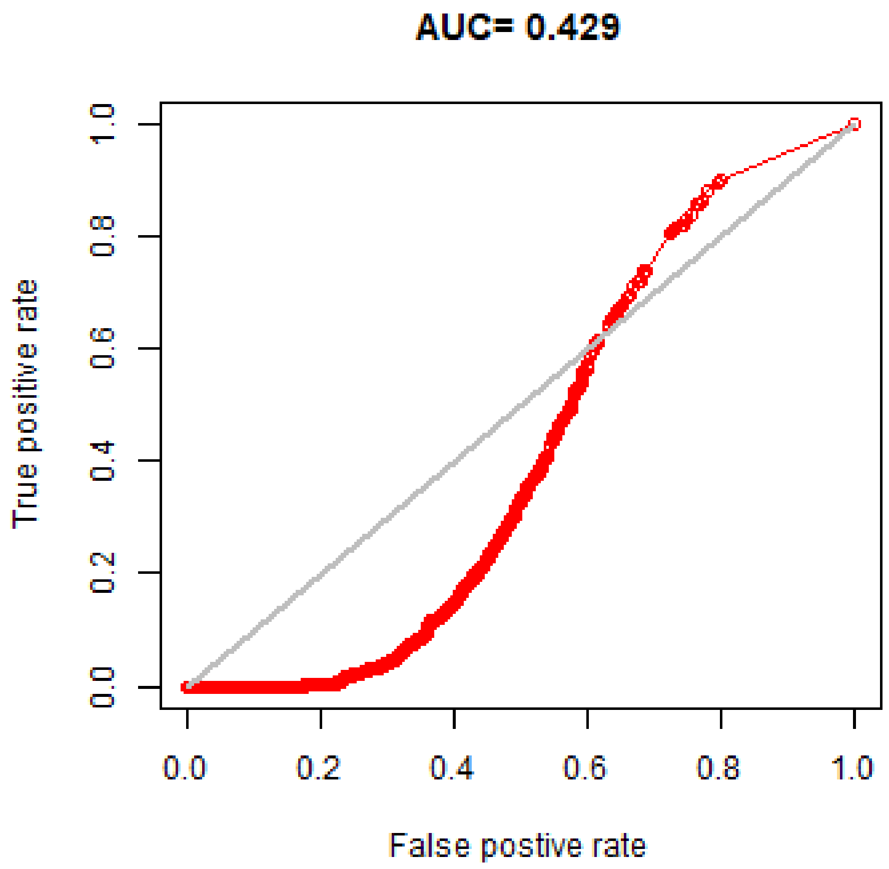
```
rc2 <- predict(wc, crf, ext=ew, type='prob', index=2)
plot(rc2)
```



5.4 Extrapolation

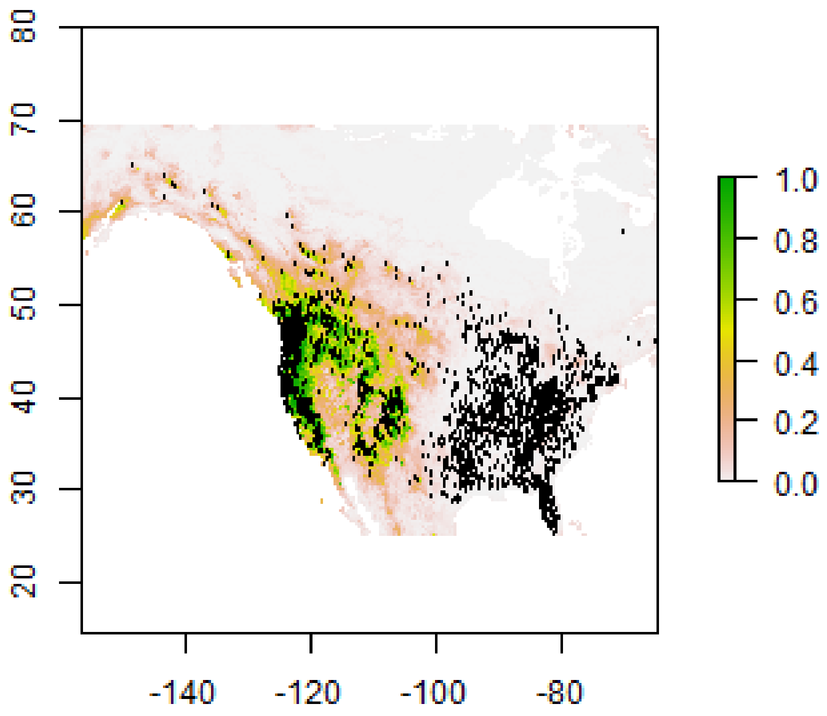
Now, let's see if our model is general enough to predict the distribution of the Eastern species.

```
de <- na.omit(de)
eva2 <- evaluate(de[de$pa==1, ], de[de$pa==0, ], rrf)
eva2
## class           : ModelEvaluation
## n presences     : 1866
## n absences      : 2978
## AUC             : 0.4294745
## cor             : -0.2609383
## max TPR+TNR at : 0.000208
plot(eva2, 'ROC')
```



We can also look at it on a map.

```
eus <- extent(SpatialPoints(bf[, 1:2]))
eus
## class       : Extent
## xmin        : -156.75
## xmax        : -64.4627
## ymin        : 25.141
## ymax        : 69.5
rcusa <- predict(wc, rrf, ext=eus)
plot(rcusa)
points(bf[,1:2], cex=.25)
```

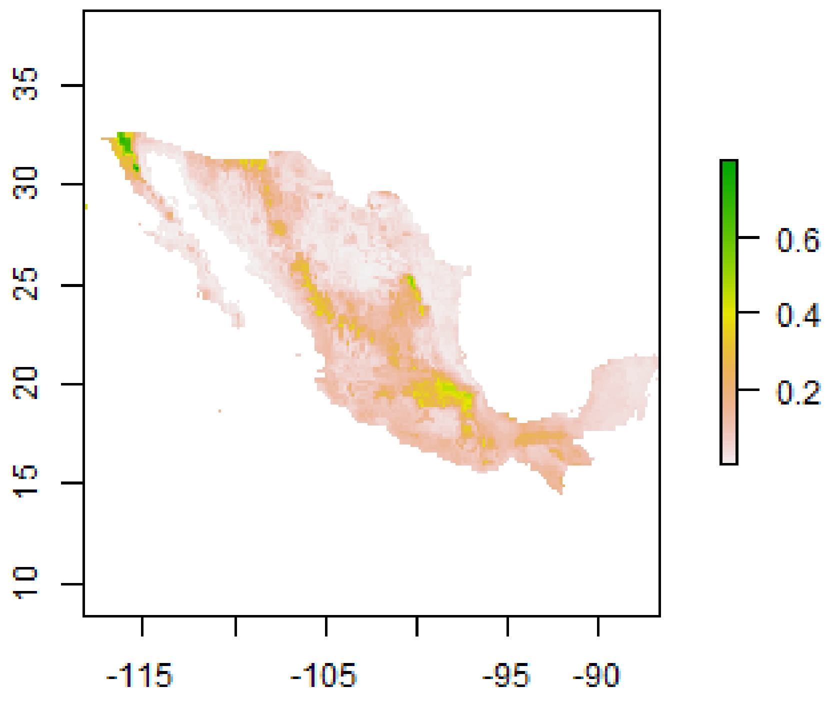


Question 4: *Why would it be that the model does not extrapolate well?*

An important question in the biogeography of the western species is why it does not occur in Mexico. Or if it does, where would that be?

Let's see.

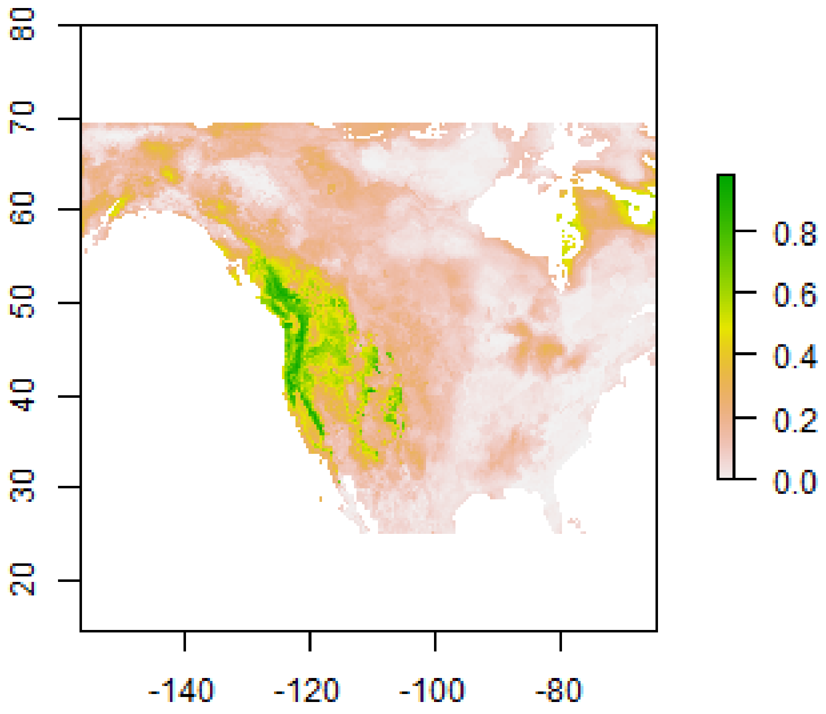
```
mex <- getData('GADM', country='MEX', level=1)
pm <- predict(wc, rrf, ext=mex)
pm <- mask(pm, mex)
plot(pm)
```



Question 5: *Where in Mexico are you most likely to encounter western bigfoot?*

We can also estimate range shifts due to climate change

```
fut <- getData('CMIP5', res=10, var='bio', rcp=85, model='AC', year=70)
names(fut)
##  [1] "ac85bi701"  "ac85bi702"  "ac85bi703"  "ac85bi704"  "ac85bi705"
##  [6] "ac85bi706"  "ac85bi707"  "ac85bi708"  "ac85bi709"  "ac85bi7010"
## [11] "ac85bi7011" "ac85bi7012" "ac85bi7013" "ac85bi7014" "ac85bi7015"
## [16] "ac85bi7016" "ac85bi7017" "ac85bi7018" "ac85bi7019"
names(wc)
##  [1] "bio1"  "bio2"  "bio3"  "bio4"  "bio5"  "bio6"  "bio7"  "bio8"
##  [9] "bio9"  "bio10" "bio11" "bio12" "bio13" "bio14" "bio15" "bio16"
## [17] "bio17" "bio18" "bio19"
names(fut) <- names(wc)
futusa <- predict(fut, rrf, ext=eus, progress='window')
## Loading required namespace: tcltk
plot(futusa)
```



Question 6: *Make a map to show where conditions are improving for western bigfoot, and where they are not. Is the species headed toward extinction?*

5.5 Further reading

More on Species distribution modeling with R; and on the use of boosted regression trees in the same context.

CHAPTER

SIX

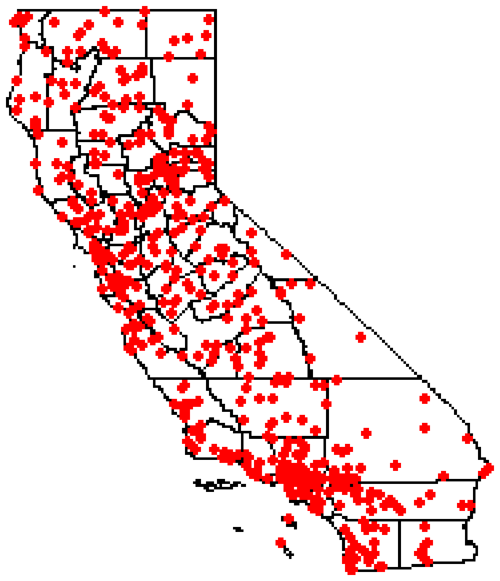
6. LOCAL REGRESSION

Regression models are typically "global". That is, all date are used simultaneously to fit a single model. In some cases it can make sense to fit more flexible "local" models. Such models exist in a general regression framework (e.g. generalized additive models), where "local" refers to the values of the predictor values. In a spatial context local refers to location. Rather than fitting a single regression model, it is possible to fit several models, one for each location (out of possibly very many) locations. This technique is sometimes called "geographically weighted regression" (GWR). GWR is a data exploration technique that allows to understand changes in importance of different variables over space (which may indicate that the model used is misspecified and can be improved).

There are two examples here. One short example with California precipitation data, and than a more elaborate example with house price data.

6.1 California precipitation

```
# If have not yet done so, first install the rspatial package (see the line below).␣
↪You may need to install the devtools package first
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
library(rspatial)
cts <- sp_data('counties')
p <- sp_data('precipitation')
head(p)
##      ID                  NAME   LAT    LONG ALT  JAN FEB MAR APR MAY JUN
## 1 ID741          DEATH VALLEY 36.47 -116.87 -59  7.4 9.5 7.5 3.4 1.7 1.0
## 2 ID743  THERMAL/FAA AIRPORT 33.63 -116.17 -34  9.2 6.9 7.9 1.8 1.6 0.4
## 3 ID744           BRAWLEY 2SW 32.96 -115.55 -31 11.3 8.3 7.6 2.0 0.8 0.1
## 4 ID753 IMPERIAL/FAA AIRPORT 32.83 -115.57 -18 10.6 7.0 6.1 2.5 0.2 0.0
## 5 ID754                NILAND 33.28 -115.51 -18  9.0 8.0 9.0 3.0 0.0 1.0
## 6 ID758         EL CENTRO/NAF 32.82 -115.67 -13  9.8 1.6 3.7 3.0 0.4 0.0
##   JUL  AUG SEP OCT NOV DEC
## 1 3.7  2.8 4.3 2.2 4.7 3.9
## 2 1.9  3.4 5.3 2.0 6.3 5.5
## 3 1.9  9.2 6.5 5.0 4.8 9.7
## 4 2.4  2.6 8.3 5.4 7.7 7.3
## 5 8.0  9.0 7.0 8.0 7.0 9.0
## 6 3.0 10.8 0.2 0.0 3.3 1.4

plot(cts)
points(p[,c('LONG', 'LAT')], col='red', pch=20)
```



Compute annual average precipitation

```
p$pan <- rowSums(p[,6:17])
```

Global regression model

```
m <- lm(pan ~ ALT, data=p)
m
##
## Call:
## lm(formula = pan ~ ALT, data = p)
##
## Coefficients:
## (Intercept)          ALT
##      523.60         0.17
```

Create `Spatial*` objects with a planar crs.

```
alb <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000␣
↪+ellps=GRS80 +datum=NAD83 +units=m +no_defs")
sp <- p
coordinates(sp) = ~ LONG + LAT
crs(sp) <- "+proj=longlat +datum=NAD83"
spt <- spTransform(sp, alb)
ctst <- spTransform(cts, alb)
```

Get the optimal bandwidth

```
library( spgwr )
## Loading required package: spData
## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source'))`
## NOTE: This package does not constitute approval of GWR
## as a method of spatial analysis; see example(gwr)
bw <- gwr.sel(pan ~ ALT, data=spt)
## Bandwidth: 526221.1 CV score: 64886883
## Bandwidth: 850593.6 CV score: 74209073
## Bandwidth: 325747.9 CV score: 54001118
## Bandwidth: 201848.6 CV score: 44611213
## Bandwidth: 125274.7 CV score: 35746320
## Bandwidth: 77949.39 CV score: 29181737
## Bandwidth: 48700.74 CV score: 22737197
## Bandwidth: 30624.09 CV score: 17457161
## Bandwidth: 19452.1 CV score: 15163436
## Bandwidth: 12547.43 CV score: 19452191
## Bandwidth: 22792.75 CV score: 15512988
## Bandwidth: 17052.67 CV score: 15709960
## Bandwidth: 20218.99 CV score: 15167438
## Bandwidth: 19767.99 CV score: 15156913
## Bandwidth: 19790.05 CV score: 15156906
## Bandwidth: 19781.39 CV score: 15156902
## Bandwidth: 19781.48 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
## Bandwidth: 19781.47 CV score: 15156902
bw
## [1] 19781.47
```

Create a regular set of points to estimate parameters for.

