Spatial Data Analysis with R

Robert J. Hijmans

Nov 30, 2023

CONTENTS

CHAPTER

ONE

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

SCALE AND DISTANCE

2.1 Introduction

Scale, aggregation, and distance are two key concepts in spatial data analysis that can be difficult 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".

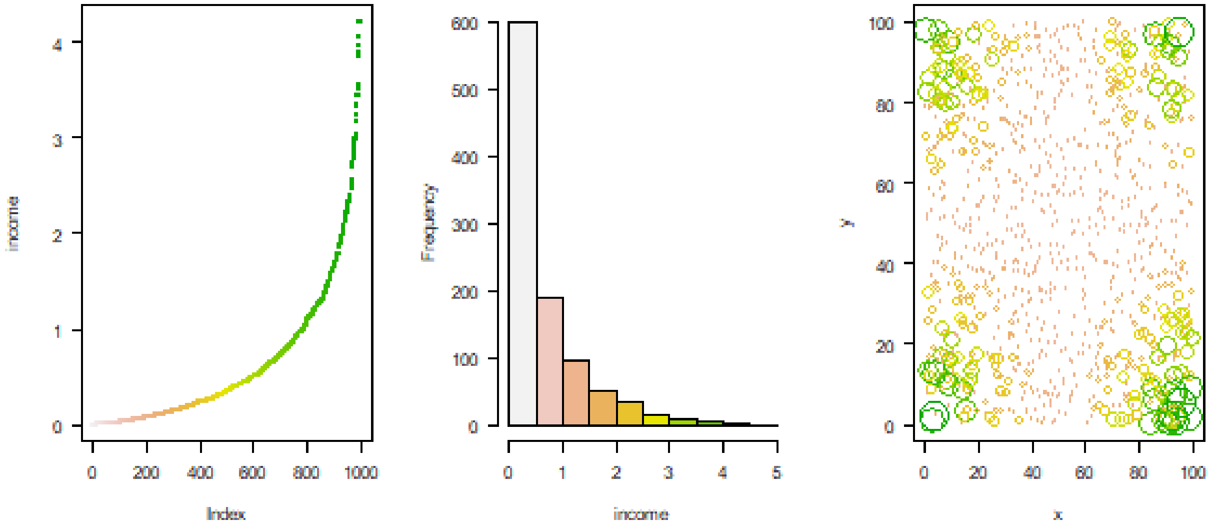
To illustrate the effect of zonation and aggregation, I create a region with 1000 households. For each household we know where they live and what their annual income is. I then aggregate the data to a set of zones.

The income distribution data

```
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 third that there is a clear spatial pattern in where the rich and the poor live.

```
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.

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

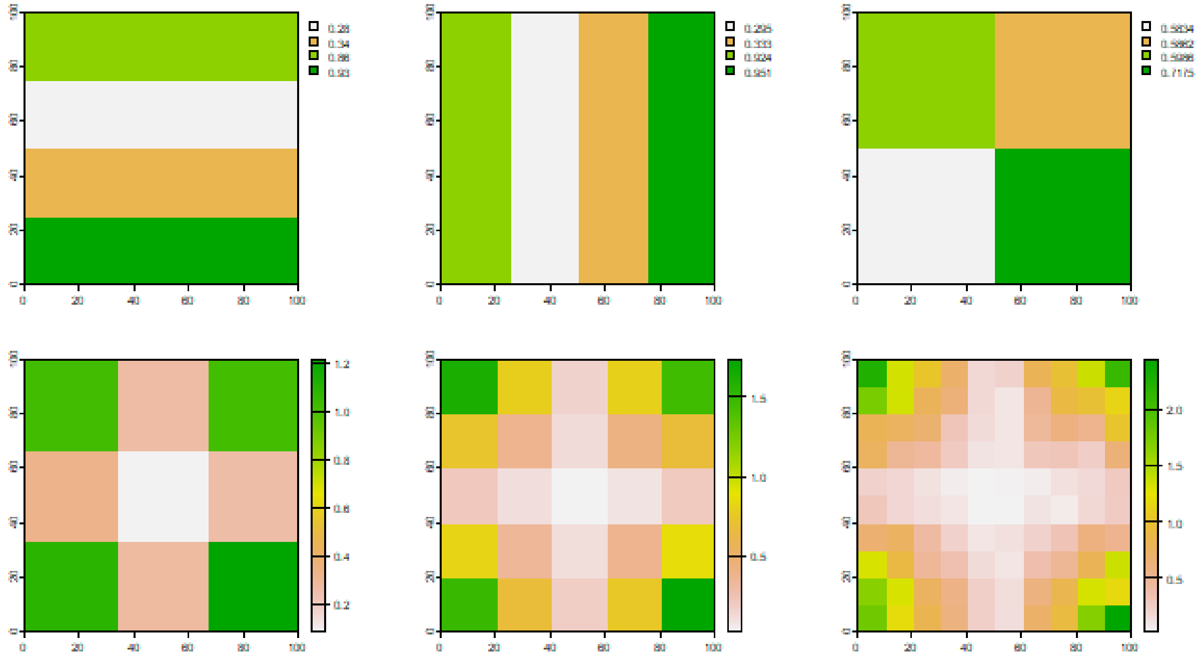
For our data set the Gini coefficient is 0.581.

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.

```r
library(terra)
## terra 1.7.62
v <- vect(xy)
v$income <- income
r1 <- rast(ncol=1, nrow=4, xmin=0, xmax=100, ymin=0, ymax=100)
r1 <- rasterize(v, r1, "income", mean)

r2 <- rast(ncol=4, nrow=1, xmin=0, xmax=100, ymin=0, ymax=100)
r2 <- rasterize(v, r2, "income", mean)

r3 <- rast(ncol=2, nrow=2, xmin=0, xmax=100, ymin=0, ymax=100)
r3 <- rasterize(v, r3, "income", mean)

r4 <- rast(ncol=3, nrow=3, xmin=0, xmax=100, ymin=0, ymax=100)
r4 <- rasterize(v, r4, "income", mean)

r5 <- rast(ncol=5, nrow=5, xmin=0, xmax=100, ymin=0, ymax=100)
r5 <- rasterize(v, r5, "income", mean)

r6 <- rast(ncol=10, nrow=10, xmin=0, xmax=100, ymin=0, ymax=100)
r6 <- rasterize(v, r6, "income", mean)
```

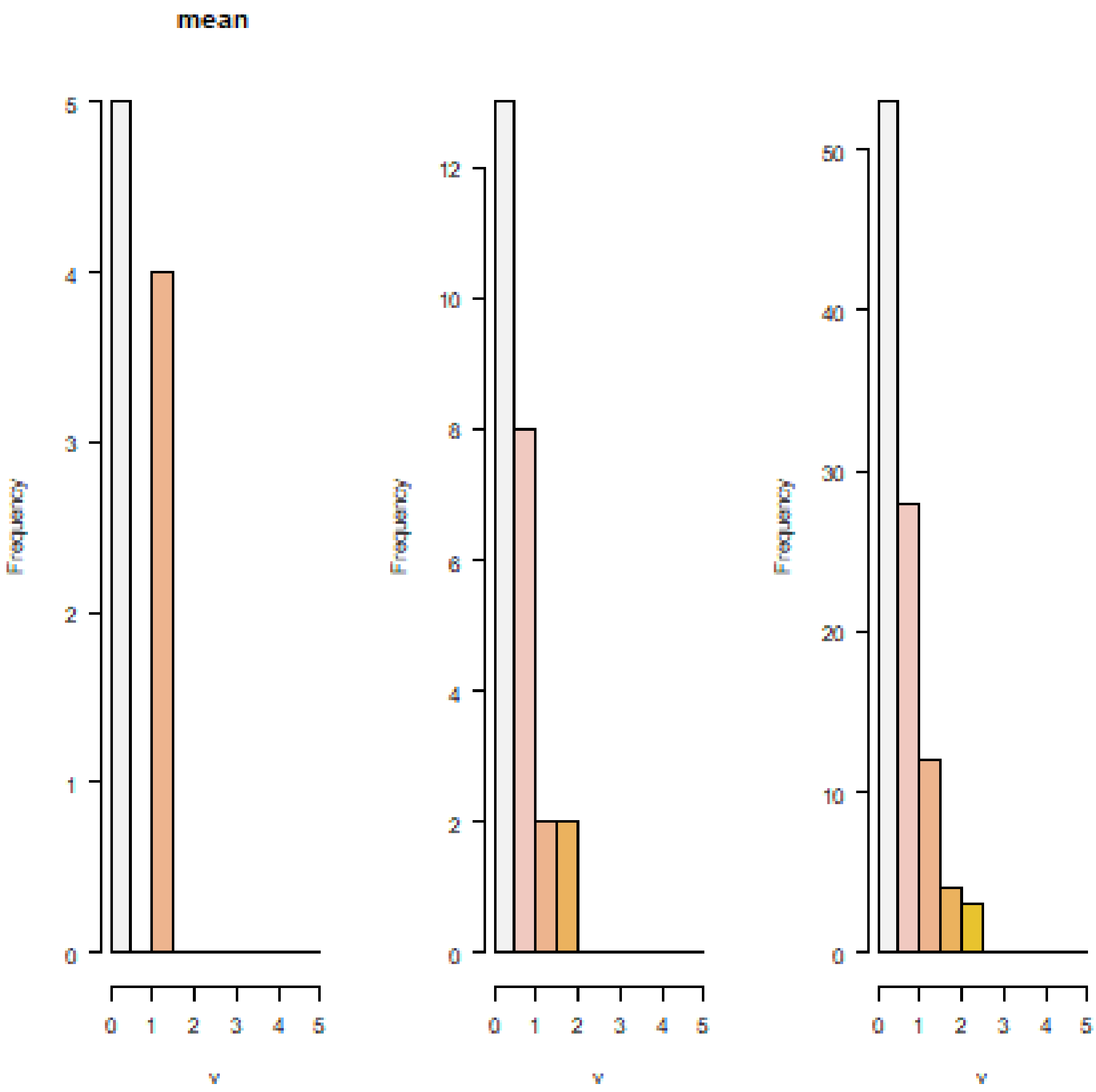
Have a look at the plots of the income distribution and the sub-regional averages.

```
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, 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))
hist(r6, main="", col=rev(terrain.colors(10)), xlim=c(0,5), breaks=seq(0, 5, 0.5))
```



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.

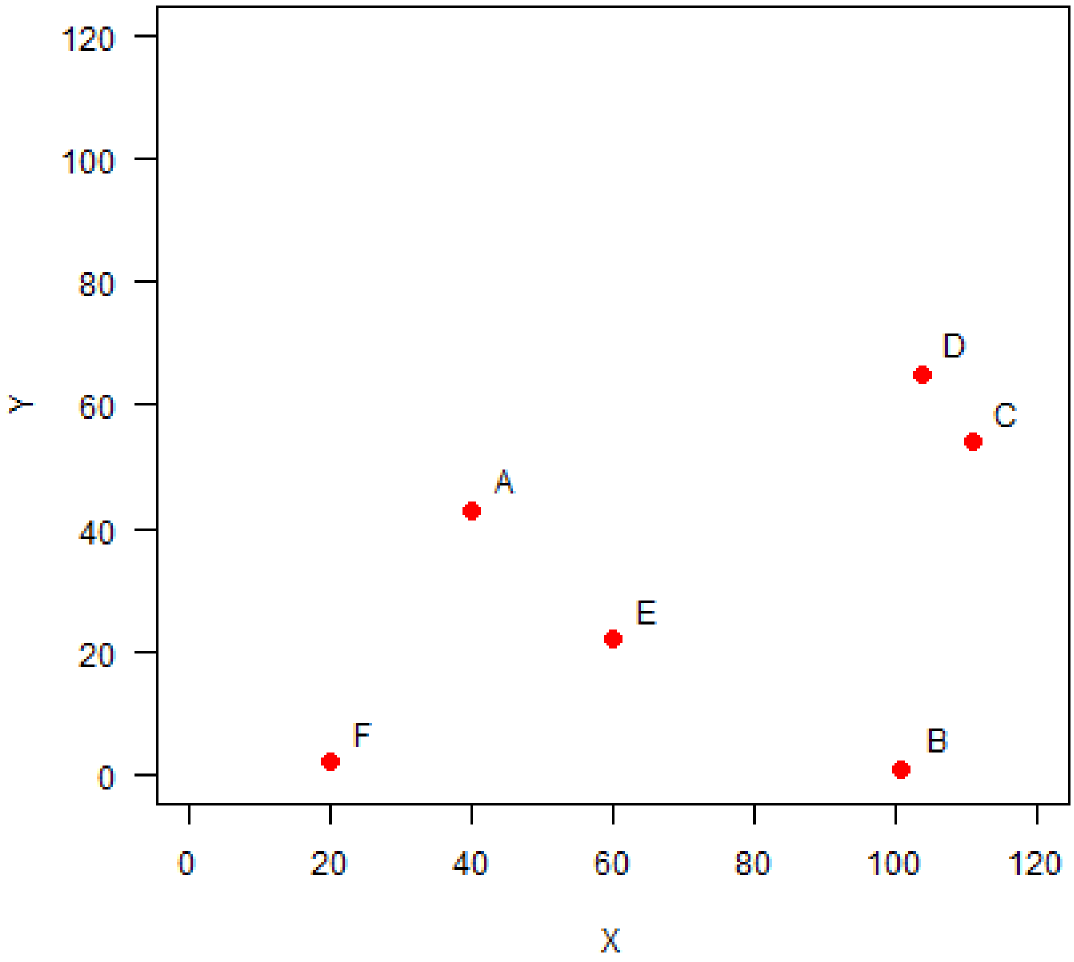
Let’s create a distance matrix from a set of points. We start with a set of points

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.

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

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]
## A    5    6
## B    5    3
## C    4    2
## D    3    5
```

(continues on next page)

(continued from previous page)

```
## 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
```

(continues on next page)

(continued from previous page)

```
# 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

```
p <- vect(system.file("ex/lux.shp", package="terra"))
```

We create a "rook's case" neighbors matrix.

```
wr <- adjacent(p, "rook", pairs=FALSE)
dim(wr)
## [1] 12 12
wr[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.

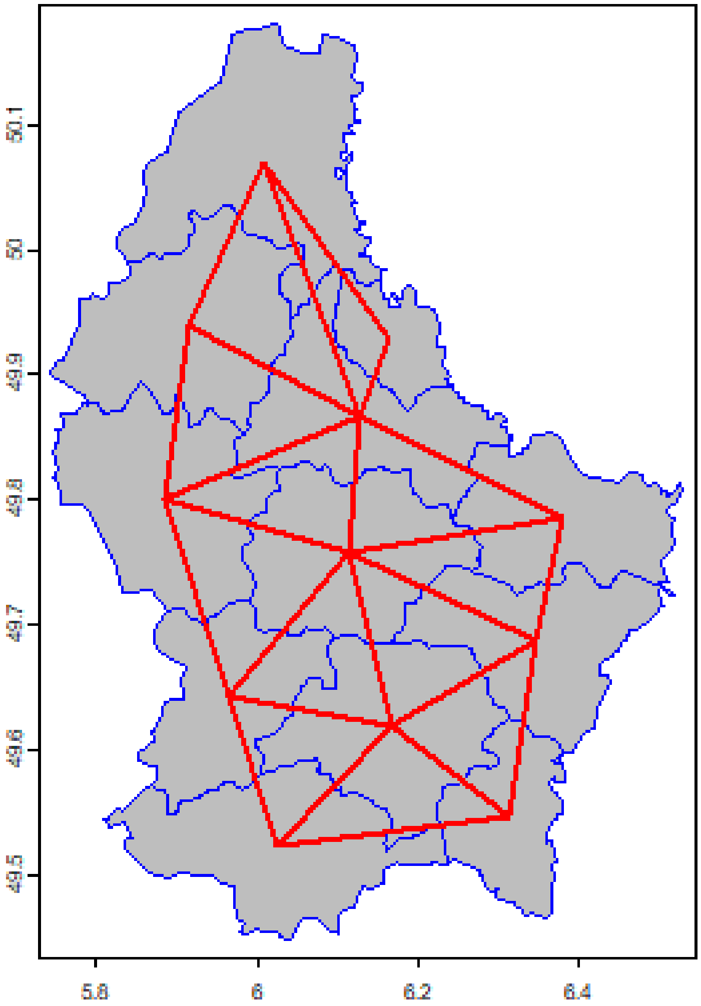
```
i <- rowSums(wr)
i
##  1  2  3  4  5  6  7  8  9 10 11 12
##  3  6  4  2  3  3  3  4  4  3  5  6
```

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")
nb <- adjacent(p, "rook")
v <- centroids(p)
p1 <- v[nb[,1], ]
p2 <- v[nb[,2], ]
lines(p1, p2, col="red", lwd=2)
```



Now some alternative approaches to compute “spatial influence”.

Distance based:

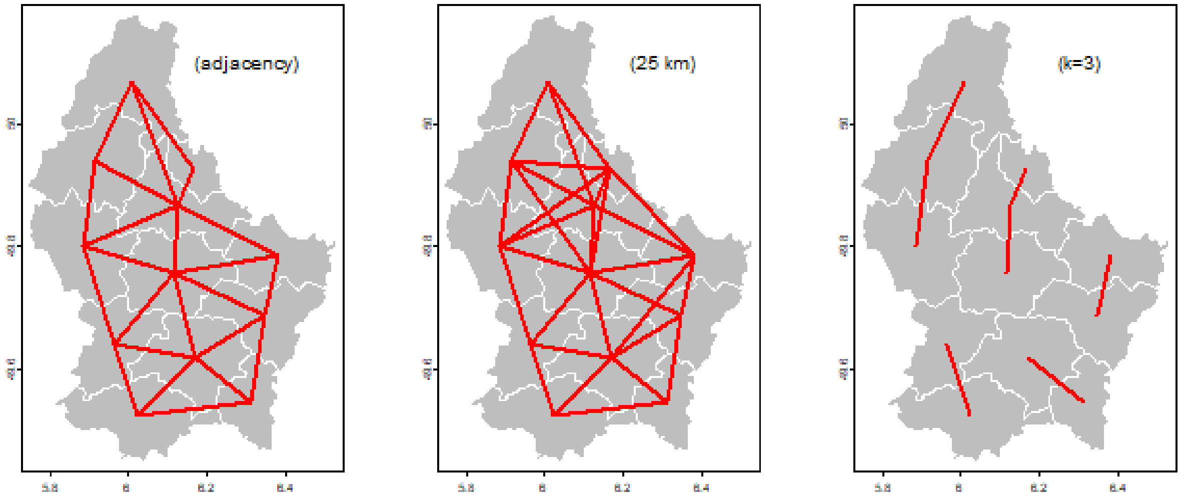
```
wd10 <- nearby(v, distance=10000)
wd25 <- nearby(v, distance=25000)
```

Nearest neighbors:

```
k3 <- nearby(v, k=3)
k6 <- nearby(v, k=6)
```

And now we plot some using the `plotit` function.

```
plotit <- function(nb, lab='') {
  plot(p, col='gray', border='white')
  v <- centroids(p)
  p1 <- v[nb[,1], ,drop=FALSE]
  p2 <- v[nb[,2], ,drop=FALSE]
  lines(p1, p2, col="red", lwd=2)
  text(6.3, 50.1, paste0('(', lab, ')'), cex=1.25)
}

par(mfrow=c(1, 3), mai=c(0,0,0,0))
plotit(nb, "adjacency")
plotit(wd25, "25 km")
plotit(k3, "k=3")
```



2.6 Raster based distance metrics

2.6.1 distance

2.6.2 cost distance

2.6.3 resistance distance

CHAPTER THREE

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

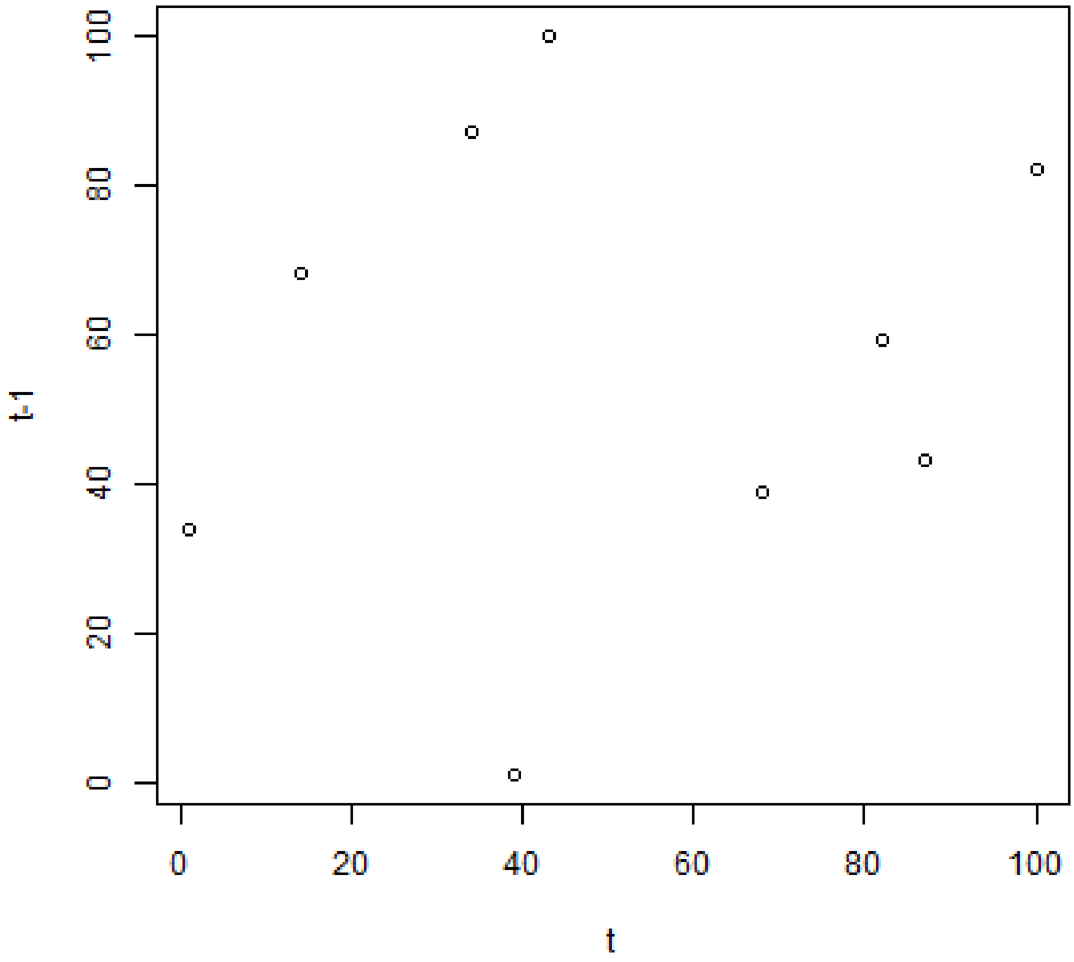
If you measure something about the same object over time, for example a persons weight or wealth, it is likely that two observations that are close to each other in time are also similar in measurement. Say that over a couple of years your weight went from 50 to 80 kg. It is unlikely that it was 60 kg 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.

```
set.seed(0)
d <- sample(100, 10)
d
##  [1]  14  68  39   1  34  87  43 100  82  59
```

Compute auto-correlation.

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

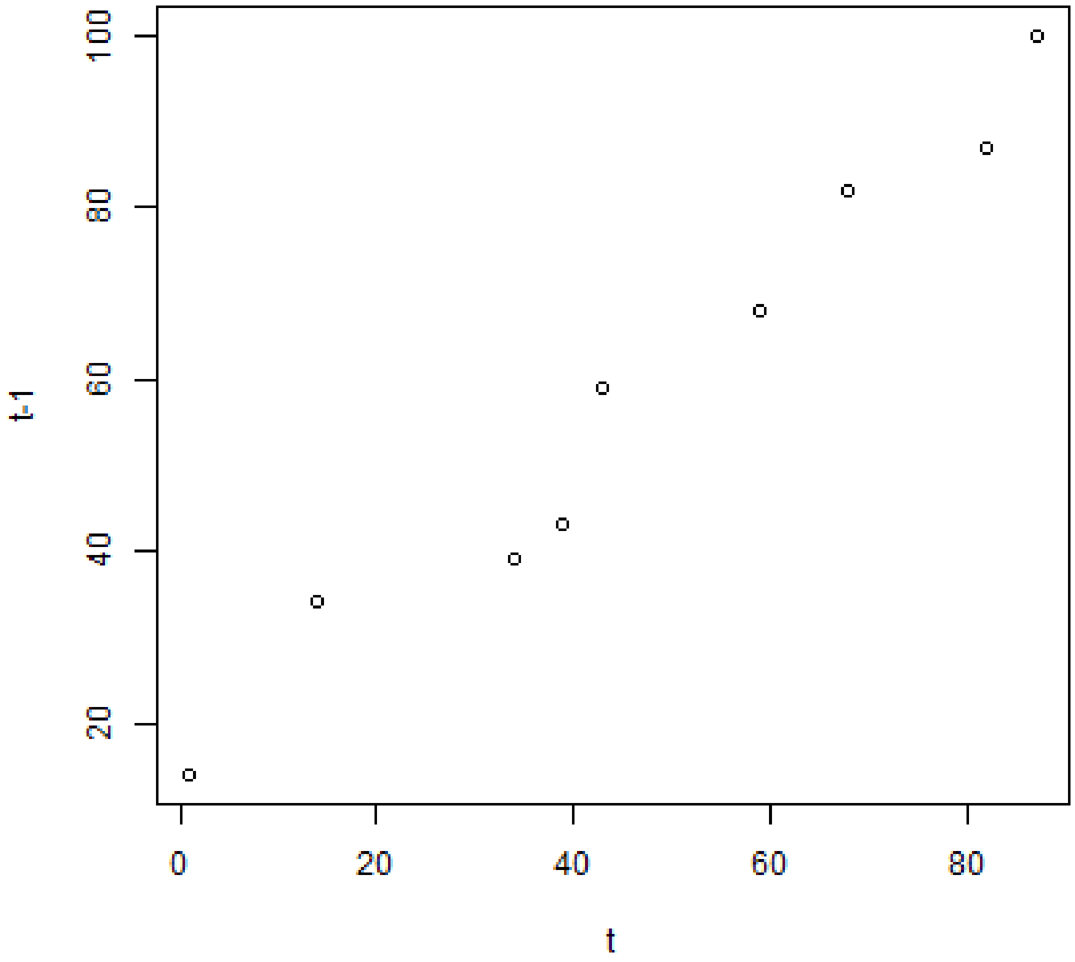


```
cor(a, b)
## [1] 0.1227634
```

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).

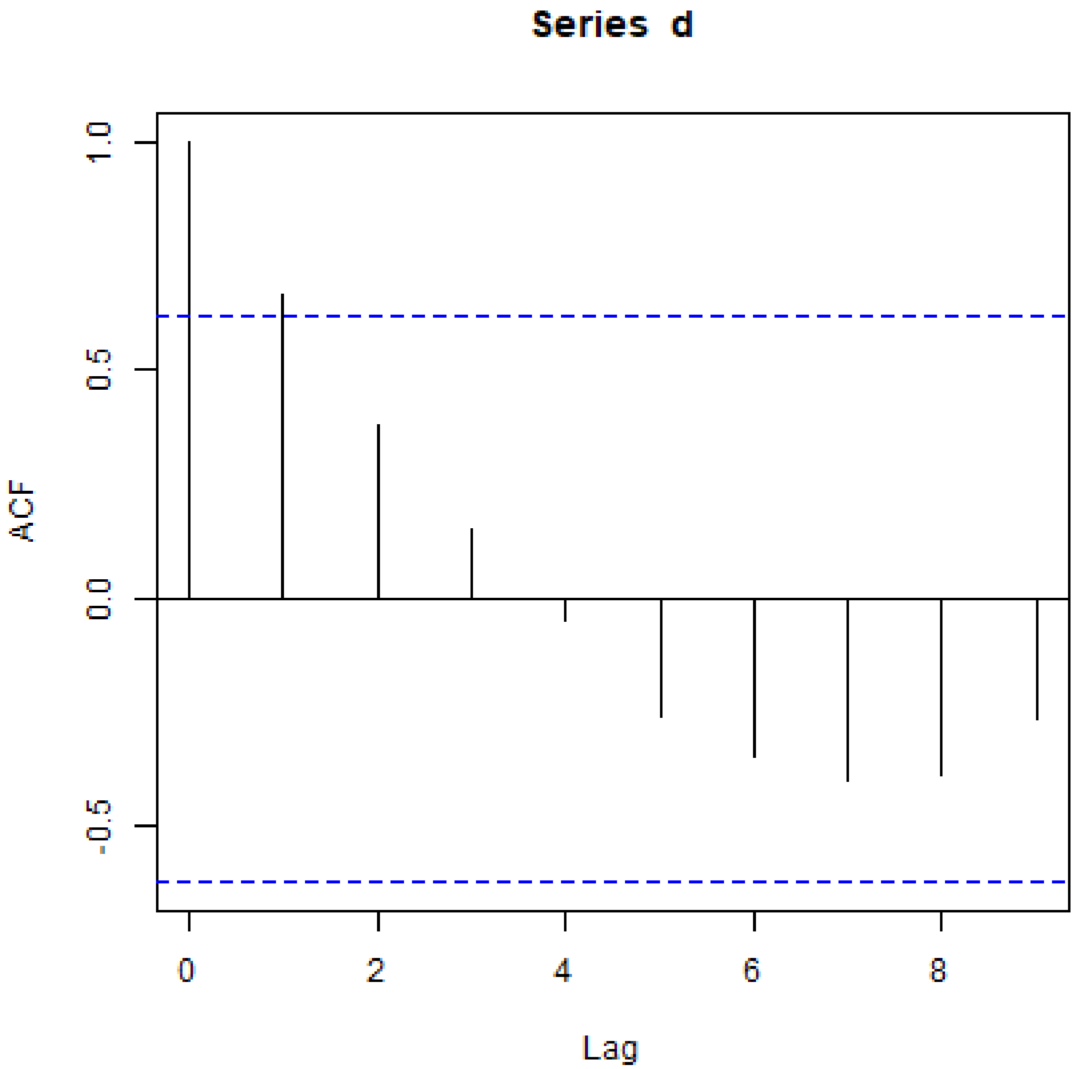
```
d <- sort(d)
d
##  [1]   1  14  34  39  43  59  68  82  87 100
a <- d[-length(d)]
b <- d[-1]
plot(a, b, xlab='t', ylab='t-1')
```



```
cor(a, b)
## [1] 0.9819258
```

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

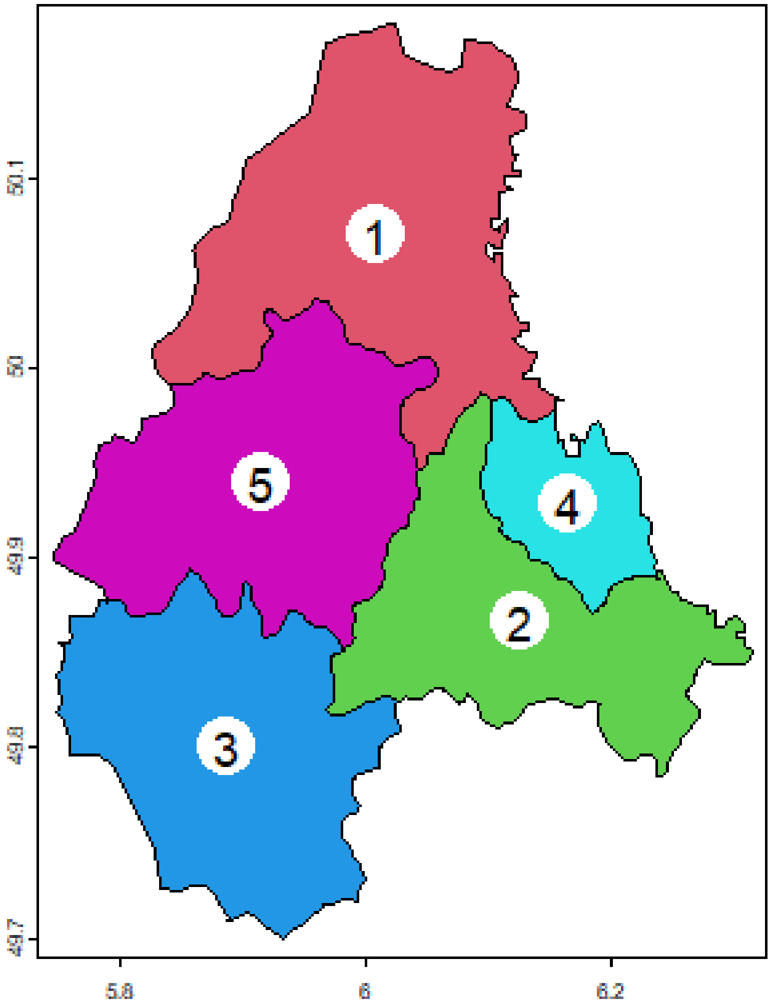
Read the example data

```
library(terra)
p <- vect(system.file("ex/lux.shp", package="terra"))
p <- p[p$NAME_1=="Diekirch", ]
p$value <- c(10, 6, 4, 11, 6)
as.data.frame(p)
##   ID_1   NAME_1 ID_2   NAME_2 AREA   POP value
## 1    1 Diekirch    1 Clervaux  312 18081    10
## 2    1 Diekirch    2 Diekirch  218 32543     6
## 3    1 Diekirch    3  Redange  259 18664     4
## 4    1 Diekirch    4  Vianden   76  5163    11
## 5    1 Diekirch    5    Wiltz  263 16735     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 <- centroids(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'.

```
w <- adjacent(p, symmetrical=TRUE)
class(w)
## [1] "matrix" "array"
head(w)
##      from to
## [1,]    1  2
## [2,]    1  4
## [3,]    1  5
## [4,]    2  3
```

(continues on next page)

(continued from previous page)

```
## [5,]     2   4
## [6,]     2   5
```

`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?).

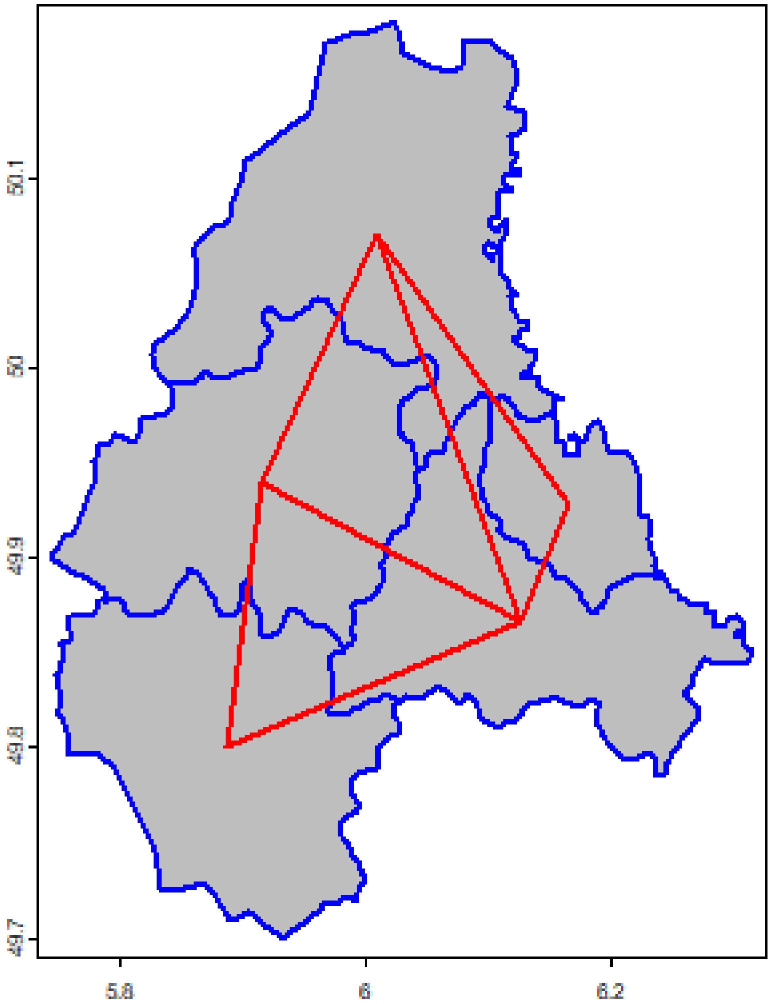
Let's have a look at `w`.