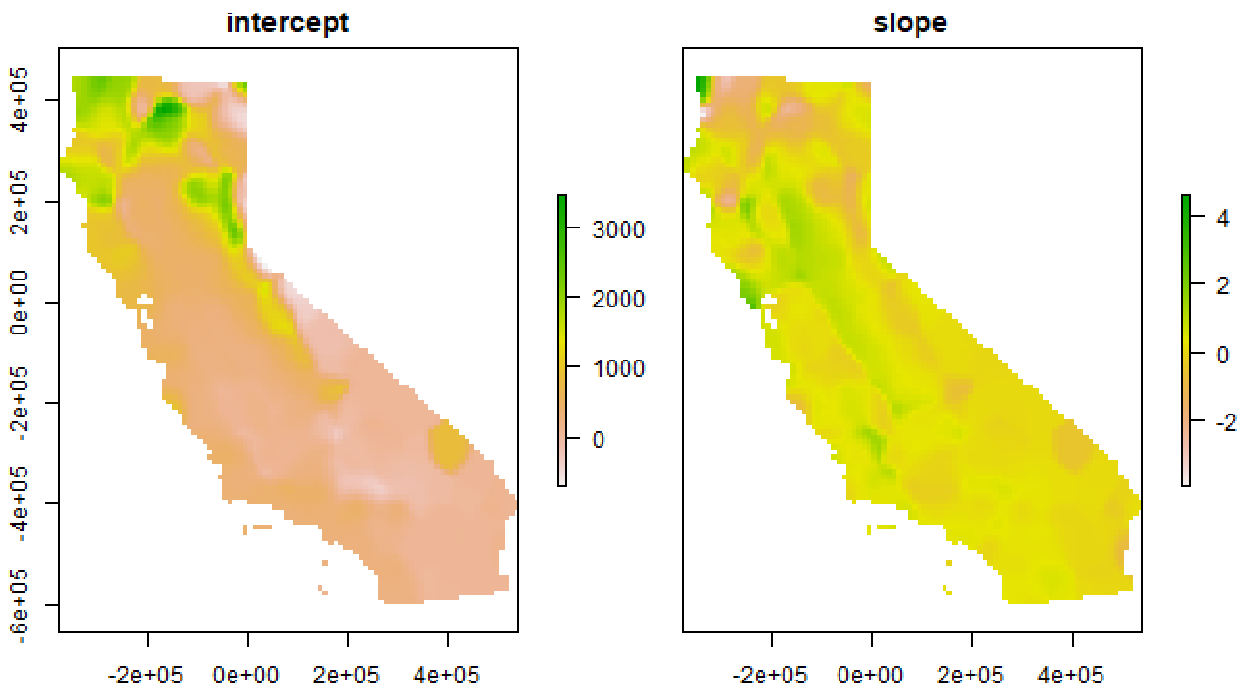
```
r <- raster(ctst, res=10000)
r <- rasterize(ctst, r)
newpts <- rasterToPoints(r)
```

Run the `gwr` function

```
g <- gwr(pan ~ ALT, data=spt, bandwidth=bw, fit.points=newpts[, 1:2])
g
## Call:
## gwr(formula = pan ~ ALT, data = spt, bandwidth = bw, fit.points = newpts[,
##     1:2])
## Kernel function: gwr.Gauss
## Fixed bandwidth: 19781.47
## Fit points: 4087
## Summary of GWR coefficient estimates at fit points:
##                     Min.     1st Qu.      Median     3rd Qu.       Max.
## X.Intercept. -702.40121    79.54254   330.48807   735.42718 3468.8702
## ALT            -3.91270     0.03058     0.20461     0.41542    4.6133
```

Link the results back to the raster

```
slope <- r
intercept <- r
slope[!is.na(slope)] <- g$SDF$ALT
intercept[!is.na(intercept)] <- g$SDF$'(Intercept)'
s <- stack(intercept, slope)
names(s) <- c('intercept', 'slope')
plot(s)
```



6.2 California House Price Data

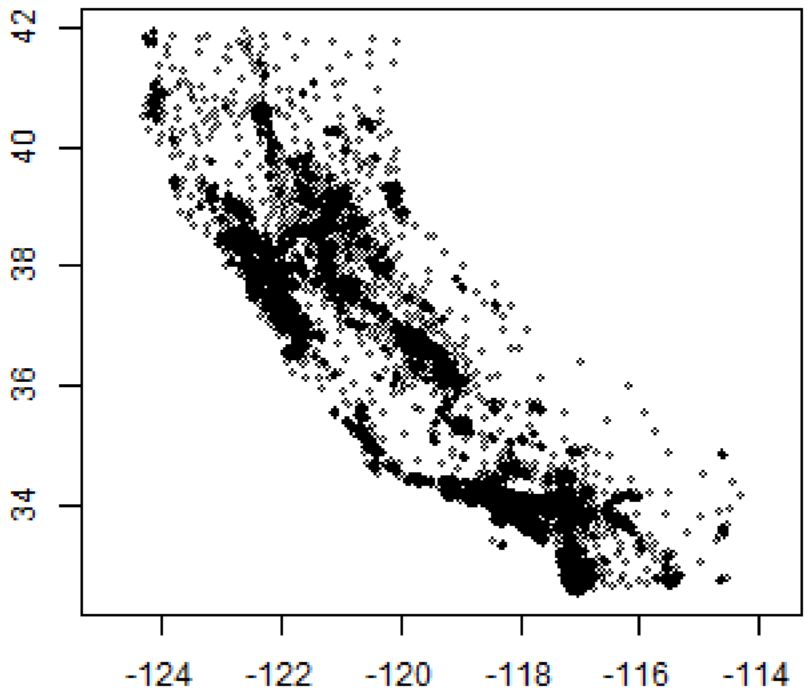
We will use house prices data from the 1990 census, taken from "Pace, R.K. and R. Barry, 1997. Sparse Spatial Autoregressions. Statistics and Probability Letters 33: 291-297." You can download the data here

```
houses <- sp_data("houses1990.csv")
dim(houses)
## [1] 20640     9
head(houses)
##   houseValue income houseAge rooms bedrooms population households latitude
## 1     452600 8.3252       41   880      129        322        126    37.88
## 2     358500 8.3014       21  7099     1106       2401       1138    37.86
## 3     352100 7.2574       52  1467      190        496        177    37.85
## 4     341300 5.6431       52  1274      235        558        219    37.85
## 5     342200 3.8462       52  1627      280        565        259    37.85
## 6     269700 4.0368       52   919      213        413        193    37.85
##   longitude
## 1   -122.23
## 2   -122.22
## 3   -122.24
## 4   -122.25
## 5   -122.25
## 6   -122.25
```

Each record represents a census "blockgroup". The longitude and latitude of the centroids of each block group are available. We can use that to make a map and we can also use these to link the data to other spatial data. For example to get county-membership of each block group. To do that, let's first turn this into a SpatialPointsDataFrame to find out to which county each point belongs.

```
library(sp)
coordinates(houses) <- ~longitude+latitude
```

```
plot(houses, cex=0.5, pch=1, axes=TRUE)
```



Now get the county boundaries and assign CRS of the houses data matches that of the counties (because they are both in longitude/latitude!).

```
library(raster)
counties <- readRDS("data/counties.rds")
## Warning in readRDS("data/counties.rds"): invalid or incomplete compressed
## data
## Error in readRDS("data/counties.rds"): error reading from connection
crs(houses) <- crs(counties)
## Error in crs(counties): object 'counties' not found
```

Do a spatial query (points in polygon)

```
cnty <- over(houses, counties)
## Error in over(houses, counties): object 'counties' not found
head(cnty)
## Error in head(cnty): object 'cnty' not found
```

6.3 Summarize

We can summarize the data by county. First combine the extracted county data with the original data.

```
hd <- cbind(data.frame(houses), cnty)
## Error in cbind(data.frame(houses), cnty): object 'cnty' not found
```

Compute the population by county

```
totpop <- tapply(hd$population, hd$NAME, sum)
## Error in tapply(hd$population, hd$NAME, sum): object 'hd' not found
totpop
## Error in eval(expr, envir, enclos): object 'totpop' not found
```

Income is harder because we have the median household income by blockgroup. But it can be approximated by first computing total income by blockgroup, summing that, and dividing that by the total number of households.

```
# total income
hd$suminc <- hd$income * hd$households
## Error in eval(expr, envir, enclos): object 'hd' not found
# now use aggregate (similar to tapply)
csum <- aggregate(hd[, c('suminc', 'households')], list(hd$NAME), sum)
## Error in aggregate(hd[, c("suminc", "households")], list(hd$NAME), sum): object 'hd
↪' not found
# divide total income by number of housefholds
csum$income <- 10000 * csum$suminc / csum$households
## Error in eval(expr, envir, enclos): object 'csum' not found
# sort
csum <- csum[order(csum$income), ]
## Error in eval(expr, envir, enclos): object 'csum' not found
head(csum)
## Error in head(csum): object 'csum' not found
tail(csum)
## Error in tail(csum): object 'csum' not found
```

6.4 Regression

Before we make a regression model, let's first add some new variables that we might use, and then see if we can build a regression model with house price as dependent variable. The authors of the paper used a lot of log tranforms, so you can also try that.

```
hd$roomhead <- hd$rooms / hd$population
## Error in eval(expr, envir, enclos): object 'hd' not found
hd$bedroomhead <- hd$bedrooms / hd$population
## Error in eval(expr, envir, enclos): object 'hd' not found
hd$hhsize <- hd$population / hd$households
## Error in eval(expr, envir, enclos): object 'hd' not found
```

Ordinary least squares regression:

```
# OLS
m <- glm( houseValue ~ income + houseAge + roomhead + bedroomhead + population,␣
↪data=hd)
## Error in is.data.frame(data): object 'hd' not found
summary(m)
##
## Call:
## lm(formula = pan ~ ALT, data = p)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -638.4 -281.2 -115.7  187.4 1793.5
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 523.60251   26.50338  19.756  < 2e-16 ***
## ALT           0.16997    0.03505   4.849  1.7e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 425.4 on 454 degrees of freedom
## Multiple R-squared:  0.04925,    Adjusted R-squared:  0.04715
## F-statistic: 23.52 on 1 and 454 DF,  p-value: 1.704e-06
coefficients(m)
## (Intercept)         ALT
## 523.6025073   0.1699685
```

6.5 Geographicaly Weighted Regression

6.5.1 By county

Of course we could make the model more complex, with e.g. squared income, and interactions. But let's see if we can do Geographically Weighted regression. One approach could be to use counties.

First I remove records that were outside the county boundaries

```
 hd2 <- hd[!is.na(hd$NAME), ]
## Error in eval(expr, envir, enclos): object 'hd' not found
```

Then I write a function to get what I want from the regression (the coefficients in this case)

```
regfun <- function(x)  {
  dat <- hd2[hd2$NAME == x, ]
  m <- glm(houseValue~income+houseAge+roomhead+bedroomhead+population, data=dat)
  coefficients(m)
}
```

And now run this for all counties using sapply:

```
countynames <- unique(hd2$NAME)
## Error in unique(hd2$NAME): object 'hd2' not found
res <- sapply(countynames, regfun)
## Error in lapply(X = X, FUN = FUN, ...): object 'countynames' not found
```

Plot of a single coefficient

```
dotchart(sort(res['income', ]), cex=0.65)
## Error in res["income", ]: object of type 'closure' is not subsettable
```

There clearly is variation in the coefficient (*beta*) for income. How does this look on a map?

First make a data.frame of the results

```
resdf <- data.frame(NAME=colnames(res), t(res))
## Error in t.default(res): argument is not a matrix
head(resdf)
## Error in head(resdf): object 'resdf' not found
```

Fix the counties object. There are too many counties because of the presence of islands. I first aggregate ('dissolve' in GIS-speak') the counties such that a single county becomes a single (multi-)polygon.

```
dim(counties)
## Error in eval(expr, envir, enclos): object 'counties' not found
dcounties <- aggregate(counties, vars='NAME')
## Error in aggregate(counties, vars = "NAME"): object 'counties' not found
dim(dcounties)
## Error in eval(expr, envir, enclos): object 'dcounties' not found
```

Now we can merge this SpatialPolygonsDataFrame with data.frame with the regression results.

```
cnres <- merge(dcounties, resdf, by='NAME')
## Error in merge(dcounties, resdf, by = "NAME"): object 'dcounties' not found
spplot(cnres, 'income')
## Error in spplot(cnres, "income"): object 'cnres' not found
```

To show all parameters in a 'conditioning plot', we need to first scale the values to get similar ranges.

```
# a copy of the data
cnres2 <- cnres
## Error in eval(expr, envir, enclos): object 'cnres' not found

# scale all variables, except the first one (county name)
# assigning values to a "@data" slot is risky, but (I think) OK here
cnres2@data = data.frame(scale(data.frame(cnres)[, -1]))
## Error in data.frame(cnres): object 'cnres' not found
spplot(cnres2)
## Error in spplot(cnres2): object 'cnres2' not found
```

Is this just random noise, or is there spatial autocorrelation?

```
library(spdep)
## Loading required package: Matrix
##
## Attaching package: 'Matrix'
## The following object is masked from 'package:spam':
##
##     det
nb <- poly2nb(cnres)
## Error in extends(class(pl), "SpatialPolygons"): object 'cnres' not found
plot(cnres)
## Error in plot(cnres): object 'cnres' not found
plot(nb, coordinates(cnres), add=T, col='red')
```

(continues on next page)

(continued from previous page)

```
## Error in plot(nb, coordinates(cnres), add = T, col = "red"): object 'nb' not found

lw <- nb2listw(nb)
## Error in nb2listw(nb): object 'nb' not found
moran.test(cnres$income, lw)
## Error in moran.test(cnres$income, lw): object 'lw' not found
moran.test(cnres$roomhead, lw, na.action=na.omit)
## Error in moran.test(cnres$roomhead, lw, na.action = na.omit): object 'lw' not found
```

6.5.2 By grid cell

An alternative approach would be to compute a model for grid cells. Let's use the 'Teale Albers' projection (often used when mapping the entire state of California).

```
TA <- CRS("+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000
              +datum=NAD83 +units=m +no_defs +ellps=GRS80 +towgs84=0,0,0")
countiesTA <- spTransform(counties, TA)
## Error in spTransform(counties, TA): object 'counties' not found
```

Create a RasteLayer using the extent of the counties, and setting an arbitrary resolution of 50 by 50 km cells

```
library(raster)
r <- raster(countiesTA)
## Error in raster(countiesTA): object 'countiesTA' not found
res(r) <- 50000
```

Get the xy coordinates for each raster cell:

```
xy <- xyFromCell(r, 1:ncell(r))
```

For each cell, we need to select a number of observations, let's say within 50 km of the center of each cell (thus the data that are used in different cells overlap). And let's require at least 50 observations to do a regression.

First transform the houses data to Teale-Albers

```
housesTA <- spTransform(houses, TA)
## Error in spTransform(xSP, CRSobj, ...): No transformation possible from NA␣
↪reference system
crds <- coordinates(housesTA)
## Error in coordinates(housesTA): object 'housesTA' not found
```

Set up a new regression function.

```
regfun2 <- function(d)  {
 m <- glm(houseValue~income+houseAge+roomhead+bedroomhead+population, data=d)
 coefficients(m)
}
```

Run the model for al cells if there are at least 50 observations within a radius of 50 km.

```
res <- list()
for (i in 1:nrow(xy)) {
    d <- sqrt((xy[i,1]-crds[,1])^2 + (xy[i,2]-crds[,2])^2)
    j <- which(d < 50000)
    if (length(j) > 49) {
```

(continues on next page)

(continued from previous page)

```
        d <- hd[j,]
        res[[i]] <- regfun2(d)
    } else {
        res[[i]] <- NA
    }
}
## Error: object 'crds' not found
```

For each cell get the income coefficient:

```
inc <- sapply(res, function(x) x['income'])
```

Use these values in a RasterLayer

```
rinc <- setValues(r, inc)
## Error in setValues(r, inc): values must be numeric, integer, logical or factor
plot(rinc)
## Error in plot(rinc): object 'rinc' not found
plot(countiesTA, add=T)
## Error in plot(countiesTA, add = T): object 'countiesTA' not found
Moran(rinc)
## Error in Moran(rinc): object 'rinc' not found
```

So that was a lot of 'home-brew-GWR'.

Question 1: *Can you comment on weaknesses (and perhaps strengths) of the approaches I have shown?*

Question 2: *Can you do it the easier and more professional way for these data, using the spgwr package?*

6.6 spgwr package

Now use the spgwr package (and the the `gwr` function) to fit the model. You can do this with all data, as long as you supply and argument `fit.points` (to avoid estimating a model for each observation point. You can use a raster similar to the one I used above (perhaps disaggregate with a factor 2 first).

This is how you can get the points to use:

Create a RasterLayer with the correct extent

```
r <- raster(countiesTA)
## Error in raster(countiesTA): object 'countiesTA' not found
```

Set to a desired resolution. I choose 25 km

```
res(r) <- 25000
```

I only want cells inside of CA, so I add some more steps.

```
ca <- rasterize(countiesTA, r)
## Error in rasterize(countiesTA, r): object 'countiesTA' not found
```

Extract the coordinates that are not `NA`.

```
fitpoints <- rasterToPoints(ca)
## Error in nlayers(x): object 'ca' not found
```

I don't want the third column

```
fitpoints <- fitpoints[,-3]
## Error in eval(expr, envir, enclos): object 'fitpoints' not found
```

Now specificy the model

```
gwr.model <- _______
```

`gwr` returns a list-like object that includes (as first element) a `SpatialPointsDataFrame` that has the model coeffients. Plot these using `spplot`, and after that, transfer them to a `RasterBrick` object.

To extract the SpatialPointsDataFrame:

```
sp <- gwr.model$SDF
spplot(sp)
```

To reconnect these values to the raster structure (etc.)

```
cells <- cellFromXY(r, fitpoints)
dd <- as.matrix(data.frame(sp))
b <- brick(r, values=FALSE, nl=nrow(dd))
b[cells] <- dd
names(b) <- colnames(dd)
plot(b)
```

Question 3: *Briefly comment on the results and the differences (if any) with the two home-brew examples.*

CHAPTER

SEVEN

7. SPATIAL REGRESSION MODELS

7.1 Introduction

This chapter deals with the problem of inference in (regression) models with spatial data. Inference from regression models with spatial data can be suspect. In essence this is because nearby things are similar, and it may not be fair to consider individual cases as independent (they may be pseudo-replicates). Therefore, such models need to be diagnosed before reporting them. Specifically, it is important to evaluate the for spatial autocorrelation in the residuals (as these are supposed to be independent, not correlated). If the residuals are spatially autocorrelated, this indicates that the model is misspecified. In that case you should try to improve the model by adding (and perhaps removing) important variables. If that is not possible (either because there is no data available, or because you have no clue as to what variable to look for), you can try formulating a regression model that controls for spatial autocorrelation. We show some examples of that approach here.

7.2 Reading & aggregating data

We use California house price data from the 2000 Census.

7.2.1 Get the data

```
# If have not yet done so, first install the rspatial package (see the line below).␣
↪You may need to install the devtools package first
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
library(rspatial)
h <- sp_data('houses2000')
```

I have selected some variables on on housing and population. You can get more data from the American Fact Finder http://factfinder2.census.gov (among other web sites).

```
library(raster)
dim(h)
## [1] 7049   29
names(h)
##  [1] "TRACT"      "GEOID"      "label"      "houseValue" "nhousingUn"
##  [6] "recHouses"  "nMobileHom" "yearBuilt"  "nBadPlumbi" "nBadKitche"
## [11] "nRooms"     "nBedrooms"  "medHHinc"   "Population" "Males"
## [16] "Females"    "Under5"     "MedianAge"  "White"      "Black"
## [21] "AmericanIn" "Asian"      "Hispanic"   "PopInHouse" "nHousehold"
## [26] "Families"   "householdS" "familySize" "County"
```

These are the variables we have:

variable	explanation
nhousingUn	number of housing units
recHouses	number of houses for recreational use
nMobileHom	number of mobile homes
nBadPlumbi	number of houses with incomplete plumbing
nBadKitche	number of houses with incomplete kitchens
Population	total population
Males	number of males
Females	number of females
Under5	number of persons under five
White	number of persons identifying themselves as white (only)
Black	number of persons identifying themselves African-american (only)
AmericanIn	number of persons identifying themselves American Indian (only)
Asian	number of persons identifying themselves as American Indian (only)
Hispanic	number of persons identifying themselves as hispanic (only)
PopInHouse	number of persons living in households
nHousehold	number of households
Families	

number of families

houseValue

value of the house

yearBuilt

year house was built

nRooms

median number of rooms per house

nBedrooms

median number of bedrooms per house

medHHinc

median household income

MedianAge

median age of population

householdS

median household size

familySize

median family size

First some data massaging. These are values for Census tracts. I want to analyze these data at the county level. So we need to aggregate the values.

```
hh <- aggregate(h, "County")
```

Now we have the county outlines, but we also need to get the values of interest at the county level. Although it is possible to do everything in one step in the aggregate function, I prefer to do this step by step. The simplest case is where we can sum the numbers. For example for the number of houses.

```
d1 <- data.frame(h)[, c("nhousingUn", "recHouses", "nMobileHom", "nBadPlumbi",
 "nBadKitche", "Population", "Males", "Females", "Under5", "White",
 "Black", "AmericanIn", "Asian", "Hispanic", "PopInHouse", "nHousehold", "Families")]

 d1a <- aggregate(d1, list(County=h$County), sum, na.rm=TRUE)
```

In other cases we need to use a weighted mean. For example for houseValue

```
d2 <- data.frame(h)[, c("houseValue", "yearBuilt", "nRooms", "nBedrooms",
       "medHHinc", "MedianAge", "householdS",  "familySize")]
d2 <- cbind(d2 * h$nHousehold, hh=h$nHousehold)

d2a <- aggregate(d2, list(County=h$County), sum, na.rm=TRUE)
d2a[, 2:ncol(d2a)] <- d2a[, 2:ncol(d2a)] / d2a$hh
```

Combine these two groups:

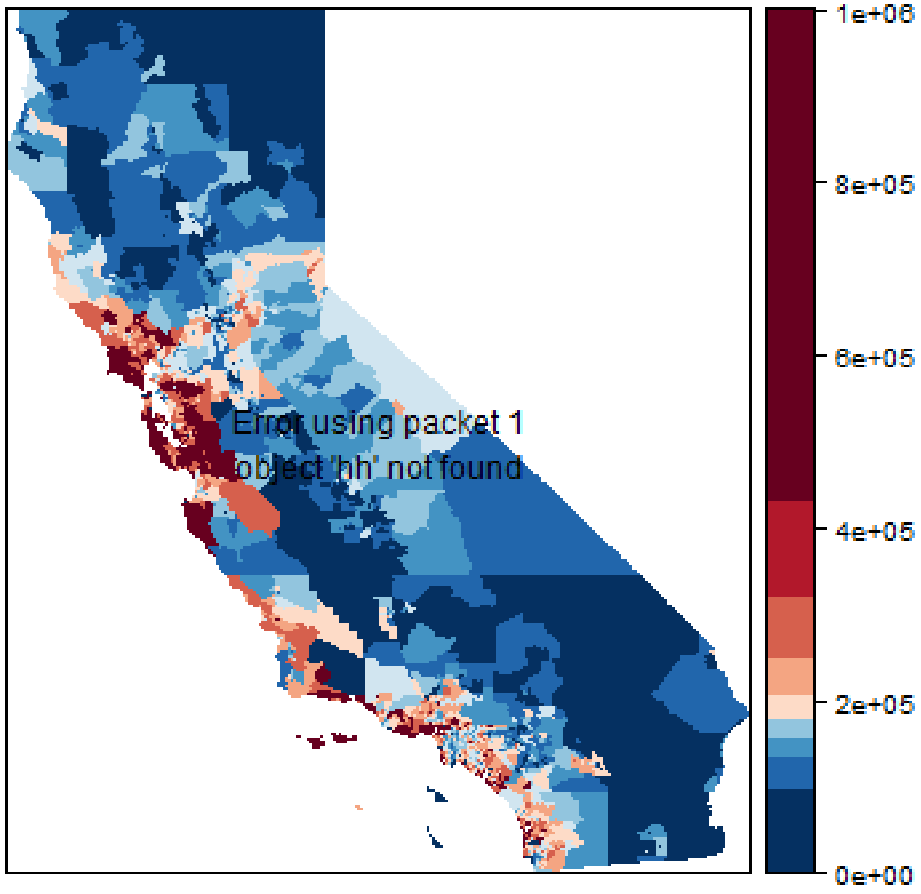
```
d12 <- merge(d1a, d2a, by='County')
```

And merge the aggregated (from census tract to county level) attribute data with the aggregated polygons

```
hh <- merge(hh, d12, by='County')
```

Let's make some maps, at the orignal Census tract level. We are using a bit more advanced (and slower) plotting methods here. First the house value, using a legend with 10 intervals.

```
library(latticeExtra)
## Loading required package: lattice
## Loading required package: RColorBrewer

grps <- 10
brks <- quantile(h$houseValue, 0:(grps-1)/(grps-1), na.rm=TRUE)

p <- spplot(h, "houseValue", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")), col=
↪"transparent" )
p + layer(sp.polygons(hh))
```



This takes very long. spplot (levelplot) is a bit slow when using a large dataset...

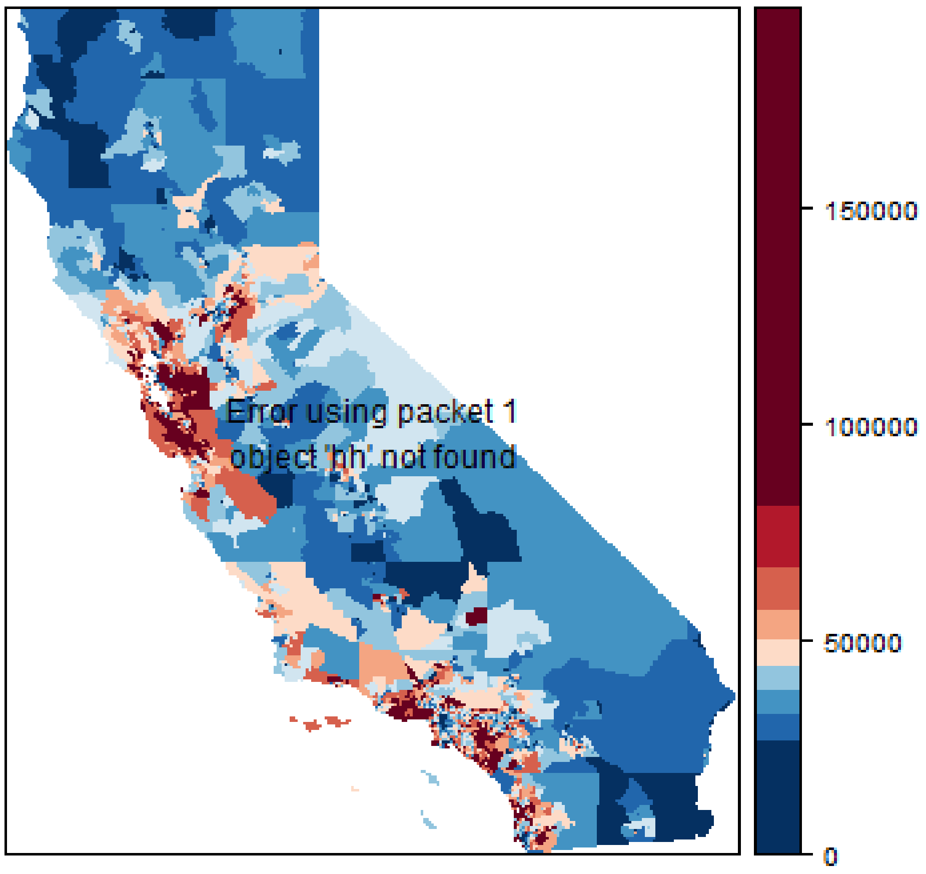
A map of the median household income.

```
brks <- quantile(h$medHHinc, 0:(grps-1)/(grps-1), na.rm=TRUE)

p <- spplot(h, "medHHinc", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")), col=
↪"transparent")
```

(continues on next page)

(continued from previous page)

```
p + layer(sp.polygons(hh))
```



7.3 Basic OLS model

I'll now make some models with the county-level data. I first compute some new variables (that I might not all use).

```
hh$fBadP <- pmax(hh$nBadPlumbi, hh$nBadKitche) / hh$nhousingUn
hh$fWhite <- hh$White / hh$Population
hh$age <- 2000 - hh$yearBuilt

f1 <- houseValue ~ age +  nBedrooms
m1 <- lm(f1, data=hh)
summary(m1)
##
## Call:
## lm(formula = f1, data = hh)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -222541  -67489   -6128   60509  217655
##
```

(continues on next page)

(continued from previous page)

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -628578     233217  -2.695  0.00931 **
## age            12695       2480   5.119 4.05e-06 ***
## nBedrooms     191889      76756   2.500  0.01543 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 94740 on 55 degrees of freedom
## Multiple R-squared:  0.3235, Adjusted R-squared:  0.2989
## F-statistic: 13.15 on 2 and 55 DF,  p-value: 2.147e-05
```

Just for illustration, here is how you can do OLS with matrix algebra. First set up the data. I add a constant variable ‘1’ to X, to get an intercept.

```
y <- matrix(hh$houseValue)
X <- cbind(1, hh$age, hh$nBedrooms)
```

Then use matrix algebra

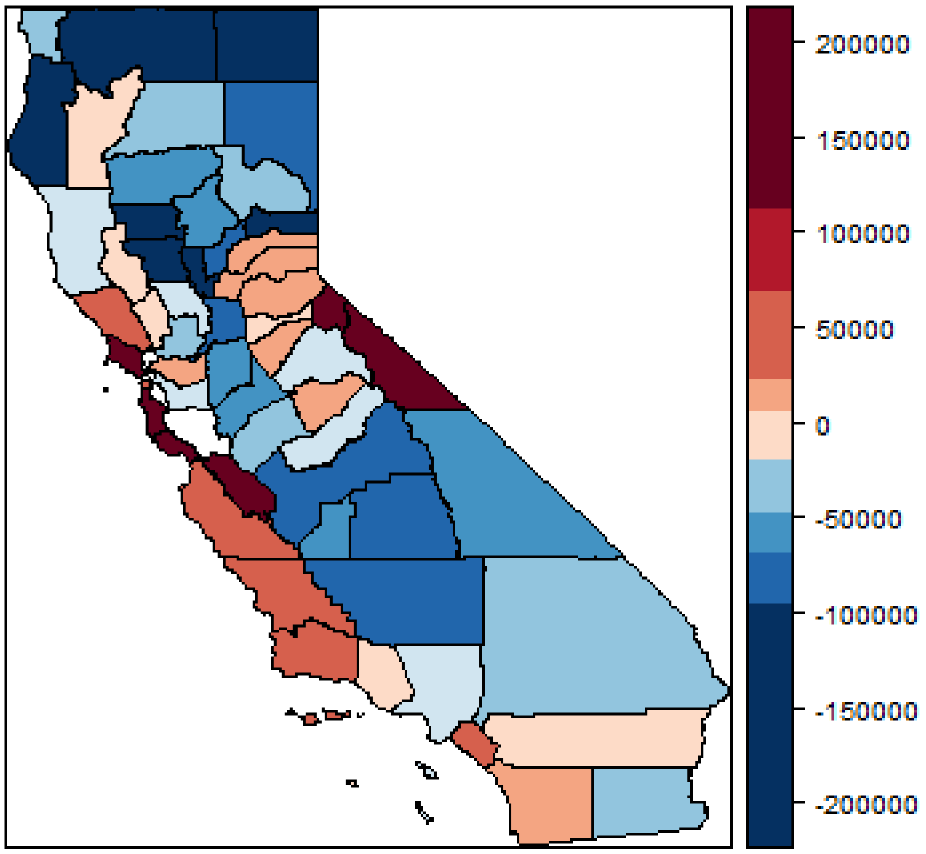
```
ols <- solve(t(X) %*% X) %*% t(X) %*% y
rownames(ols) <- c('intercept', 'age', 'nBedroom')
ols
##                 [,1]
## intercept -628577.95
## age          12694.75
## nBedroom    191888.89
```

So, according to this simple model, “age” is highly significant. The older a house, the more expensive. You pay 1,269,475 dollars more for a house that is 100 years old than a for new house! While the p-value for the number of bedrooms is not impressive, but every bedroom adds about 200,000 dollars to the value of a house.

Question 1: *What would be the price be of a house built in 1999 with three bedrooms?*

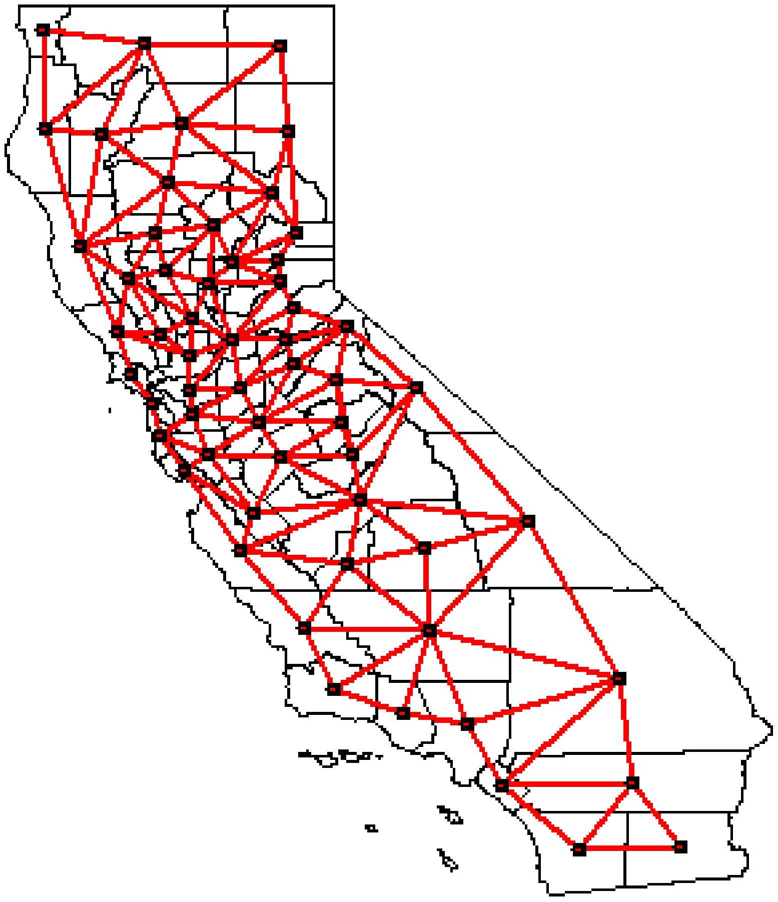
Let’s see if the errors (model residuals) appear to be randomly distributed in space.