```
w
##       from to
## [1,]     1   2
## [2,]     1   4
## [3,]     1   5
## [4,]     2   3
## [5,]     2   4
## [6,]     2   5
## [7,]     3   5
```

Question 1:*Explain the meaning of the values returned by w*

Plot the links between the polygons.

```
plot(p, col='gray', border='blue', lwd=2)
p1 <- xy[w[,1], ]
p2 <- xy[w[,2], ]
lines(p1, p2, col='red', lwd=2)
```



We can also make a spatial weights matrix, reflecting the intensity of the geographic relationship between observations (see previous chapter).

```
wm <- adjacent(p, pairs=FALSE)
wm
##   1 2 3 4 5
## 1 0 1 0 1 1
## 2 1 0 1 1 1
## 3 0 1 0 0 1
## 4 1 1 0 0 0
## 5 1 1 1 0 0
```

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
## 1  0.00 -3.64 0.00  9.36 -3.64
## 2 -3.64  0.00 4.76 -5.04  1.96
## 3  0.00  4.76 0.00  0.00  4.76
## 4  9.36 -5.04 0.00  0.00  0.00
## 5 -3.64  1.96 4.76  0.00  0.00
```

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 first need to create a spatial weights matrix

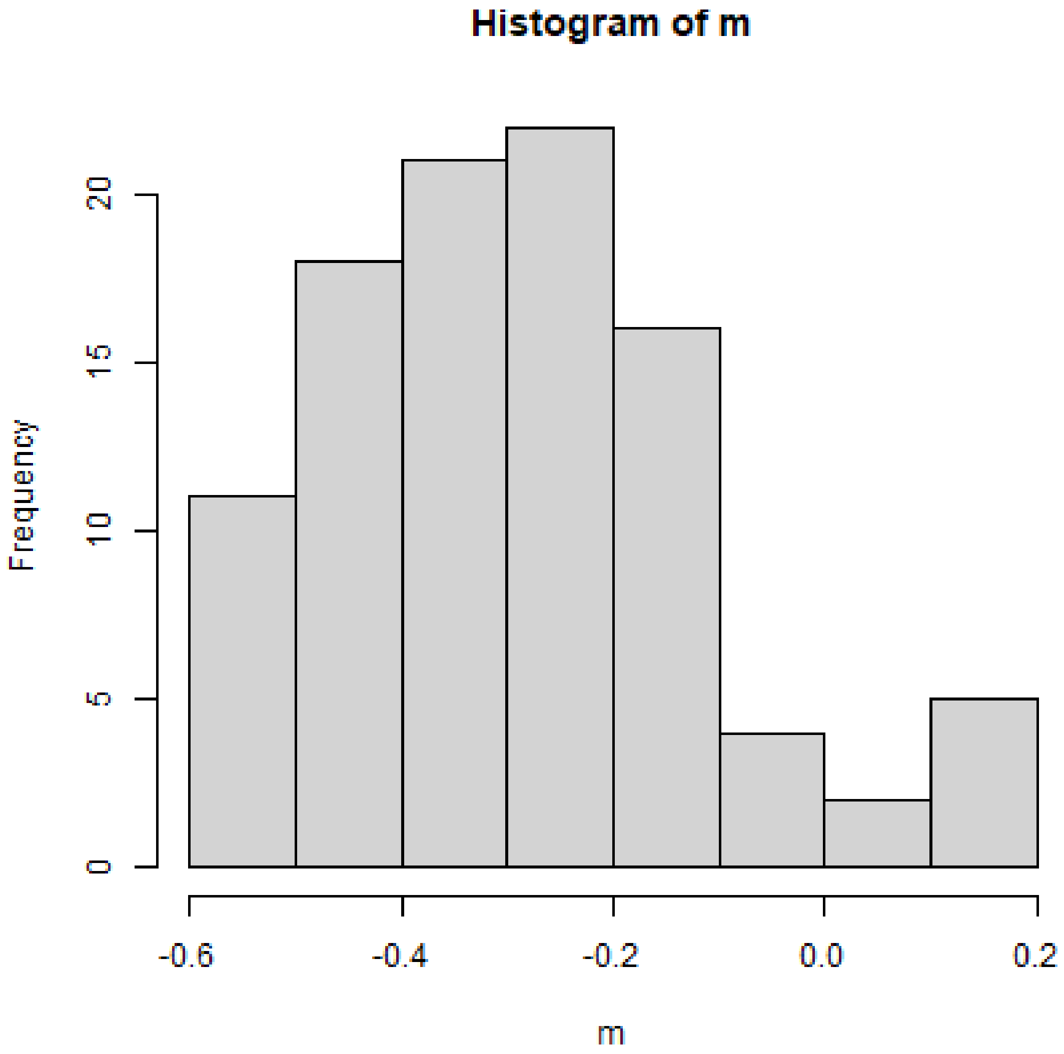
```
ww <-  adjacent(p, "queen", pairs=FALSE)
ww
##   1 2 3 4 5
## 1 0 1 0 1 1
## 2 1 0 1 1 1
## 3 0 1 0 0 1
## 4 1 1 0 0 0
## 5 1 1 1 0 0
```

Now we can use the `autocor` function.

```
ac <- autocor(p$value, ww, "moran")
ac
## [1] 0.1728896
```

We can test for significance using Monte Carlo simulation. That is the preferred method (in fact, the only good method). The way 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.

```
m <- sapply(1:99, function(i) {
    autocor(sample(p$value), ww, "moran")
})
hist(m)
```



```
pval <- sum(m >= ac) / 100
pval
## [1] 0.04
```

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

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

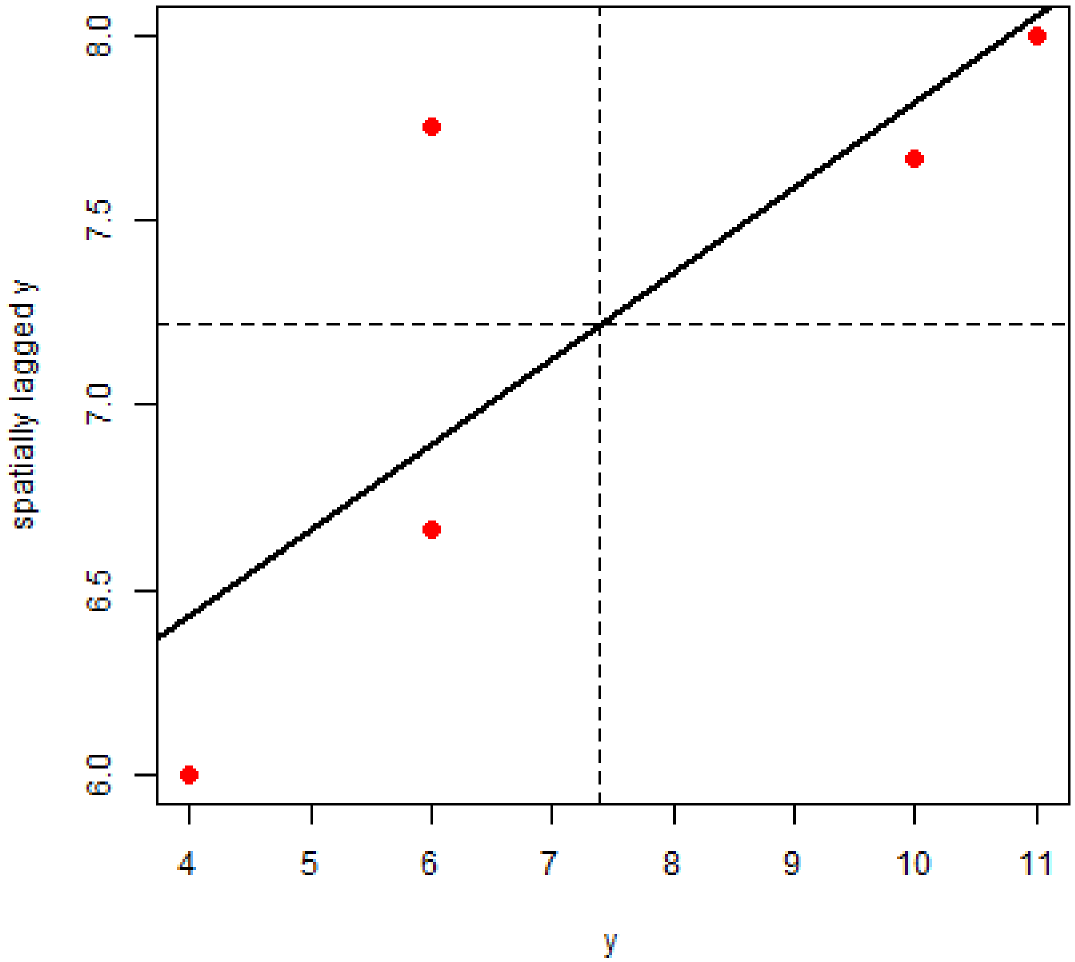
```
ms <- ms[ms[,3] > 0, ]
```

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, pch=20, col="red", cex=2)
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
```

has a similar magnitude as Moran's *I*.

CHAPTER

FOUR

INTERPOLATION

4.1 Introduction

Almost any geographic 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.

4.2 Temperature in California

We will be working with temperature data for California, USA. If have not yet done so, first install the `rspat` package to get the data. You may need to install the `remotes` package first.

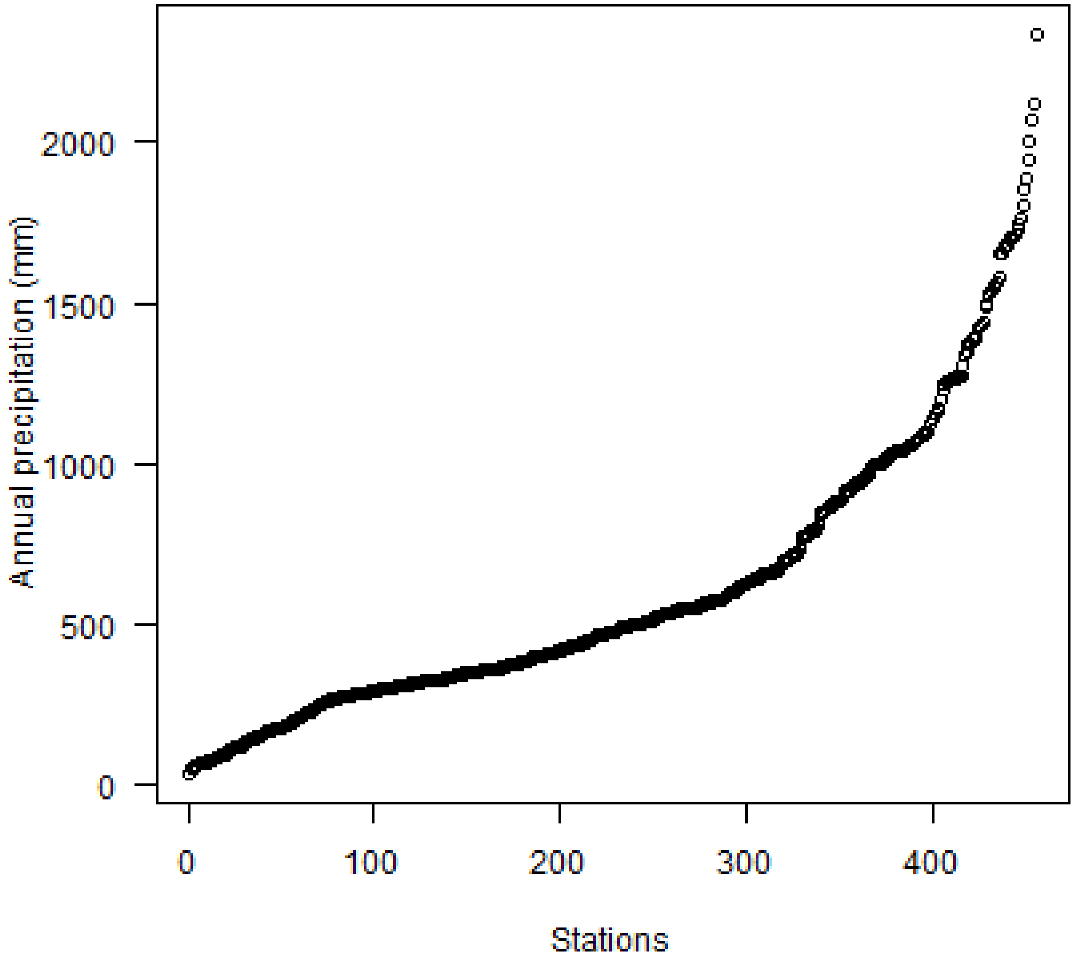
```
if (!require("rspat")) remotes::install_github('rspatial/rspat')
## Loading required package: rspat
## Loading required package: terra
## terra 1.7.62
```

Now get the data:

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

Compute annual precipitation

```
mnts <- toupper(month.abb)
d$prec <- rowSums(d[, mnts])
plot(sort(d$prec), ylab="Annual precipitation (mm)", las=1, xlab="Stations")
```



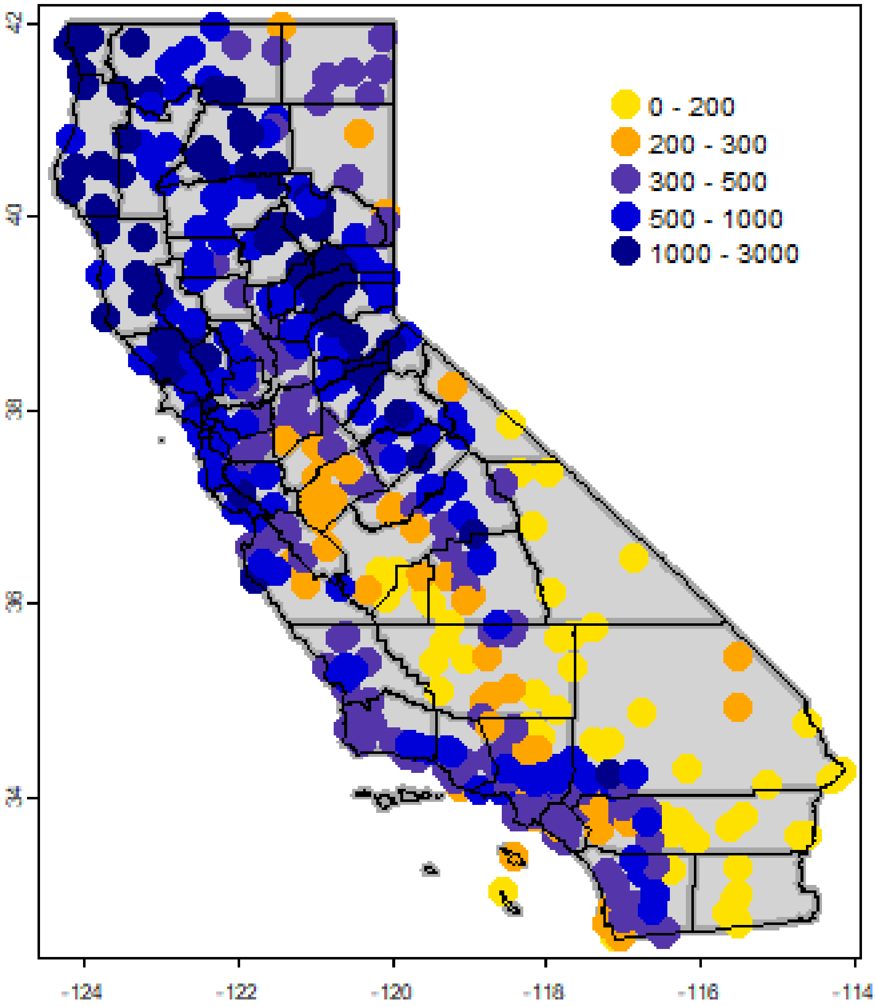
Now make a quick map.

```
dsp <- vect(d, c("LONG", "LAT"), crs="+proj=longlat +datum=NAD83")
CA <- spat_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'))

plot(CA, col="light gray", lwd=4, border="dark gray")
plot(dsp, "prec", type="interval", col=blues(10), legend=TRUE, cex=2,
```

(continues on next page)

(continued from previous page)

```
    breaks=cuts, add=TRUE, plg=list(x=-117.27, y=41.54))
lines(CA)
```



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 <- "+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000 +datum=WGS84 +units=m"
dta <- project(dsp, TA)
cata <- project(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

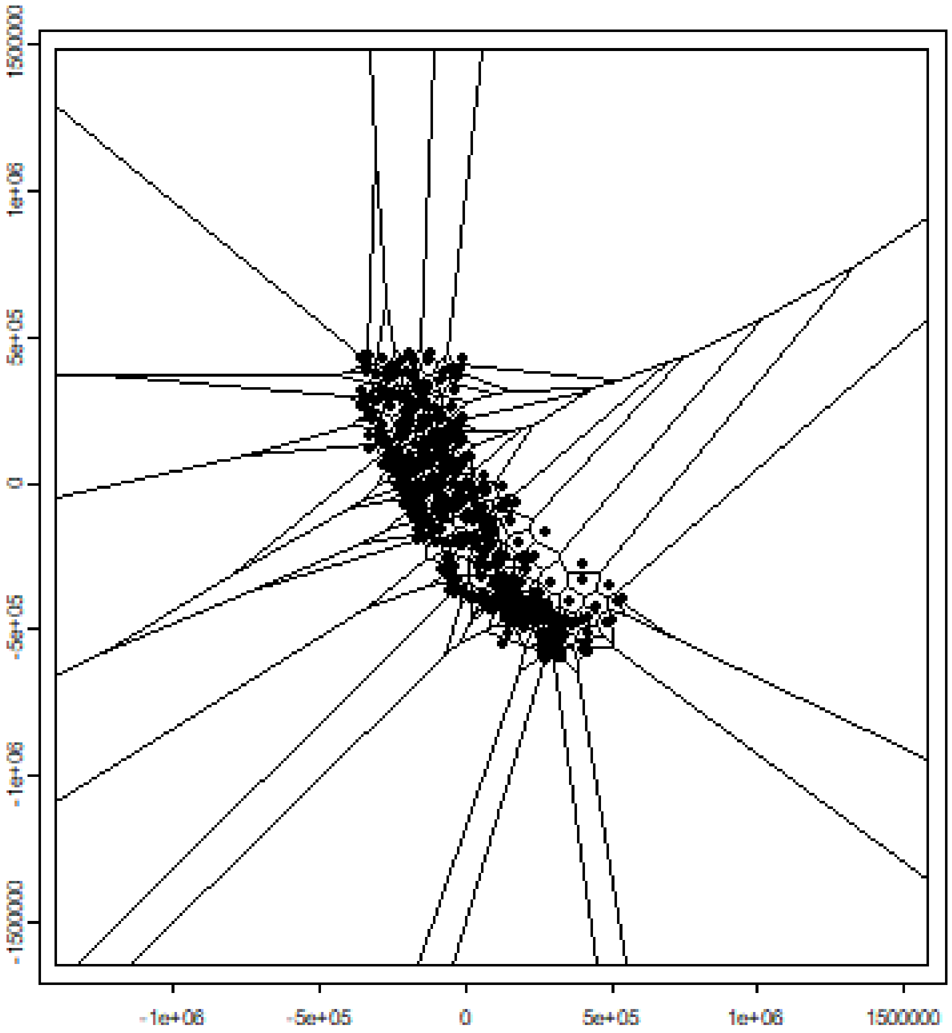
```
null <- RMSE(mean(dsp$prec), dsp$prec)
null
## [1] 435.3217
```

So 435 is our target. Can we do better (have a smaller RMSE)?

4.2.2 proximity polygons

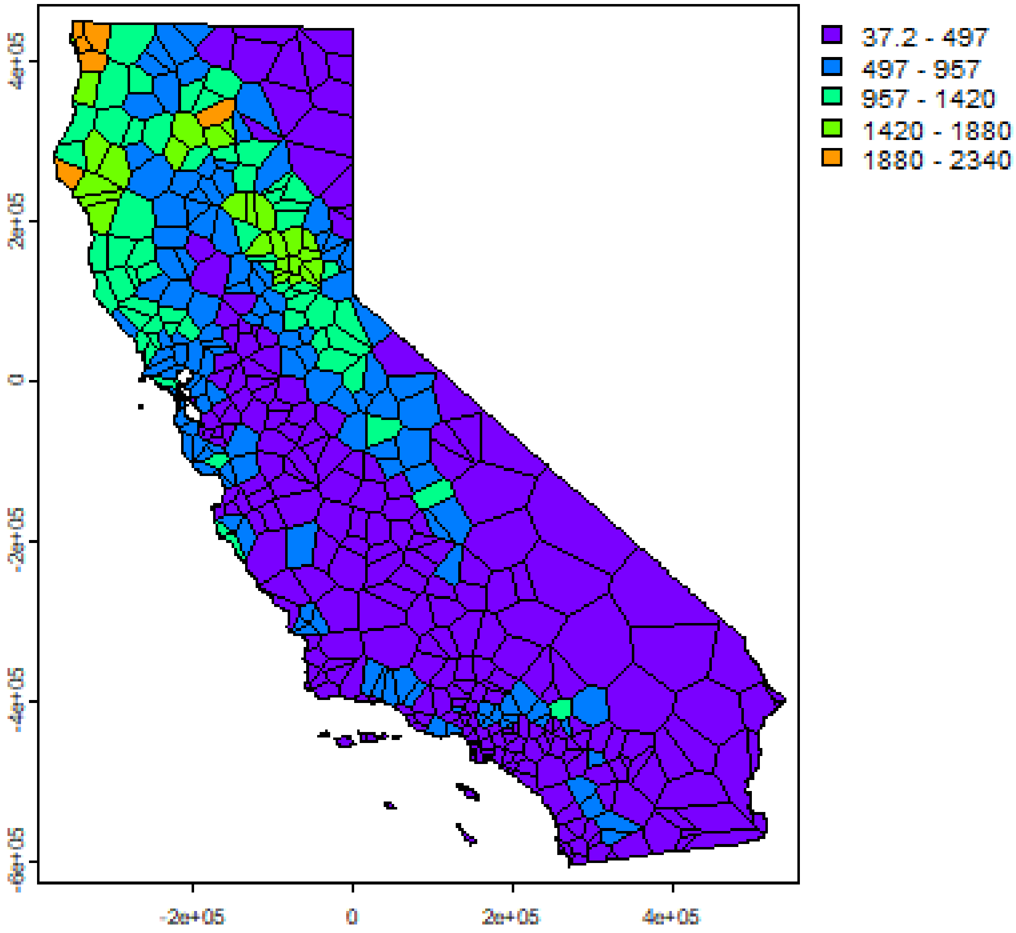
Proximity polygons can be used to interpolate categorical variables. Another term for this is "nearest neighbour" interpolation.

```
v <- voronoi(dta)
plot(v)
points(dta)
```



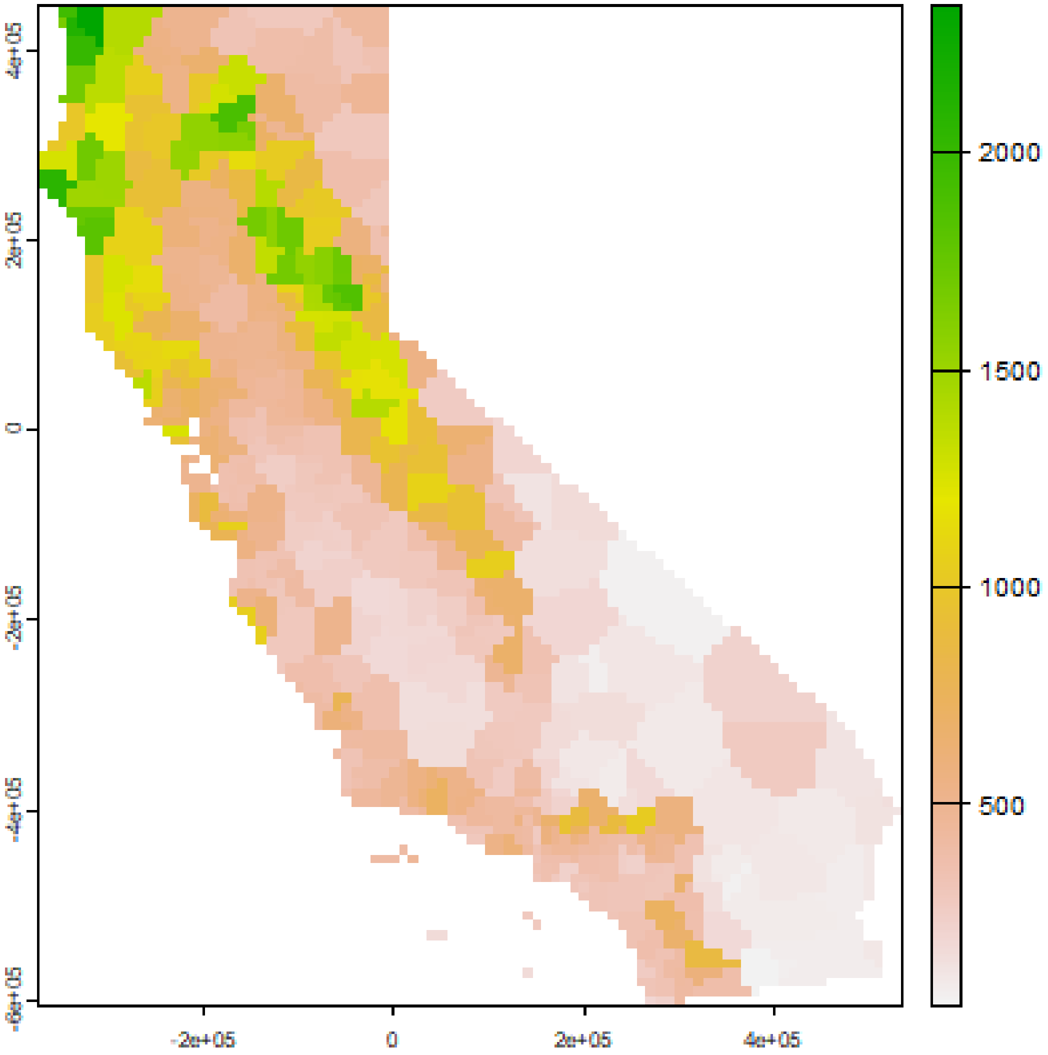
Let’s cut out what is not California, and map precipitation.

```
vca <- crop(v, cata)
plot(vca, "prec")
```



Now we can `rasterize` the results like this.

```
r <- rast(vca, res=10000)
vr <- rasterize(vca, r, "prec")
plot(vr)
```



And use 5-fold cross-validation to evaluate this model.

```
set.seed(5132015)
kf <- sample(1:5, nrow(dta), replace=TRUE)

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] 192.0568 203.1304 183.5556 177.5523 205.6921
mean(rmse)
## [1] 192.3974
```

(continues on next page)

(continued from previous page)

```
# relative model performance
perf <- 1 - (mean(rmse) / null)
round(perf, 3)
## [1] 0.558
```

Question 1: *Describe what each step in the code chunk above does (that is, how does cross-validation work?)*

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

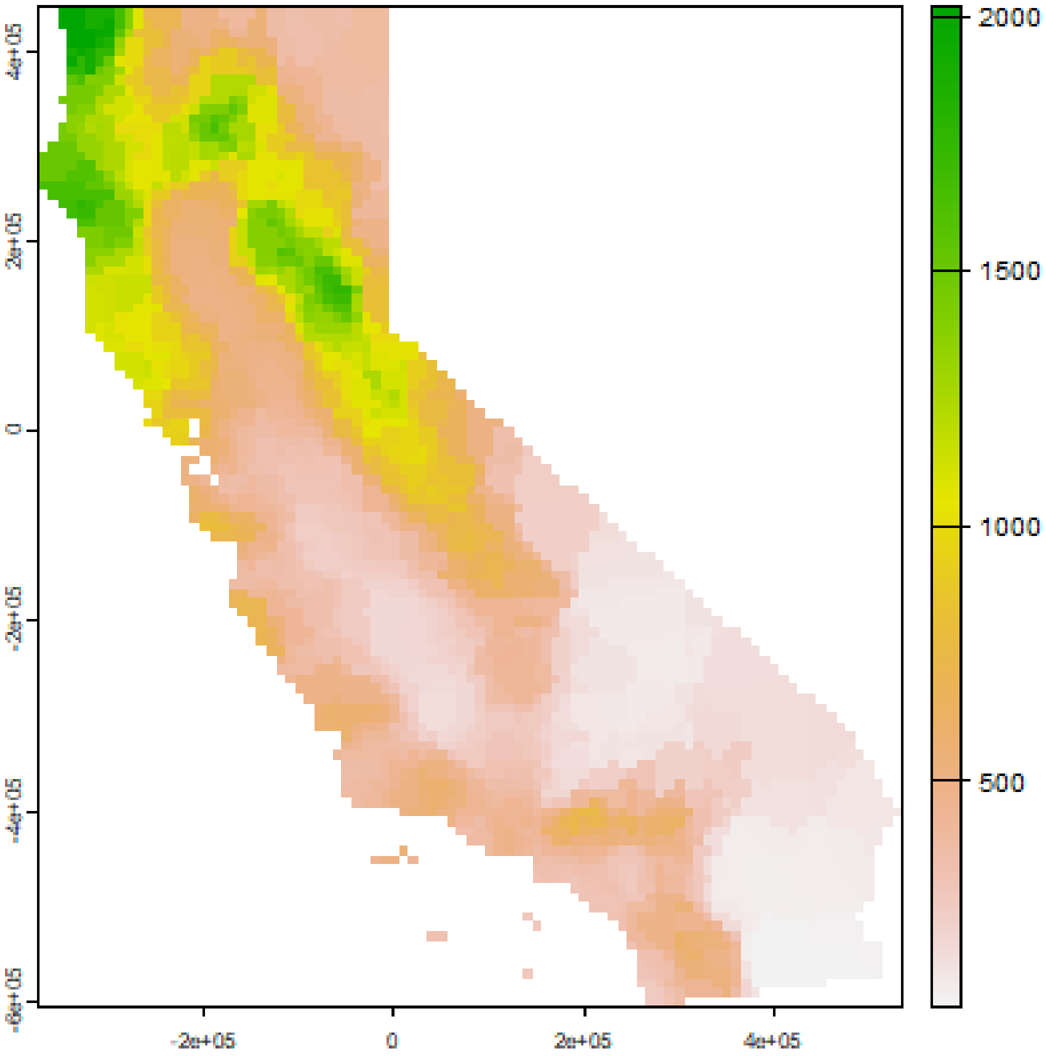
Question 3: *You would not typically use proximty polygons for rainfall data. For what kind of data might you use them?*

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)
d <- data.frame(geom(dta)[,c("x", "y")], as.data.frame(dta))
head(d)
##          x         y    ID                 NAME ALT  JAN FEB MAR APR MAY JUN
## 1 280058.6 -167265.4 ID741         DEATH VALLEY -59  7.4 9.5 7.5 3.4 1.7 1.0
## 2 355394.7 -480020.3 ID743  THERMAL/FAA AIRPORT -34  9.2 6.9 7.9 1.8 1.6 0.4
## 3 416370.9 -551681.2 ID744          BRAWLEY 2SW -31 11.3 8.3 7.6 2.0 0.8 0.1
## 4 415173.4 -566152.9 ID753 IMPERIAL/FAA AIRPORT -18 10.6 7.0 6.1 2.5 0.2 0.0
## 5 418432.1 -516087.7 ID754               NILAND -18  9.0 8.0 9.0 3.0 0.0 1.0
## 6 405858.6 -567692.3 ID758        EL CENTRO/NAF -13  9.8 1.6 3.7 3.0 0.4 0.0
##   JUL  AUG SEP OCT NOV DEC prec
## 1 3.7  2.8 4.3 2.2 4.7 3.9 52.1
## 2 1.9  3.4 5.3 2.0 6.3 5.5 52.2
## 3 1.9  9.2 6.5 5.0 4.8 9.7 67.2
## 4 2.4  2.6 8.3 5.4 7.7 7.3 60.1
## 5 8.0  9.0 7.0 8.0 7.0 9.0 78.0
## 6 3.0 10.8 0.2 0.0 3.3 1.4 37.2
gs <- gstat(formula=prec~1, locations=~x+y, data=d, nmax=5, set=list(idp = 0))
nn <- interpolate(r, gs, debug.level=0)
nnmsk <- mask(nn, vr)
plot(nnmsk, 1)
```



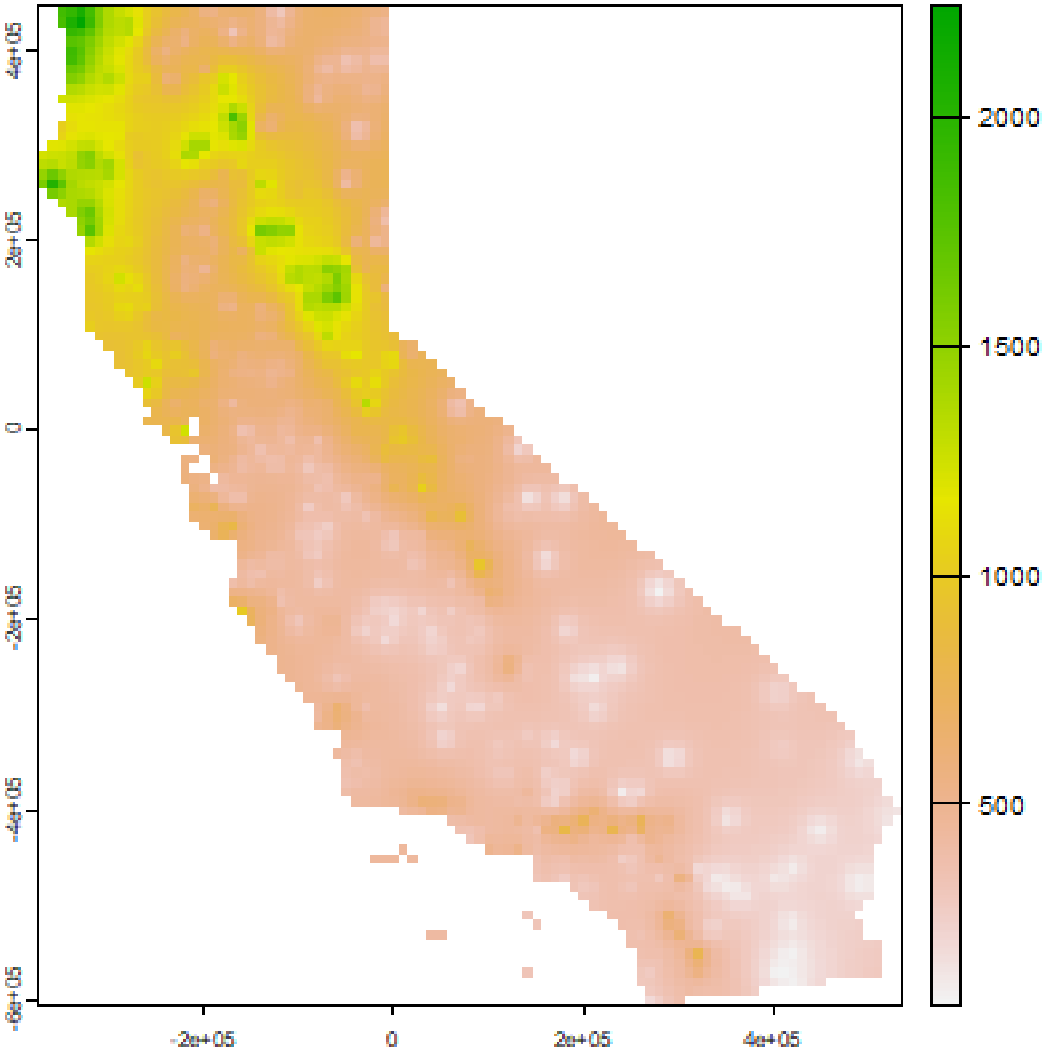
Again we cross-validate the result. Note that we can use the `predict` method to get predictions for the locations of the test points.

```
rmsenn <- rep(NA, 5)
for (k in 1:5) {
  test <- d[kf == k, ]
  train <- d[kf != k, ]
  gscv <- gstat(formula=prec~1, locations=~x+y, data=train, nmax=5, set=list(idp = 0))
  p <- predict(gscv, test, debug.level=0)$var1.pred
  rmsenn[k] <- RMSE(test$prec, p)
}
rmsenn
## [1] 215.0993 209.5838 197.0604 177.1946 189.8130
mean(rmsenn)
## [1] 197.7502
1 - (mean(rmsenn) / null)
## [1] 0.5457377
```

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=~x+y, data=d)
idw <- interpolate(r, gs, debug.level=0)
idwr <- mask(idw, vr)
plot(idwr, 1)
```



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

Cross-validate again. We can use `predict` for the locations of the test points

```
rmse <- rep(NA, 5)
for (k in 1:5) {
```

(continues on next page)

(continued from previous page)

```
  test <- d[kf == k, ]
  train <- d[kf != k, ]
  gs <- gstat(formula=prec~1, locations=~x+y, data=train)
  p <- predict(gs, test, debug.level=0)
  rmse[k] <- RMSE(test$prec, p$var1.pred)
}
rmse
## [1] 243.3256 212.6271 206.8982 180.1828 207.5790
mean(rmse)
## [1] 210.1225
1 - (mean(rmse) / null)
## [1] 0.5173166
```

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? Illustrate with a map*

```
gs2 <- gstat(formula=prec~1, locations=~x+y, data=d, 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 <- rspat::spat_data("airqual")
x$OZDLYAV <- x$OZDLYAV * 1000
x <- vect(x, c("LONGITUDE", "LATITUDE"), crs="+proj=longlat +datum=WGS84")
```

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

```
TAkm <- "+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000␣
↪+datum=WGS84 +units=km"
aq <- project(x, TAkm)
```

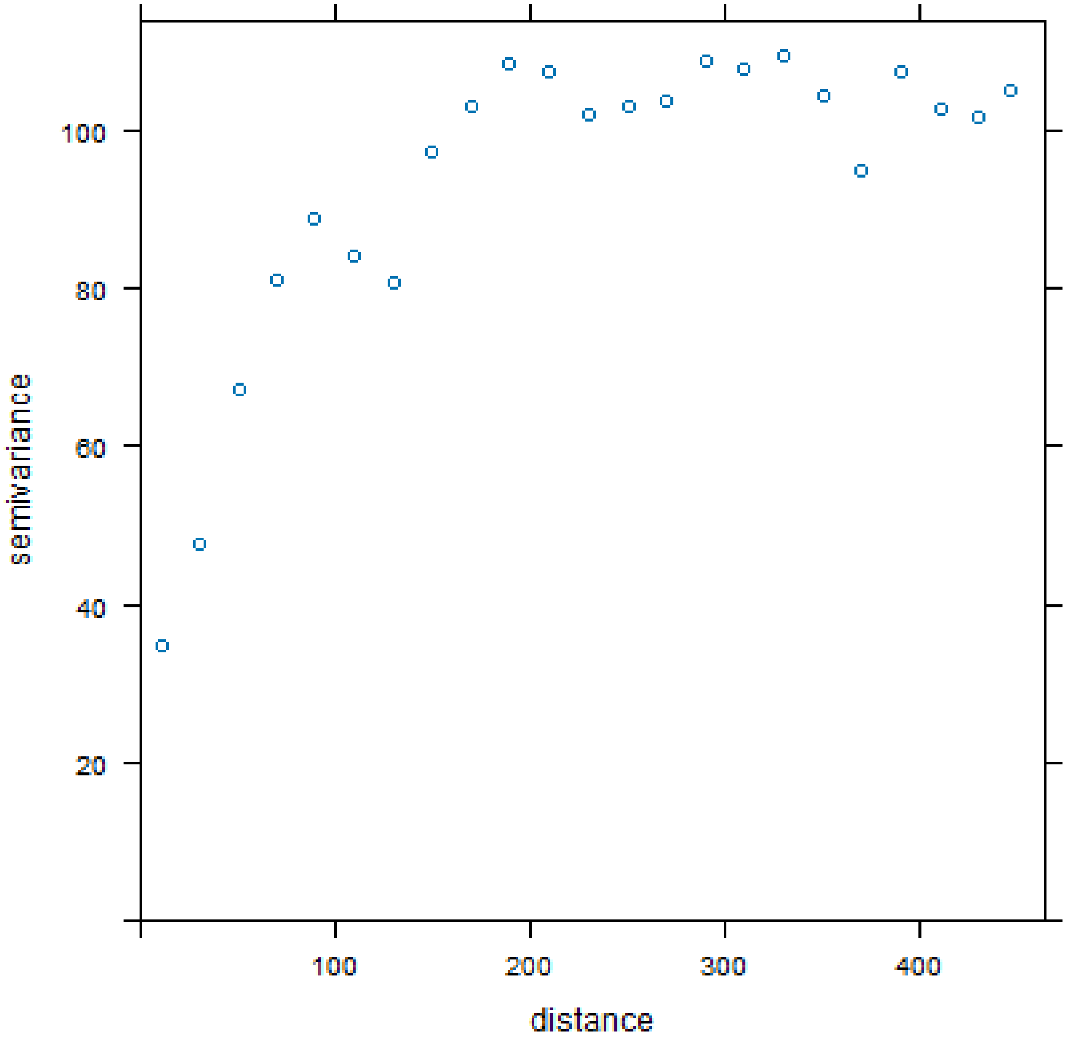
Create an template SpatRaster to interpolate to.

```
ca <- project(CA, TAkm)
r <- rast(ca)
res(r) <- 10  # 10 km if your CRS's units are in km
```

4.3.2 Fit a variogram

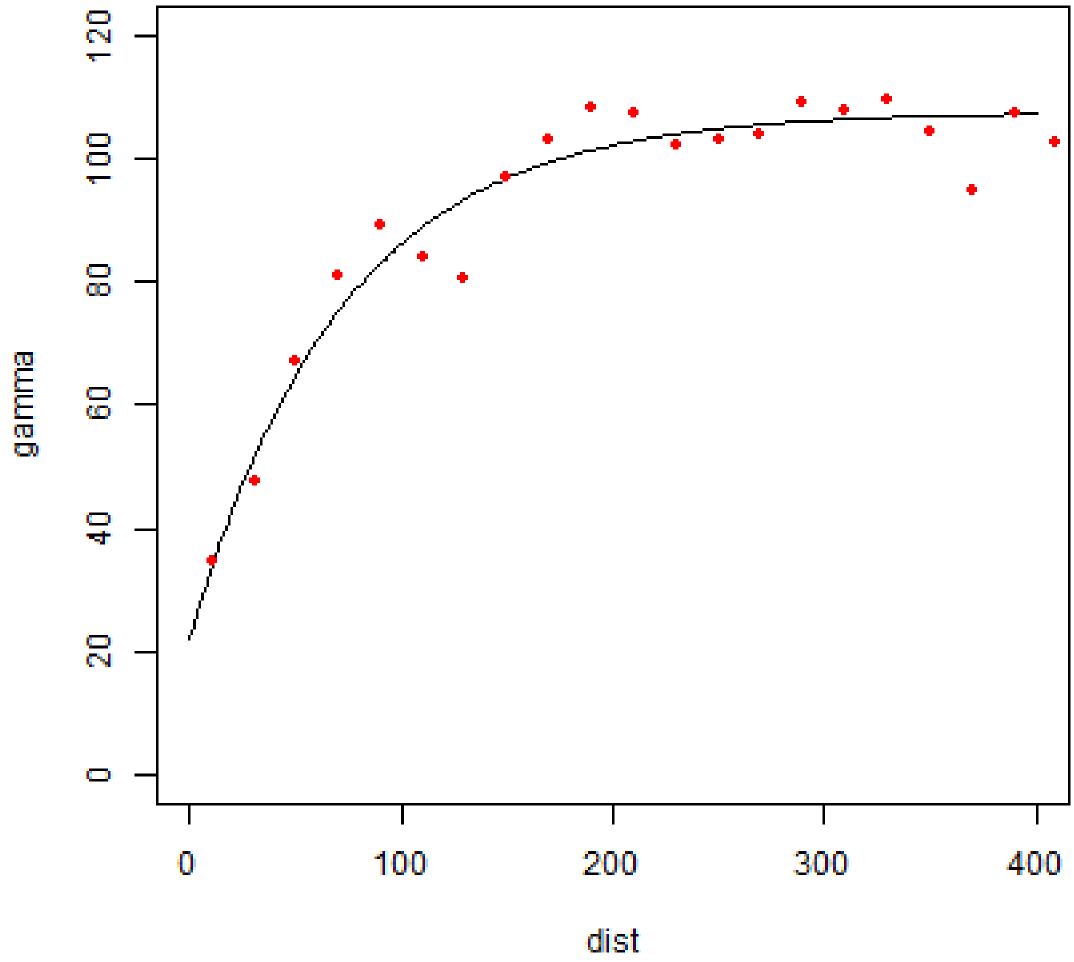
Use gstat to create an emperical variogram 'v'

```
p <- data.frame(geom(aq)[, c("x", "y")], as.data.frame(aq))
gs <- gstat(formula=OZDLYAV~1, locations=~x+y, data=p)
v <- variogram(gs, width=20)
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
## 6  3437 110.42302  84.13589       0       0 var1
## 7  3581 130.07080  80.59402       0       0 var1
## 8  3808 149.75625  97.06451       0       0 var1
## 9  3589 170.13526 102.97593       0       0 var1
## 10 3569 189.70054 108.28135       0       0 var1
## 11 3489 210.01413 107.48915       0       0 var1
## 12 3583 230.17040 101.95520       0       0 var1
## 13 3529 250.22845 103.06846       0       0 var1
## 14 3394 269.58370 103.63122       0       0 var1
## 15 3267 290.04602 108.81122       0       0 var1
## 16 3046 309.73363 107.58961       0       0 var1
## 17 2824 329.92996 109.52365       0       0 var1
## 18 2860 349.91455 104.27218       0       0 var1
## 19 2641 369.71992  94.76248       0       0 var1
## 20 2430 389.97879 107.47451       0       0 var1
## 21 2570 409.87266 102.55504       0       0 var1
## 22 2385 429.90866 101.55894       0       0 var1
## 23 1584 446.54929 105.00524       0       0 var1
plot(v)
```



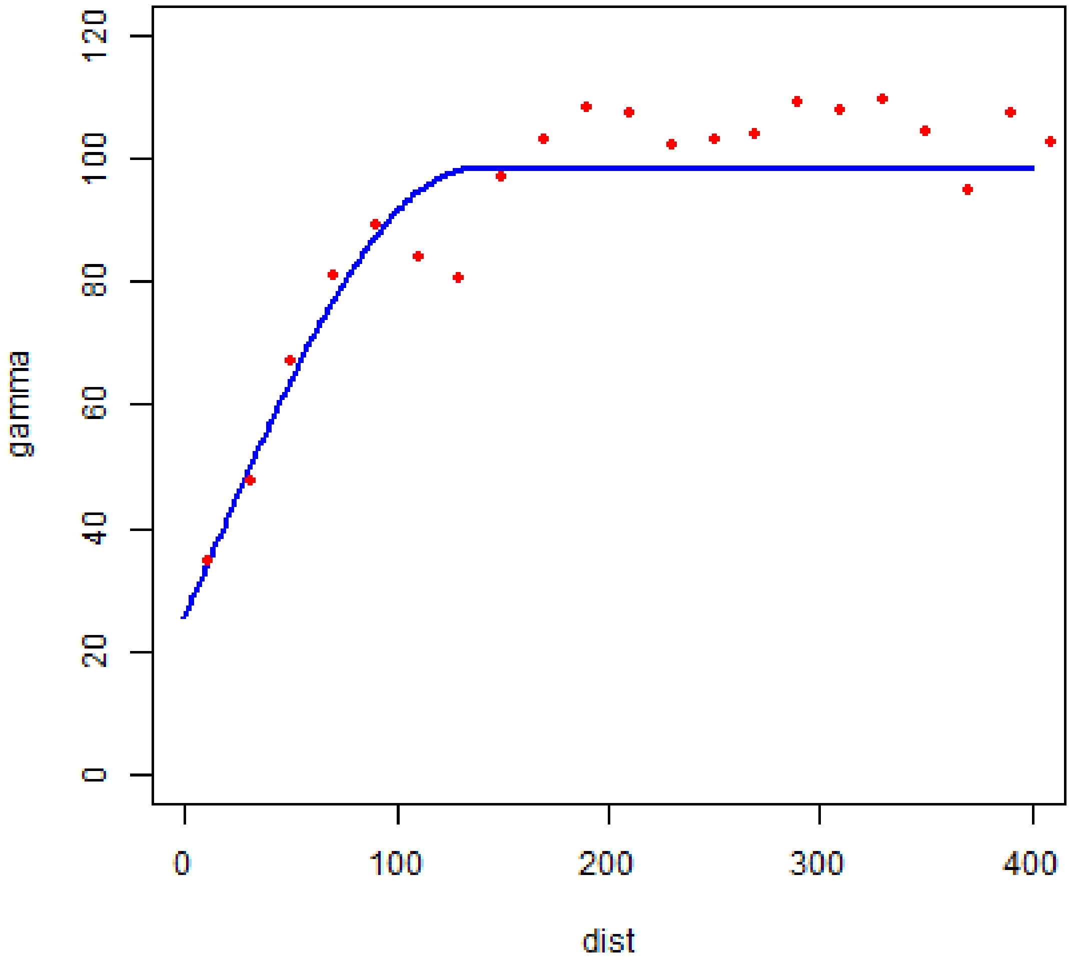
Now, fit a model variogram

```
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)

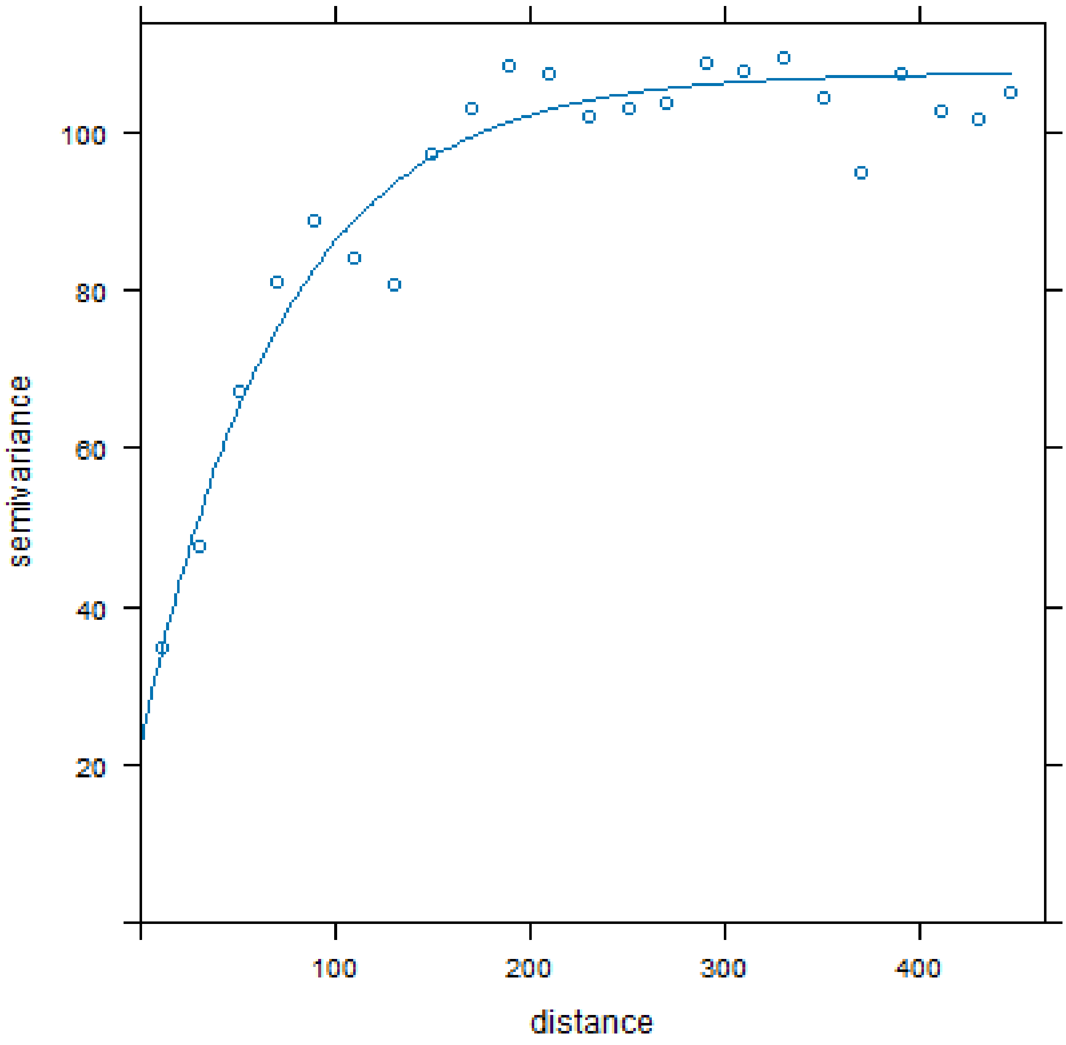
```
fvs <- fit.variogram(v, vgm(85, "Sph", 75, 20))
fvs
##   model    psill    range
## 1   Nug 25.57019   0.0000
## 2   Sph 72.65881 135.7744
plot(variogramLine(fvs, 400), type='l', ylim=c(0,120) ,col='blue', lwd=2)
points(v[,2:3], pch=20, col='red')
```



Both look pretty good in this case.

Another way to plot the variogram and the model

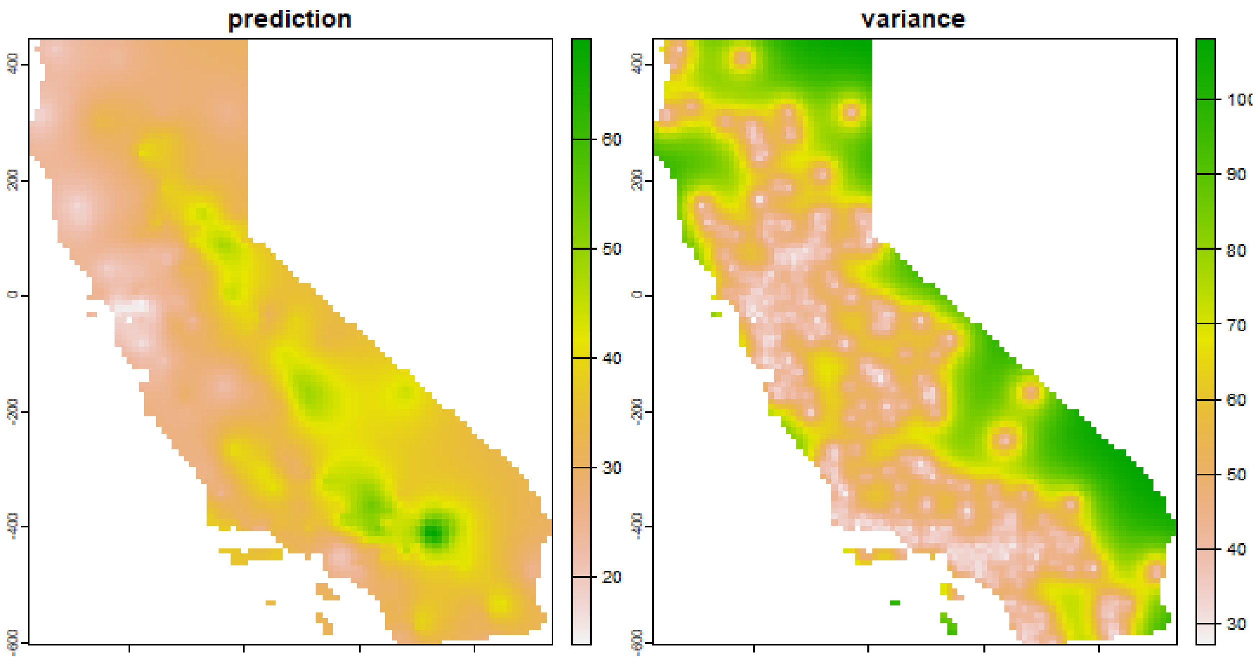
```
plot(v, fve)
```



4.3.3 Ordinary kriging

Use variogram fve in a kriging interpolation

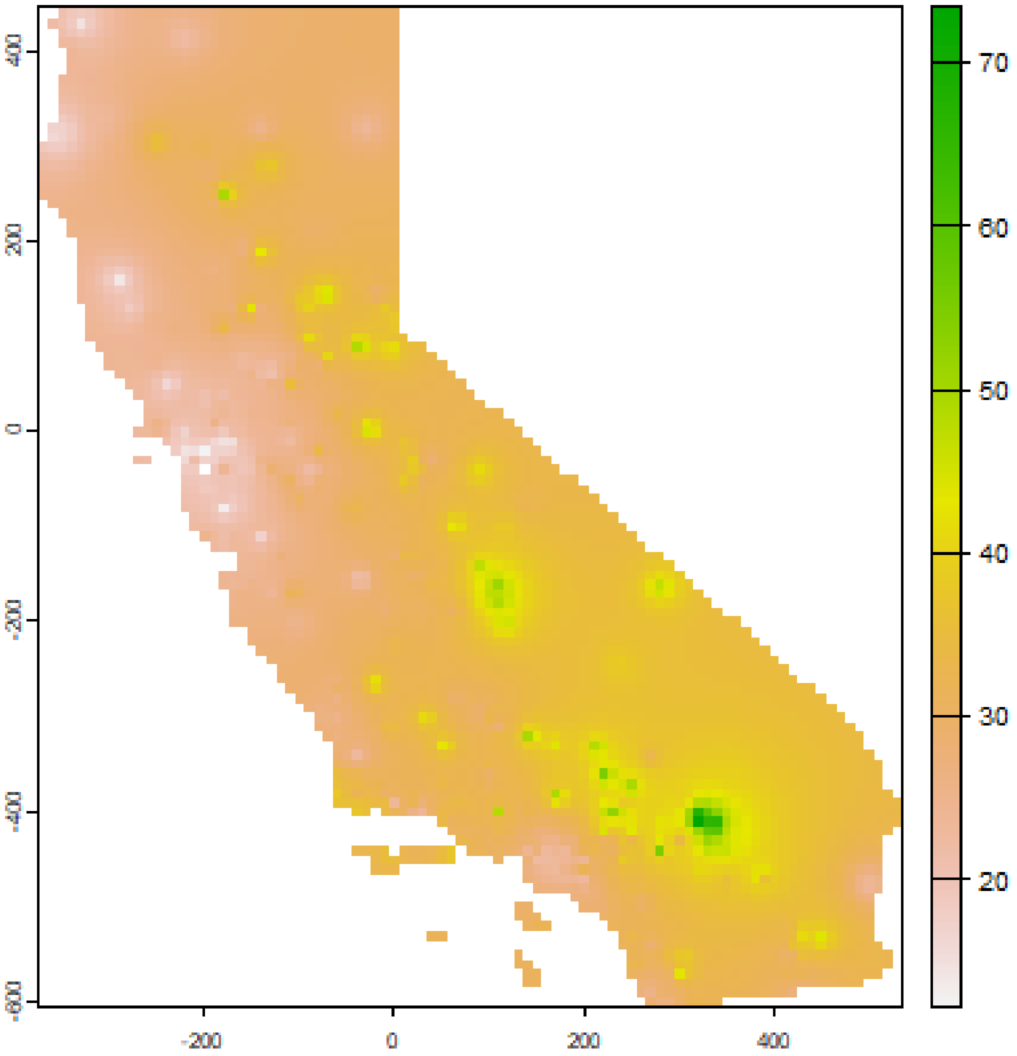
```
k <- gstat(formula=OZDLYAV~1, locations=~x+y, data=p, model=fve)
# predicted values
kp <- interpolate(r, k, debug.level=0)
ok <- mask(kp, ca)
names(ok) <- c('prediction', 'variance')
plot(ok)
```



4.3.4 Compare with other methods

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

```
idm <- gstat(formula=OZDLYAV~1, locations=~x+y, data=p)
idp <- interpolate(r, idm, debug.level=0)
idp <- mask(idp, ca)
plot(idp, 1)
```



We can find good values for the idw parameters (distance decay and number of neighbours) through optimization. For simplicity's sake I only do that once here, not *k* times. 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. You also need to provide initial values for these parameters. `optim` then searches for the optimal values (for which the function returns the lowest number).

```
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=~x+y, data=train, nmax=nmx, set=list(idp=idp))
  p <- predict(m, newdata=test, debug.level=0)$var1.pred
  RMSE(test$OZDLYAV, p)
}
set.seed(20150518)
i <- sample(nrow(aq), 0.2 * nrow(aq))
```

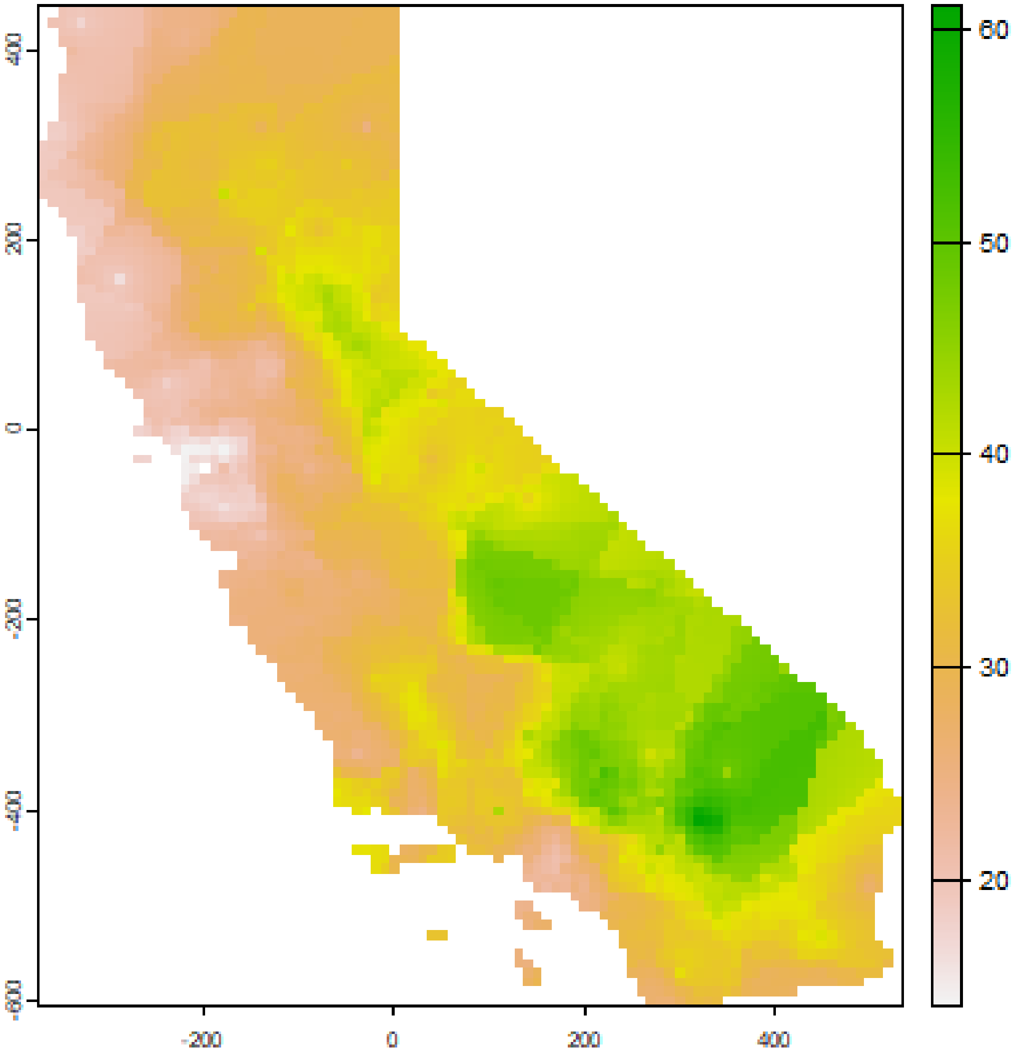
(continues on next page)

(continued from previous page)

```
tst <- p[i,]
trn <- p[-i,]
opt <- optim(c(8, .5), f1, test=tst, train=trn)
str(opt)
## List of 5
##  $ par        : num [1:2] 9.259 0.682
##  $ value      : num 7.86
##  $ counts     : Named int [1:2] 35 NA
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

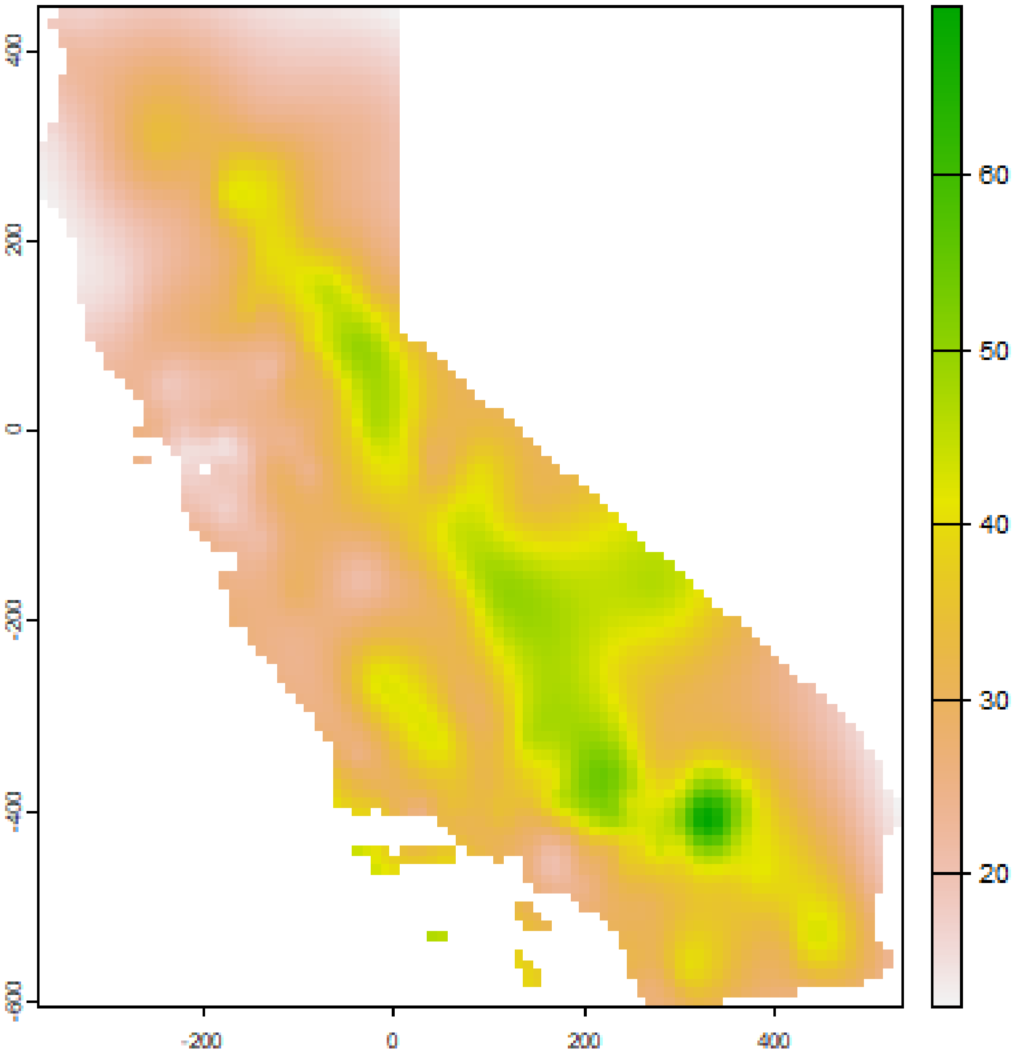
Our optimal IDW model

```
m <- gstat(formula=OZDLYAV~1, locations=~x+y, data=p, nmax=opt$par[1], set=list(idp=opt
↪$par[2]))
idw <- interpolate(r, m, debug.level=0)
idw <- mask(idw, ca)
plot(idw, 1)
```



And now, for something completely different, a thin plate spline model:

```
library(fields)
m <- fields::Tps(p[,c("x", "y")], p$OZDLYAV)
tps <- interpolate(r, m)
tps <- mask(tps, idw[[1]])
plot(tps)
```



4.3.5 Cross-validation

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

```
k <- sample(5, nrow(p), replace=TRUE)

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

for (i in 1:5) {
  test <- p[k!=i,]
  train <- p[k==i,]
  m <- gstat(formula=OZDLYAV~1, locations=~x+y, data=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)
```

(continues on next page)

(continued from previous page)

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

  m <- Tps(train[,c("x", "y")], train$OZDLYAV)
  p3 <- predict(m, test[,c("x", "y")])
  tpsrmse[i] <-  RMSE(test$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)

}
## Warning:
## Grid searches over lambda (nugget and sill variances) with  minima at the endpoints:
##   (GCV) Generalized Cross-Validation
##    minimum at  right endpoint  lambda  =  1.582376e-07 (eff. df= 89.30001 )
rmi <- mean(idwrmse)
rmk <- mean(krigrmse)
rmt <- mean(tpsrmse)
rms <- c(rmi, rmt, rmk)
rms
## [1] 8.011006 9.120307 7.736301
rme <- mean(ensrmse)
rme
## [1] 7.936466
```