```
hh$residuals <- residuals(m1)

brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)

spplot(hh, "residuals", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")), col="black
↪")
```



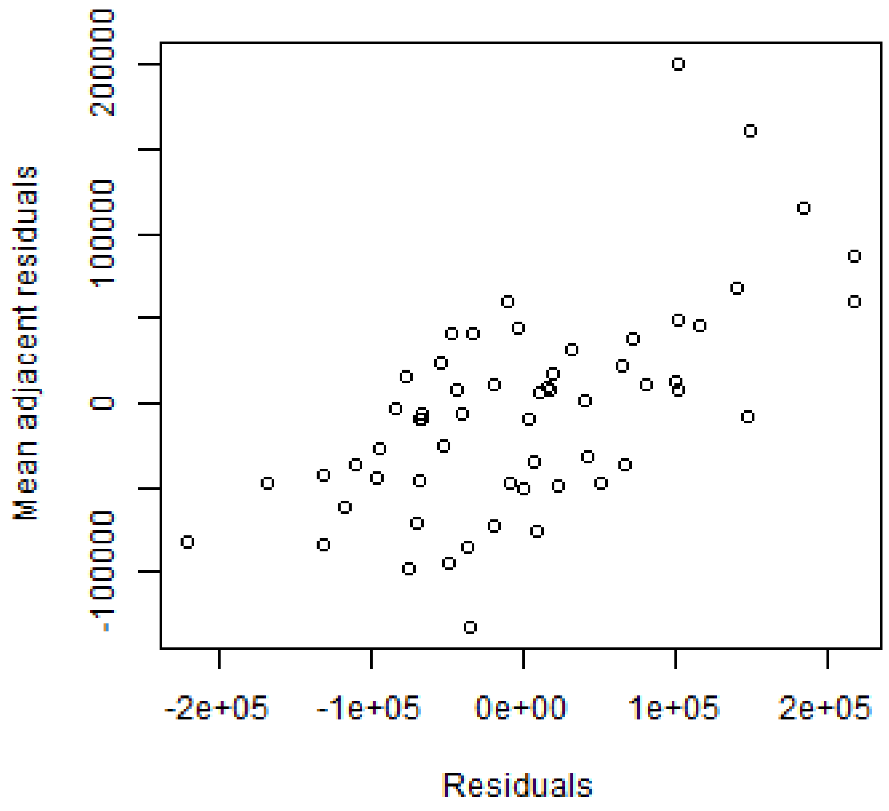
What do think? Is this random? Let's see what Mr. Moran would say. First make a neighborhoods list. I add two links: between San Francisco and Marin County and vice versa (to consider the Golden Gate bridge).

```
library(spdep)
nb <- poly2nb(hh)
nb[[21]] <- sort(as.integer(c(nb[[21]], 38)))
nb[[38]] <- sort(as.integer(c(21, nb[[38]])))
nb
## Neighbour list object:
## Number of regions: 58
## Number of nonzero links: 278
## Percentage nonzero weights: 8.263971
## Average number of links: 4.793103

par(mai=c(0,0,0,0))
plot(hh)
plot(nb, coordinates(hh), col='red', lwd=2, add=TRUE)
```



We can use the neighbour list object to get the average value for the neighbors of each polygon.

```
resnb <- sapply(nb, function(x) mean(hh$residuals[x]))
cor(hh$residuals, resnb)
## [1] 0.6311218
plot(hh$residuals, resnb, xlab='Residuals', ylab='Mean adjacent residuals')
```



```
lw <- nb2listw(nb)
```

That does not look independent.

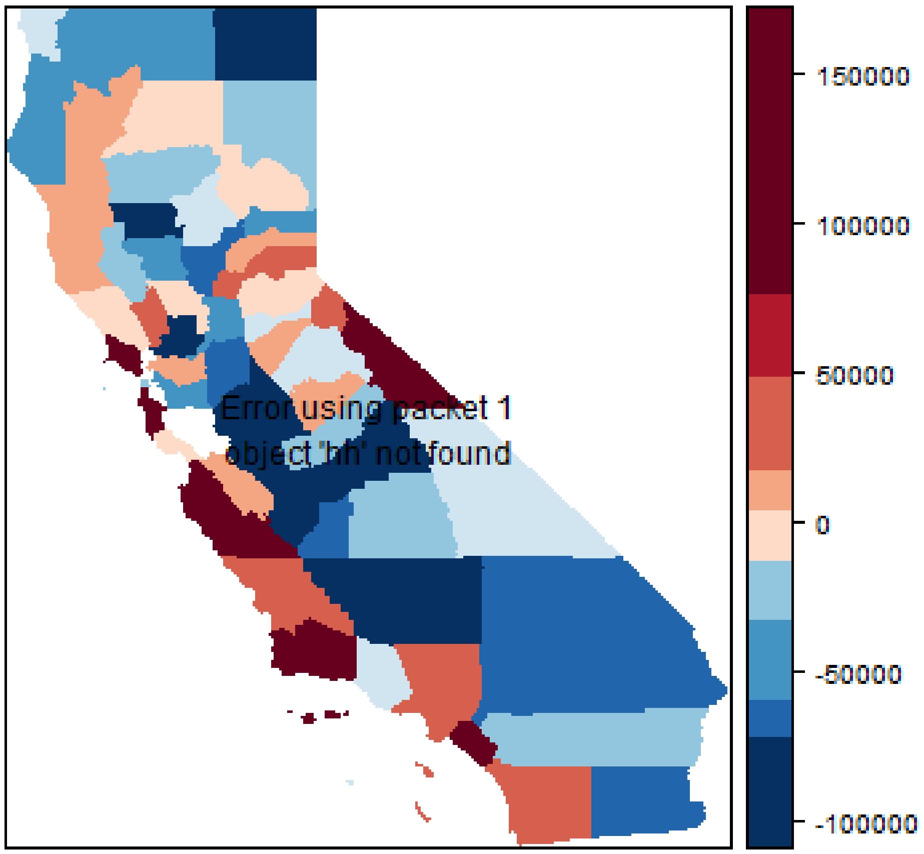
```
moran.mc(hh$residuals, lw, 999)
##
##  Monte-Carlo simulation of Moran I
##
## data:  hh$residuals
## weights: lw
## number of simulations + 1: 1000
##
## statistic = 0.41428, observed rank = 1000, p-value = 0.001
## alternative hypothesis: greater
```

Clearly, there is spatial autocorrelation. Our *p*-values and regression model coefficients cannot be trusted. so let's try SAR models.

7.4 Spatial lag model

Here I show a how to do spatial regression with a spatial lag model (lagsarlm)

```
m1s = lagsarlm(f1, data=hh, lw, tol.solve=1.0e-30)

summary(m1s)
##
## Call:lagsarlm(formula = f1, data = hh, listw = lw, tol.solve = 1e-30)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -108145.2  -49816.3   -1316.3   44604.9  171536.0
##
## Type: lag
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -418674.1   153693.6 -2.7241 0.006448
## age            5533.6     1698.2  3.2584 0.001120
## nBedrooms    127912.8    50859.7  2.5150 0.011903
##
## Rho: 0.77413, LR test value: 34.761, p-value: 3.7282e-09
## Asymptotic standard error: 0.08125
##     z-value: 9.5277, p-value: < 2.22e-16
## Wald statistic: 90.778, p-value: < 2.22e-16
##
## Log likelihood: -727.9964 for lag model
## ML residual variance (sigma squared): 3871700000, (sigma: 62223)
## Number of observations: 58
## Number of parameters estimated: 5
## AIC: 1466, (AIC for lm: 1498.8)
## LM test for residual autocorrelation
## test value: 0.12431, p-value: 0.72441

hh$residuals <- residuals(m1s)
moran.mc(hh$residuals, lw, 999)
##
##  Monte-Carlo simulation of Moran I
##
## data:  hh$residuals
## weights: lw
## number of simulations + 1: 1000
##
## statistic = -0.016, observed rank = 523, p-value = 0.477
## alternative hypothesis: greater

brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)
p <- spplot(hh, "residuals", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")), col=
↪"transparent")
print( p + layer(sp.polygons(hh)) )
```



7.5 Spatial error model

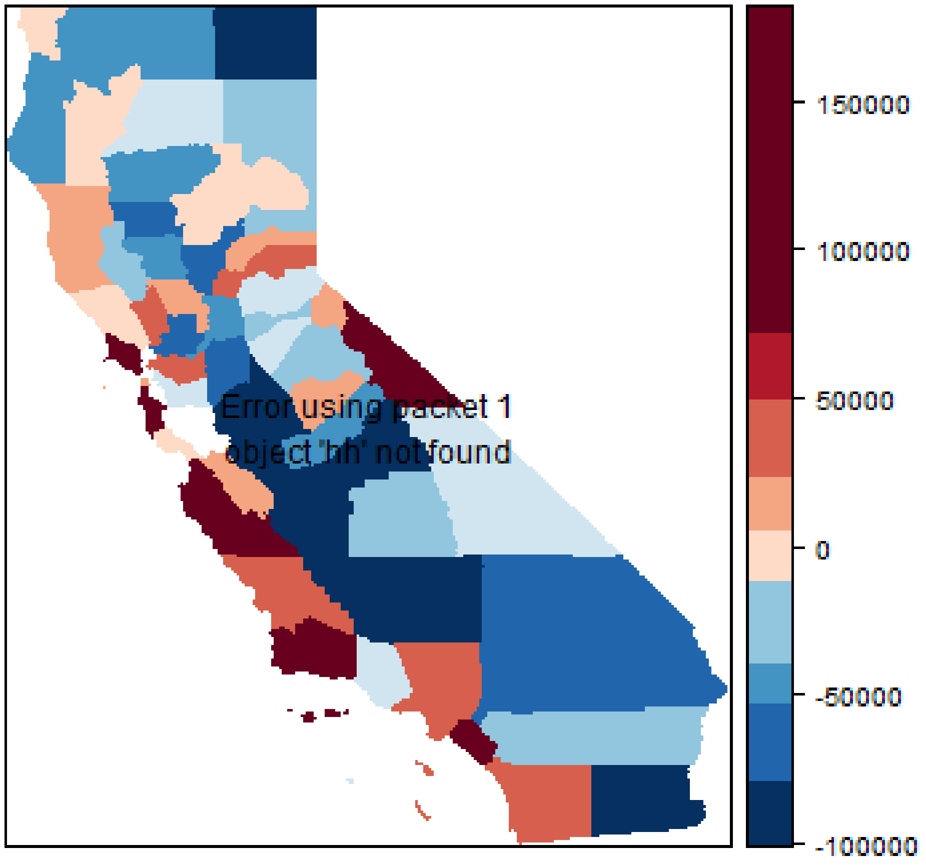
And now with a "Spatial error" (or spatial moving average) models (errorsarlm)

```
m1e = errorsarlm(f1, data=hh, lw, tol.solve=1.0e-30)
summary(m1e)
##
## Call:errorsarlm(formula = f1, data = hh, listw = lw, tol.solve = 1e-30)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -100640.7  -47783.1   -2364.5   44180.6  181876.5
##
## Type: error
## Coefficients: (asymptotic standard errors)
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -185443.8   180133.7 -1.0295  0.30325
## age            4313.6     2214.9  1.9475  0.05147
## nBedrooms    117864.4    51564.7  2.2858  0.02227
##
## Lambda: 0.82151, LR test value: 29.781, p-value: 4.8373e-08
## Asymptotic standard error: 0.071111
```

(continues on next page)

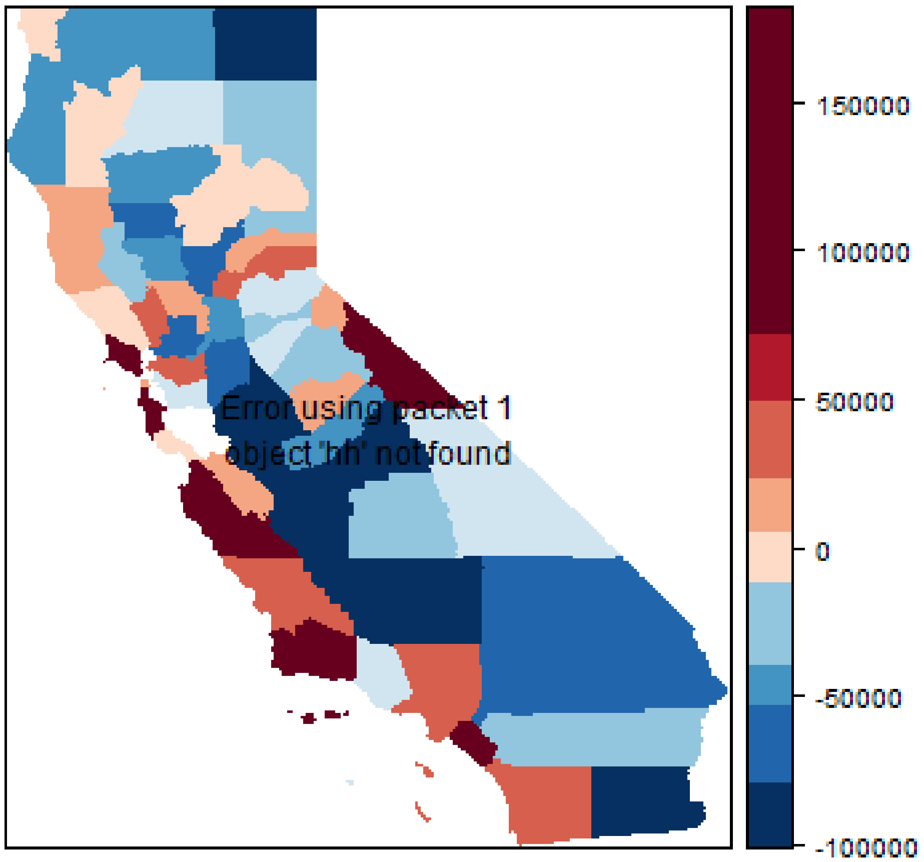
(continued from previous page)

```
##     z-value: 11.552, p-value: < 2.22e-16
## Wald statistic: 133.46, p-value: < 2.22e-16
##
## Log likelihood: -730.4863 for error model
## ML residual variance (sigma squared): 4.07e+09, (sigma: 63797)
## Number of observations: 58
## Number of parameters estimated: 5
## AIC: 1471, (AIC for lm: 1498.8)

hh$residuals <- residuals(m1e)
moran.mc(hh$residuals, lw, 999)
##
##  Monte-Carlo simulation of Moran I
##
## data:  hh$residuals
## weights: lw
## number of simulations + 1: 1000
##
## statistic = 0.039033, observed rank = 763, p-value = 0.237
## alternative hypothesis: greater

brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)
p <- spplot(hh, "residuals", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")),
 col="transparent")
print( p + layer(sp.polygons(hh)) )
```



Are the resiudals spatially autocorrelated for either of these models? Let's plot them for the spatial error model.

```
brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)

p <- spplot(hh, "residuals", at=brks, col.regions=rev(brewer.pal(grps, "RdBu")),
 col="transparent")

print( p + layer(sp.polygons(hh)) )
```



7.6 Questions

Question 2: *The last two maps still seem to show a lot of spatial autocorrelation. But according to the tests there is none. Now why might that be?*

Question 3: *One of the most important, or perhaps THE most important aspect of modeling is variable selection. A misspecified model is never going to be any good, no matter how much you do to, e.g., correct for spatial autocorrelation.* a) Which variables would you choose from the list?

2. Which new variables could you propose to create from the variables in the list.
3. Which other variables could you add, created from the geometries/location (perhaps other geographic data).
4. add a lot of variables and use stepAIC to select an 'optimal' OLS model
5. check for spatial autocorrelation in the residuals of that model

CHAPTER

EIGHT

8. POINT PATTERN ANALYSIS

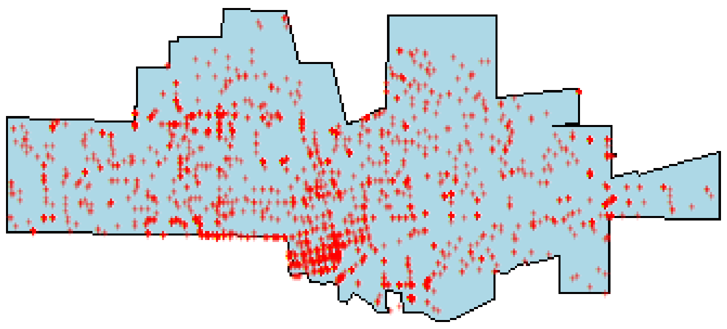
8.1 Introduction

We are using a dataset of crimes in a city. Start by reading in the data.

```
# If have not yet done so, first install the rspatial package (see the line below).␣
↪You may need to install the devtools package first
if (!require("rspatial")) devtools::install_github('rspatial/rspatial')
library(rspatial)
city <- sp_data('city')
crime <- sp_data('crime')
```

Here is a map of both datasets.

```
plot(city, col='light blue')
points(crime, col='red', cex=.5, pch='+')
```



A sorted table of the incidence of crime types.

```
tb <- sort(table(crime$CATEGORY))[-1]
tb
##
```

(continues on next page)

(continued from previous page)

```
##                  Arson                Weapons                Robbery
##                      9                     15                     49
##             Auto Theft     Drugs or Narcotics    Commercial Burglary
##                     86                    134                    143
##            Grand Theft               Assaults                    DUI
##                    143                    172                    212
## Residential Burglary      Vehicle Burglary         Drunk in Public
##                    219                    221                    232
##              Vandalism           Petty Theft
##                    355                    665
```

Let's get the coordinates of the crime data, and for this exercise, remove duplicate crime locations. These are the 'events' we will use below (later we'll go back to the full data set).

```
xy <- coordinates(crime)
dim(xy)
## [1] 2661    2
xy <- unique(xy)
dim(xy)
## [1] 1208    2
head(xy)
##      coords.x1 coords.x2
## [1,]   6628868   1963718
## [2,]   6632796   1964362
## [3,]   6636855   1964873
## [4,]   6626493   1964343
## [5,]   6639506   1966094
## [6,]   6640478   1961983
```

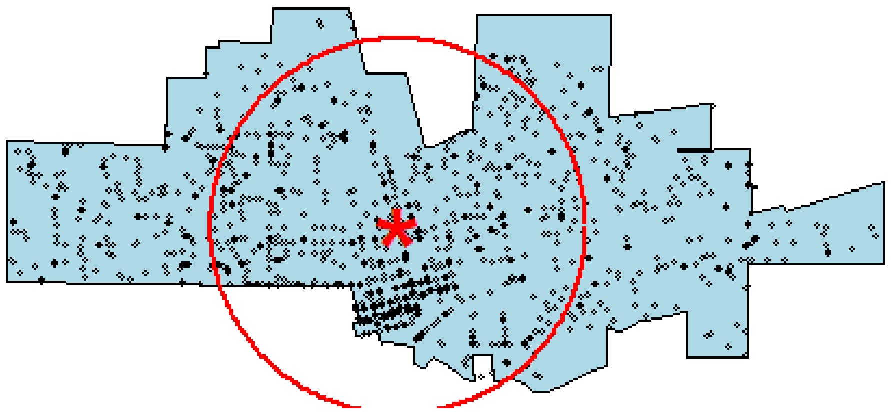
8.2 Basic statistics

Compute the mean center and standard distance for the crime data.

```
# mean center
mc <- apply(xy, 2, mean)
# standard distance
sd <- sqrt(sum((xy[,1] - mc[1])^2 + (xy[,2] - mc[2])^2) / nrow(xy))
```

Plot the data to see what we've got. I add a summary circle (as in Fig 5.2) by dividing the circle in 360 points and compute bearing in radians. I do not think this is particularly helpful, but it might be in other cases. And it is always fun to figure out how to do tis.

```
plot(city, col='light blue')
points(crime, cex=.5)
points(cbind(mc[1], mc[2]), pch='*', col='red', cex=5)

# make a circle
bearing <- 1:360 * pi/180
cx <- mc[1] + sd * cos(bearing)
cy <- mc[2] + sd * sin(bearing)
circle <- cbind(cx, cy)
lines(circle, col='red', lwd=2)
```



8.3 Density

Here is a basic approach to computing point density.

```
CityArea <- area(city)
dens <- nrow(xy) / CityArea
```

Question 1a:*What is the unit of 'dens'?*

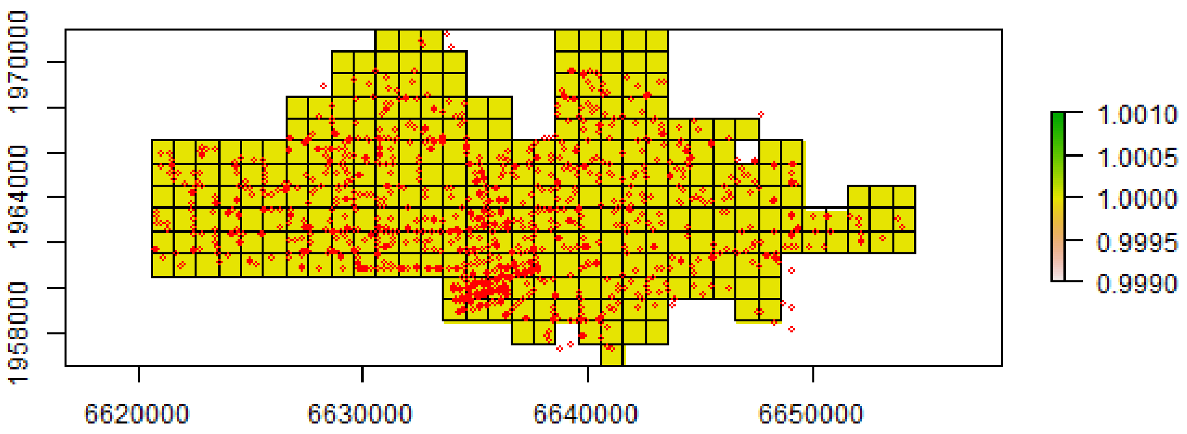
Question 1b:*What is the number of crimes per square km?*

To compute quadrat counts I first create quadrats (a RasterLayer). I get the extent for the raster from the city polygon, and then assign an an arbitrary resolution of 1000. (In real life one should always try a range of resolutions, I think).

```
r <- raster(city)
res(r) <- 1000
r
## class       : RasterLayer
## dimensions  : 15, 34, 510  (nrow, ncol, ncell)
## resolution  : 1000, 1000  (x, y)
## extent      : 6620591, 6654591, 1956519, 1971519  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=lcc +lat_1=38.33333333333334 +lat_2=39.83333333333334 +lat_
↪0=37.66666666666666 +lon_0=-122 +x_0=2000000 +y_0=500000.0000000001 +datum=NAD83␣
↪+units=us-ft +no_defs +ellps=GRS80 +towgs84=0,0,0
```

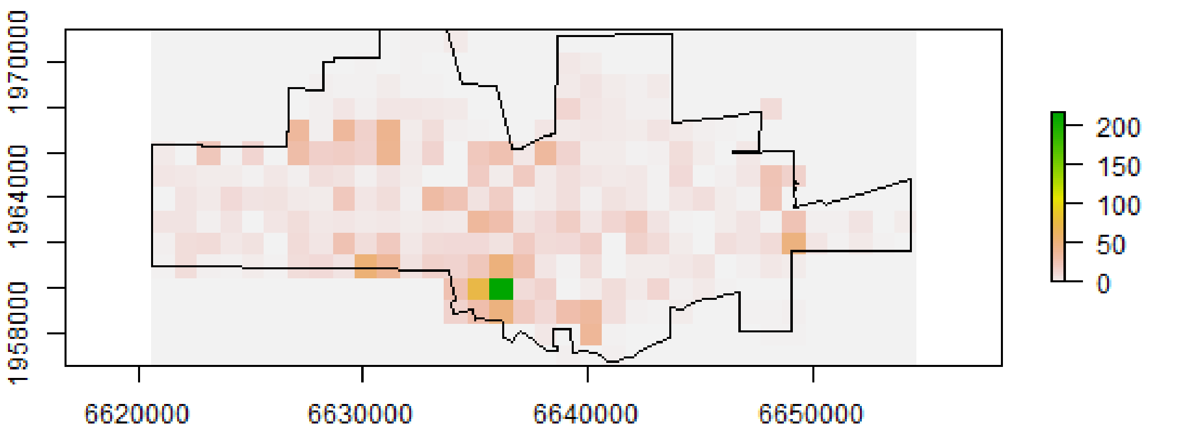
To find the cells that are in the city, and for easy display, I create polygons from the RasterLayer.

```
r <- rasterize(city, r)
plot(r)
quads <- as(r, 'SpatialPolygons')
plot(quads, add=TRUE)
points(crime, col='red', cex=.5)
```



The number of events in each quadrat can be counted using the 'rasterize' function. That function can be used to summarize the number of points within each cell, but also to compute statistics based on the 'marks' (attributes). For example we could compute the number of different crime types) by changing the 'fun' argument to another function (see ?rasterize).

```
nc <- rasterize(coordinates(crime), r, fun='count', background=0)
plot(nc)
plot(city, add=TRUE)
```



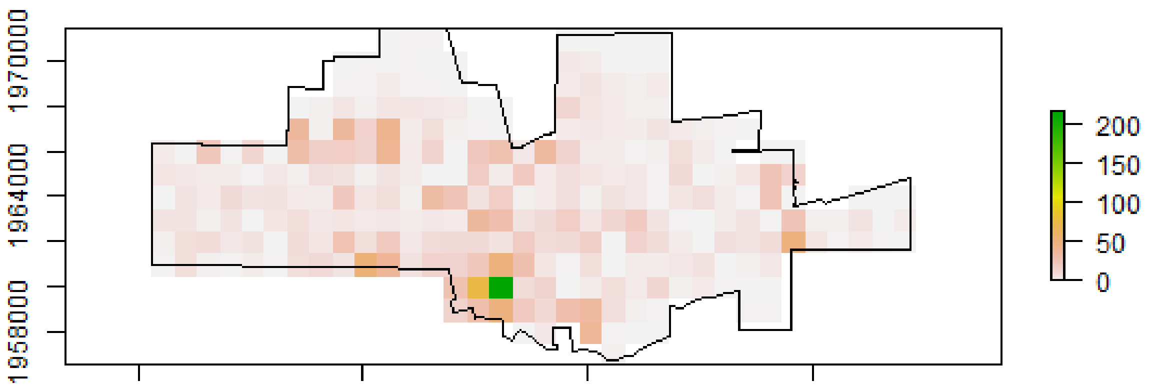
nc has crime counts. As we only have data for the city, the areas outside of the city need to be excluded. We can do that with the mask function (see ?mask).

```
ncrimes <- mask(nc, r)
plot(ncrimes)
```

(continues on next page)

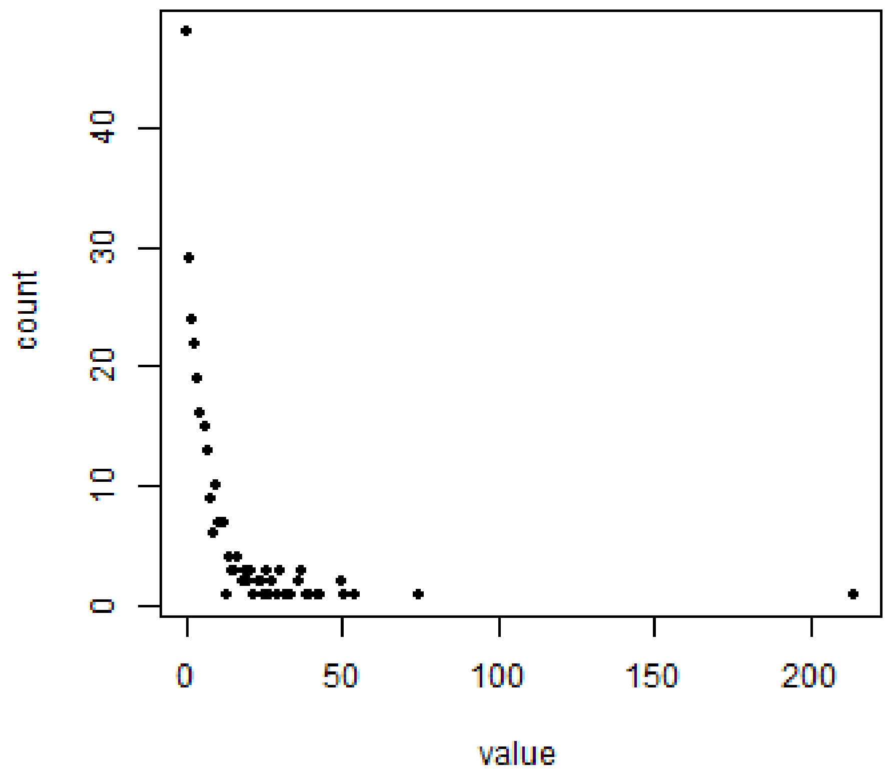
(continued from previous page)

```
plot(city, add=TRUE)
```



Better. Now the frequencies.

```
f <- freq(ncrimes, useNA='no')
head(f)
##      value count
## [1,]     0    48
## [2,]     1    29
## [3,]     2    24
## [4,]     3    22
## [5,]     4    19
## [6,]     5    16
plot(f, pch=20)
```



Does this look like a pattern you would have expected? Now compute average number of cases per quadrat.