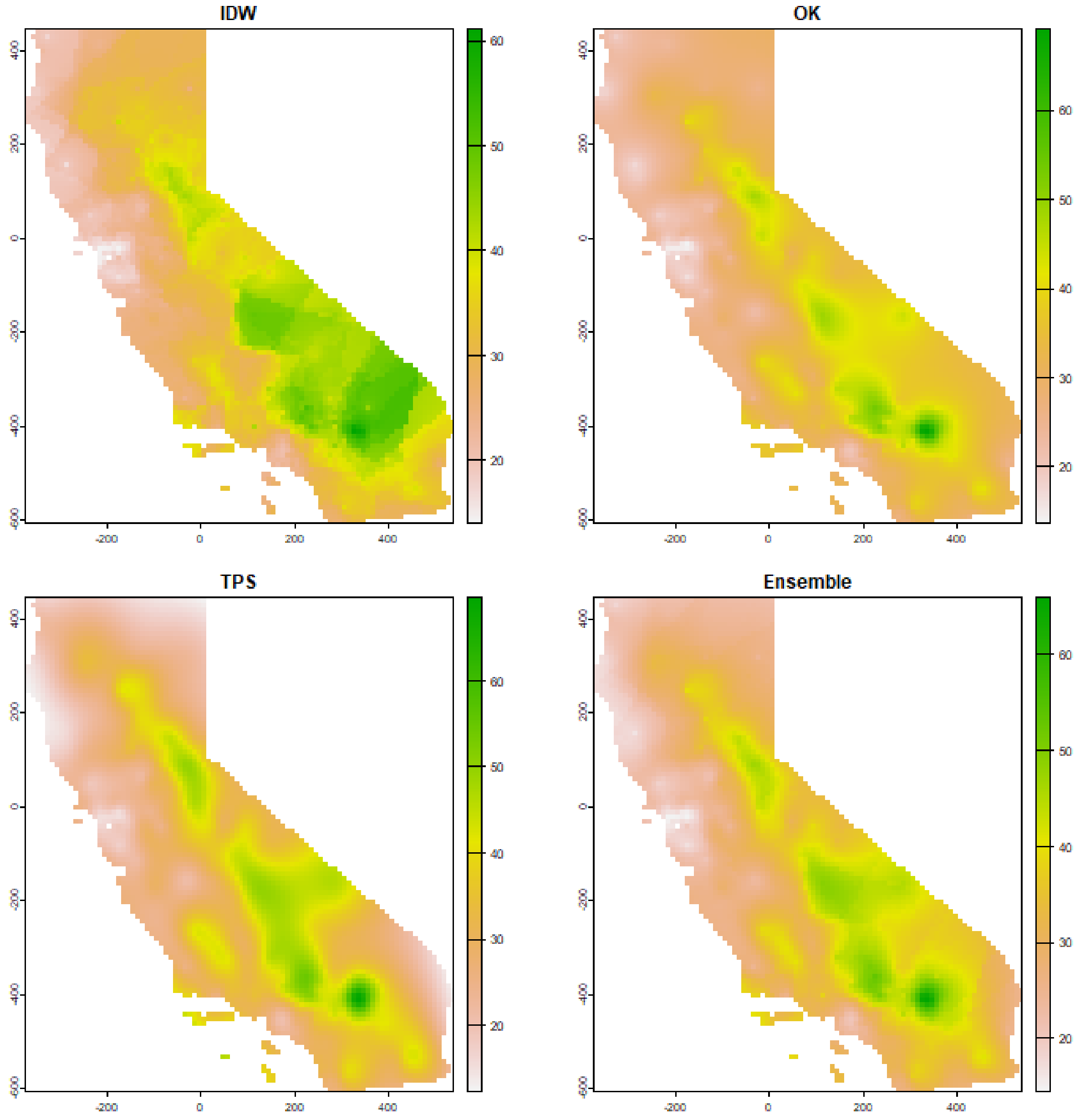
Question 6: *Which method performed best?*

We can use the RMSE values to make a weighted ensemble. I use the normalized difference between a model's RMSE and the NULL model as weights.

```
nullrmse <- RMSE(test$OZDLYAV, mean(test$OZDLYAV))
w <- nullrmse - rms
# normalize weights to sum to 1
weights <- ( w / sum(w) )
# check
sum(weights)
## [1] 1
s <- c(idw[[1]], ok[[1]], tps)
ensemble <- sum(s * weights)
```

And compare maps.

```
s <- c(idw[[1]], 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 the 95% confidence interval of the OK prediction.*

CHAPTER FIVE

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 believed to occur in the United States, but it is so hard to find by scientists that its very existence is commonly denied by the mainstream media — despite the many reports on Twitter! 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 will use “citizen-science” data to find out:

a) What the complete range of the species might be.

b) How good (general) our model is by predicting the range of the Eastern sub-species, with data from the Western sub-species.

c) 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.

First make sure we have the packages needed:

```
if (!require("rspat")) remotes::install_github("rspatial/rspat")
## Loading required package: rspat
## Loading required package: terra
## terra 1.7.62
if (!require("predicts")) install.packages("predicts")
## Loading required package: predicts
if (!require("geodata")) install.packages("geodata")
## Loading required package: geodata
```

5.1 Data

5.1.1 Observations

We get a data set of reported Bigfoot observations

```
library(terra)
library(rspat)
bf <- spat_data("bigfoot")
```

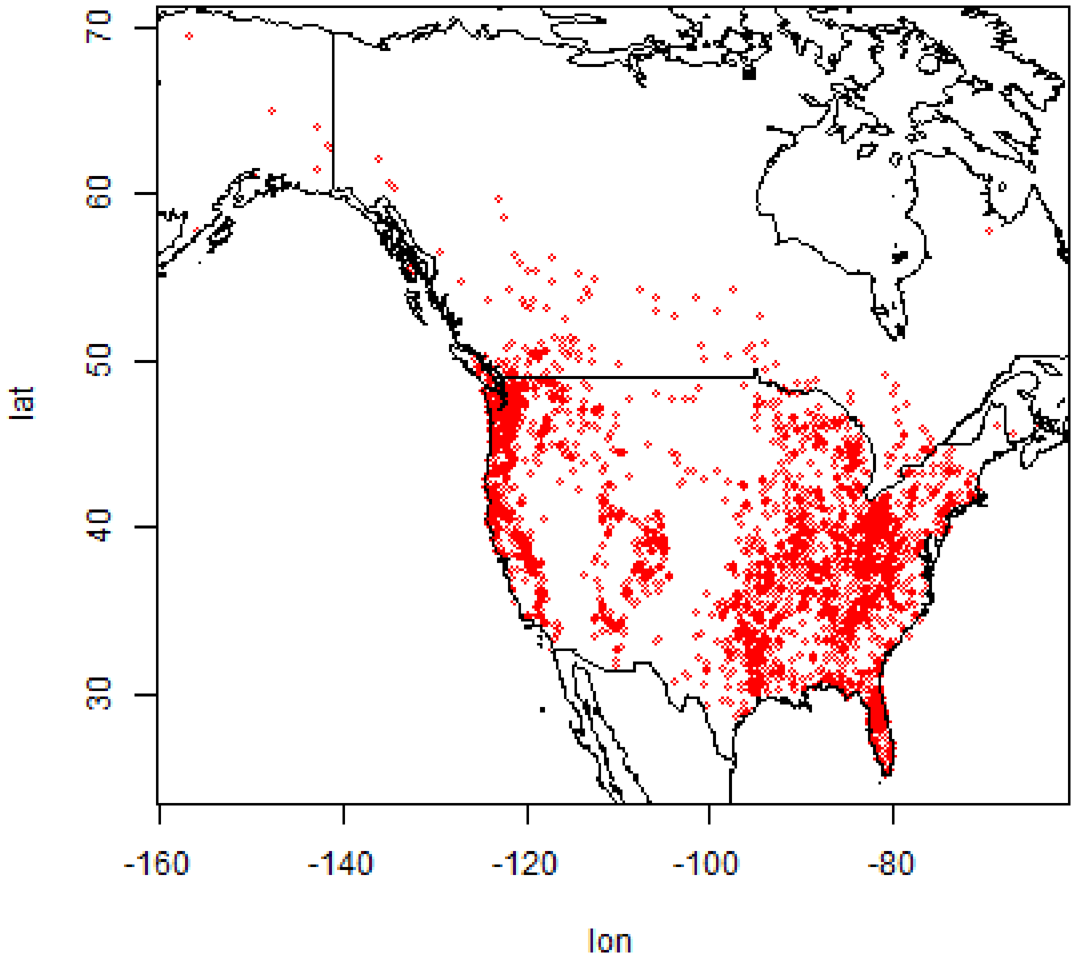
(continues on next page)

(continued from previous page)

```
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
```

It is always good to first plot the locations to see what we are dealing with.

```
plot(bf[,1:2], cex=0.5, col="red")

library(geodata)
wrld <- geodata::world(path=".")
bnds <- wrld[wrld$NAME_0 %in% c("Canada", "Mexico", "United States"), ]
lines(bnds)
```

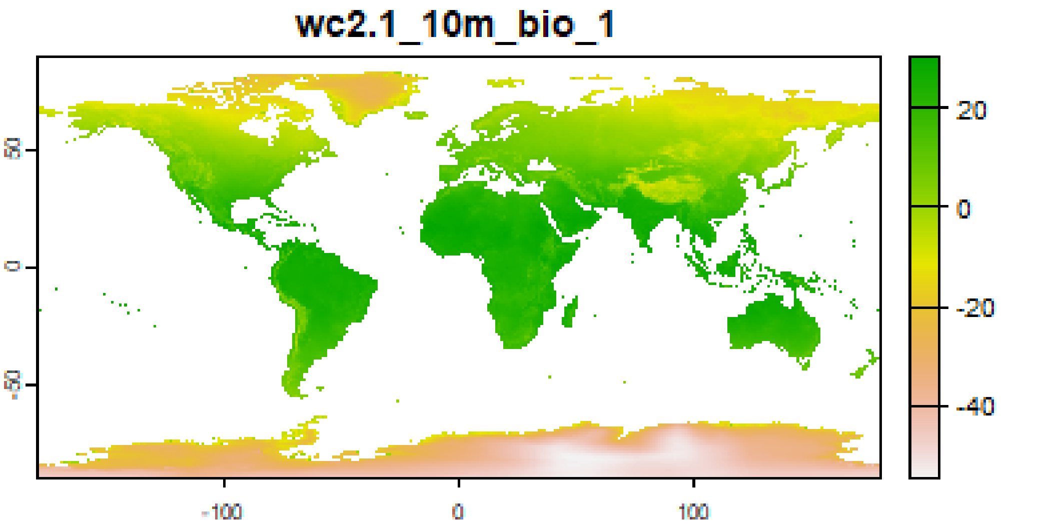


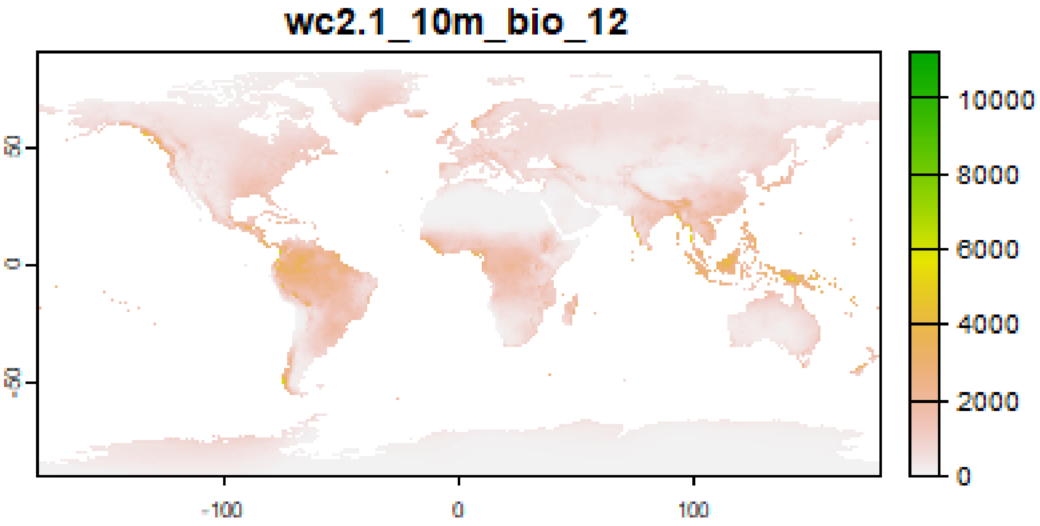
So the are in Canada and in the United States, but no reports from Mexico, so far.

5.1.2 Predictor variables

Here, as is common in species distribution modeling, we use climate data as predictor variables in our model. Specifically, we use “bioclimatic variables”, see: https://www.worldclim.org/data/bioclim.html. Here we used a spatial resolution of 10 minutes (one sixt of a degree). That is relatively coarse but it makes the download and processing faster.

```
wc <- geodata::worldclim_global("bio", res=10, ".")
plot(wc[[c(1, 12)]], nr=2)
```


wc2.1_10m_bio_1


wc2.1_10m_bio_12

Now extract climate data for the locations of our observations. In that way, we can find out what the climate conditions are that the species likes, apparently.

```
bfc <- extract(wc, bf[,1:2])
head(bfc, 3)
##   ID wc2.1_10m_bio_1 wc2.1_10m_bio_2 wc2.1_10m_bio_3 wc2.1_10m_bio_4
## 1  1       -1.832979       12.504708        28.95899       1152.4308
## 2  2        6.360650        5.865935        32.27475        462.5731
## 3  3        6.360650        5.865935        32.27475        462.5731
##   wc2.1_10m_bio_5 wc2.1_10m_bio_6 wc2.1_10m_bio_7 wc2.1_10m_bio_8
## 1        20.34075      -22.840000        43.18075        5.327750
## 2        16.65505       -1.519947        18.17500        3.964495
## 3        16.65505       -1.519947        18.17500        3.964495
##   wc2.1_10m_bio_9 wc2.1_10m_bio_10 wc2.1_10m_bio_11 wc2.1_10m_bio_12
## 1      -0.6887083         11.80792       -16.038542              991
## 2      10.4428196         12.28183         1.467686             3079
## 3      10.4428196         12.28183         1.467686             3079
```

(continues on next page)

(continued from previous page)

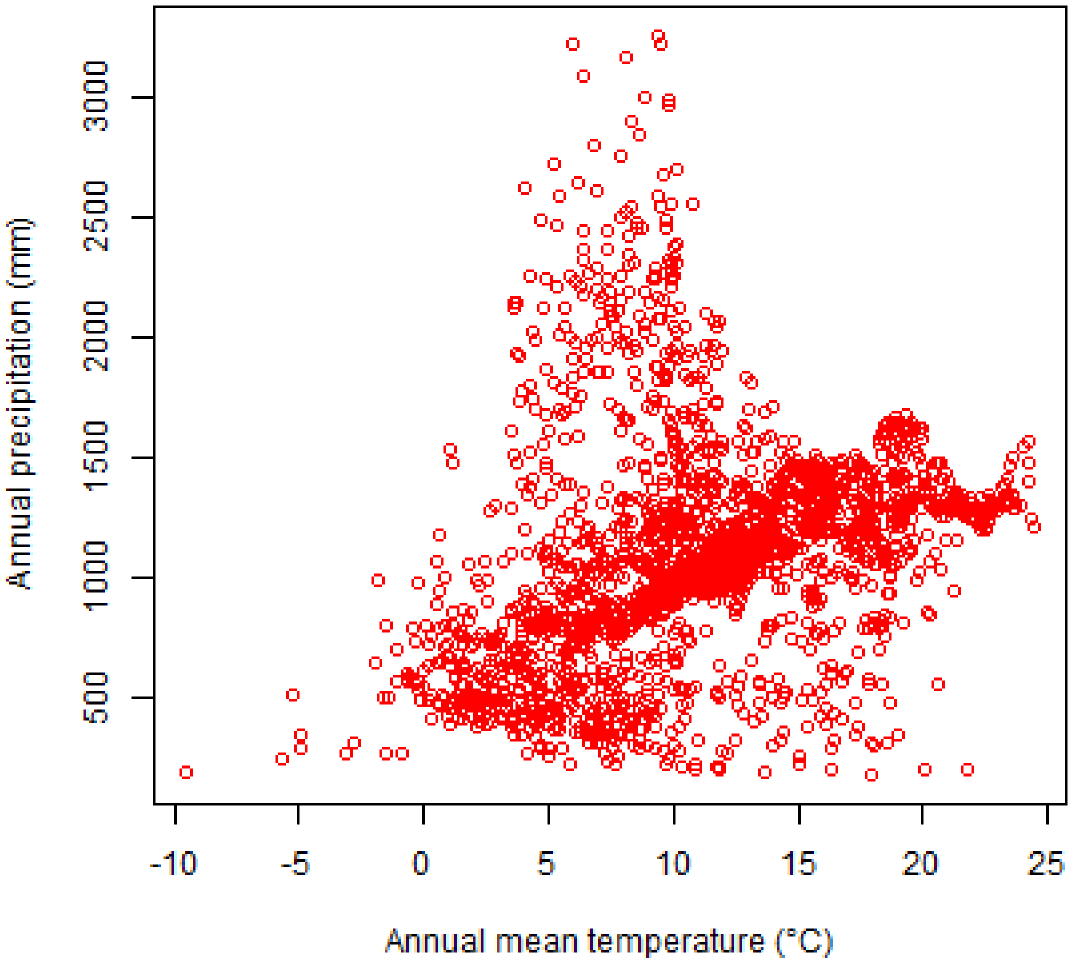
```
##   wc2.1_10m_bio_13 wc2.1_10m_bio_14 wc2.1_10m_bio_15 wc2.1_10m_bio_16
## 1              120               42         31.32536              337
## 2              448              141         35.27518             1127
## 3              448              141         35.27518             1127
##   wc2.1_10m_bio_17 wc2.1_10m_bio_18 wc2.1_10m_bio_19
## 1              157              288              216
## 2              468              630              873
## 3              468              630              873
```

I remove the first column with the ID that we do not need.

```
bfc <- bfc[,-1]
```

Now we can plot the species' distribution in a part of the *environmental* space. Here is a plot of temperature vs rainfall of sites where Bigfoot was observed.

```
plot(bfc[ ,"wc2.1_10m_bio_1"], bfc[, "wc2.1_10m_bio_12"], col="red",
        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 at 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. You blink and Bigfoot is gone!).

The common approach to deal with these type of data is to not model presence versus absence, but presence versus "background". The "background" is the random (or maximum entropy) expectation; it 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.

To do so, I first get the extent of all points

```
ext_bf <- ext(vect(bf[, 1:2])) + 1
ext_bf
## SpatExtent : -157.75, -63.4627, 24.141, 70.5 (xmin, xmax, ymin, ymax)
```

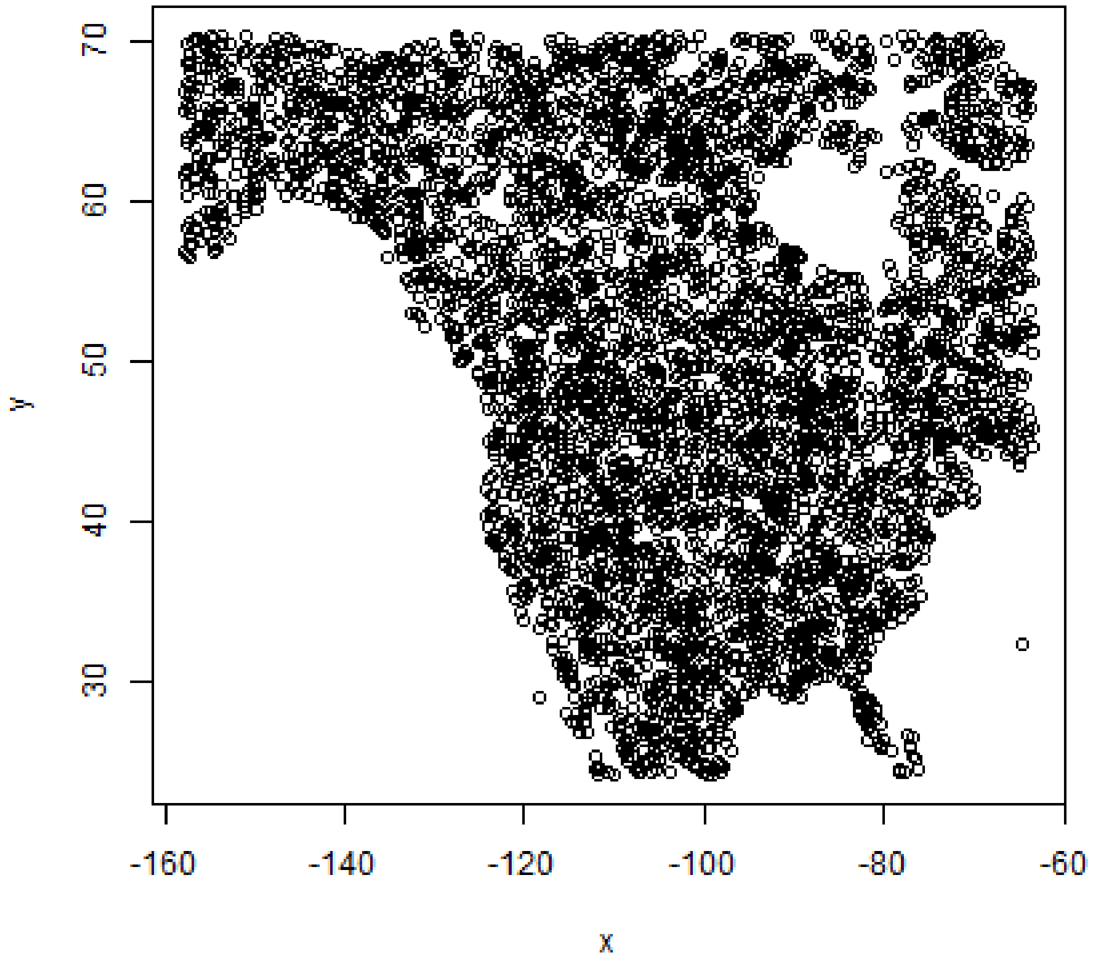
And then I take 5000 random samples (excluding `NA` cells) from SpatExtent `e`, by using it as a "window" (blacking out all other areas) on the climate SpatRaster.

```
set.seed(0)
window(wc) <- ext_bf
bg <- spatSample(wc, 5000, "random", na.rm=TRUE, xy=TRUE)
head(bg)
##           x        y wc2.1_10m_bio_1 wc2.1_10m_bio_2 wc2.1_10m_bio_3
## 1  -99.2500 66.75000     -13.2934895        7.870646        14.96619
## 2 -106.0833 42.08333       5.6722708       14.530958        36.82943
## 3 -111.9167 46.58333       6.7605939       14.135854        35.23372
## 4 -106.9167 54.75000       0.4086979       11.528605        24.43290
## 5 -118.2500 67.08333      -9.1363859        8.185354        16.34505
## 6 -111.2500 38.91667       8.4194584       15.997125        38.84047
##   wc2.1_10m_bio_4 wc2.1_10m_bio_5 wc2.1_10m_bio_6 wc2.1_10m_bio_7
## 1       1638.6833        15.42850       -37.16100        52.58950
## 2        894.3715        27.86600       -11.58875        39.45475
## 3        927.7927        28.14375       -11.97650        40.12025
## 4       1290.1088        22.55225       -24.63250        47.18475
## 5       1567.0846        17.46575       -32.61275        50.07850
## 6        904.0610        30.49050       -10.69625        41.18675
##   wc2.1_10m_bio_8 wc2.1_10m_bio_9 wc2.1_10m_bio_10 wc2.1_10m_bio_11
## 1        6.484917      -31.617332         7.518209        -31.76942
## 2        9.226916       -4.839750        17.168291         -4.83975
## 3       15.638333       -4.921750        18.186209         -4.92175
## 4       15.417084      -13.864500        15.417084        -16.31392
## 5        8.609292      -21.353209        10.573625        -27.49783
## 6       19.076958        3.179209        19.812834         -2.50475
##   wc2.1_10m_bio_12 wc2.1_10m_bio_13 wc2.1_10m_bio_14 wc2.1_10m_bio_15
## 1              171               33                4         70.29919
## 2              288               42               13         38.78144
## 3              293               48                9         53.40759
## 4              471               86               16         58.32499
## 5              223               43                7         61.21693
## 6              228               28               11         32.40370
##   wc2.1_10m_bio_16 wc2.1_10m_bio_17 wc2.1_10m_bio_18 wc2.1_10m_bio_19
## 1               90               13               78               13
## 2              112               41               90               41
## 3              129               35              115               35
## 4              220               53              220               56
## 5              108               27               93               29
## 6               83               40               72               44
```

Instead of using `window` you could also subset the climate data like this `wc <- crop(wc, ext_bf)`

Above, with `spatSample`, I used the argument `xy=TRUE` to be able to show were these points are from:

```
plot(bg[, c("x", "y")])
```

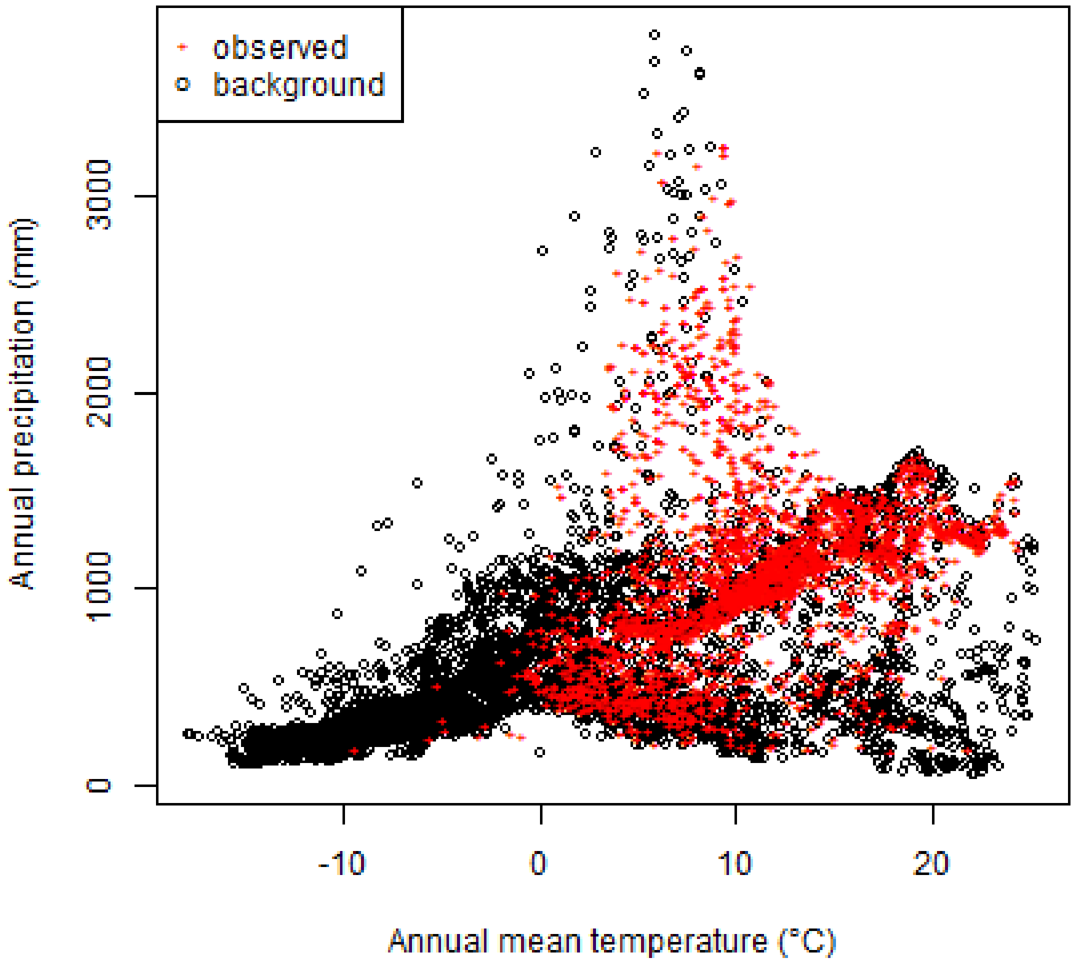


But we otherwise do not need them so I remove them again,

```
bg <- bg[, -c(1:2)]
```

We can now compare the climate of the presence and background points, for example, for temperature and rainfall

```
plot(bg[,1], bg[,12], xlab="Annual mean temperature (°C)",
          ylab="Annual precipitation (mm)", cex=.8)

points(bfc[,1], bfc[,12], col="red", cex=.6, pch="+")
legend("topleft", c("observed", "background"), col=c("red", "black"), pch=c("+", "o"),␣
↪pt.cex=c(.6, .8))
```



So we see that while Bigfoot is widespread, it is not common in cold areas, nor in hot and dry areas.

5.1.4 East vs West

I am first going to split the data into East and West. This is because I believe there are two sub-species (The Eastern Sasquatch is darker, less hairy, and has more pointy ears). I am principally interested in the western sub-species. Note how I use the original coordinates to subset the climate data. We can do this because they are in the same order.

```
#eastern points
bfe <- bfc[bf[,1] > -102, ]
#western points
bfw <- bfc[bf[,1] <= -102, ]
```

And now I combine the presence ("1") with the background ("0") data (I use the same background data for both subspecies)

```
dw <- rbind(cbind(pa=1, bfw), cbind(pa=0, bg))
de <- rbind(cbind(pa=1, bfe), cbind(pa=0, bg))

dw <- data.frame(dw)
de <- data.frame(na.omit(de))

dim(dw)
## [1] 6224   20
dim(de)
## [1] 6866   20
```

5.2 Fit a model

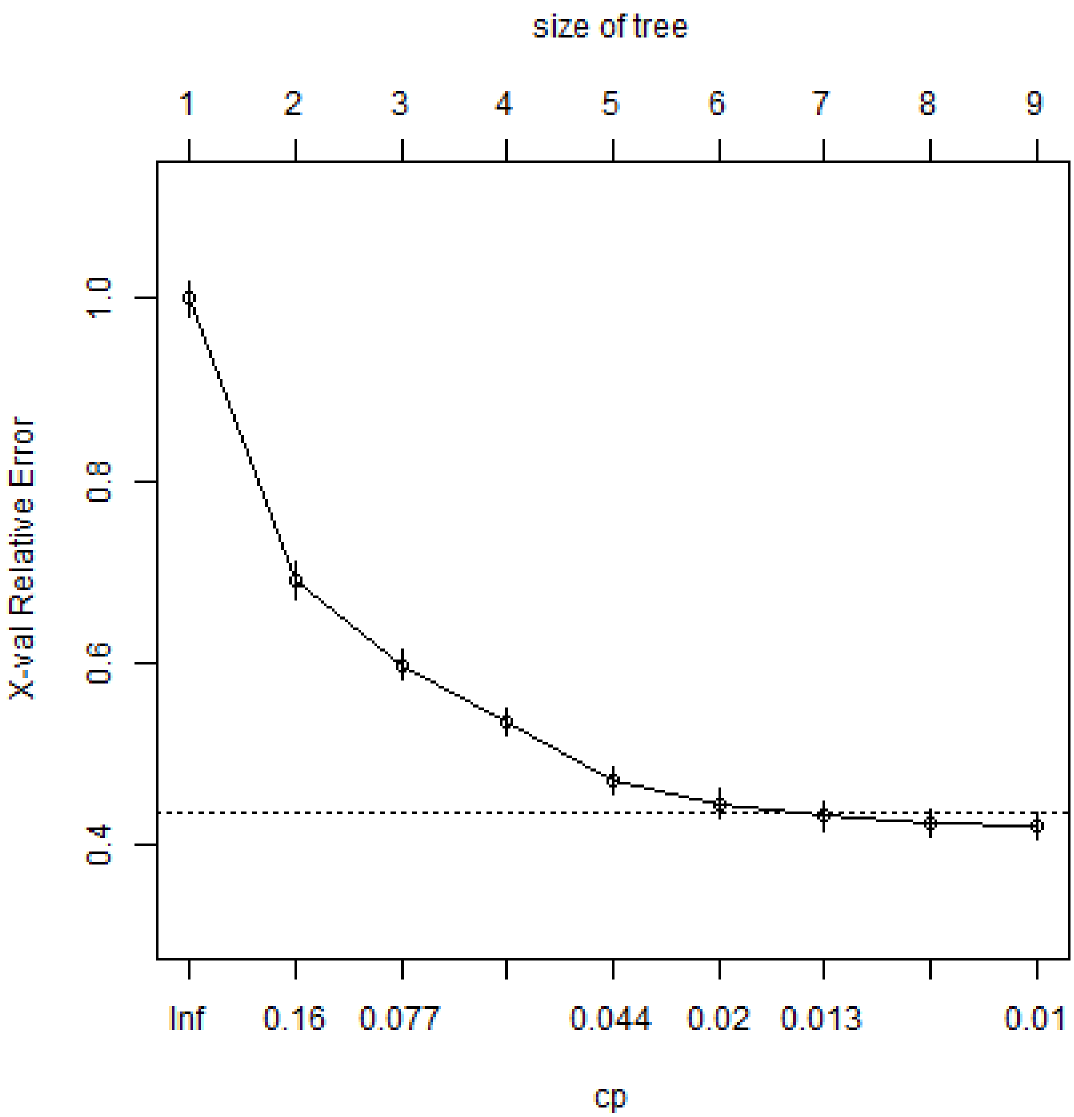
Now we have the data to fit a model. Let's first look at a Classification and Regression Trees (CART) model.

5.2.1 CART

```
library(rpart)
cart <- rpart(pa~., data=dw)
```

The "complexity parameter" can be used as a stopping parameter to avoid overfitting.

```
printcp(cart)
##
## Regression tree:
## rpart(formula = pa ~ ., data = dw)
##
## Variables actually used in tree construction:
## [1] wc2.1_10m_bio_10 wc2.1_10m_bio_12 wc2.1_10m_bio_14 wc2.1_10m_bio_18
## [5] wc2.1_10m_bio_19 wc2.1_10m_bio_3  wc2.1_10m_bio_4
##
## Root node error: 983.29/6224 = 0.15798
##
## n= 6224
##
##          CP nsplit rel error  xerror     xstd
## 1 0.322797      0   1.00000 1.00049 0.019357
## 2 0.080521      1   0.67720 0.68995 0.019745
## 3 0.073325      2   0.59668 0.59709 0.015583
## 4 0.068645      3   0.52336 0.53441 0.015405
## 5 0.027920      4   0.45471 0.47035 0.014765
## 6 0.014907      5   0.42679 0.44483 0.015067
## 7 0.010869      6   0.41188 0.43042 0.015332
## 8 0.010197      7   0.40102 0.42283 0.015159
## 9 0.010000      8   0.39082 0.42075 0.015069
plotcp(cart)
```

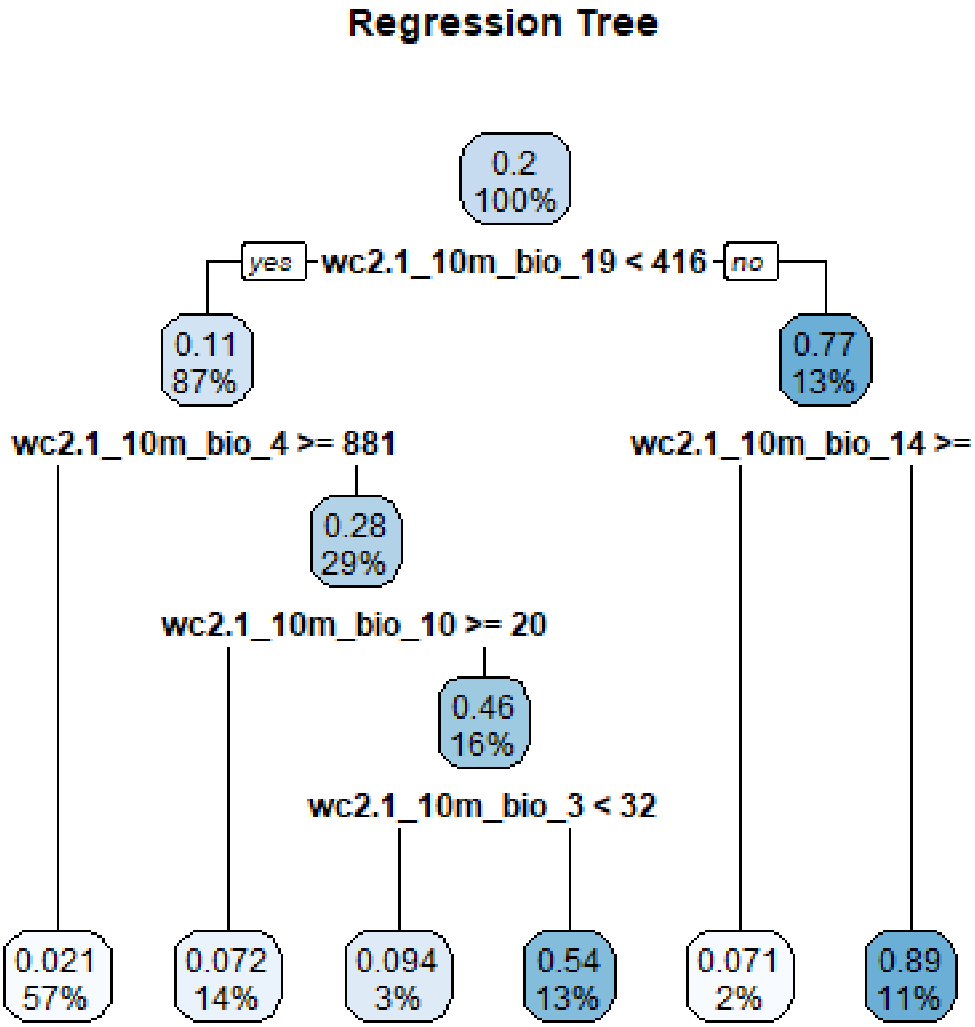


Fit the model again, with fewer splits

```
cart <- rpart(pa~., data=dw, cp=0.02)
```

And here is the tree

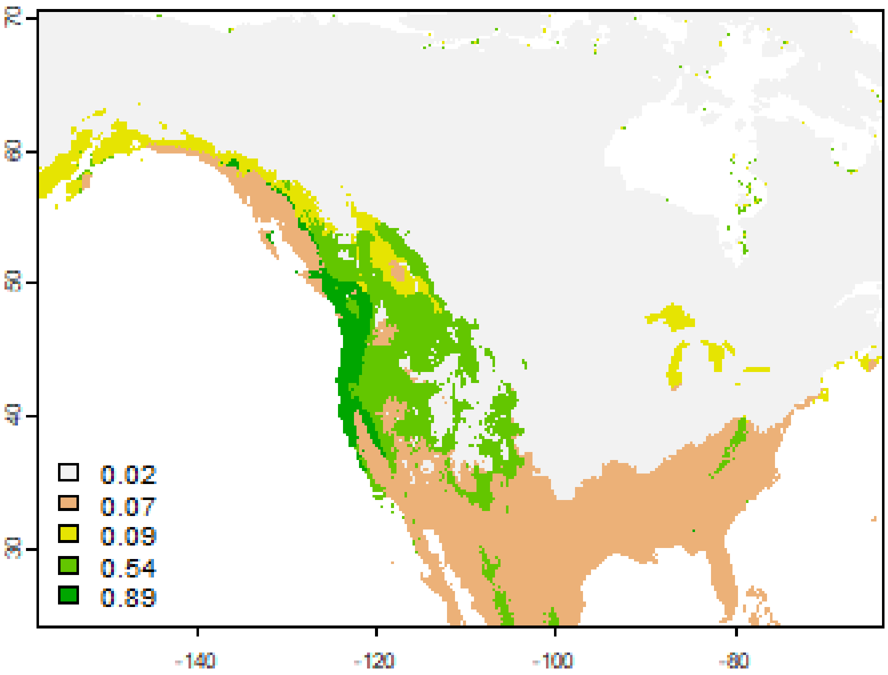
```
library(rpart.plot)
rpart.plot(cart, uniform=TRUE, main="Regression Tree")
```



Question 1: *Describe the environmental conditions that Bigfoot appears to enjoy most?*

And now we can use the model to show how attractive the climate is for this species.

```
x <- predict(wc, cart)
x <- mask(x, wc[[1]])
x <- round(x, 2)
plot(x, type="class", plg=list(x="bottomleft"))
```



Notice that there are six values, because the regression tree has six leaves.

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 tends to be over-fit, it is different each time a somewhat different datasets are used); and the quality of its predictions suffers from that. Random Forest does not have that problem as much. Above, with CART, we use regression, let's do both regression and classification here.

But first I set some points aside for validation (normally we would do k-fold cross-validation, but we keep it simple here).

```
set.seed(123)
i <- sample(nrow(dw), 0.2 * nrow(dw))
test <- dw[i,]
train <- dw[-i,]
```

First we do classification, by making a categorical variable for presence/background.

```
fpa <- as.factor(train[, 'pa'])
```

Now fit the RandomForest model

```
library(randomForest)
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
crf <- randomForest(train[, 2:ncol(train)], fpa)
crf
##
```

(continues on next page)

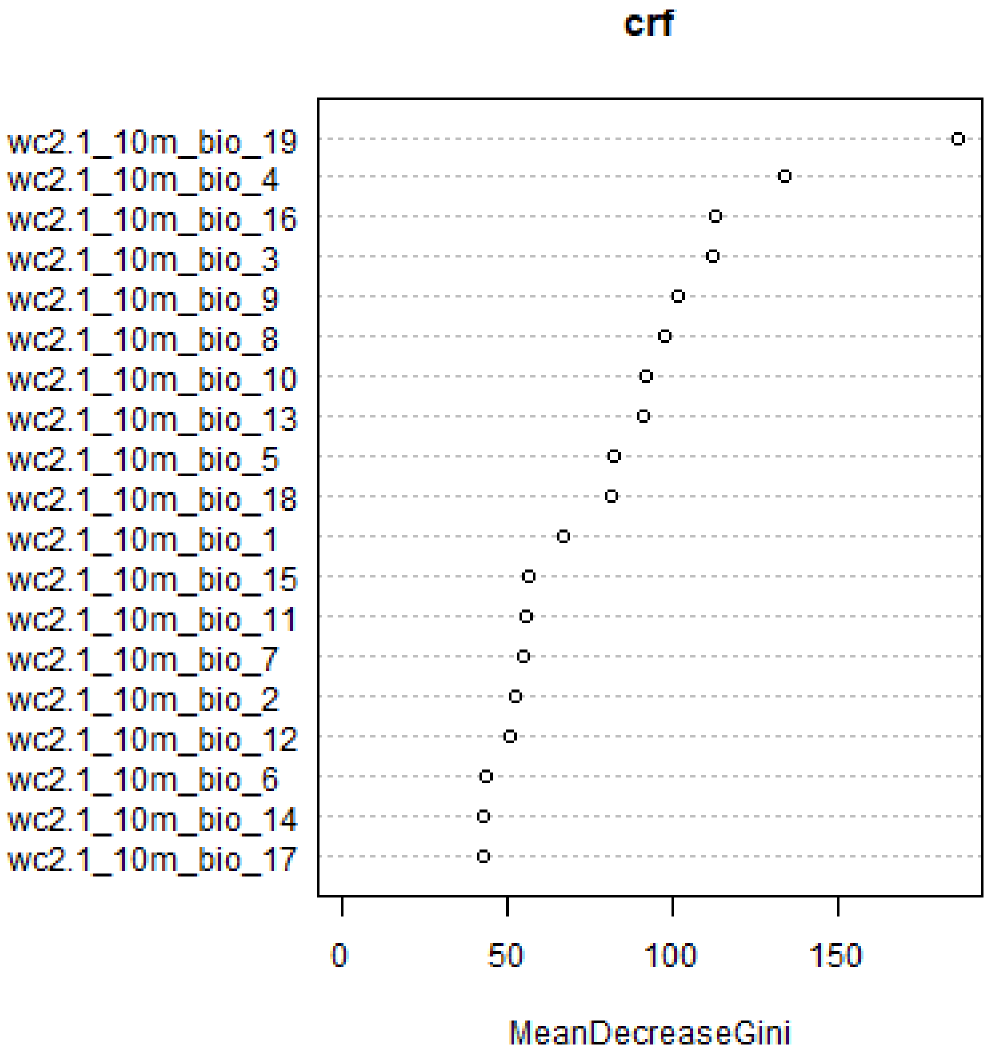
(continued from previous page)

```
## Call:
##  randomForest(x = train[, 2:ncol(train)], y = fpa)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 7.19%
## Confusion matrix:
##      0   1 class.error
## 0 3832 165  0.04128096
## 1  193 790  0.19633774
```

The Out-Of-Bag error rate is very small.

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

```
varImpPlot(crf)
```



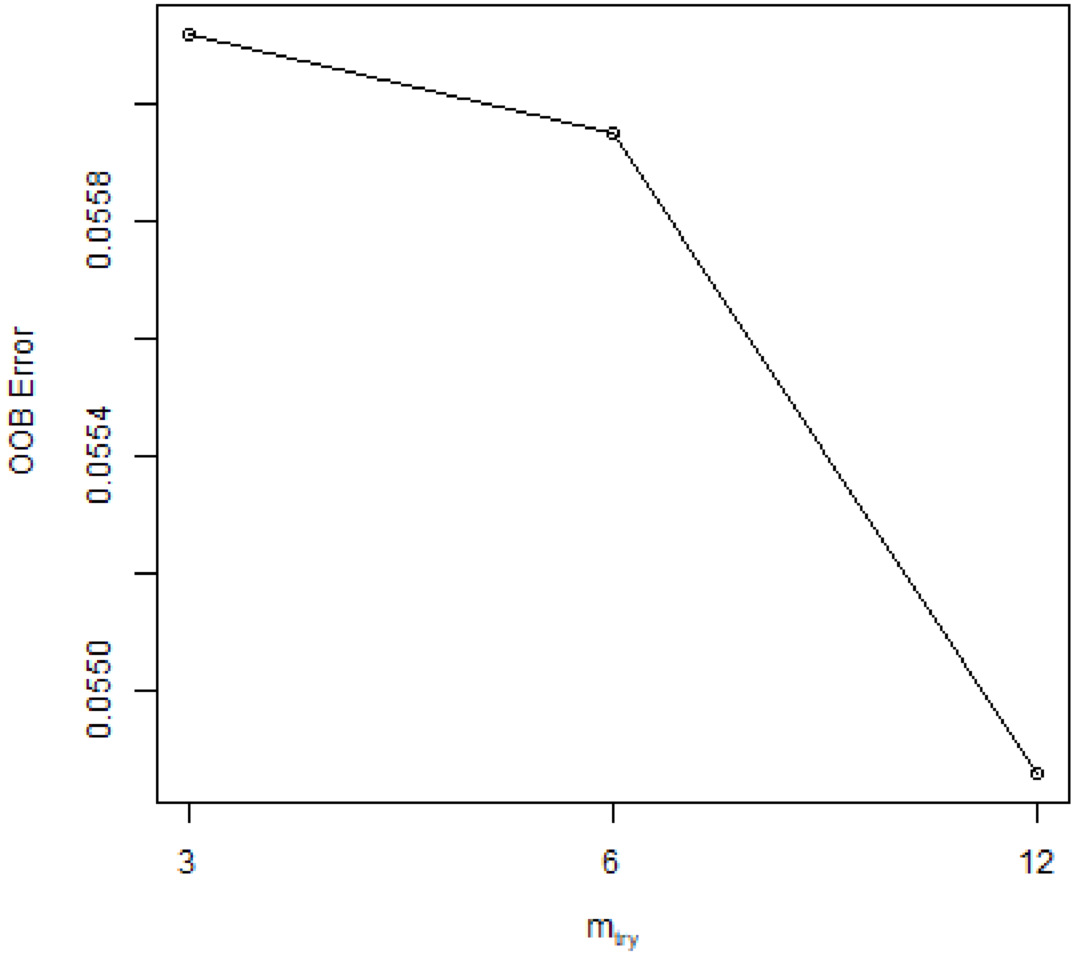
Now we use regression, rather than classification. First we tune a parameter.

```
trf <- tuneRF(train[, 2:ncol(train)], train[, "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 = 0.05594974
## 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 = 0.05612125
## -0.003065481 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?
```

(continues on next page)

(continued from previous page)

```
## mtry = 12     OOB error = 0.05485775
## 0.01951734 0.05
```



```
trf
##    mtry   OOBError
## 3     3 0.05612125
## 6     6 0.05594974
## 12   12 0.05485775
mt <- trf[which.min(trf[,2]), 1]
mt
## [1] 12
```

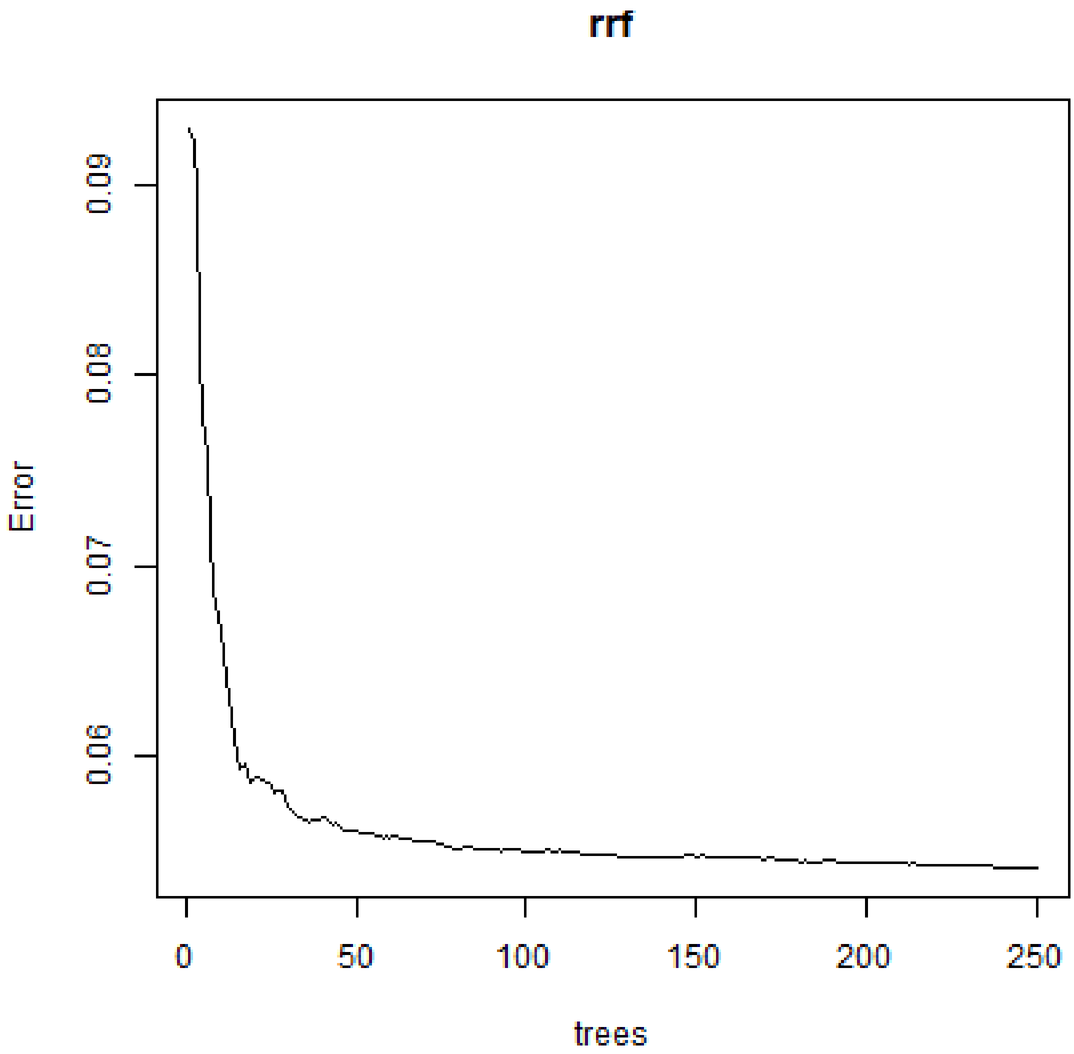
Question 2: *What did tuneRF help us find? What does the values of mt represent?*

```
rrf <- randomForest(train[, 2:ncol(train)], train[, "pa"], mtry=mt, ntree=250)
```

(continues on next page)

(continued from previous page)

```
## Warning in randomForest.default(train[, 2:ncol(train)], train[, "pa"], mtry =
## mt, : The response has five or fewer unique values.  Are you sure you want to
## do regression?
rrf
##
## Call:
##  randomForest(x = train[, 2:ncol(train)], y = train[, "pa"], ntree = 250,      mtry =␣
↪mt)
##                Type of random forest: regression
##                      Number of trees: 250
## No. of variables tried at each split: 12
##
##           Mean of squared residuals: 0.05421534
##                     % Var explained: 65.78
plot(rrf)
```



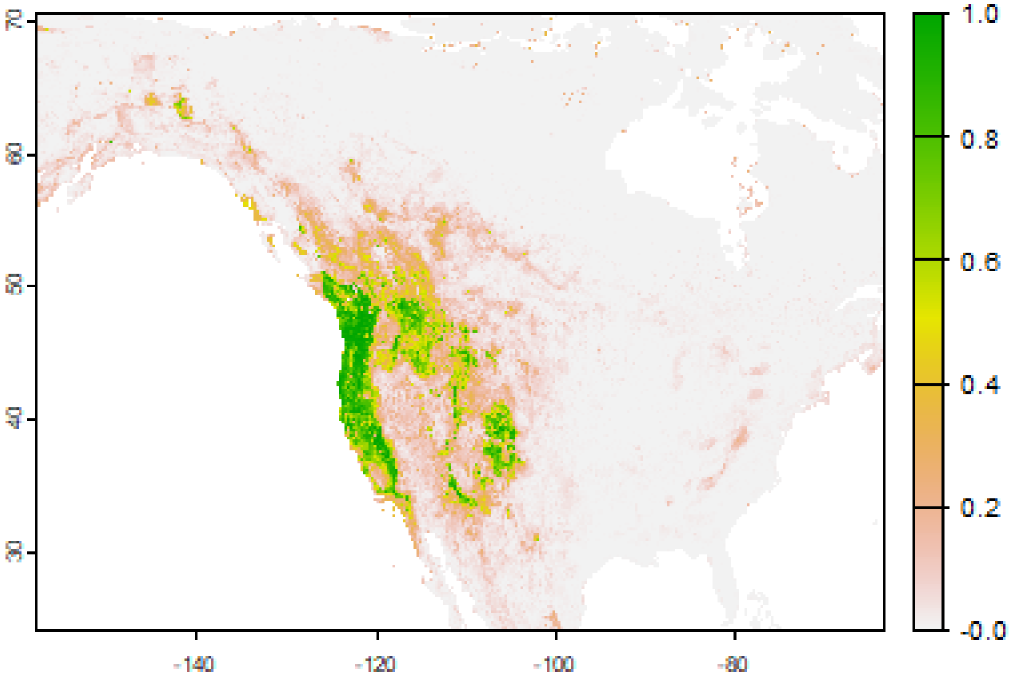
Question 3: *What does ``plot(rrf)`` show us?*

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 areas for Bigfoot in Australia, but let's stick to North America for now.

5.3.1 Regression

```
rp <- predict(wc, rrf, na.rm=TRUE)
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 use the training data for simplicity.

```
library(predicts)
eva <- pa_evaluate(predict(rrf, test[test$pa==1, ]), predict(rrf, test[test$pa==0, ]))
eva
## @stats
##    np   na prevalence   auc cor pcor   ODP
## 1 241 1003      0.194 0.965 0.8    0 0.806
##
## @thresholds
##   max_kappa max_spec_sens no_omission equal_prevalence equal_sens_spec
## 1     0.447         0.322       0.004            0.195           0.217
```

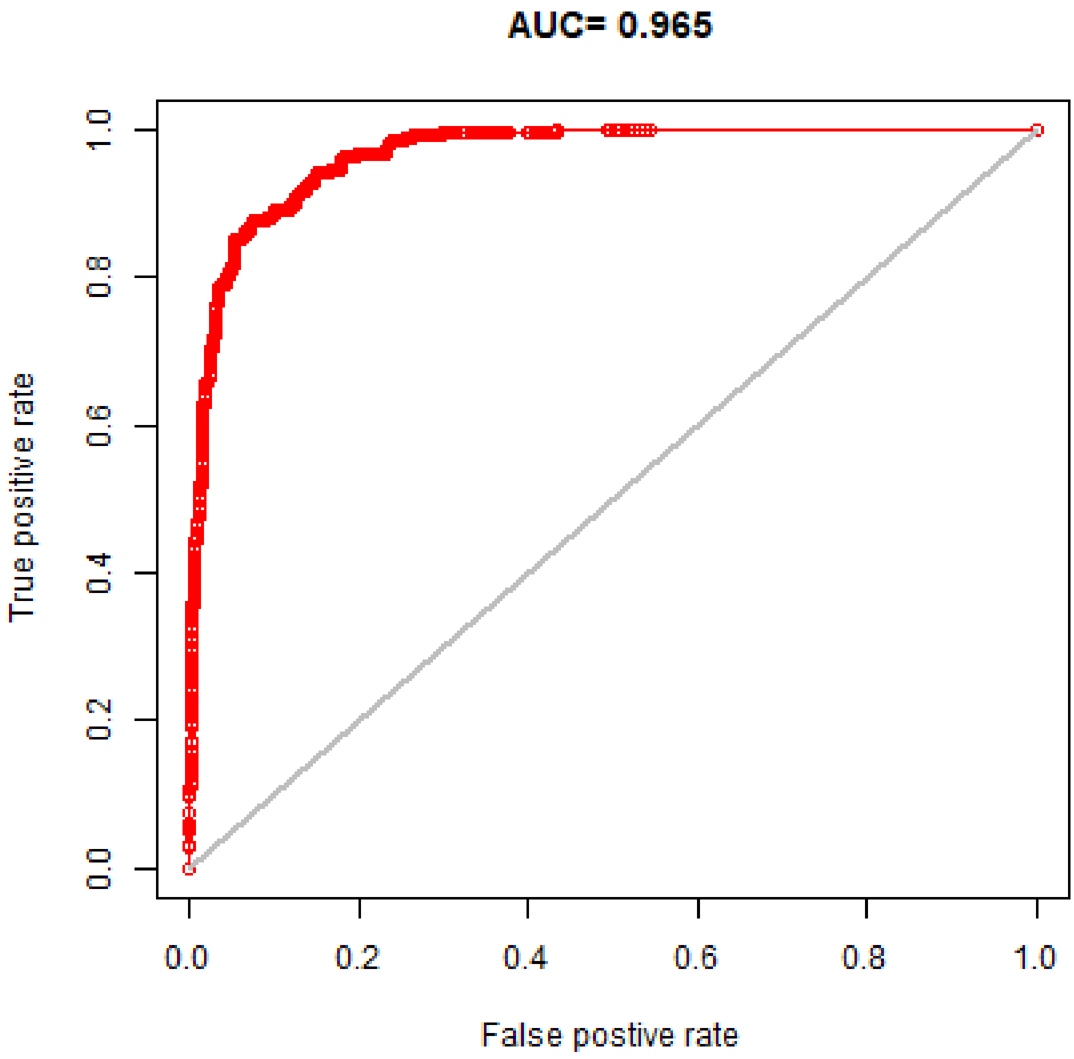
(continues on next page)

(continued from previous page)

```
##
## @tr_stats
##     treshold kappa  CCR  TPR  TNR  FPR  FNR  PPP  NPP  MCR  OR
## 1          0     0 0.19    1    0    1    0 0.19  NaN 0.81 NaN
## 2          0  0.25 0.56    1 0.46 0.54    0 0.31    1 0.44 Inf
## 3          0  0.25 0.56    1 0.46 0.54    0 0.31    1 0.44 Inf
## 4        ...   ...  ...  ...  ...  ...  ...  ...  ...  ... ...
## 594        1  0.05 0.81 0.03    1    0 0.97    1 0.81 0.19 Inf
## 595        1  0.05 0.81 0.03    1    0 0.97    1 0.81 0.19 Inf
## 596        1     0 0.81    0    1    0    1  NaN 0.81 0.19 NaN
```

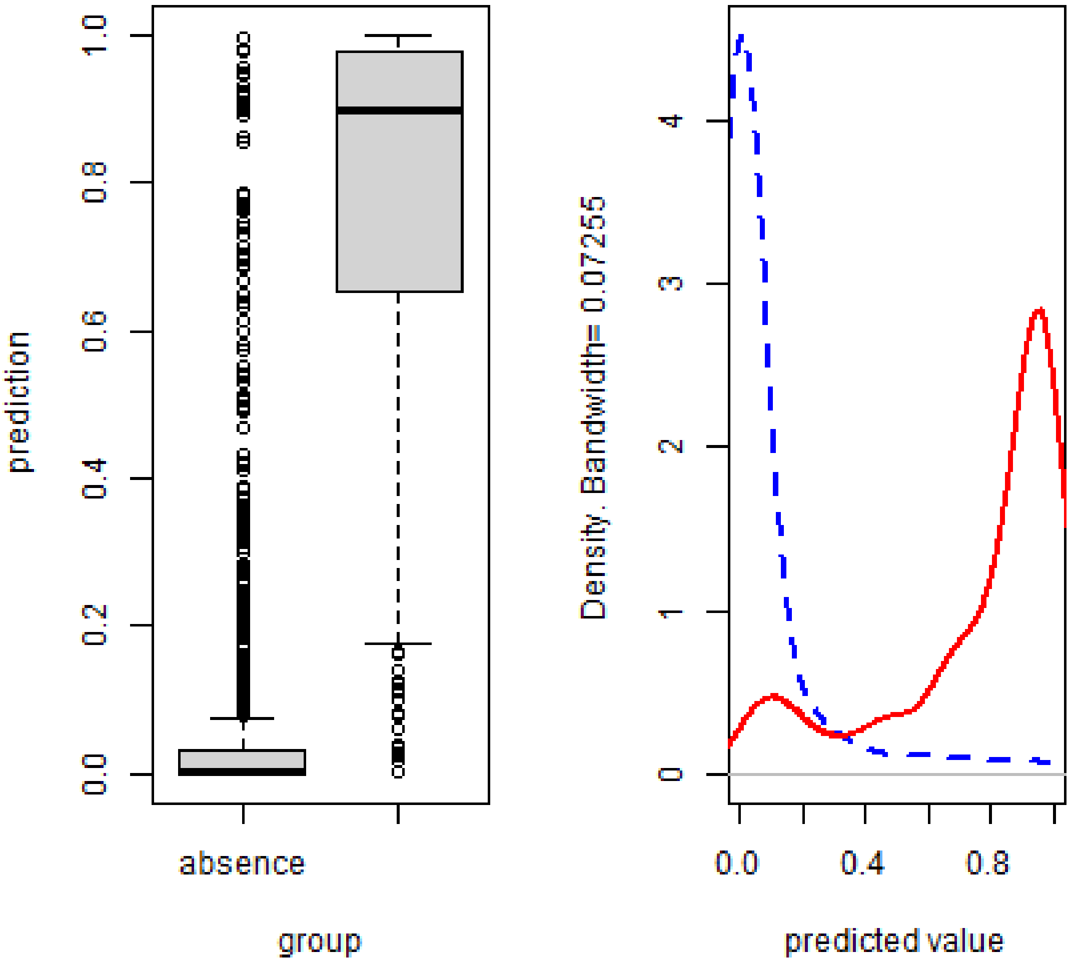
We can make a ROC plot

```
plot(eva, "ROC")
```



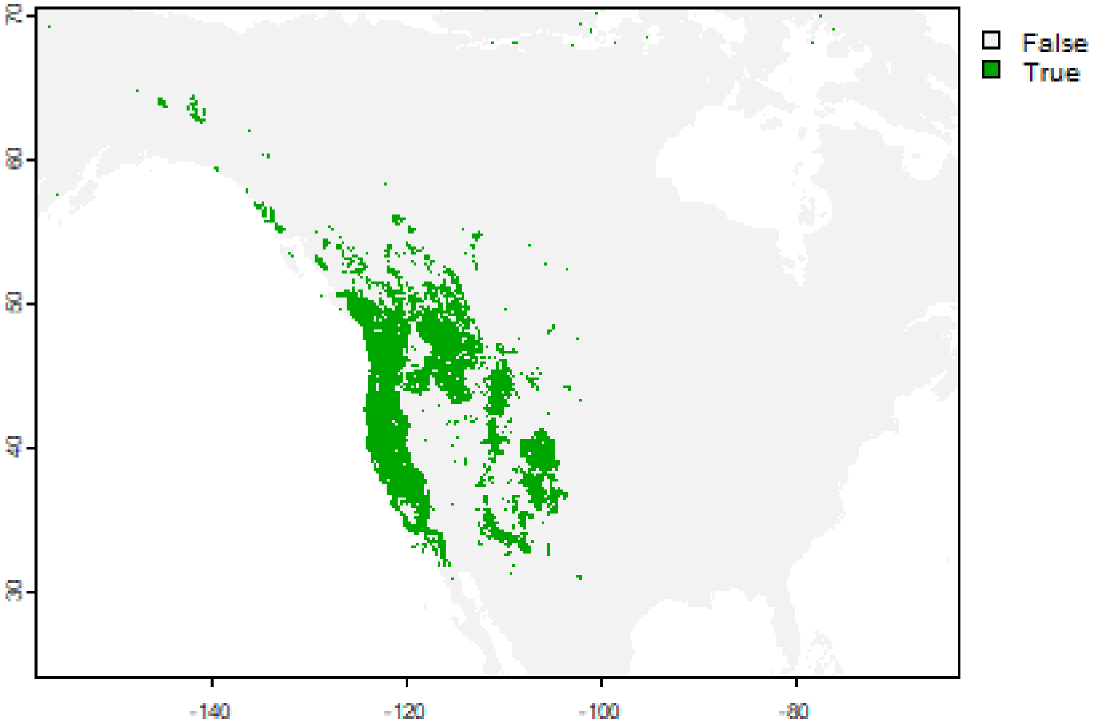
This suggests that the model is (very near) perfect in disinguising presence from background points. This is perhaps better illustrated with these plots:

```
par(mfrow=c(1,2))
plot(eva, "boxplot")
plot(eva, "density")
```



To get a good threshold to determine presence/absence and plot the prediction, we can use the "max specificity + sensitivity" threshold.

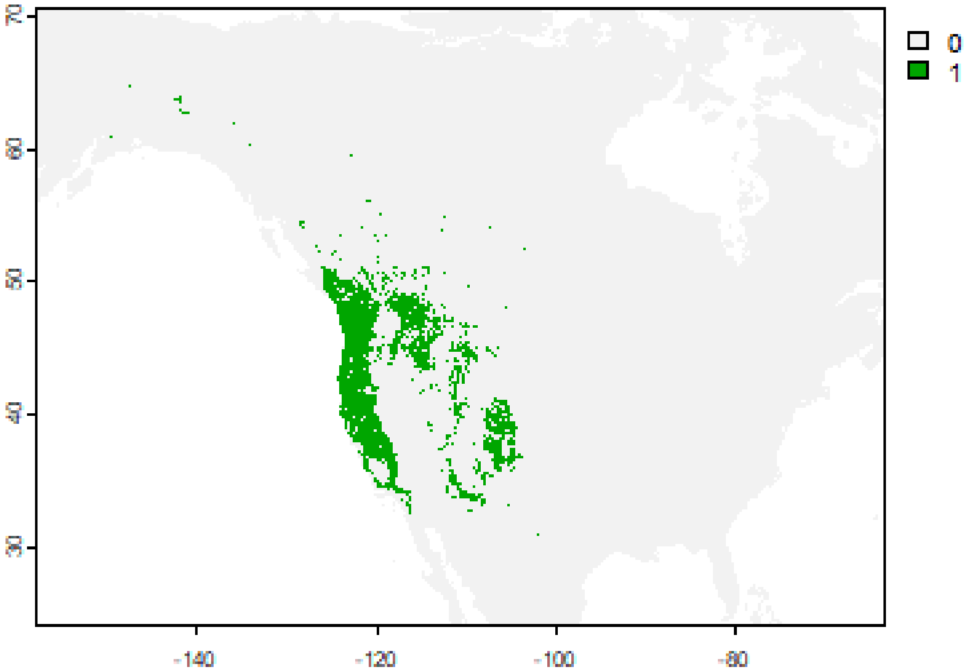
```
tr <- eva@thresholds
tr
##   max_kappa max_spec_sens no_omission equal_prevalence equal_sens_spec
## 1 0.4469667     0.3219856  0.00425973        0.1952333       0.2167239
plot(rp > tr$max_spec_sens)
```



5.3.2 Classification

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

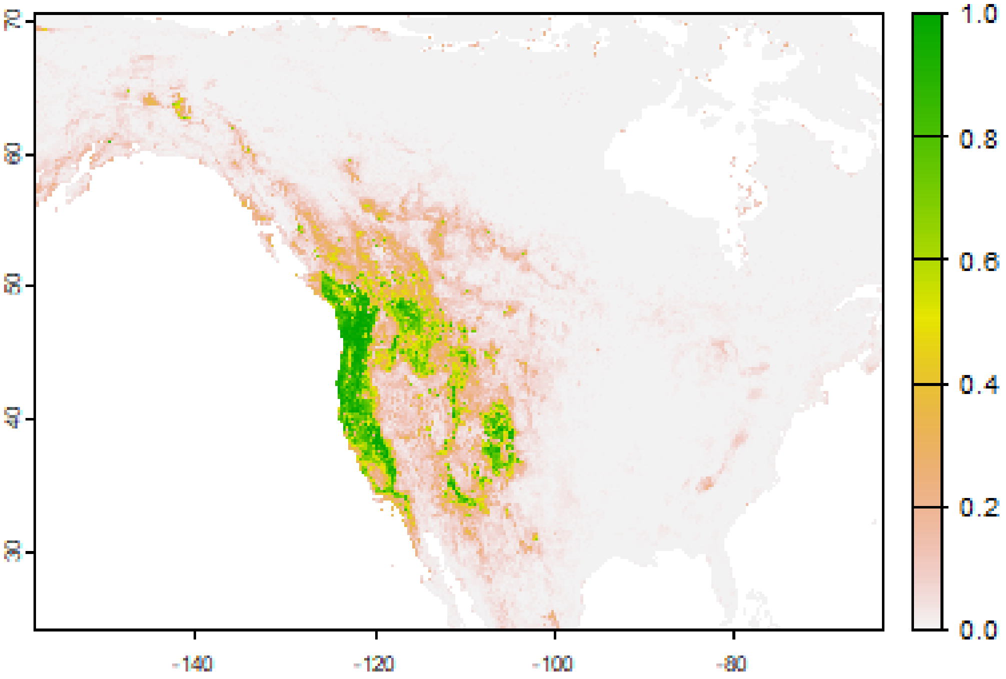
```
rc <- predict(wc, crf, na.rm=TRUE)
plot(rc)
```



They are different because the classification used a threshold of 0.5, which is not necessarily appropriate.

You can get probabilities for the classes (in this case there are 2 classes, presence and absence, and I only plot presence)

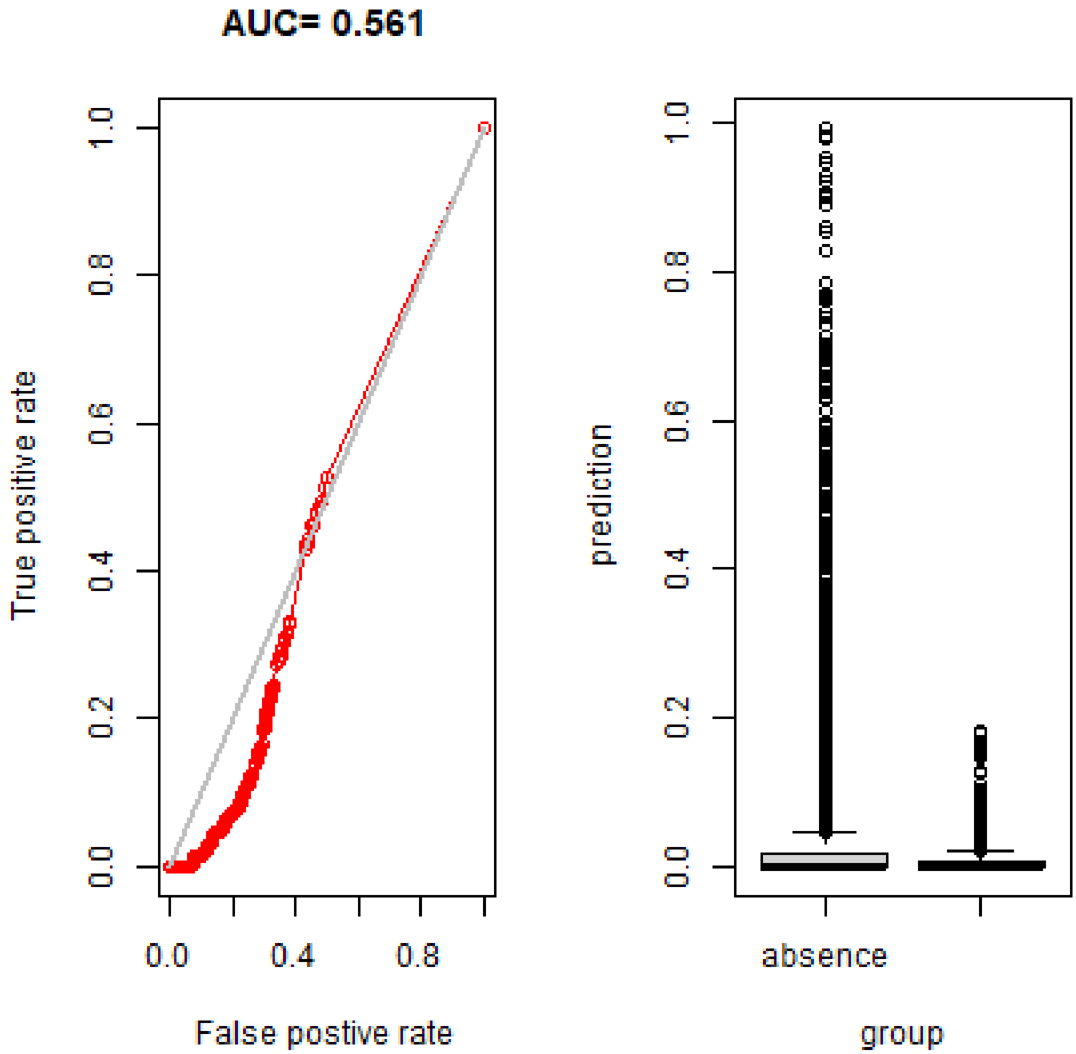
```
rc2 <- predict(wc, crf, type="prob", na.rm=TRUE)
plot(rc2, 2)
```



5.4 Extrapolation

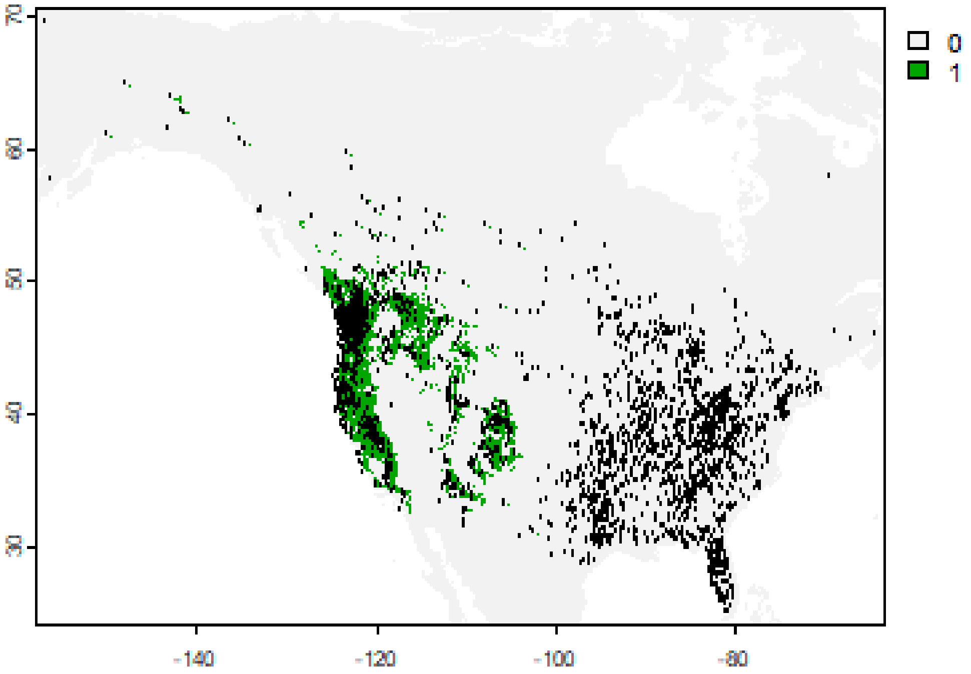
Now, let's see if our model is general enough to predict the distribution of the Eastern species.

```
eva2 <- pa_evaluate(predict(rrf, de[de$pa==1, ]), predict(rrf, de[de$pa==0, ]))
eva2
## @stats
##      np   na prevalence   auc    cor pcor   ODP
## 1 1866 5000      0.272 0.561 -0.137    0 0.728
##
## @thresholds
##   max_kappa max_spec_sens no_omission equal_prevalence equal_sens_spec
## 1     0.001         0.001           0            0.271           0.001
##
## @tr_stats
##     treshold kappa  CCR  TPR TNR FPR  FNR  PPP  NPP  MCR   OR
## 1          0     0 0.27    1   0   1    0 0.27  NaN 0.73  NaN
## 2          0  0.02 0.51 0.53 0.5 0.5 0.47 0.28 0.74 0.49 1.12
## 3          0  0.02 0.51 0.53 0.5 0.5 0.47 0.28 0.74 0.49 1.12
## 4        ...   ...  ...  ... ... ...  ...  ...  ...  ...  ...
## 522     0.99     0 0.73    0   1   0    1    0 0.73 0.27    0
## 523     0.99     0 0.73    0   1   0    1  NaN 0.73 0.27  NaN
## 524     0.99     0 0.73    0   1   0    1  NaN 0.73 0.27  NaN
par(mfrow=c(1,2))
plot(eva2, "ROC")
plot(eva2, "boxplot")
```



By this measure, it is a *terrible* model – as we already saw on the map. So our model is really good in predicting the range of the West, but it cannot extrapolate at all to the East.

```
plot(rc)
points(bf[,1:2], cex=.25)
```

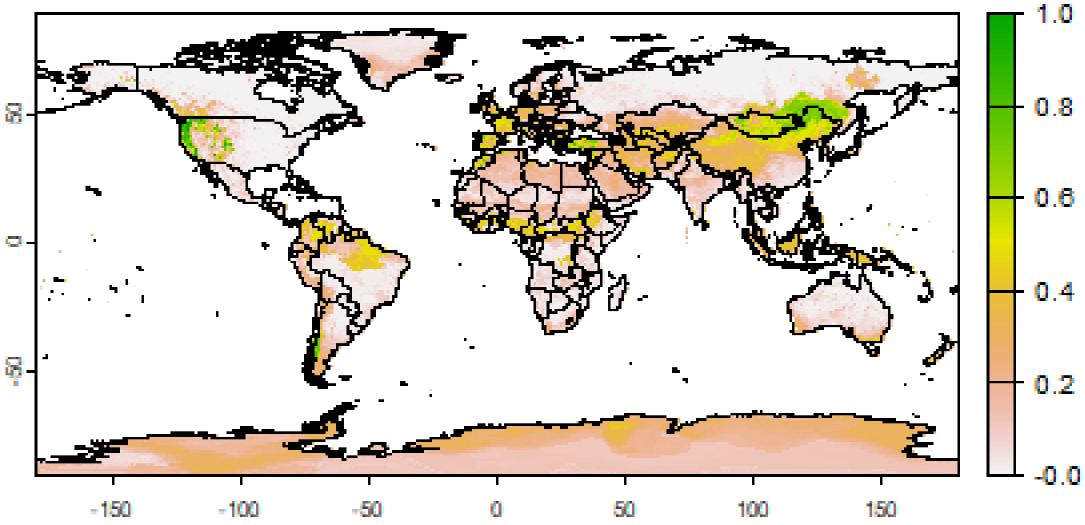


Question 4: *Why would it be that the model does not extrapolate well?*

An important question in the biogeography of the Bigfoot would be if it can survive in other parts of the world (it has been spotted trying to get on commerical flights leaving North America).

Let's see.

```
window(wc) <- NULL
pm <- predict(wc, rrf, na.rm=TRUE)
plot(pm)
lines(wrld)
```

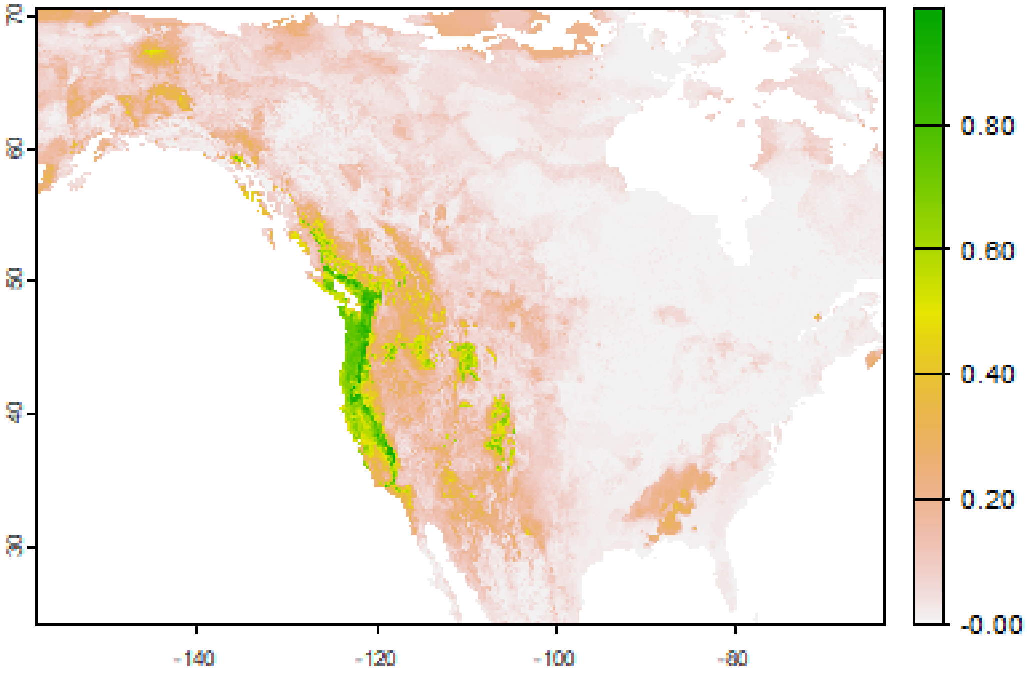


Question 5: *What are some countries that should consider Bigfoot as a potential invasive species?*

5.5 Climate change

We can also estimate range shifts due to climate change. We can use the same model, but now extrapolate in time (and space).

```
fut <- cmip6_world("CNRM-CM6-1", "585", "2061-2080", var="bio", res=10, path=".")
names(fut)
##  [1] "bio01" "bio02" "bio03" "bio04" "bio05" "bio06" "bio07" "bio08" "bio09"
## [10] "bio10" "bio11" "bio12" "bio13" "bio14" "bio15" "bio16" "bio17" "bio18"
## [19] "bio19"
names(wc)
##  [1] "wc2.1_10m_bio_1"  "wc2.1_10m_bio_2"  "wc2.1_10m_bio_3"  "wc2.1_10m_bio_4"
##  [5] "wc2.1_10m_bio_5"  "wc2.1_10m_bio_6"  "wc2.1_10m_bio_7"  "wc2.1_10m_bio_8"
##  [9] "wc2.1_10m_bio_9"  "wc2.1_10m_bio_10" "wc2.1_10m_bio_11" "wc2.1_10m_bio_12"
## [13] "wc2.1_10m_bio_13" "wc2.1_10m_bio_14" "wc2.1_10m_bio_15" "wc2.1_10m_bio_16"
## [17] "wc2.1_10m_bio_17" "wc2.1_10m_bio_18" "wc2.1_10m_bio_19"
names(fut) <- names(wc)
window(fut) <- ext_bf
pfut <- predict(fut, rrf, na.rm=TRUE)
plot(pfut)
```



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?*

5.6 Further reading

More on Species distribution modeling with R.

CHAPTER

SIX

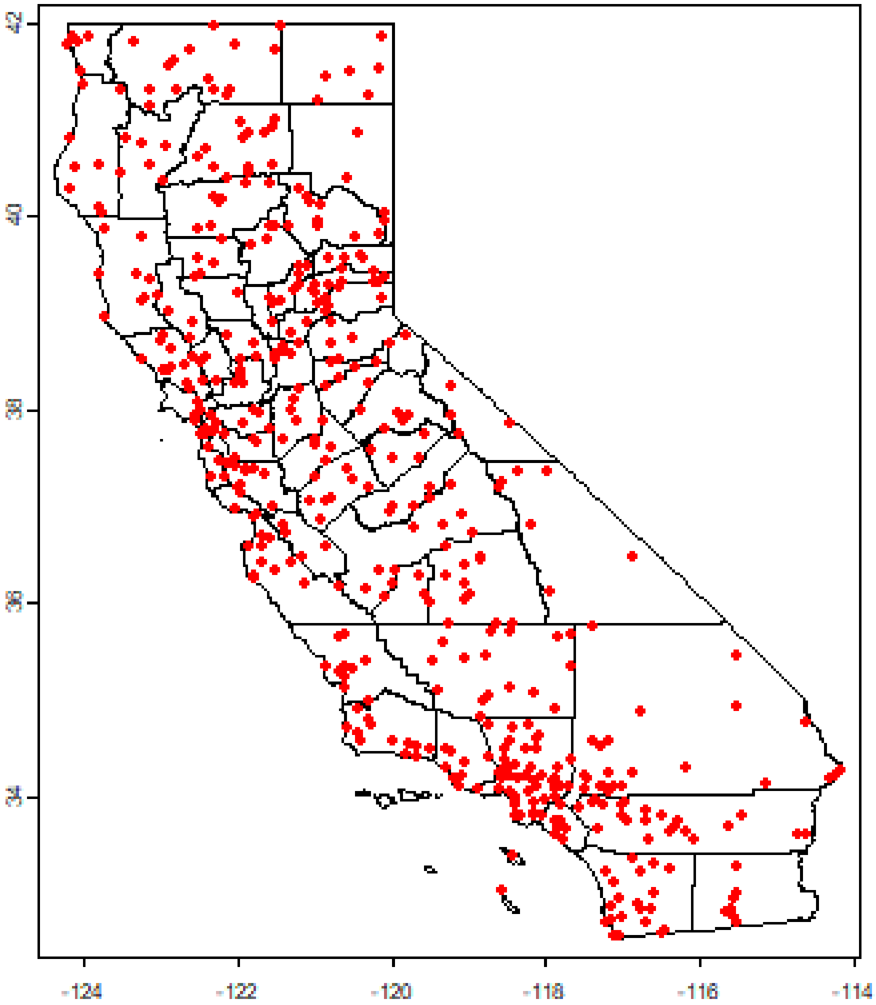
LOCAL REGRESSION

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 mis-specified 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 (!require("rspat")) remotes::install_github('rspatial/rspat')
## Loading required package: rspat
## Loading required package: terra
## terra 1.7.62

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

plot(counties)
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 a `SpatVector` objects with a planar crs.

```
alb <- "+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000␣
↪+datum=WGS84 +units=m"
sp <- vect(p, c("LONG", "LAT"), crs="+proj=longlat +datum=WGS84")
spt <- project(sp, alb)
ctst <- project(counties, alb)
```

Get the optimal bandwidth

```
library( spgwr )
## Loading required package: sp
## 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=as.data.frame(spt), coords=geom(spt)[,c("x", "y")])
## 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 <- rast(ctst, res=10000)
r <- rasterize(ctst, r)
newpts <- as.points(r)
```

Run the gwr function

```
g <- gwr(pan ~ ALT, data=as.data.frame(spt), coords=geom(spt)[,c("x", "y")],␣
↪bandwidth=bw, fit.points=geom(newpts)[,c("x", "y")])
g
## Call:
```

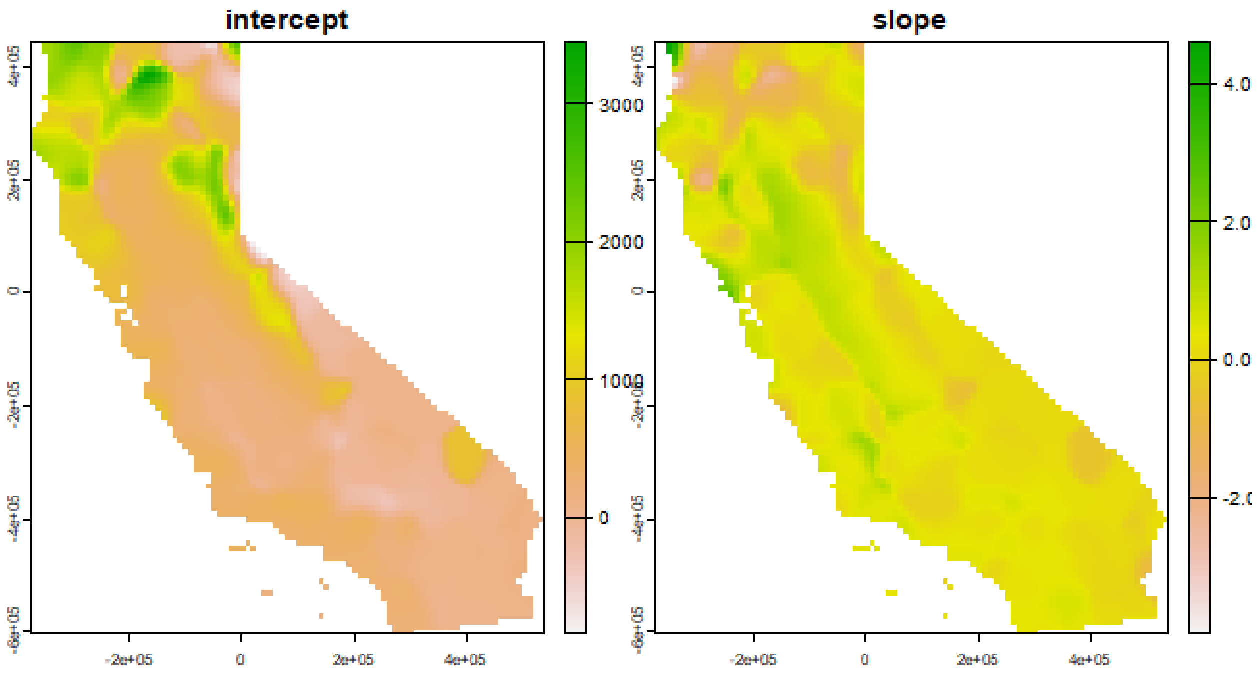
(continues on next page)

(continued from previous page)

```
## gwr(formula = pan ~ ALT, data = as.data.frame(spt), coords = geom(spt)[,
##     c("x", "y")], bandwidth = bw, fit.points = geom(newpts)[,
##     c("x", "y")])
## Kernel function: gwr.Gauss
## Fixed bandwidth: 19781.47
## Fit points: 4090
## Summary of GWR coefficient estimates at fit points:
##                      Min.     1st Qu.      Median     3rd Qu.      Max.
## X.Intercept. -846.314308   77.986476  328.579339  729.588996 3452.1972
## ALT            -3.961701    0.034149    0.201568    0.418716    4.6022
```

Link the results back to the raster

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



6.2 California House Price Data

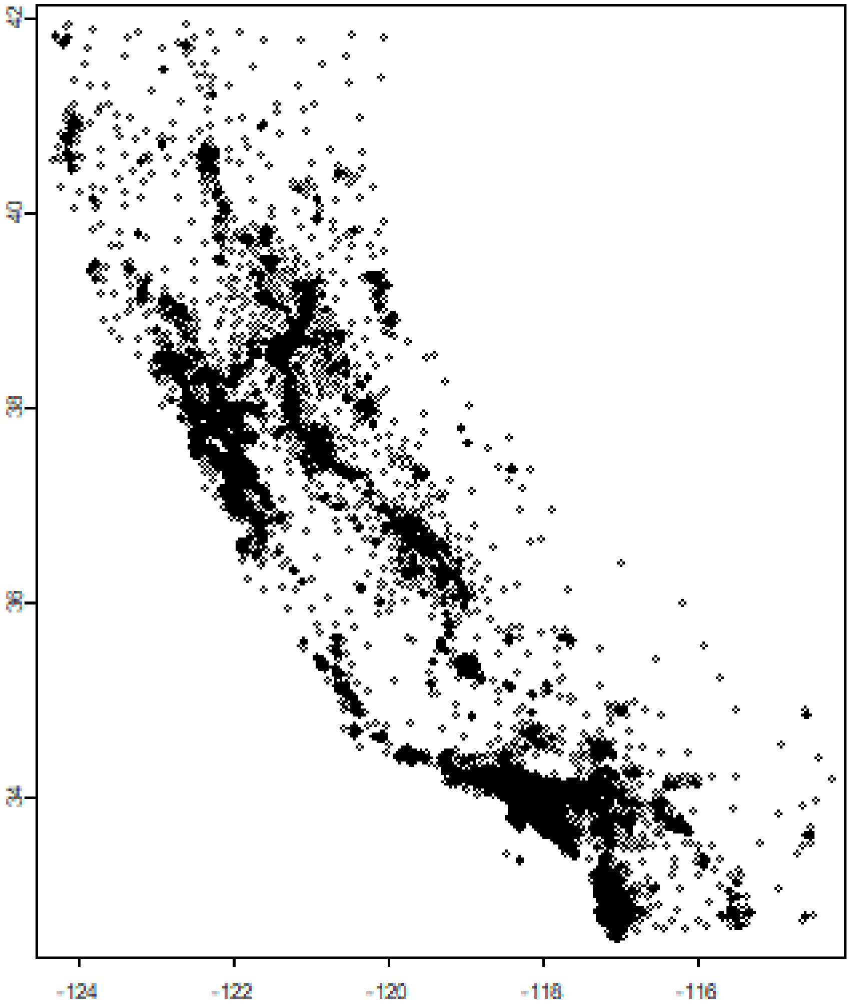
We will use house prices data from the 1990 census, taken from "Pace, R.K. and R. Barry, 1997. Sparse Spatial Autoregressions. Statistics and Probability Letters 33: 291-297."

```
houses <- spat_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.

```
hvect <- vect(houses, c("longitude", "latitude"))
```

```
plot(hvect, 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!).

```
crs(hvect) <- crs(counties)
```

Do a spatial query (points in polygon)

```
cnty <- extract(counties, hvect)
head(cnty)
##   id.y STATE COUNTY    NAME LSAD LSAD_TRANS
## 1    1    06    001 Alameda   06     County
## 2    2    06    001 Alameda   06     County
## 3    3    06    001 Alameda   06     County
## 4    4    06    001 Alameda   06     County
## 5    5    06    001 Alameda   06     County
## 6    6    06    001 Alameda   06     County
```

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)
```

Compute the population by county

```
totpop <- tapply(hd$population, hd$NAME, sum)
totpop
##         Alameda          Alpine          Amador           Butte       Calaveras
##         1241779            1113           30039          182120           31998
##          Colusa    Contra Costa       Del Norte       El Dorado          Fresno
##           16275          799017           16045          128624          662261
##           Glenn        Humboldt        Imperial            Inyo            Kern
##           24798          116418          108633           18281          528995
##           Kings            Lake          Lassen     Los Angeles          Madera
##           91842           50631           27214         8721937           88089
##           Marin        Mariposa       Mendocino          Merced           Modoc
##          204241           14302           75061          176457            9678
##            Mono        Monterey            Napa          Nevada          Orange
##            9956          342314          108030           78510         2340204
##          Placer          Plumas       Riverside      Sacramento      San Benito
##          170761           19739         1162787         1038540           36697
##  San Bernardino       San Diego   San Francisco     San Joaquin San Luis Obispo
##         1409740         2425153          683068          477184          203764
##       San Mateo   Santa Barbara     Santa Clara      Santa Cruz          Shasta
##          614816          335177         1486054          216732          147036
##          Sierra        Siskiyou          Solano          Sonoma      Stanislaus
##            3318           43531          337429          385296          370821
##          Sutter          Tehama         Trinity          Tulare        Tuolumne
##           63689           49625           13063          309073           48456
##         Ventura            Yolo            Yuba
##          649935          138799           58954
```

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$households
# now use aggregate (similar to tapply)
csum <- aggregate(hd[, c('suminc', 'households')], list(hd$NAME), sum)
# divide total income by number of housefholds
csum$income <- 10000 * csum$suminc / csum$households
# sort
csum <- csum[order(csum$income), ]
head(csum)
##     Group.1    suminc households   income
## 53  Trinity 11198.985       5156 21720.30
## 58     Yuba 43739.708      19882 21999.65
## 25    Modoc  8260.597       3711 22259.76
## 47 Siskiyou 38769.952      17302 22407.79
## 17     Lake 47612.899      20805 22885.32
```

(continues on next page)

(continued from previous page)

```
## 11    Glenn 20497.683       8821 23237.37
tail(csum)
##         Group.1    suminc households   income
## 56      Ventura  994094.8     210418 47243.81
## 7  Contra Costa 1441734.6     299123 48198.72
## 30       Orange 3938638.1     800968 49173.48
## 43  Santa Clara 2621895.6     518634 50553.87
## 41    San Mateo 1169145.6     230674 50683.89
## 21        Marin  436808.4      85869 50869.17
```

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$population
hd$bedroomhead <- hd$bedrooms / hd$population
hd$hhsize <- hd$population / hd$households
```

Ordinary least squares regression:

```
# OLS
m <- glm( houseValue ~ income + houseAge + roomhead + bedroomhead + population, data=hd)
summary(m)
##
## Call:
## glm(formula = houseValue ~ income + houseAge + roomhead + bedroomhead +
##     population, data = hd)
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -6.508e+04  2.533e+03 -25.686  < 2e-16 ***
## income       5.179e+04  3.833e+02 135.092  < 2e-16 ***
## houseAge     1.832e+03  4.575e+01  40.039  < 2e-16 ***
## roomhead    -4.720e+04  1.489e+03 -31.688  < 2e-16 ***
## bedroomhead  2.648e+05  6.820e+03  38.823  < 2e-16 ***
## population   3.947e+00  5.081e-01   7.769 8.27e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6022427437)
##
##     Null deviance: 2.7483e+14  on 20639  degrees of freedom
## Residual deviance: 1.2427e+14  on 20634  degrees of freedom
## AIC: 523369
##
## Number of Fisher Scoring iterations: 2
coefficients(m)
##   (Intercept)        income      houseAge      roomhead   bedroomhead
```

(continues on next page)

(continued from previous page)

```
## -65075.701407  51786.005862   1831.685266 -47198.908765 264766.186284
##    population
##      3.947461
```

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), ]
```

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

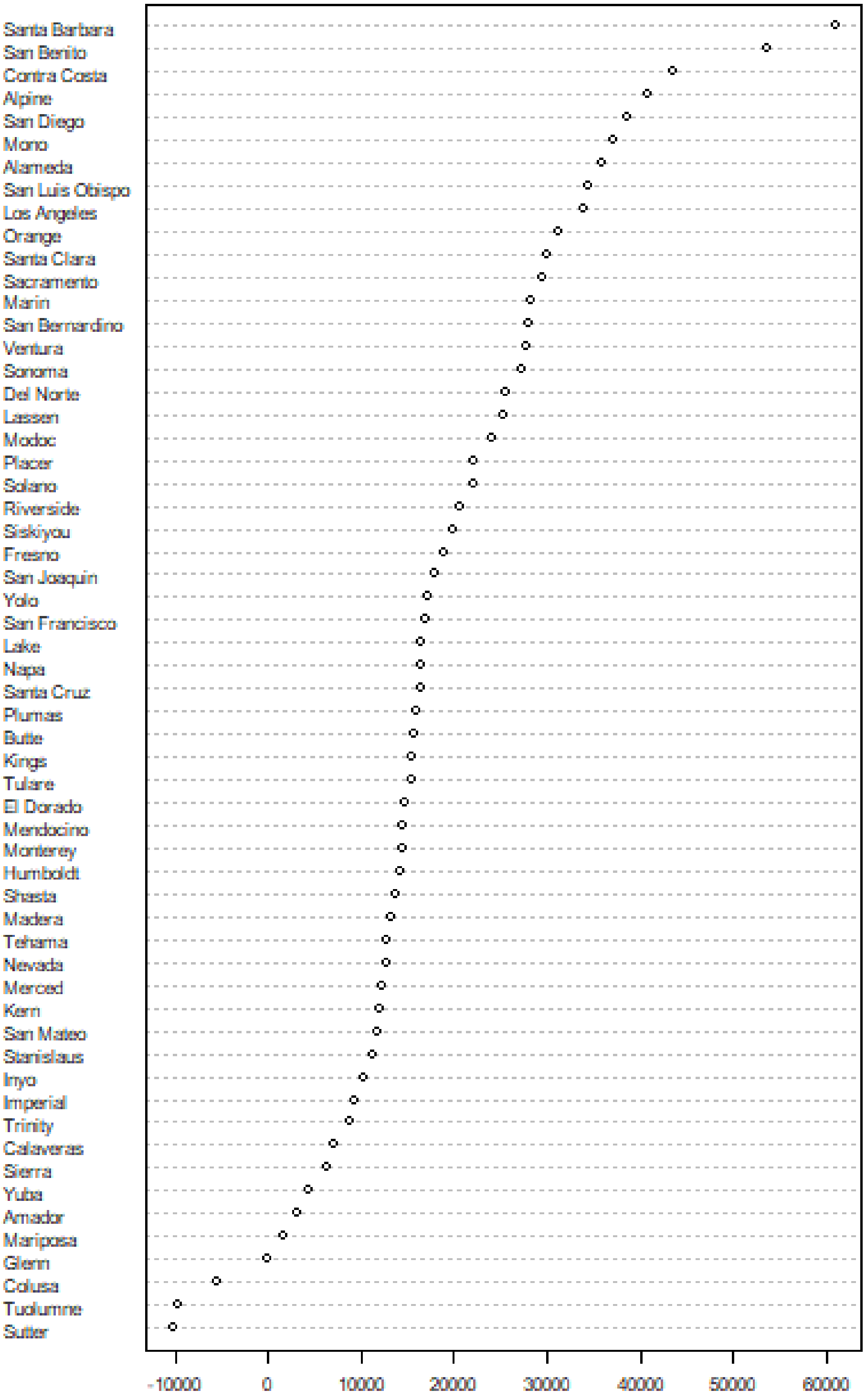
```
regfun <- function(x)  {
  dat <- hd2[hd2$NAME == x, ]
  m <- glm(houseValue~income+houseAge+roomhead+bedroomhead+population, data=dat)
  coefficients(m)
}
```

And now run this for all counties using sapply:

```
countynames <- unique(hd2$NAME)
res <- sapply(countynames, regfun)
```

Plot of a single coefficient