```
# number of quadrats
quadrats <- sum(f[,2])
# number of cases
cases <- sum(f[,1] * f[,2])
mu <- cases / quadrats
mu
## [1] 9.261484
```

And create a table like Table 5.1 on page 130

```
ff <- data.frame(f)
colnames(ff) <- c('K', 'X')
ff$Kmu <- ff$K - mu
ff$Kmu2 <- ff$Kmu^2
ff$XKmu2 <- ff$Kmu2 * ff$X
head(ff)
##   K  X       Kmu     Kmu2     XKmu2
## 1 0 48 -9.261484 85.77509 4117.2042
## 2 1 29 -8.261484 68.25212 1979.3115
## 3 2 24 -7.261484 52.72915 1265.4996
## 4 3 22 -6.261484 39.20618  862.5360
## 5 4 19 -5.261484 27.68321  525.9811
## 6 5 16 -4.261484 18.16025  290.5639
```

The observed variance s^2 is

```
s2 <- sum(ff$XKmu2) / (sum(ff$X)-1)
s2
## [1] 276.5555
```

And the VMR is

```
VMR <- s2 / mu
VMR
## [1] 29.86082
```

Question 2:*What does this VMR score tell us about the point pattern?*

8.4 Distance based measures

As we are using a *planar coordinate system* we can use the dist function to compute the distances between pairs of points. If we were using longitude/latitude we could compute distance via spherical trigonometry functions. These are available in the sp, raster, and notably the geosphere package (among others). For example, see `raster::pointDistance`.

```
d <- dist(xy)
class(d)
## [1] "dist"
```

I want to coerce the dist object to a matrix, and ignore distances from each point to itself (the zeros on the diagonal).

```
dm <- as.matrix(d)
dm[1:5, 1:5]
##           1        2         3         4         5
## 1     0.000 3980.843  8070.429  2455.809 10900.016
## 2  3980.843    0.000  4090.992  6303.450  6929.439
## 3  8070.429 4090.992     0.000 10375.958  2918.349
## 4  2455.809 6303.450 10375.958     0.000 13130.236
## 5 10900.016 6929.439  2918.349 13130.236     0.000
diag(dm) <- NA
dm[1:5, 1:5]
##           1        2         3         4         5
## 1        NA 3980.843  8070.429  2455.809 10900.016
## 2  3980.843       NA  4090.992  6303.450  6929.439
## 3  8070.429 4090.992        NA 10375.958  2918.349
## 4  2455.809 6303.450 10375.958        NA 13130.236
## 5 10900.016 6929.439  2918.349 13130.236        NA
```

To get, for each point, the minimum distance to another event, we can use the 'apply' function. Think of the rows as each point, and the columns of all other points (vice versa could also work).

```
dmin <- apply(dm, 1, min, na.rm=TRUE)
head(dmin)
##         1         2         3         4         5         6
## 266.07892 293.58874  47.90260 140.80688  40.06865 510.41231
```

Now it is trivial to get the mean nearest neighbour distance according to formula 5.5, page 131.

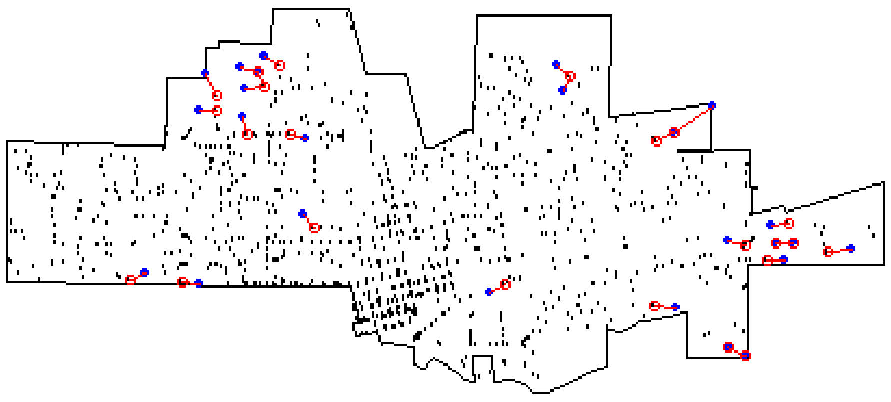
```
mdmin <- mean(dmin)
```

Do you want to know, for each point, *Which* point is its nearest neighbour? Use the 'which.min' function (but note that this ignores the possibility of multiple points at the same minimum distance).

```
wdmin <- apply(dm, 1, which.min)
```

And what are the most isolated cases? That is the furtest away from their nearest neigbor. I plot the top 25. A bit complicated.

```
plot(city)
points(crime, cex=.1)
ord <- rev(order(dmin))

far25 <- ord[1:25]
neighbors <- wdmin[far25]

points(xy[far25, ], col='blue', pch=20)
points(xy[neighbors, ], col='red')

# drawing the lines, easiest via a loop
for (i in far25) {
    lines(rbind(xy[i, ], xy[wdmin[i], ]), col='red')
}
```



Note that some points, but actually not that many, are used as isolated and as a neighbor to an isolated points.

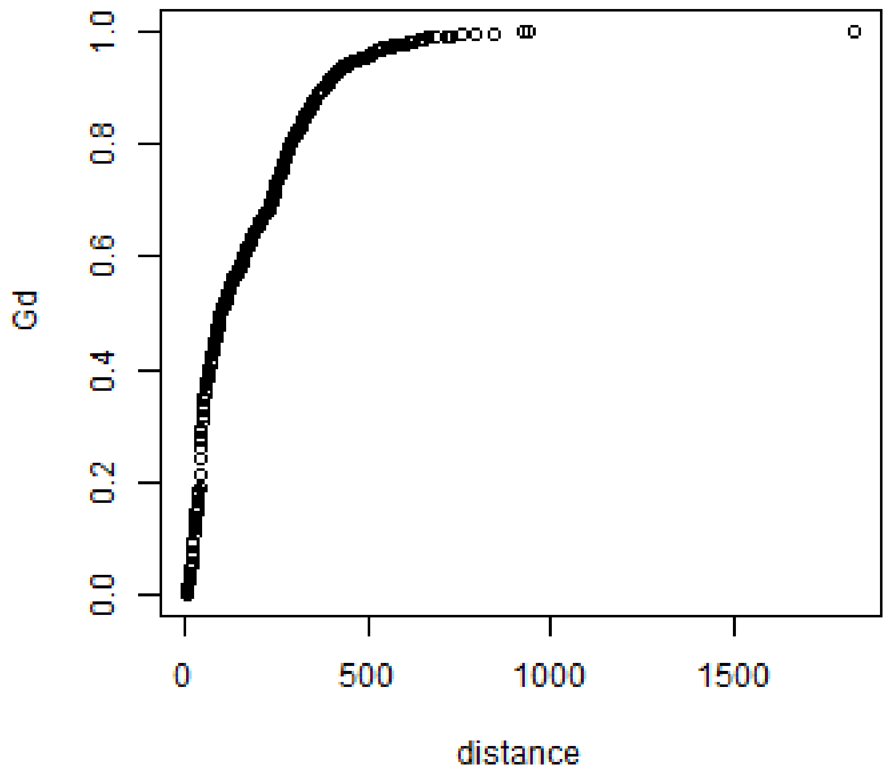
Now on to the G function

```
max(dmin)
## [1] 1829.738
# get the unique distances (for the x-axis)
distance <- sort(unique(round(dmin)))
# compute how many cases there with distances smaller that each x
Gd <- sapply(distance, function(x) sum(dmin < x))
```

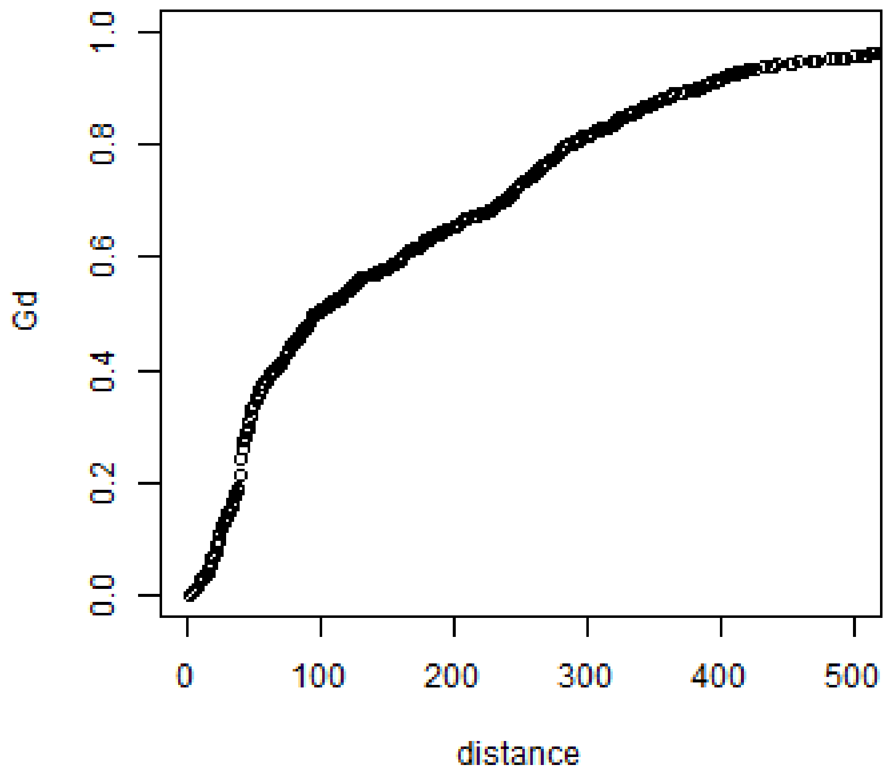
(continues on next page)

(continued from previous page)

```
# normalize to get values between 0 and 1
Gd <- Gd / length(dmin)
plot(distance, Gd)
```



```
# using xlim to exclude the extremes
plot(distance, Gd, xlim=c(0,500))
```

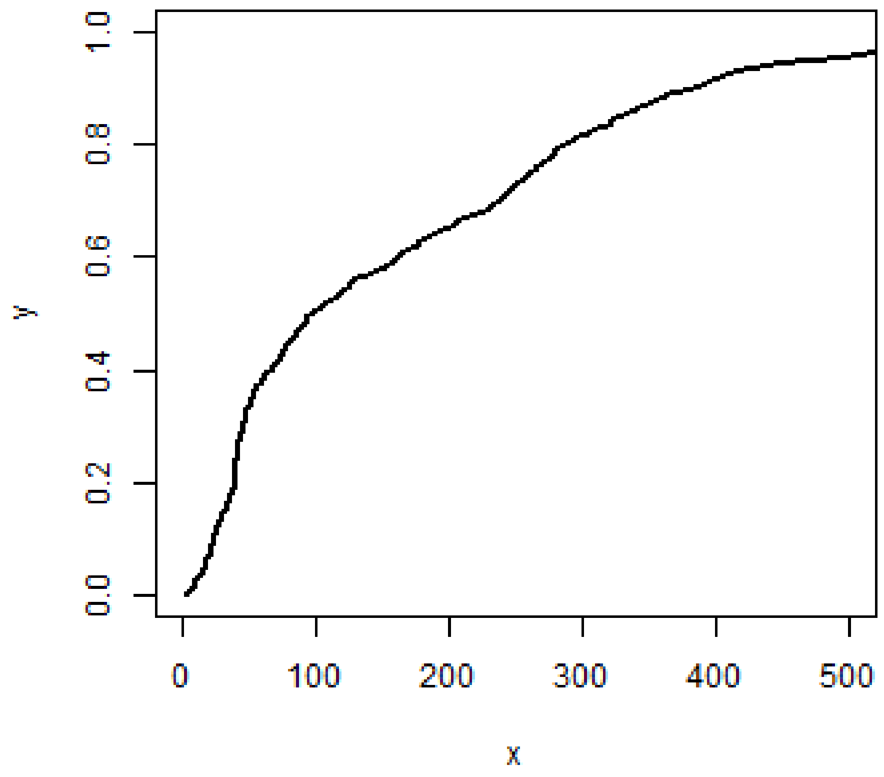


Here is a function to show these values in a more standard way.

```
stepplot <- function(x, y, type='l', add=FALSE, ...) {
    x <- as.vector(t(cbind(x, c(x[-1], x[length(x)]))))
    y <- as.vector(t(cbind(y, y)))
  if (add) {
     lines(x,y, ...)
  } else {
       plot(x,y, type=type, ...)
  }
}
```

And use it for our G function data.

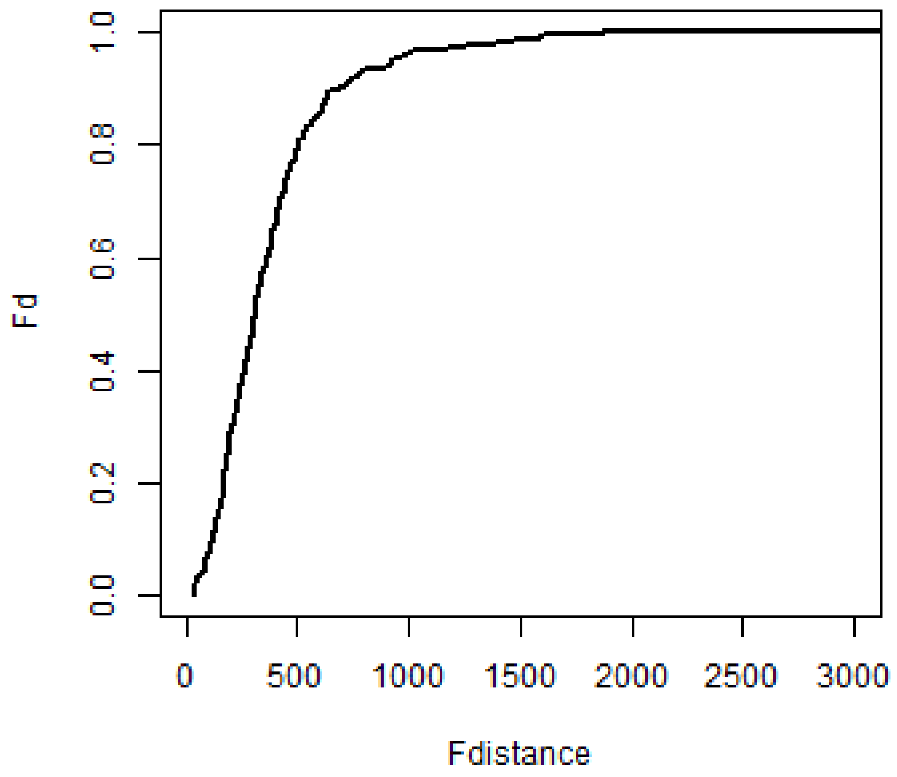
```
stepplot(distance, Gd, type='l', lwd=2, xlim=c(0,500))
```



The steps are so small in our data, that you hardly see the difference.

I use the centers of previously defined raster cells to compute the *F* function.