```
dotchart(sort(res["income", ]), cex=0.65)
```


Santa Barbara
San Benito
Contra Costa
Alpine
San Diego
Mono
Alameda
San Luis Obispo
Los Angeles
Orange
Santa Clara
Sacramento
Marin
San Bernardino
Ventura
Sonoma
Del Norte
Lassen
Modoc
Placer
Solano
Riverside
Siskiyou
Fresno
San Joaquin
Yolo
San Francisco
Lake
Napa
Santa Cruz
Plumas
Butte
Kings
Tulare
El Dorado
Mendocino
Monterey
Humboldt
Shasta
Madera
Tehama
Nevada
Merced
Kern
San Mateo
Stanislaus
Inyo
Imperial
Trinity
Calaveras
Sierra
Yuba
Amador
Mariposa
Glenn
Colusa
Tuolumne
Sutter
-10000
0
10000
20000
30000
40000
50000
60000

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

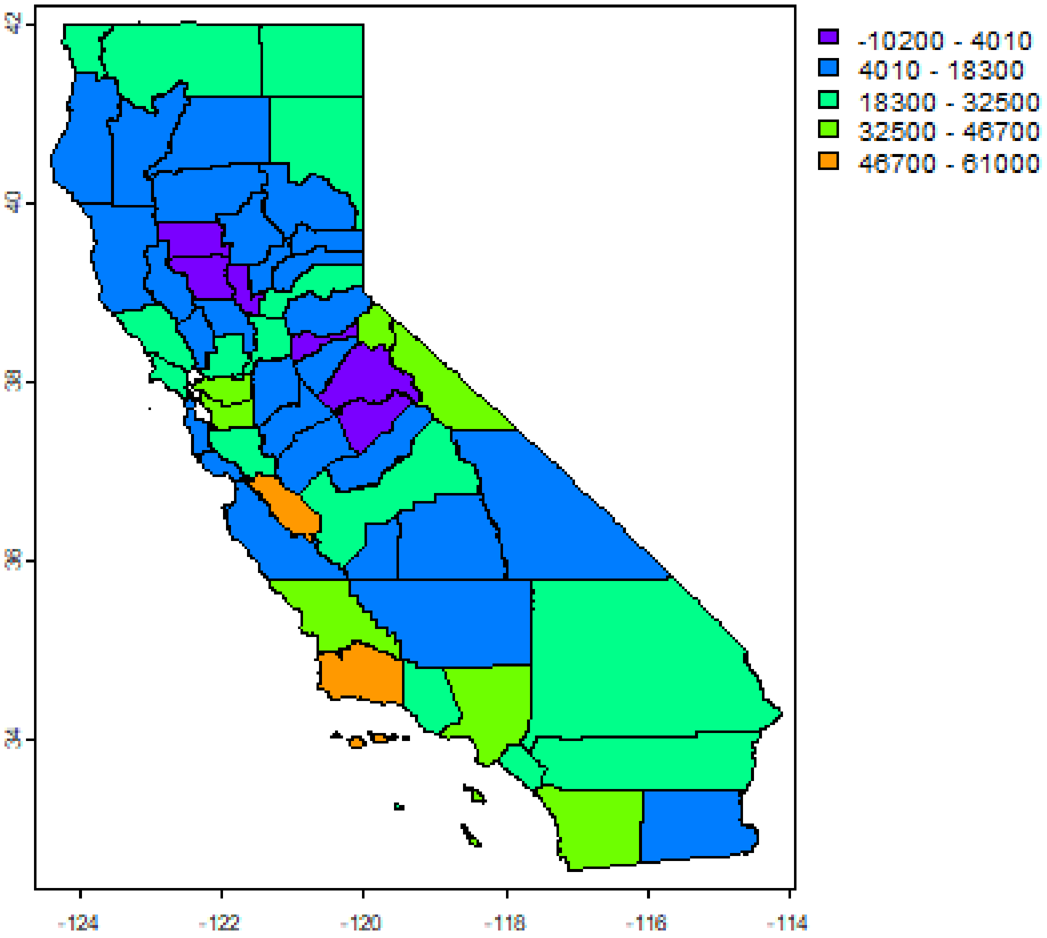
```
resdf <- data.frame(NAME=colnames(res), t(res))
head(resdf)
##                      NAME X.Intercept.    income  houseAge   roomhead
## Alameda           Alameda    -62373.62 35842.330  591.1001 24147.3182
## Contra Costa Contra Costa    -61759.84 43668.442  465.8897  -356.6085
## Alpine             Alpine    -77605.93 40850.588 5595.4113         NA
## Amador             Amador    120480.71  3234.519 -771.5857 37997.0069
## Butte               Butte     50935.36 15577.745 -380.5824  9078.9315
## Calaveras       Calaveras     91364.72  7126.668 -929.4065 16843.3456
##              bedroomhead population
## Alameda        129814.33  8.0570859
## Contra Costa   150662.89  0.8869663
## Alpine                NA         NA
## Amador        -194176.65  0.9971630
## Butte          -32272.68  5.7707597
## Calaveras      -78749.86  8.8865713
```

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)
## [1] 68  5
dcounties <- aggregate(counties[, "NAME"], "NAME")
dim(dcounties)
## [1] 58  2
```

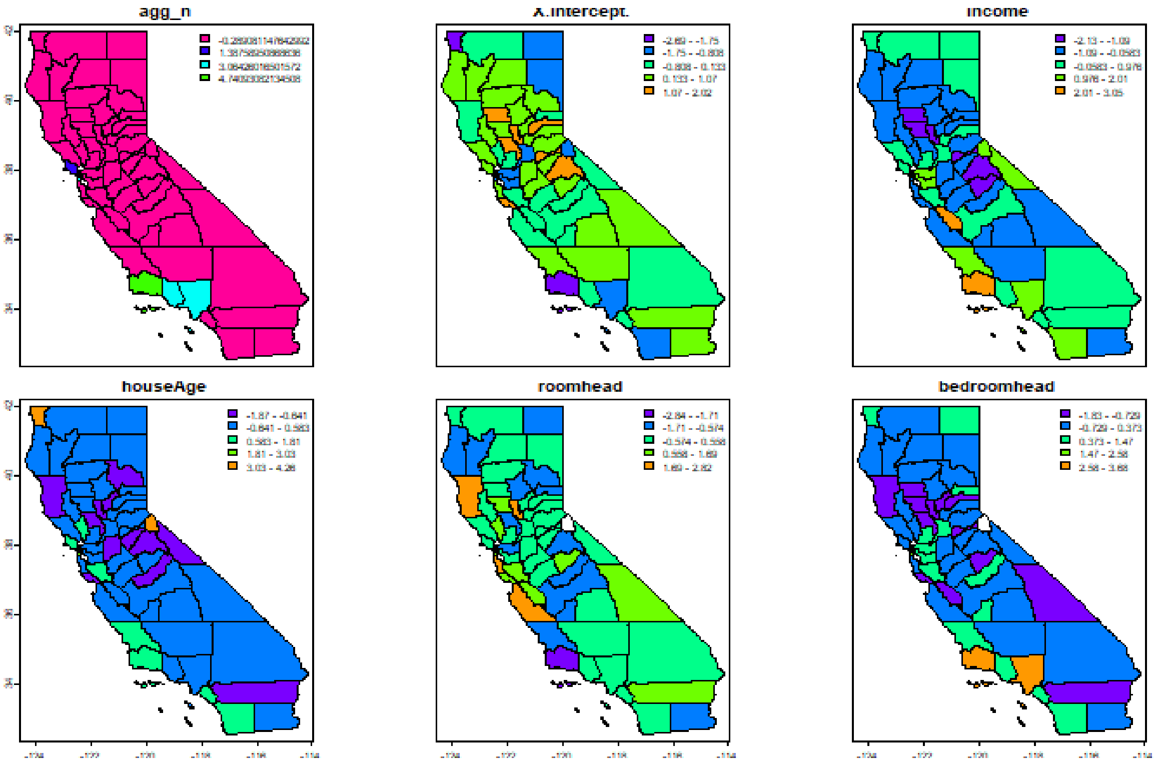
Now we can merge this SpatVector with the data.frame with the regression results.

```
cnres <- merge(dcounties, resdf, by="NAME")
plot(cnres, "income")
```



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

# scale all variables, except the first one (county name)
values(cnres2) <- as.data.frame(scale(as.data.frame(cnres)[,-1]))
plot(cnres2, names(cnres2)[1:6], plg=list(x="topright"), mar=c(1,1,1,1))
```



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

```
lw <- adjacent(cnres2, pairs=FALSE)
autocor(cnres$income, lw)
## [1] 0.1565227
autocor(cnres$houseAge, lw)
## [1] -0.02057022
```

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 <- "+proj=aea +lat_1=34 +lat_2=40.5 +lat_0=0 +lon_0=-120 +x_0=0 +y_0=-4000000␣
↪+datum=WGS84 +units=m"
countiesTA <- project(counties, TA)
```

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

```
r <- rast(countiesTA)
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 <- project(hvect, TA)
crds <- geom(housesTA)[, c("x", "y")]
```

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.

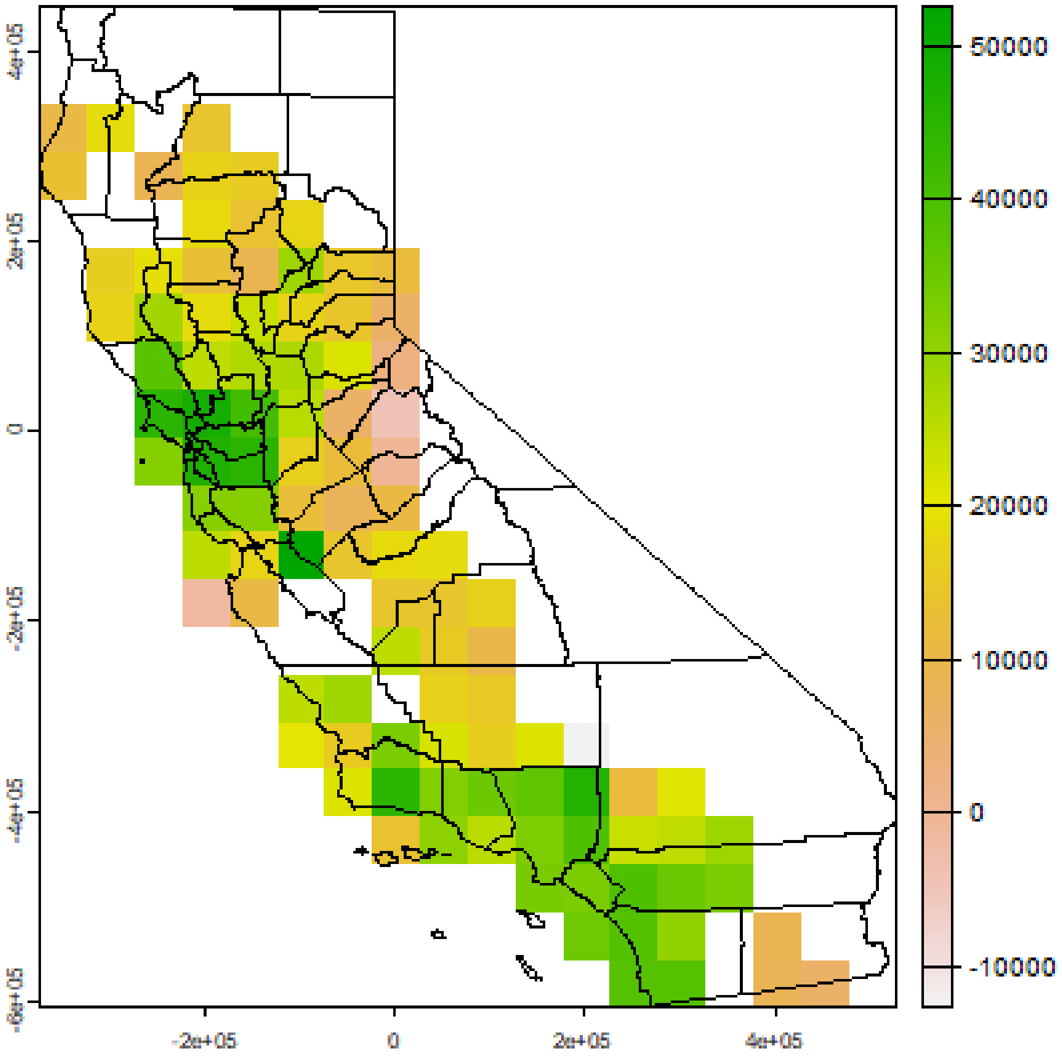
```
res <- list()
for (i in 1:nrow(xy)) {
    d <- sqrt((xy[i,1]-crds[,1])^2 + (xy[i,2]-crds[,2])^2)
    j <- which(d < 50000)
    if (length(j) > 49) {
        d <- hd[j,]
        res[[i]] <- regfun2(d)
    } else {
        res[[i]] <- NA
    }
}
```

For each cell get the income coefficient:

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

Use these values in a SpatRaster

```
rinc <- setValues(r, inc)
plot(rinc)
plot(countiesTA, add=T)
```



```
autocor(rinc)
##    lyr.1
## 1.326968
```

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?*

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 SpatRaster with the correct extent

```
r <- rast(countiesTA)
```

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)
```

Extract the coordinates that are not `NA`.

```
fitpoints <- crds(ca)
```

Now specify the model

```
gwr.model <- ______
```

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

To extract the SpatialPointsDataFrame:

```
sp <- gwr.model$SDF
v <- vect(sp)
v
```

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

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

Question 2: *spgwr shows a remarkable startup message. What is that about?*

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

CHAPTER

SEVEN

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 (!require("rspat")) remotes::install_github("rspatial/rspat")
## Loading required package: rspat
## Loading required package: terra
## terra 1.7.62

library(rspat)
h <- spat_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).

```
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"
```

(continues on next page)

(continued from previous page)

```
## [21] "AmericanIn" "Asian"      "Hispanic"   "PopInHouse" "nHousehold"
## [26] "Families"   "householdS" "familySize" "County"
```

These are the variables we have:

variabl e	description
nhousin gUn	number of housing units
recHous es	number of houses for recreational use
nMobile Hom	number of mobile homes
nBadPlu mbi	number of houses with incomplete plumbing
nBadKit che	number of houses with incomplete kitchens
Populat ion	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)
America nIn	number of persons identifying themselves American Indian (only)
Asian	number of persons identifying themselves as American Indian (only)
Hispani c	number of persons identifying themselves as hispanic (only)
PopInHo use	number of persons living in households
nHouseh old	number of households
Familie s	number of families
houseVa lue	value of the house
yearBui lt	year house was built
nRooms	median number of rooms per house
nBedroo ms	median number of bedrooms per house
medHHin c	median household income
MedianA ge	median age of population
househo ldS	median household size
familyS ize	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.

```
# using a tiny buffer to get a cleaner aggregation
hb <- buffer(h, 1)
values(hb) <- values(h)
hha <- aggregate(hb, "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 <- as.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. We should weight it by the number of houses (households) in each tract.

```
d2 <- as.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:

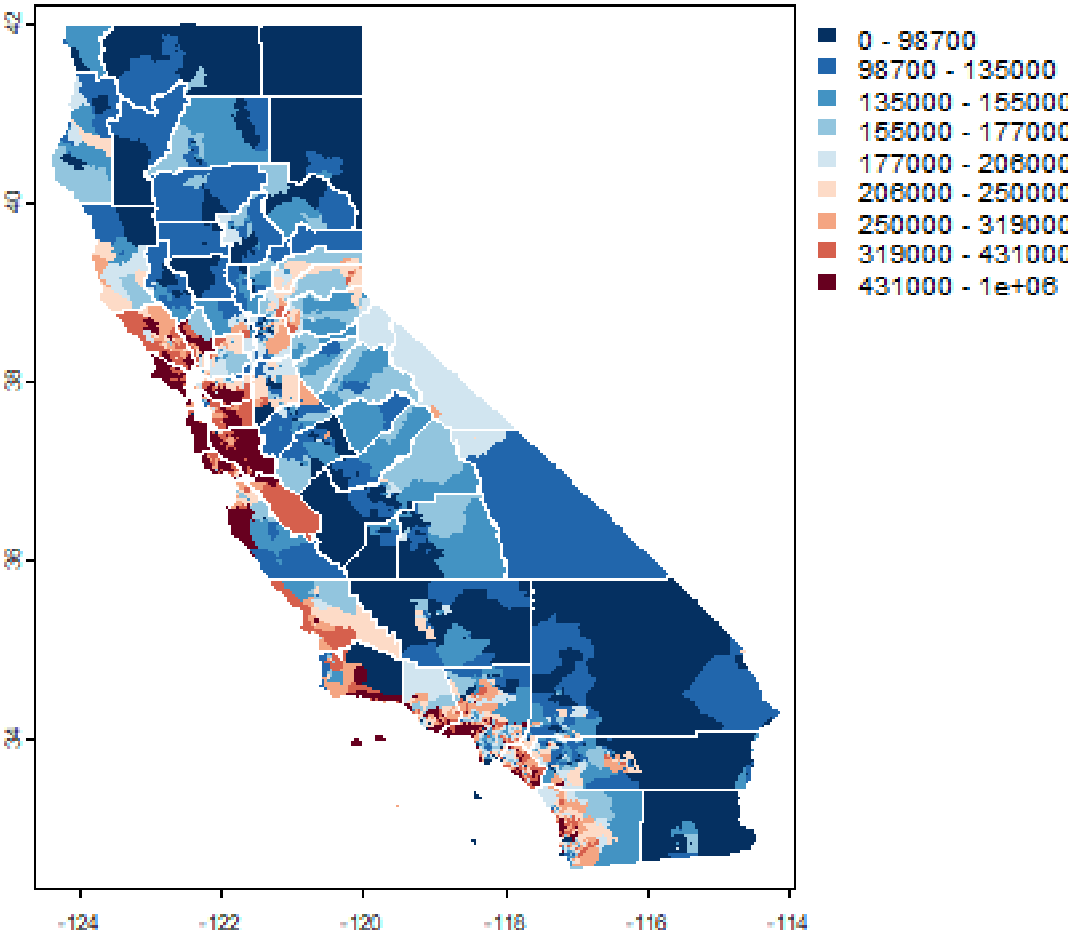
```
d12 <- merge(d1a, d2a, by='County')
```

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

```
hh <- merge(hha[, "County"], d12, by='County')
```

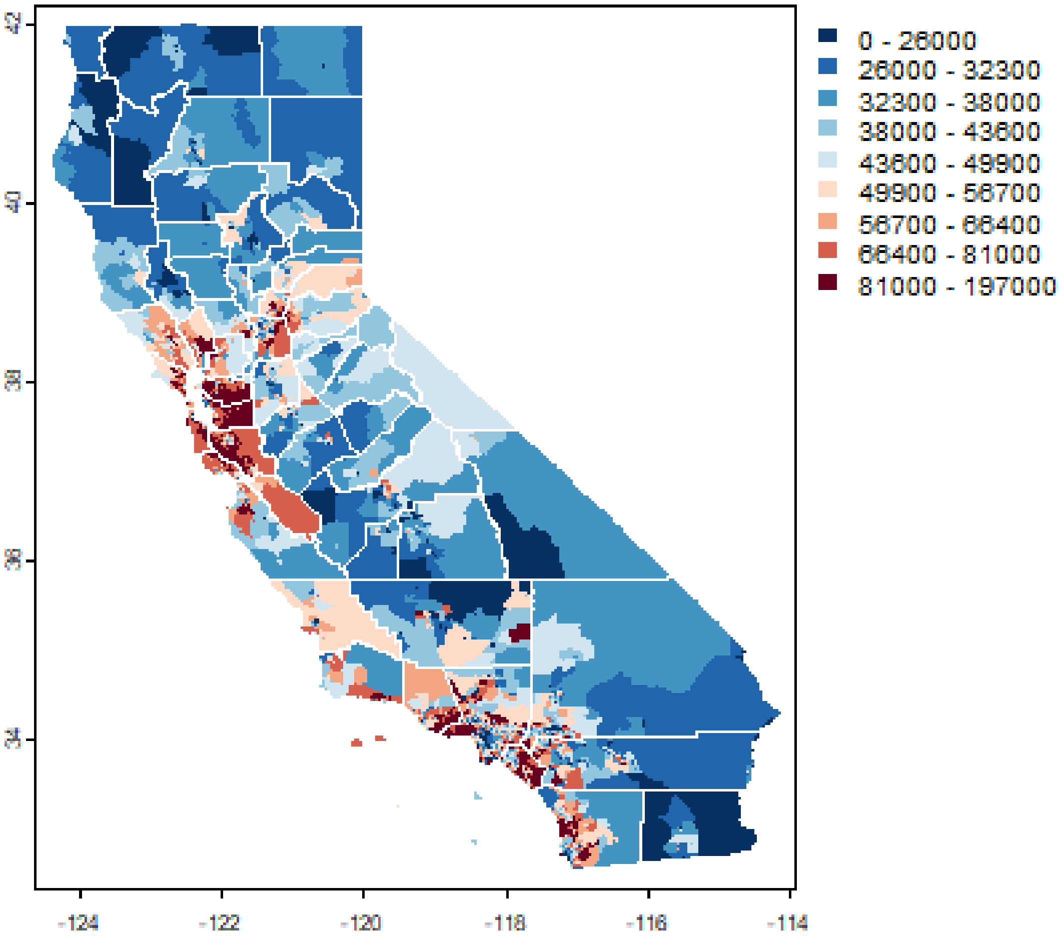
Let’s make some maps, at the orignal Census tract level. First the house value, using a legend with 10 intervals.

```
library(RColorBrewer)
grps <- 10
brks <- quantile(h$houseValue, 0:(grps-1)/(grps-1), na.rm=TRUE)
plot(h, "houseValue", breaks=brks, col=rev(brewer.pal(grps, "RdBu")), border=NA)
lines(hh, col="white")
```



A map of the median household income.

```
brks <- quantile(h$medHHinc, 0:(grps-1)/(grps-1), na.rm=TRUE)
plot(h, "medHHinc", breaks=brks, col=rev(brewer.pal(grps, "RdBu")), border=NA)
lines(hh, col="white")
```



7.3 Basic OLS model

I 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=as.data.frame(hh))
summary(m1)
##
## Call:
## lm(formula = f1, data = as.data.frame(hh))
```

(continues on next page)

(continued from previous page)

```
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -222541  -67489   -6128   60509  217655
##
## 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
```

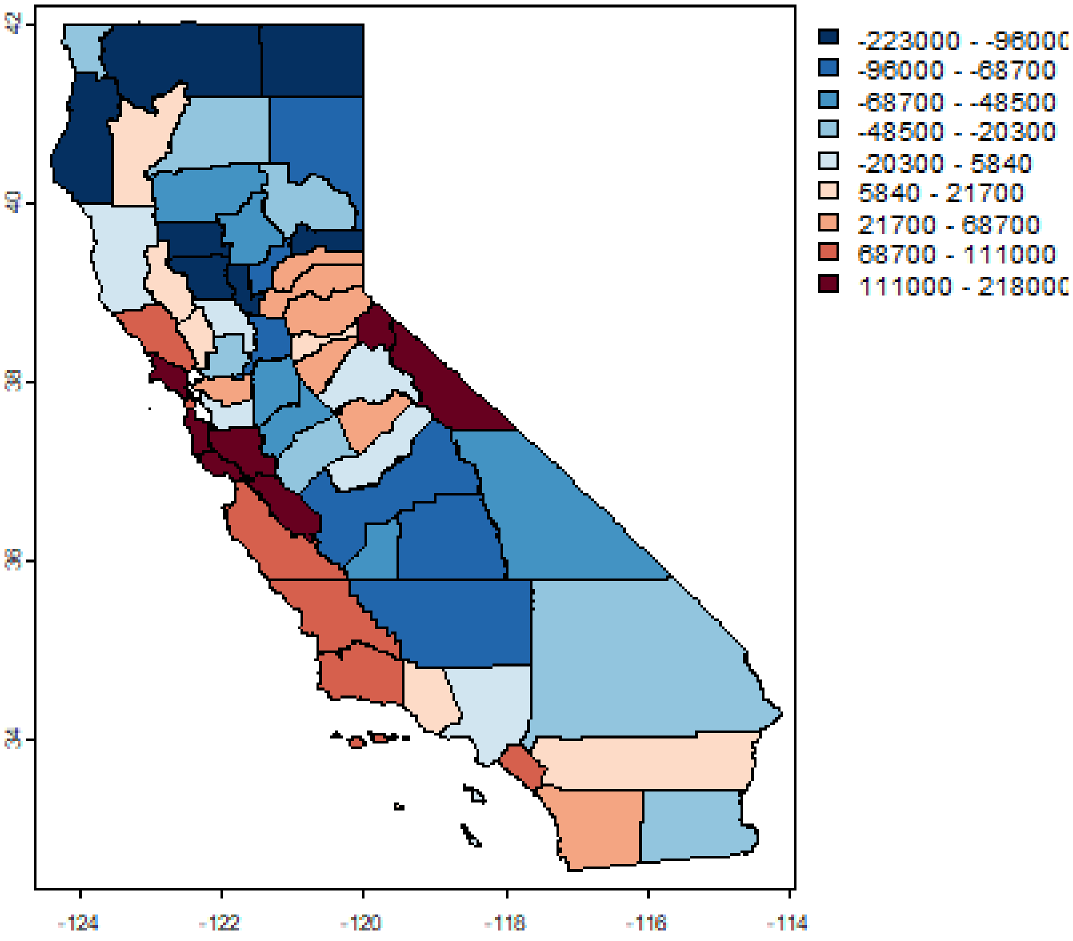
So, according to this simple model, "age" is highly significant. The older a house, the more expensive. You pay 1,269,475 dollars more for a house that is 100 years old than a for new house! While the p-value for the number of bedrooms is not impressive, but every bedroom adds about 200,000 dollars to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

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

(the answer may surprise you),

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)
plot(hh, "residuals", breaks=brks, col=rev(brewer.pal(grps, "RdBu")))
```



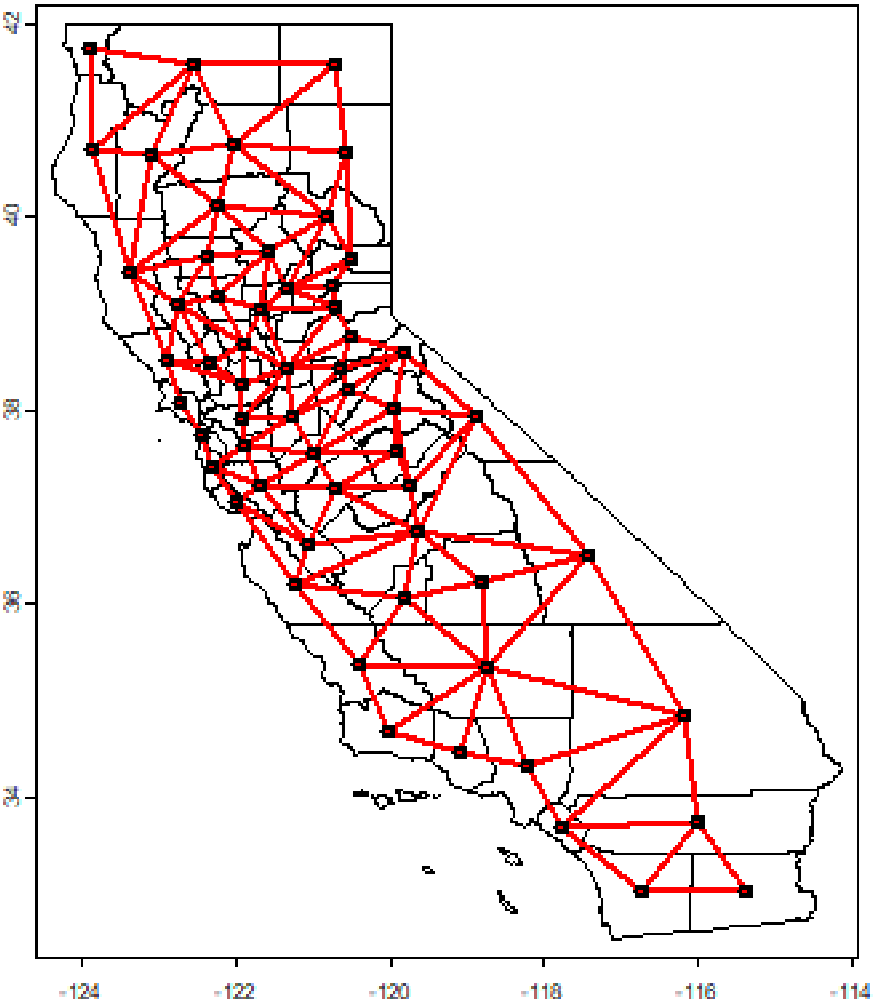
What do think? Is this a random pattern? Let's see what Mr. Moran would say. First make a neighborhoods list. I add two links: between San Francisco and Marin County and vice versa (to consider the Golden Gate bridge).

```
library(spdep)

sfhh <- sf::st_as_sf(hh)
nb <- poly2nb(sfhh, snap=1/120)
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))
```

(continues on next page)

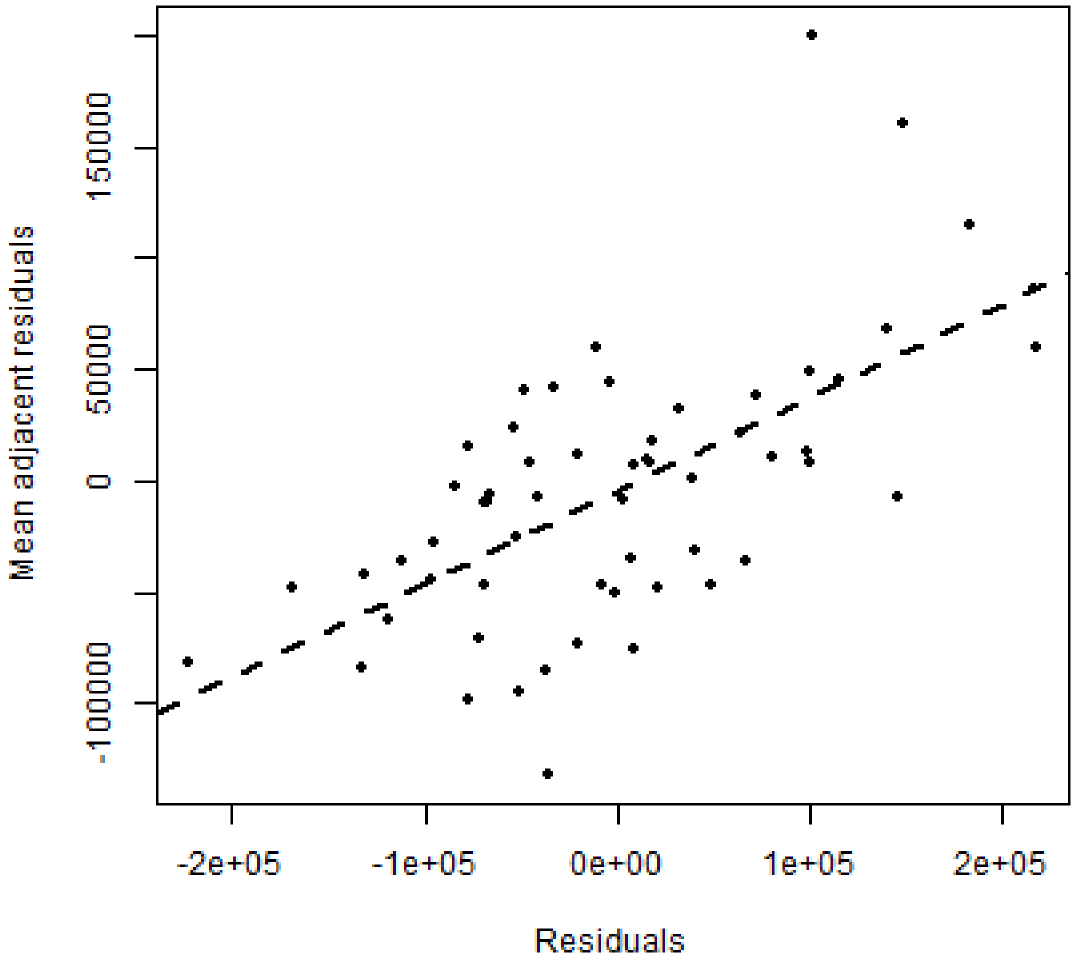
(continued from previous page)

```
plot(hh)
plot(nb, crds(centroids(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$residuals, resnb, xlab="Residuals", ylab="Mean adjacent residuals", pch=20)
abline(lm(resnb ~ hh$residuals), lwd=2, lty=2)
```



The residualso appear to be autocorrelated. A formal test:

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

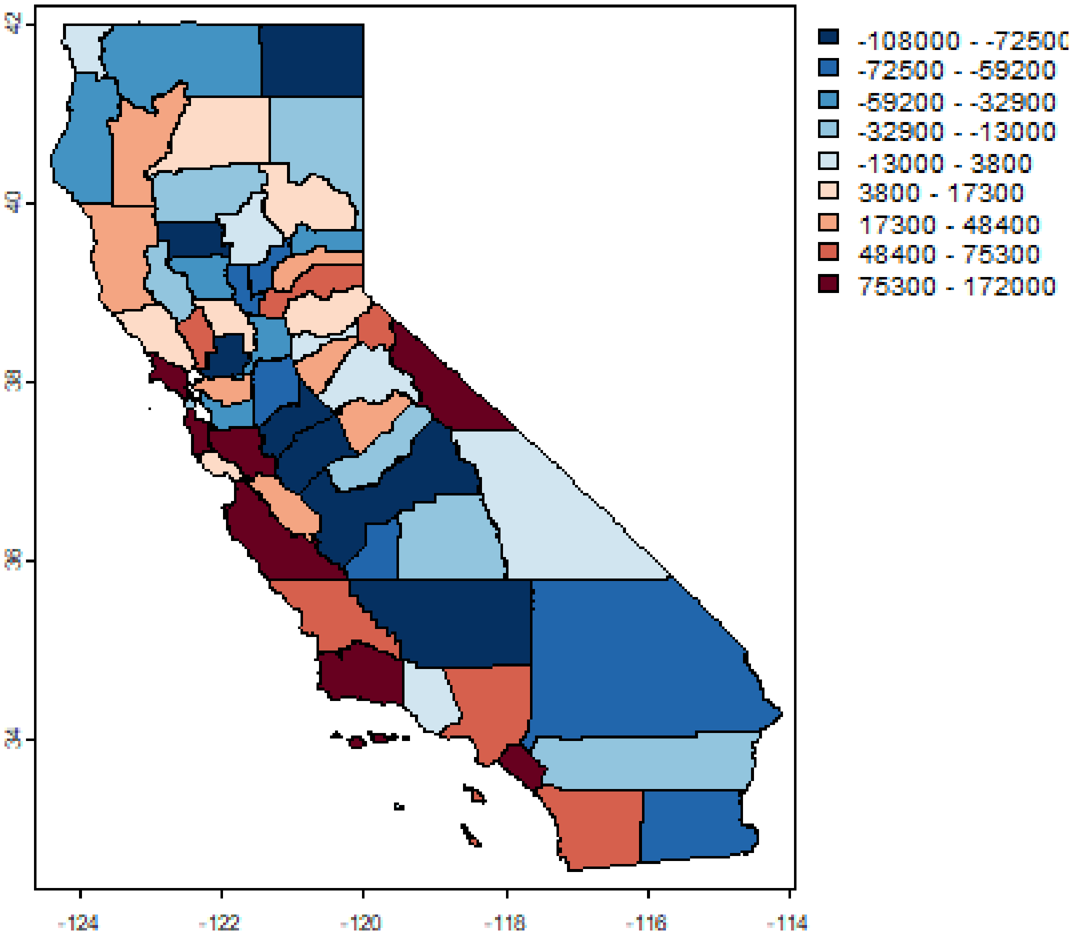
Clearly, there is spatial autocorrelation. Our model cannot be trusted. so let's try SAR models.

7.4 Spatial lag model

Here I show a how to do spatial regression with a spatial lag model (lagsarlm), using the spatialreg package.