```
# get the centers of the 'quadrats' (raster cells)
p <- rasterToPoints(r)
# compute distance from all crime sites to these cell centers
d2 <- pointDistance(p[,1:2], xy, longlat=FALSE)

# the remainder is similar to the G function
Fdistance <- sort(unique(round(d2)))
mind <- apply(d2, 1, min)
Fd <- sapply(Fdistance, function(x) sum(mind < x))
Fd <- Fd / length(mind)
plot(Fdistance, Fd, type='l', lwd=2, xlim=c(0,3000))
```

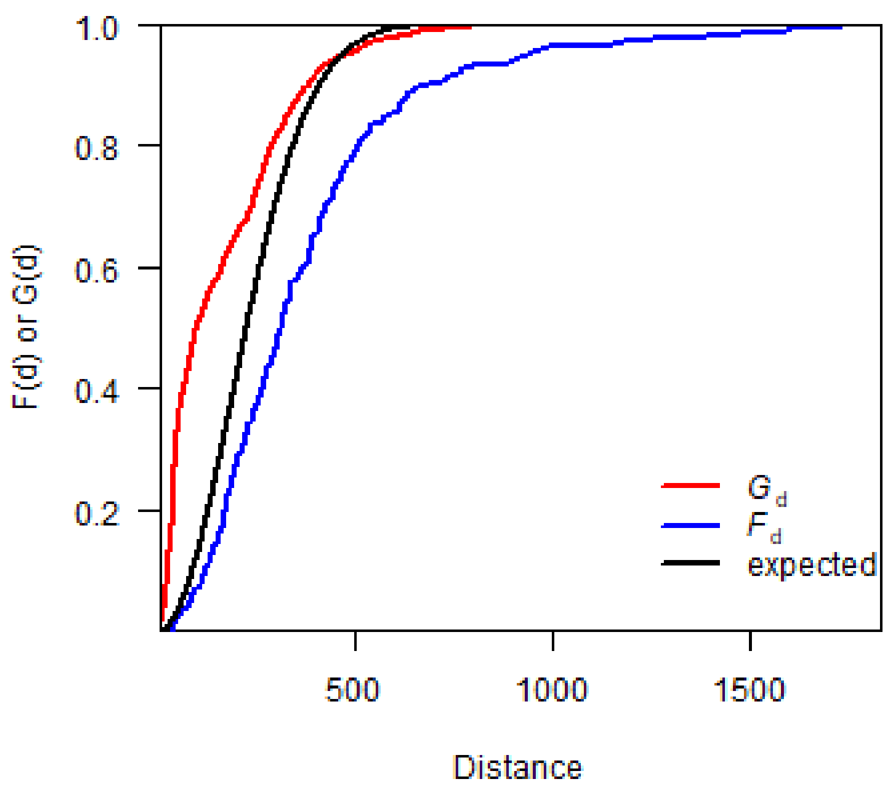


Compute the expected distributon (5.12 on page 145)

```
ef <- function(d, lambda) {
  E <- 1 - exp(-1 * lambda * pi * d^2)
}
expected <- ef(0:2000, dens)
```

Now, let’s combine F and G on one plot.

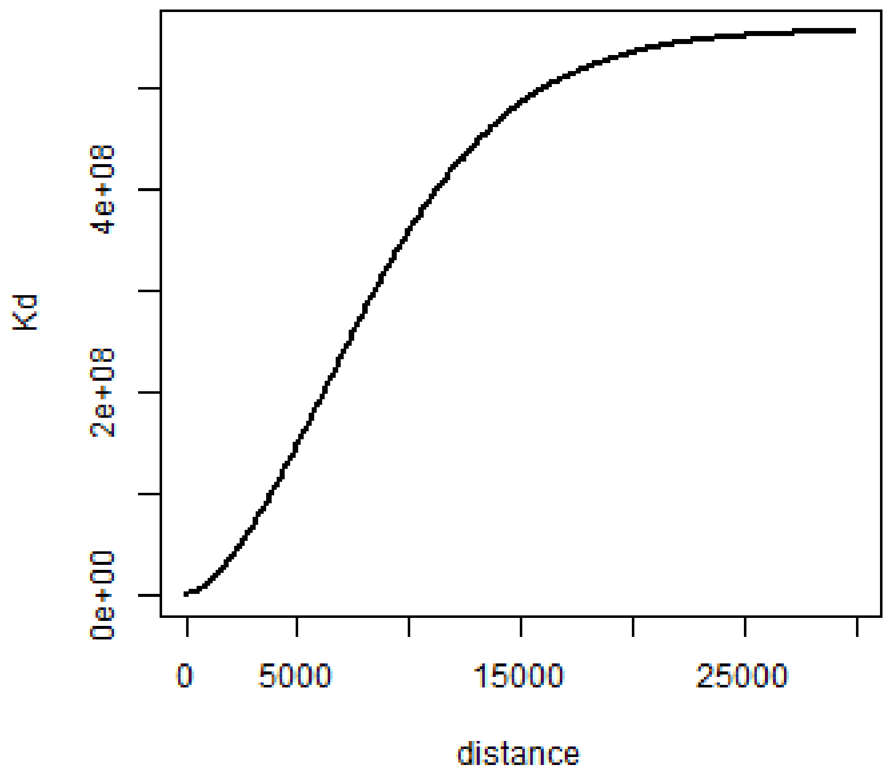
```
plot(distance, Gd, type='l', lwd=2, col='red', las=1,
    ylab='F(d) or G(d)', xlab='Distance', yaxs="i", xaxs="i")
lines(Fdistance, Fd, lwd=2, col='blue')
lines(0:2000, expected, lwd=2)

legend(1200, .3,
   c(expression(italic("G")["d"]), expression(italic("F")["d"]), 'expected'),
   lty=1, col=c('red', 'blue', 'black'), lwd=2, bty="n")
```



Question 3: *What does this plot suggest about the point pattern?*

Finally, let's compute K. Note that I use the original distance matrix 'd' here.

```
distance <- seq(1, 30000, 100)
Kd <- sapply(distance, function(x) sum(d < x)) # takes a while
Kd <- Kd / (length(Kd) * dens)
plot(distance, Kd, type='l', lwd=2)
```



Question 4: *Create a single random pattern of events for the city, with the same number of events as the crime data (object xy). Use function 'spsample'*

Question 5: *Compute the G function, and plot it on a single plot, together with the G function for the observed crime data, and the theoretical expectation (formula 5.12).*

Question 6: *(Difficult!) Do a Monte Carlo simulation (page 149) to see if the 'mean nearest distance' of the observed crime data is significantly different from a random pattern. Use a 'for loop'. First write 'pseudo-code'. That is, say in natural language what should happen. Then try to write R code that implements this.*

8.5 Spatstat package

Above we did some 'home-brew' point pattern analysis, we will now use the spatstat package. In research you would normally use spatstat rather than your own functions, at least for standard analysis. I showed how you make some of these functions in the previous sections, because understanding how to go about that may allow you to take things in directions that others have not gone. The good thing about spatstat is that it very well documented (see http://spatstat.github.io/). The bad thing is that it uses an entirly different sets of classes (ways to represent spatial data) that we we will use in all other labs (classes from sp and raster); but it is not hard to get used to that.

```
library(spatstat)
```

We start with making make a Kernel Density raster. I first create a 'ppp' (point pattern) object, as defined in the spatstat package.

A ppp object has the coordinates of the points **and** the analysis 'window' (study region). To assign the points locations we need to extract the coordinates from our SpatialPoints object. To set the window, we first need to to coerce our SpatialPolygons into an 'owin' object. We need a function from the maptools package for this coercion.

Coerce from SpatialPolygons to an object of class "owin" (observation window)

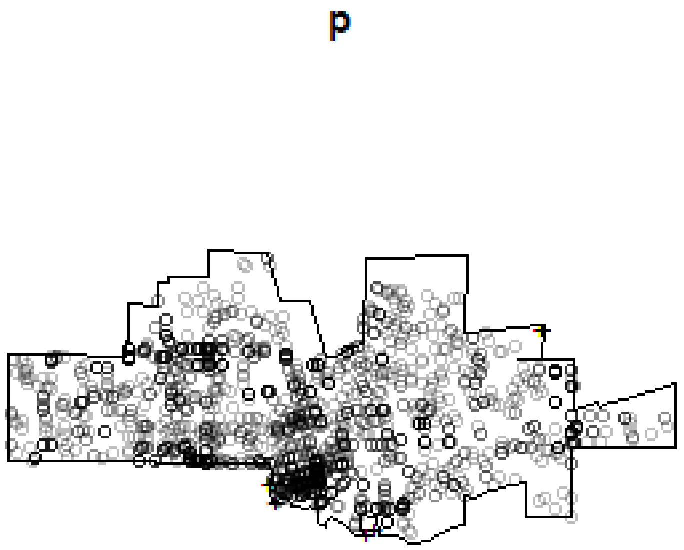
```
library(maptools)
cityOwin <- as.owin(city)
class(cityOwin)
## [1] "owin"
cityOwin
## window: polygonal boundary
## enclosing rectangle: [6620591, 6654380] x [1956729.8, 1971518.9] units
```

Extract coordinates from SpatialPointsDataFrame:

```
pts <- coordinates(crime)
head(pts)
##       coords.x1 coords.x2
## [1,]   6628868   1963718
## [2,]   6632796   1964362
## [3,]   6636855   1964873
## [4,]   6626493   1964343
## [5,]   6639506   1966094
## [6,]   6640478   1961983
```

Now we can create a 'ppp' (point pattern) object

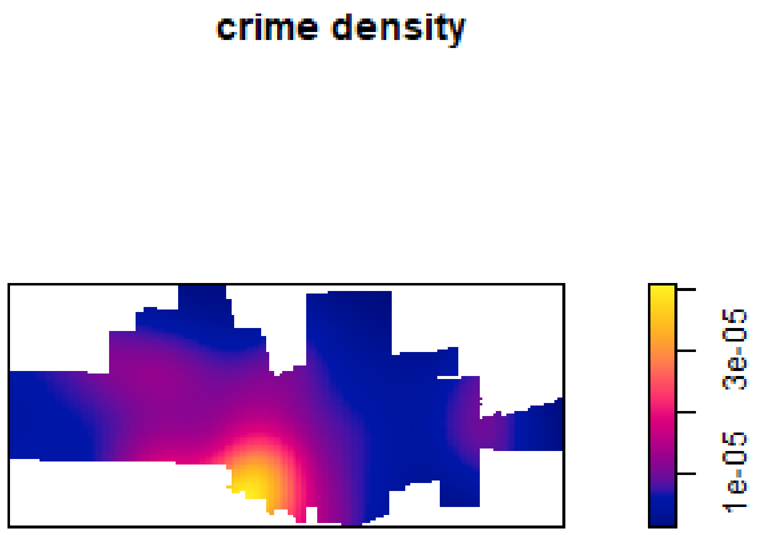
```
p <- ppp(pts[,1], pts[,2], window=cityOwin)
## Warning: 20 points were rejected as lying outside the specified window
## Warning: data contain duplicated points
class(p)
## [1] "ppp"
p
## Planar point pattern: 2641 points
## window: polygonal boundary
## enclosing rectangle: [6620591, 6654380] x [1956729.8, 1971518.9] units
## *** 20 illegal points stored in attr(,"rejects") ***
plot(p)
## Warning in plot.ppp(p): 20 illegal points also plotted
```



Note the warning message about 'illegal' points. Do you see them and do you understand why they are illegal?

Having all the data well organized, it is now easy to compute Kernel Density

```
ds <- density(p)
class(ds)
## [1] "im"
plot(ds, main='crime density')
```



Density is the number of points per unit area. Let's ceck if the numbers makes sense, by adding them up and mulitplying with the area of the raster cells. I use raster package functions for that.

```
nrow(pts)
## [1] 2661
r <- raster(ds)
s <- sum(values(r), na.rm=TRUE)
s * prod(res(r))
## [1] 2640.556
```

Looks about right. We can also get the information directly from the "im" (image) object

```
str(ds)
## List of 10
##  $ v     : num [1:128, 1:128] NA NA NA NA NA NA NA NA NA NA ...
##  $ dim   : int [1:2] 128 128
##  $ xrange: num [1:2] 6620591 6654380
##  $ yrange: num [1:2] 1956730 1971519
##  $ xstep : num 264
##  $ ystep : num 116
##  $ xcol  : num [1:128] 6620723 6620987 6621251 6621515 6621779 ...
##  $ yrow  : num [1:128] 1956788 1956903 1957019 1957134 1957250 ...
##  $ type  : chr "real"
##  $ units :List of 3
```

(continues on next page)

(continued from previous page)

```
##   ..$ singular  : chr "unit"
##   ..$ plural    : chr "units"
##   ..$ multiplier: num 1
##   ..- attr(*, "class")= chr "unitname"
##  - attr(*, "class")= chr "im"
##  - attr(*, "sigma")= num 1849
##  - attr(*, "kernel")= chr "gaussian"
sum(ds$v, na.rm=TRUE) * ds$xstep * ds$ystep
## [1] 2640.556
p$n
## [1] 2641
```

Here's another, lenghty, example of generalization. We can interpolate population density from (2000) census data; assigning the values to the centroid of a polygon (as explained in the book, but not a great technique). We use a shapefile with census data.

```
census <- sp_data("census2000.rds")
```

To compute population density for each census block, we first need to get the area of each polygon. I transform density from persons per feet^2 to persons per mile^2, and then compute population density from POP2000 and the area

```
census$area <- area(census)
census$area <- census$area/27878400
census$dens <- census$POP2000 / census$area
```

Now to get the centroids of the census blocks we can use the 'coordinates' function again. Note that it actually does something quite different (with a `SpatialPolygons*` object) then in the case above (with a SpatialPoints* object).

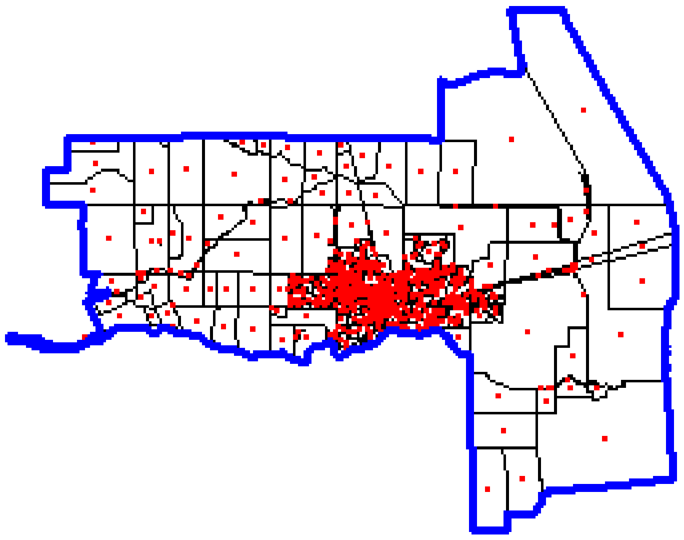
```
p <- coordinates(census)
head(p)
##      [,1]    [,2]
## 0 6666671 1991720
## 1 6655379 1986903
## 2 6604777 1982474
## 3 6612242 1981881
## 4 6613488 1986776
## 5 6616743 1986446
```

To create the 'window' we dissolve all polygons into a single polygon.

```
win <- aggregate(census)
```

Let's look at what we have:

```
plot(census)
points(p, col='red', pch=20, cex=.25)
plot(win, add=TRUE, border='blue', lwd=3)
```



Now we can use ‘smooth.ppp’ to interpolate. Population density at the points is referred to as the ‘marks’

```
owin <- as.owin(win)
pp <- ppp(p[,1], p[,2], window=owin, marks=census$dens)
## Warning: 1 point was rejected as lying outside the specified window
pp
## Marked planar point pattern: 645 points
## marks are numeric, of storage type  'double'
## window: polygonal boundary
## enclosing rectangle: [6576938, 6680926] x [1926586.1, 2007558.2] units
## *** 1 illegal point stored in attr(,"rejects") ***
```

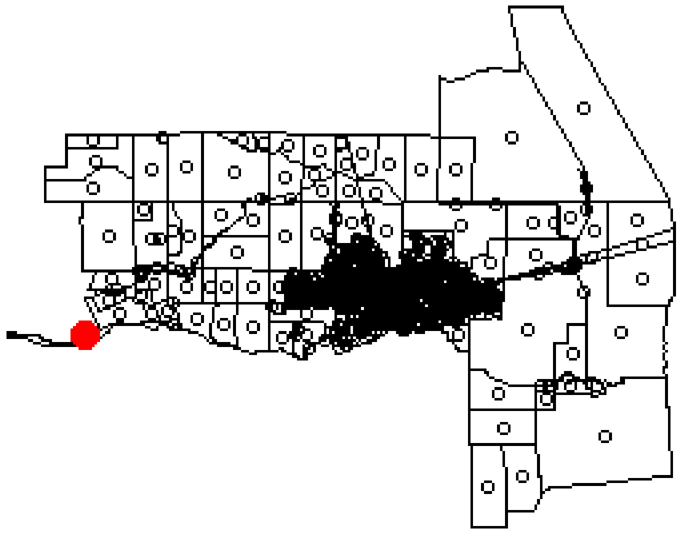
Note the warning message: “1 point was rejected as lying outside the specified window”. That is odd, there is a polygon that has a centroid that is outside of the polygon. This can happen with, e.g., kidney shaped polygons.

Let’s find and remove this point that is outside the study area.

```
sp <- SpatialPoints(p, proj4string=CRS(proj4string(win)))
library(rgeos)
i <- gIntersects(sp, win, byid=TRUE)
which(!i)
## [1] 588
```

Let’s see where it is:

```
plot(census)
points(sp)
points(sp[!i,], col='red', cex=3, pch=20)
```



You can zoom in using the code below. After running the next line, click on your map twice to zoom to the red dot, otherwise you cannot continue:

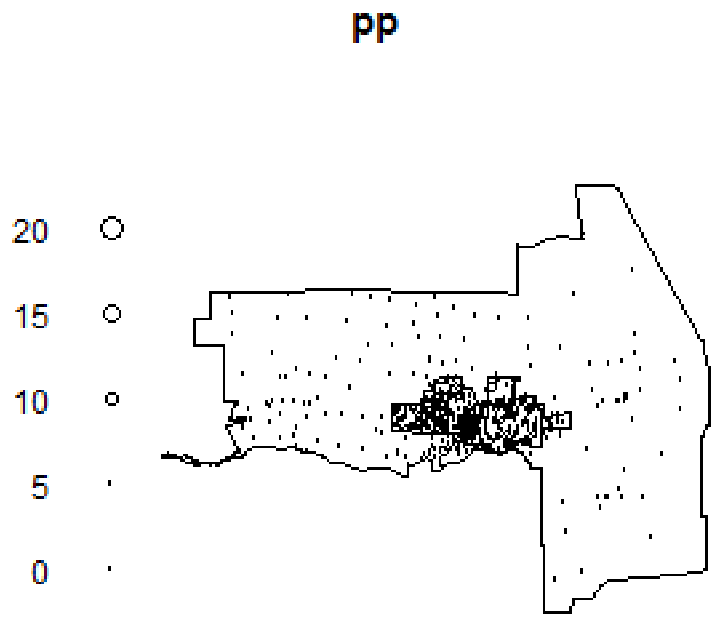
```
zoom(census)
```

And add the red points again

```
points(sp[!i,], col='red')
```

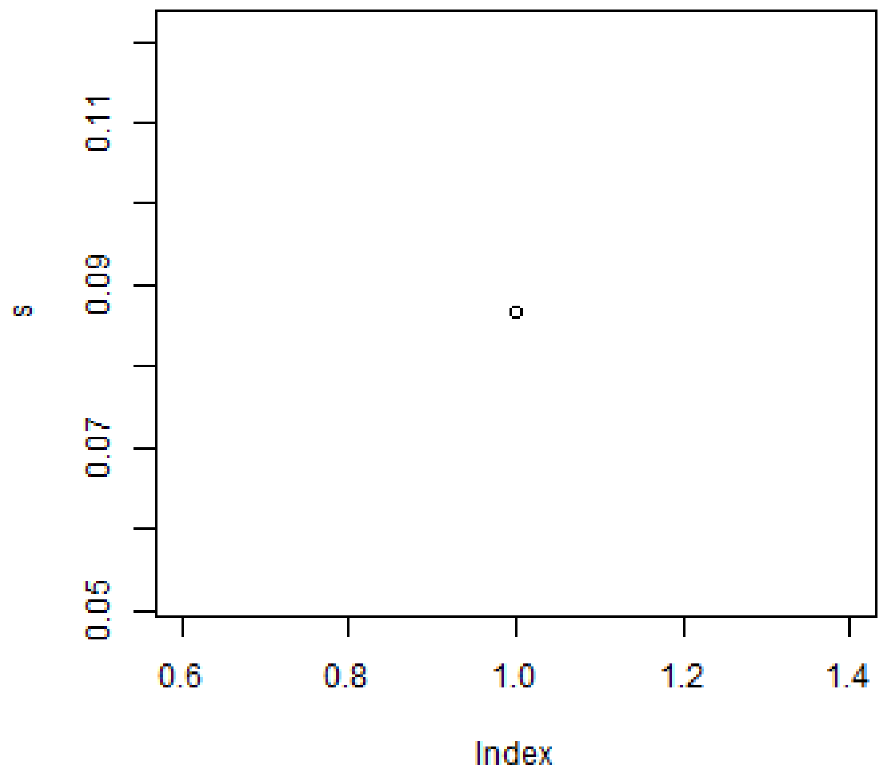
To only use points that intersect with the window polygon, that is, where 'i == TRUE':

```
pp <- ppp(p[i,1], p[i,2], window=owin, marks=census$dens[i])
plot(pp)
plot(city, add=TRUE)
```



And to get a smooth interpolation of population density.

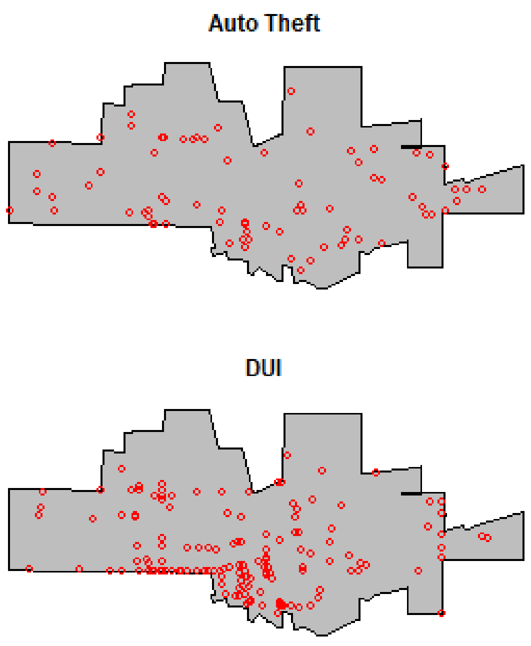
```
s <- smooth.ppp(pp)
## Error in smooth.ppp(pp): could not find function "smooth.ppp"
plot(s)
plot(city, add=TRUE)
```



Population density could establish the "population at risk" (to commit a crime) for certain crimes, but not for others.

Maps with the city limits and the incidence of 'auto-theft', 'drunk in public', 'DUI', and 'Arson'.

```
par(mfrow=c(2,2), mai=c(0.25, 0.25, 0.25, 0.25))
for (offense in c("Auto Theft", "Drunk in Public", "DUI", "Arson")) {
  plot(city, col='grey')
    acrime <- crime[crime$CATEGORY == offense, ]
    points(acrime, col = "red")
    title(offense)
}
```

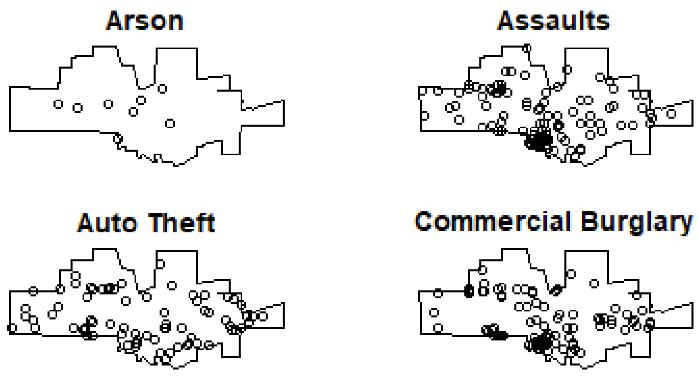


Create a marked point pattern object (ppp) for all crimes. It is important to coerce the marks to a factor variable.

```
crime$fcat <- as.factor(crime$CATEGORY)
w <- as.owin(city)
xy <- coordinates(crime)
mpp <- ppp(xy[,1], xy[,2], window = w, marks=crime$fcat)
## Warning: 20 points were rejected as lying outside the specified window
## Warning: data contain duplicated points
```

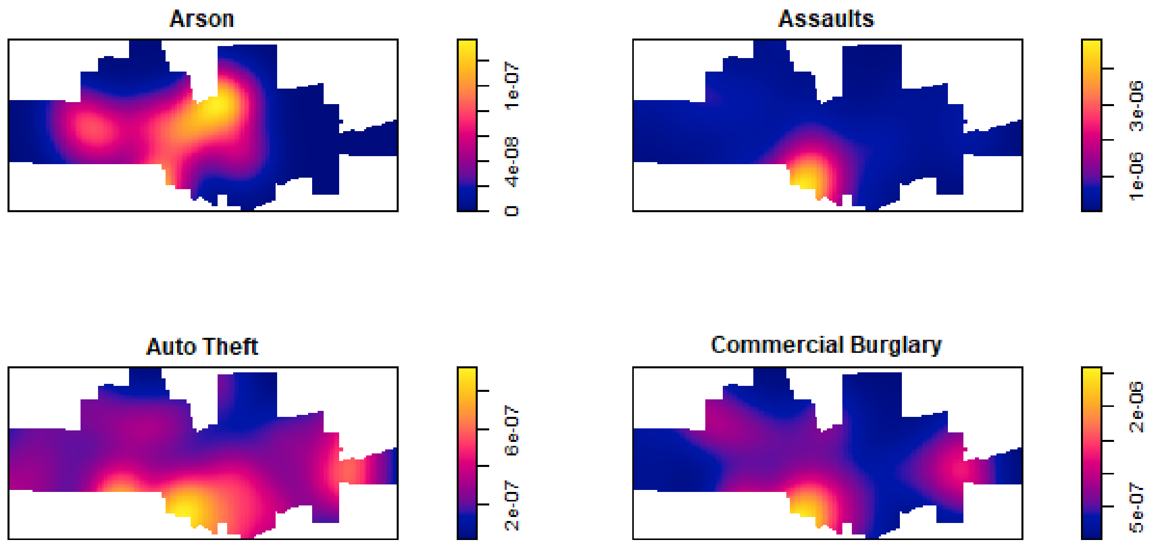
We can split the mpp object by category (crime)

```
spp <- split(mpp)
plot(spp[1:4], main='')
```



The crime density by category:

```
plot(density(spp[1:4]), main='')
```



And produce K-plots (with an envelope) for ‘drunk in public’ and ‘Arson’. Can you explain what they mean?

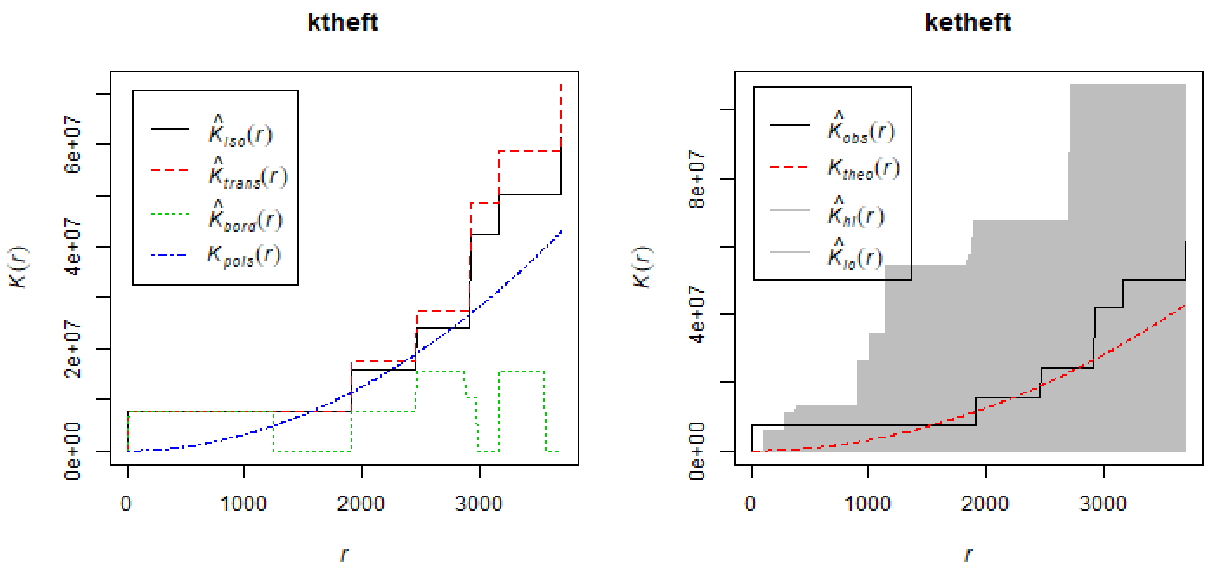
```
spatstat.options(checksegments = FALSE)
ktheft <- Kest(spp$"Auto Theft")
ketheft <- envelope(spp$"Auto Theft", Kest)
```

(continues on next page)

(continued from previous page)

```
## Generating 99 simulations of CSR  ...
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,␣
↪24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
## 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
↪ 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
## 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
↪ 98,  99.
##
## Done.
ktheft <- Kest(spp$"Arson")
ketheft <- envelope(spp$"Arson", Kest)
## Generating 99 simulations of CSR  ...
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,␣
↪24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
## 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59,
↪ 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76,
## 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97,
↪ 98,  99.
##
## Done.
```

```
par(mfrow=c(1,2))
plot(ktheft)
plot(ketheft)
```



Let's try to answer the question you have been wanting to answer all along. Is population density a good predictor of being (booked for) "drunk in public" and for "Arson"? One approach is to do a Kolmogorov-Smirnov ('kstest') on 'Drunk in Public' and 'Arson', using population density as a covariate:

```
KS.arson <- kstest(spp$Arson, ds)
## Error in kstest(spp$Arson, ds): could not find function "kstest"
KS.arson
## Error in eval(expr, envir, enclos): object 'KS.arson' not found
KS.drunk <- kstest(spp$'Drunk in Public', ds)
```

(continues on next page)

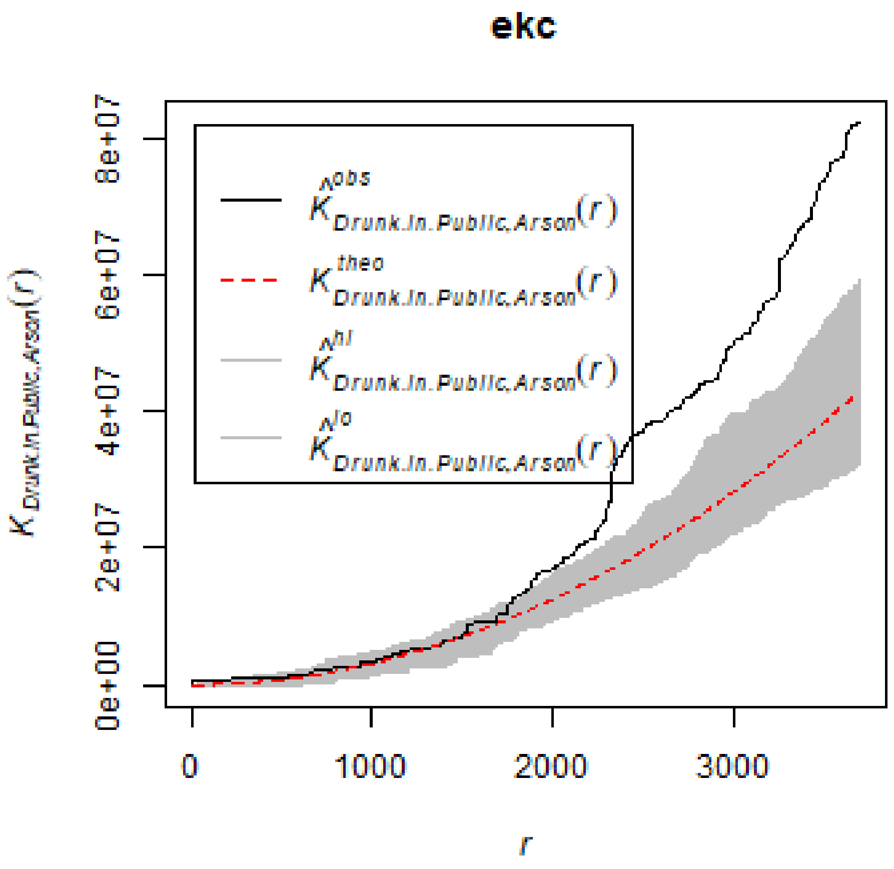
(continued from previous page)

```
## Error in kstest(spp$"Drunk in Public", ds): could not find function "kstest"
KS.drunk
## Error in eval(expr, envir, enclos): object 'KS.drunk' not found
```

Question 7: *Why is the result surprising, or not surprising?*

We can also compare the patterns for “drunk in public” and for “Arson” with the KCross function.

```
kc <- Kcross(mpp, i = "Drunk in Public", j = "Arson")
ekc <- envelope(mpp, Kcross, nsim = 50, i = "Drunk in Public", j = "Arson")
## Generating 50 simulations of CSR  ...
## 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,␣
↪24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38,
## 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,  50.
##
## Done.
plot(ekc)
```



Much more about point pattern analysis with spatstat is available here