```
library(spatialreg )
```

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

summary(m1s)
##
## Call:lagsarlm(formula = f1, data = as.data.frame(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: NA (not available for weighted model), (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 = 511, p-value = 0.489
## alternative hypothesis: greater

brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)
plot(hh, "residuals", breaks=brks, col=rev(brewer.pal(grps, "RdBu")))
```



7.5 Spatial error model

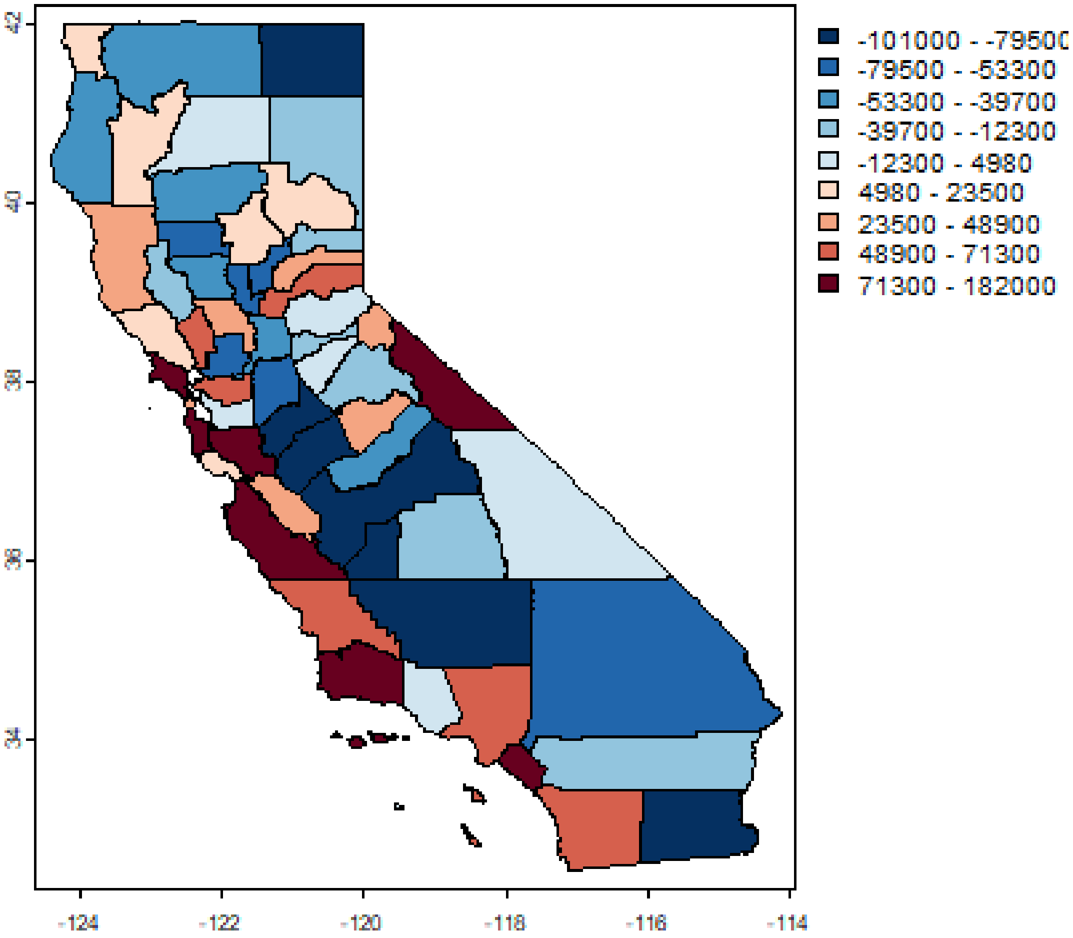
And now with a "Spatial error" (or spatial moving average) models (errorsarlm). Note the use of the lw argument.

```
m1e <- errorsarlm(f1, data=as.data.frame(hh), lw, tol.solve=1.0e-30)
summary(m1e)
##
## Call:errorsarlm(formula = f1, data = as.data.frame(hh), listw = lw,
##     tol.solve = 1e-30)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -100640.7  -47783.1   -2364.5   44180.6  181876.5
##
```

(continues on next page)

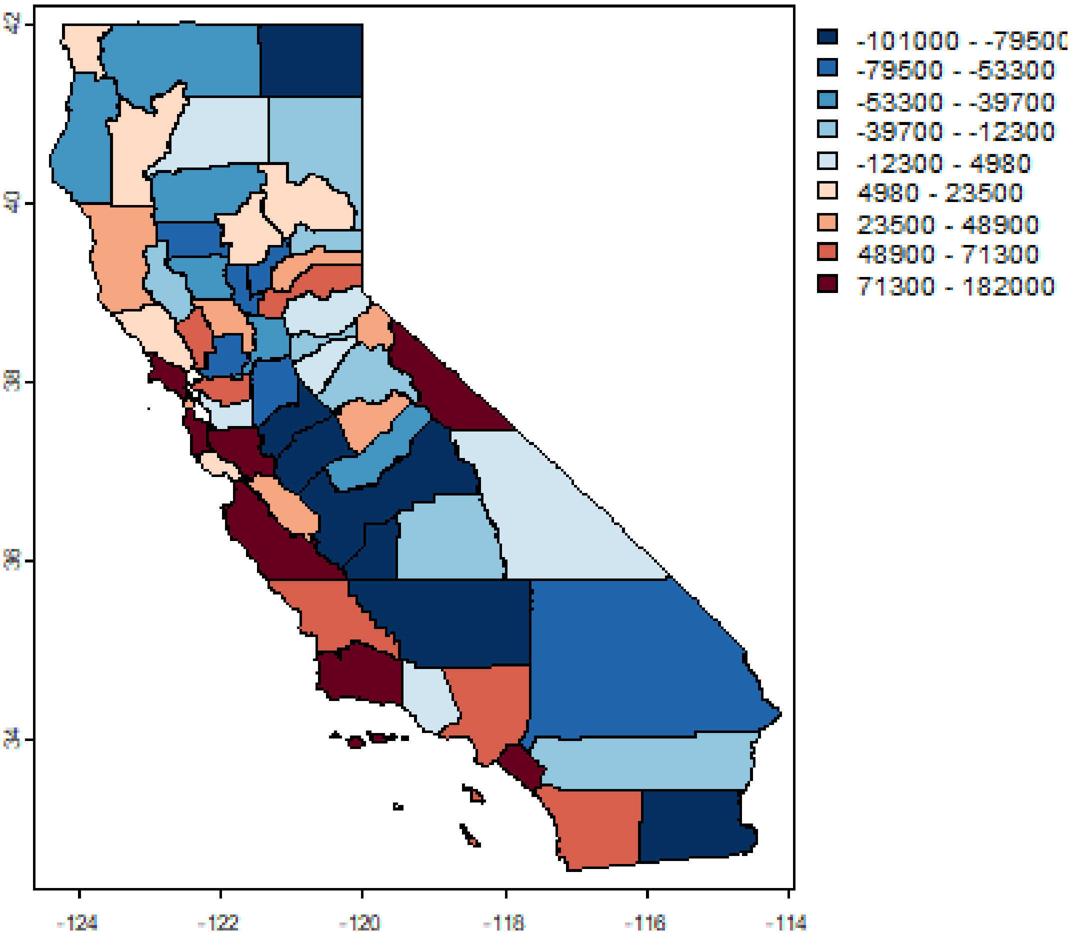
(continued from previous page)

```
## 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    117864.5    51564.7  2.2858  0.02227
##
## Lambda: 0.82151, LR test value: 29.781, p-value: 4.8373e-08
## Asymptotic standard error: 0.071111
##     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 = 768, p-value = 0.232
## alternative hypothesis: greater

brks <- quantile(hh$residuals, 0:(grps-1)/(grps-1), na.rm=TRUE)
plot(hh, "residuals", breaks=brks, col=rev(brewer.pal(grps, "RdBu")))
```



Are the residuals 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)
plot(hh, "residuals", breaks=brks, col=rev(brewer.pal(grps, "RdBu")))
```



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?

b) Which new variables could you propose to create from the variables in the list.

c) Which other variables could you add, created from the geometries/location (perhaps other geographic data).

d) add a lot of variables and use stepAIC to select an 'optimal' OLS model

e) check for spatial autocorrelation in the residuals of that model

CHAPTER

EIGHT

POINT PATTERN ANALYSIS

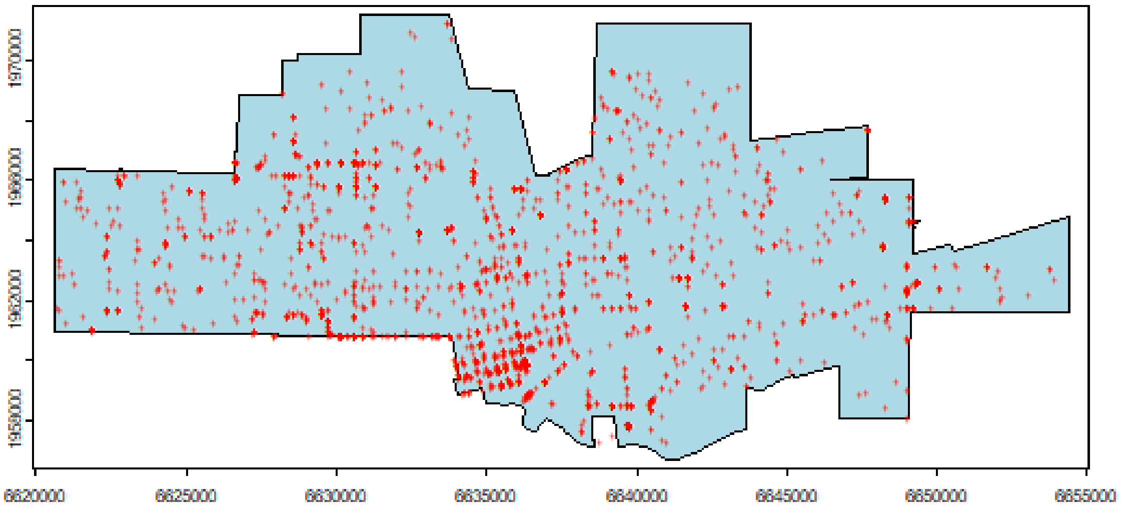
8.1 Introduction

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

```
if (!require("rspat")) remotes::install_github("rspatial/rspat")
## Loading required package: rspat
## Loading required package: terra
## terra 1.7.62
library(rspat)
city <- spat_data("city")
crime <- spat_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
##
##                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 <- crds(crime)
dim(xy)
## [1] 2661    2
xy <- unique(xy)
dim(xy)
## [1] 1208    2
head(xy)
##            x       y
## [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))
```

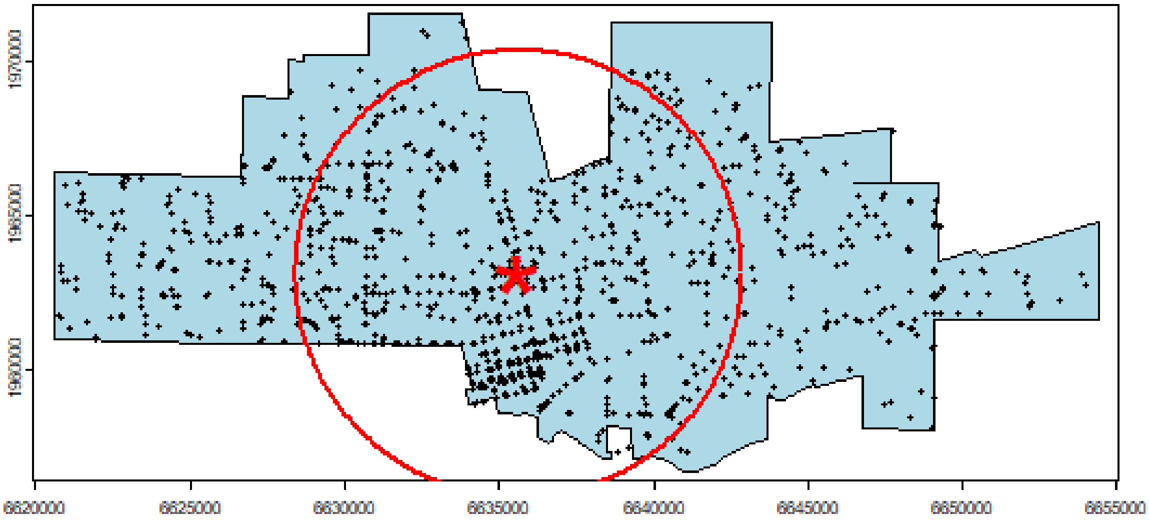
Plot the data to see what we've got. I add a summary circle (as in Fig 5.2) by dividing the circle in 360 points and compute bearing in radians. I do not think this is particularly helpful, but it might be in other cases. And it is always fun to figure out how to do tis.

```
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
```

(continues on next page)

(continued from previous page)

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

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

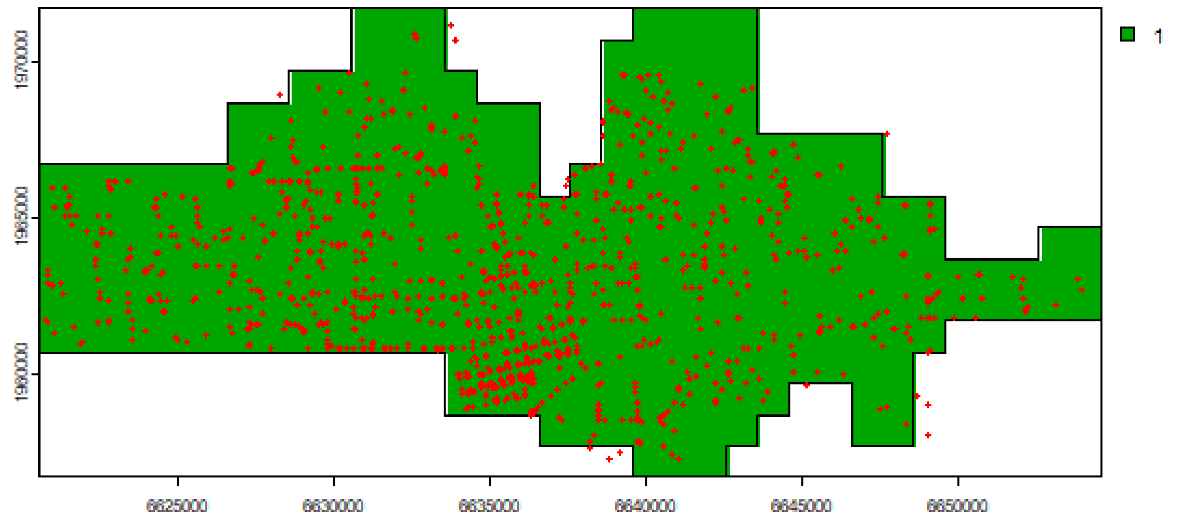
Question 1b:*What is the number of crimes per square km?*

To compute quadrat counts I first create quadrats (a SpatRaster). 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 <- rast(city, res=1000)
```

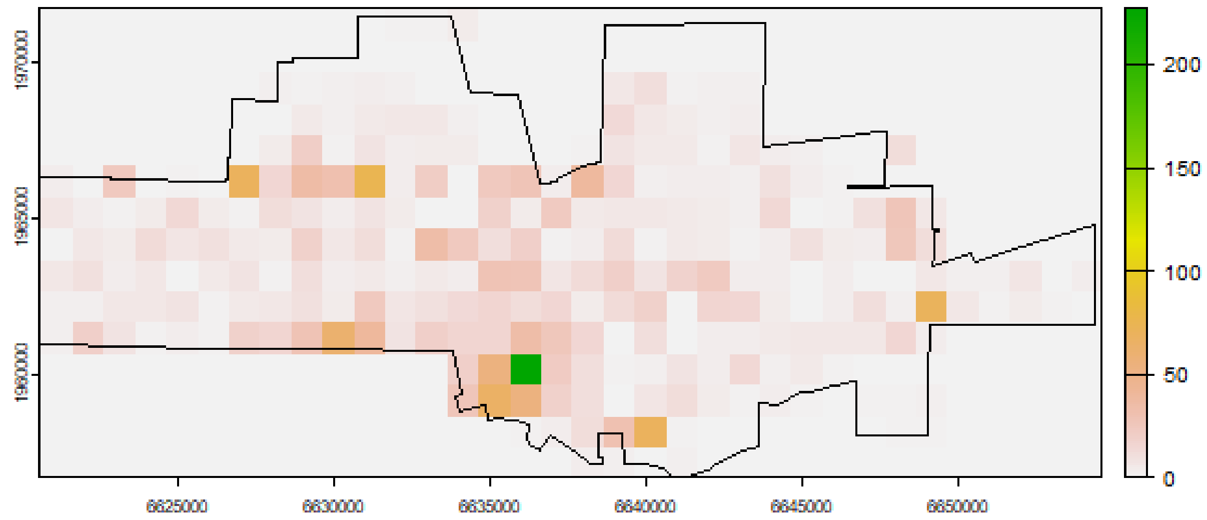
To find the cells that are in the city, and for easy display, I create polygons from the SpatRaster.

```
r <- rasterize(city, r)
plot(r)
quads <- as.polygons(r)
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(crime, r, fun=function(i){length(i)}, background=0)
plot(nc)
plot(city, add=TRUE)
```



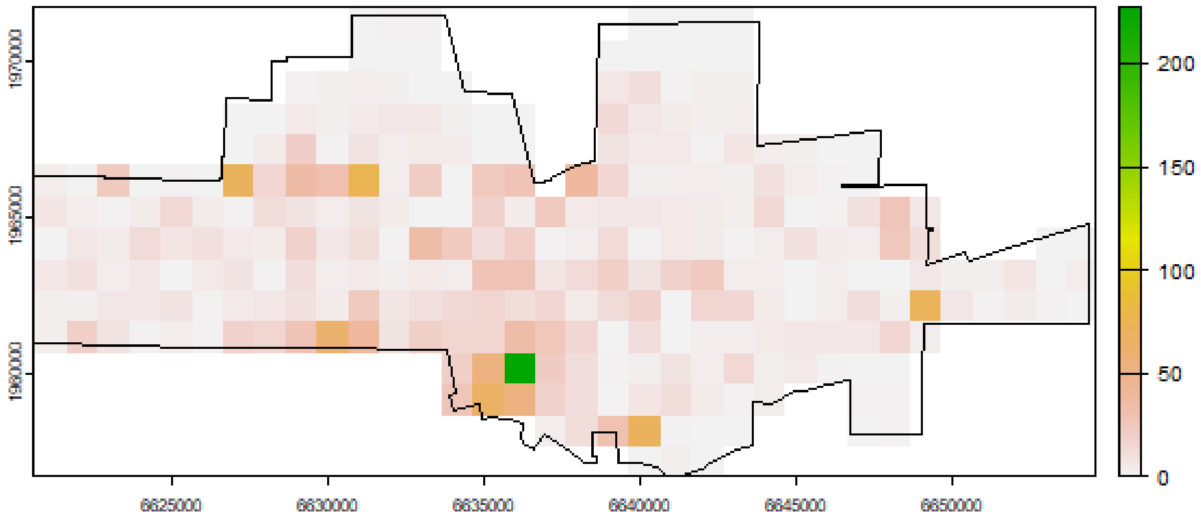
nc has crime counts. As we only have data for the city, the areas outside of the city need to be excluded. We can do that with the mask function (see ?mask).

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

(continues on next page)

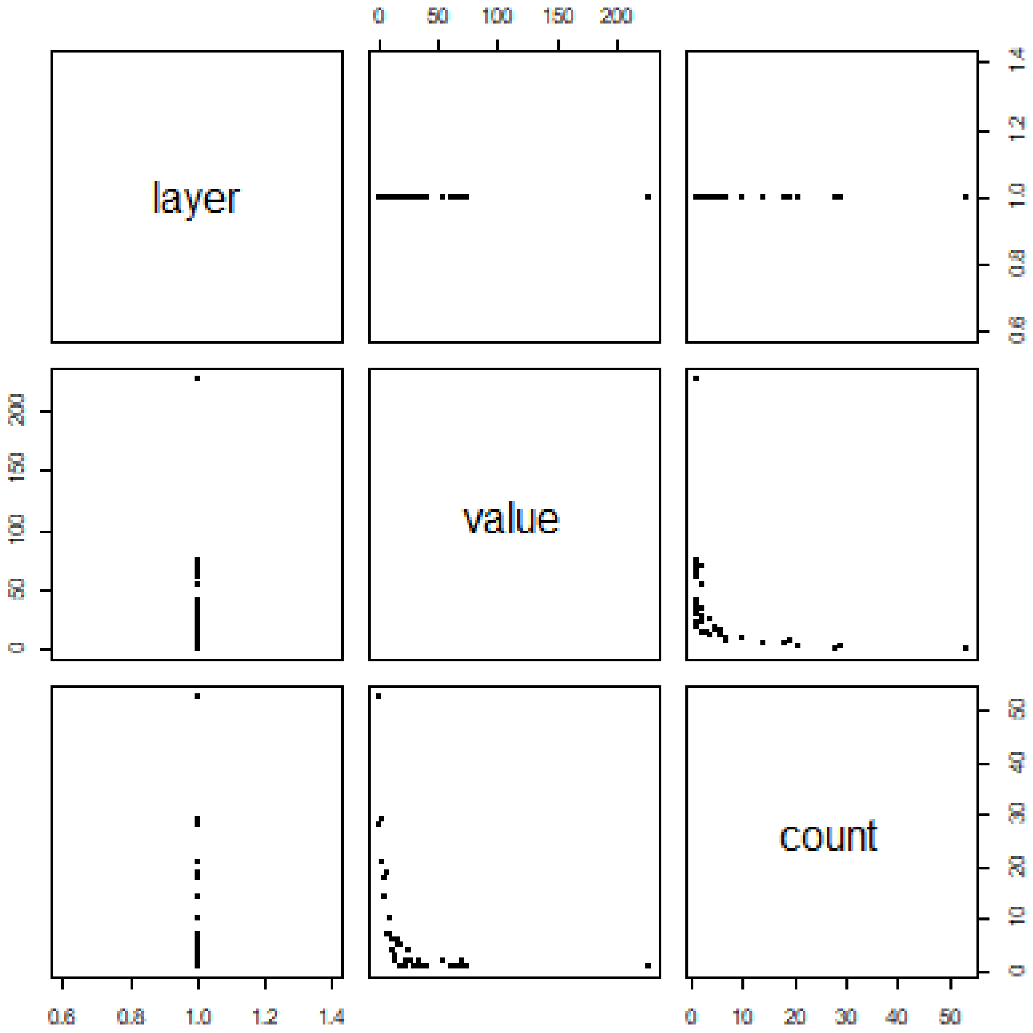
(continued from previous page)

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



Better. Now the frequencies.

```
f <- freq(ncrimes)
head(f)
##   layer value count
## 1     1     0    53
## 2     1     1    28
## 3     1     2    21
## 4     1     3    29
## 5     1     4    18
## 6     1     5    14
plot(f, pch=20)
```



Does this look like a pattern you would have expected? Now compute the 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] 1
```

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
```

(continues on next page)

(continued from previous page)

```
head(ff)
##   K X NA Kmu Kmu2 XKmu2
## 1 1 0 53   0    0     0
## 2 1 1 28   0    0     0
## 3 1 2 21   0    0     0
## 4 1 3 29   0    0     0
## 5 1 4 18   0    0     0
## 6 1 5 14   0    0     0
```

The observed variance s^2 is

```
s2 <- sum(ff$XKmu2) / (sum(ff$X)-1)
s2
## [1] 0
```

And the VMR is

```
VMR <- s2 / mu
VMR
## [1] 0
```

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 `terra::distance`.

```
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.

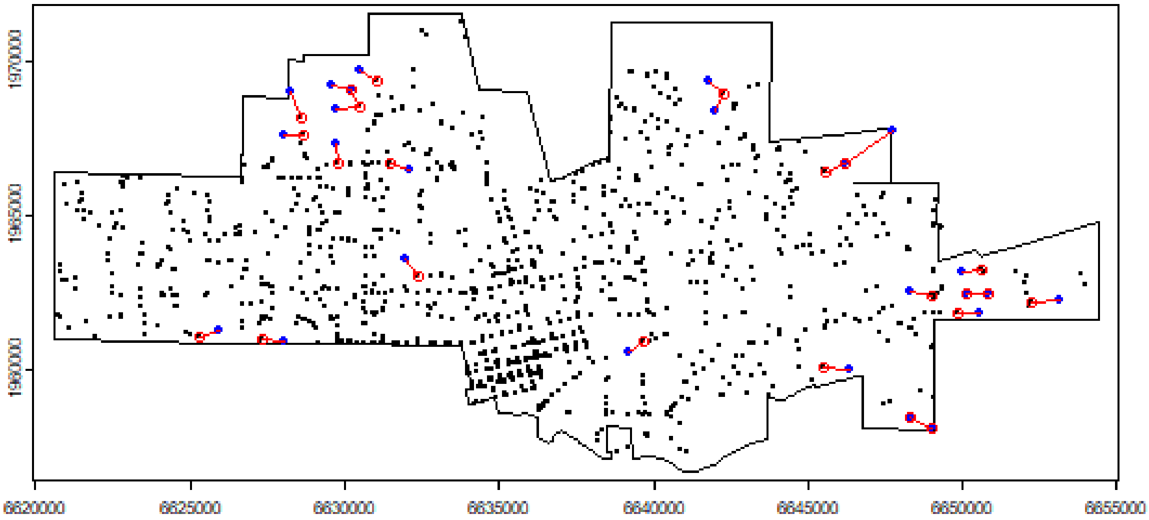
```
mdmin <- mean(dmin)
```

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.

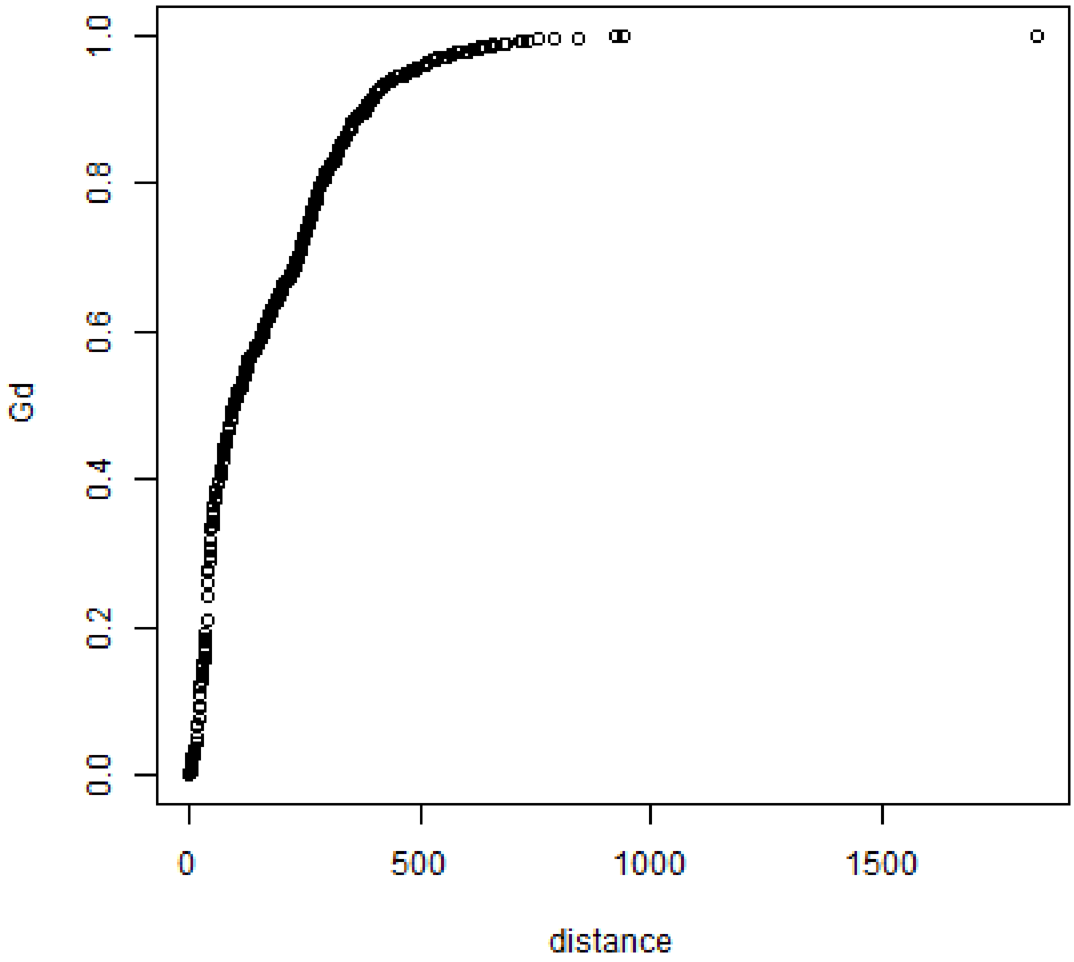
```
plot(city)
points(crime, cex=.1)
ord <- rev(order(dmin))

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')
}
```



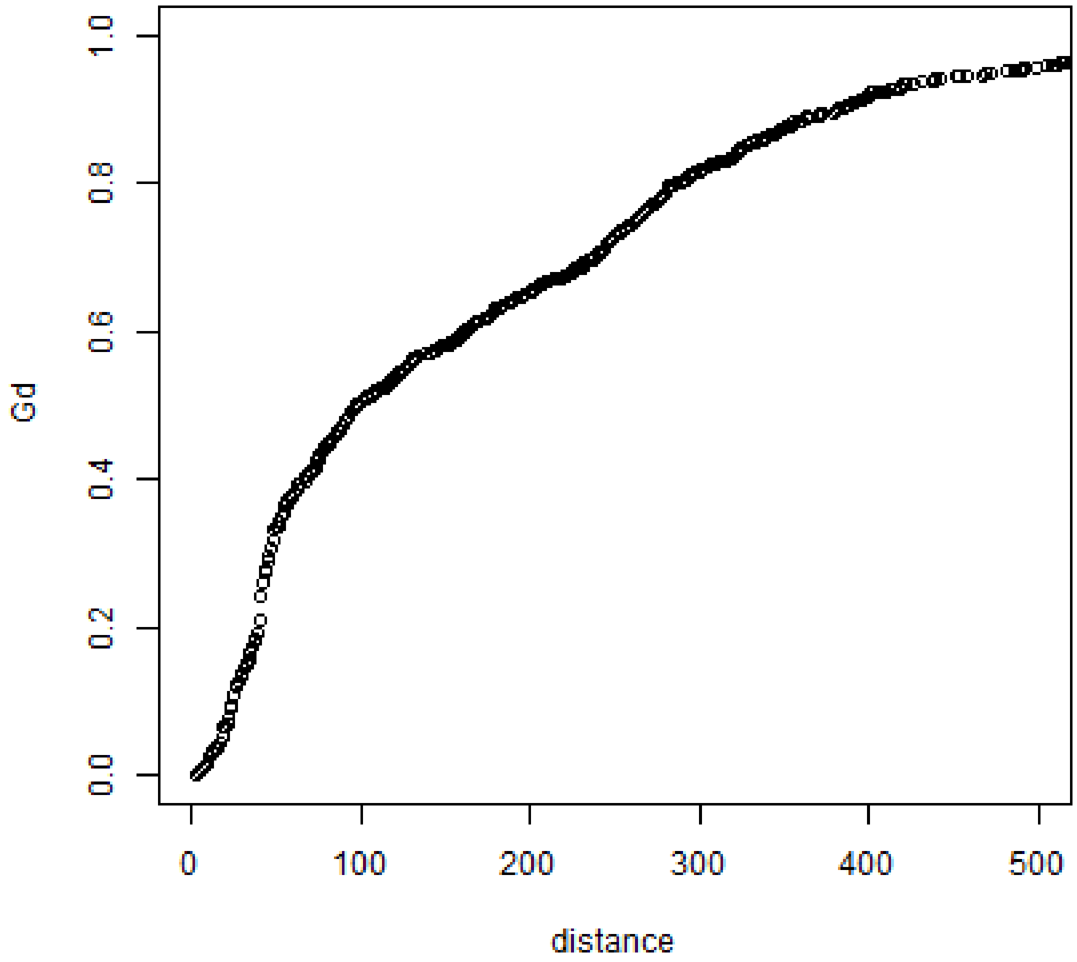
Note that some points, but actually not that many, are used as isolated and as a neighbor to an isolated points.

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))
# 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))
```

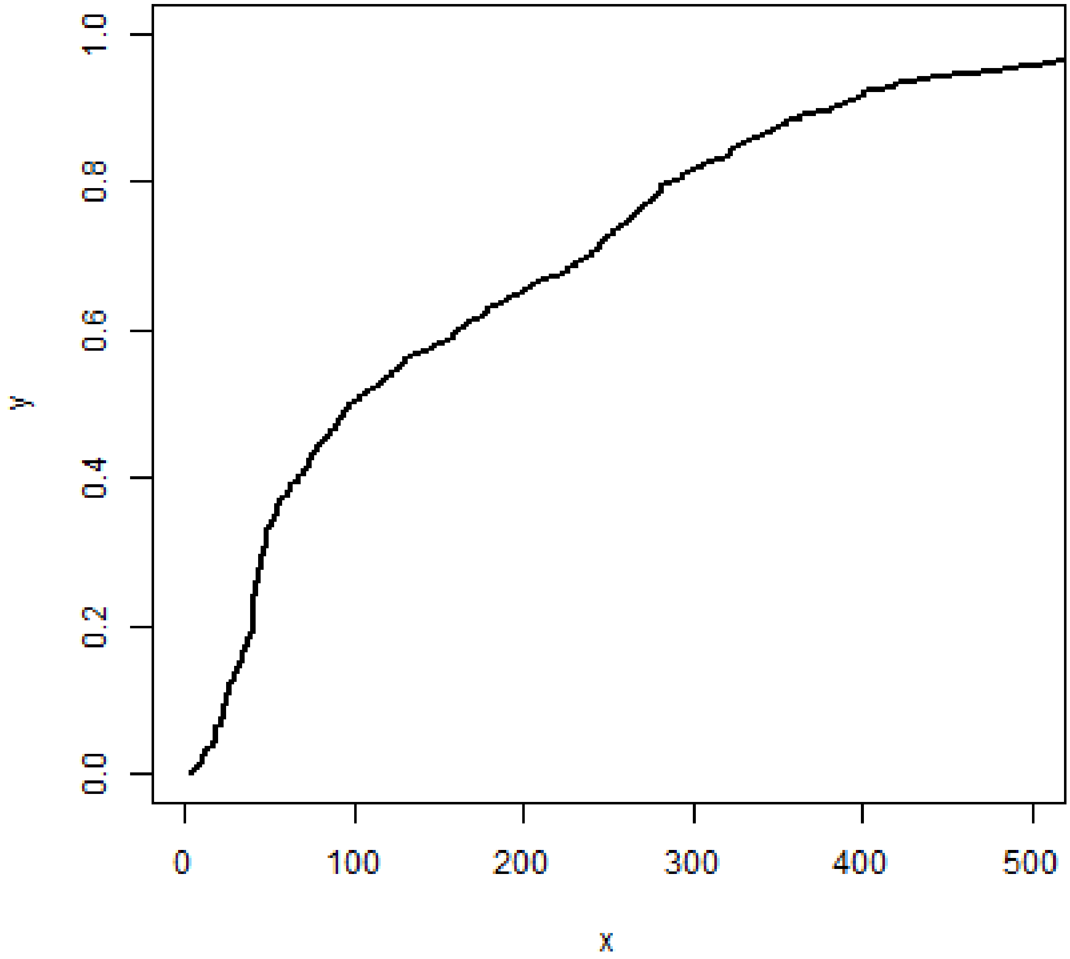


Here is a function to show these values in a more standard way.

```
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.

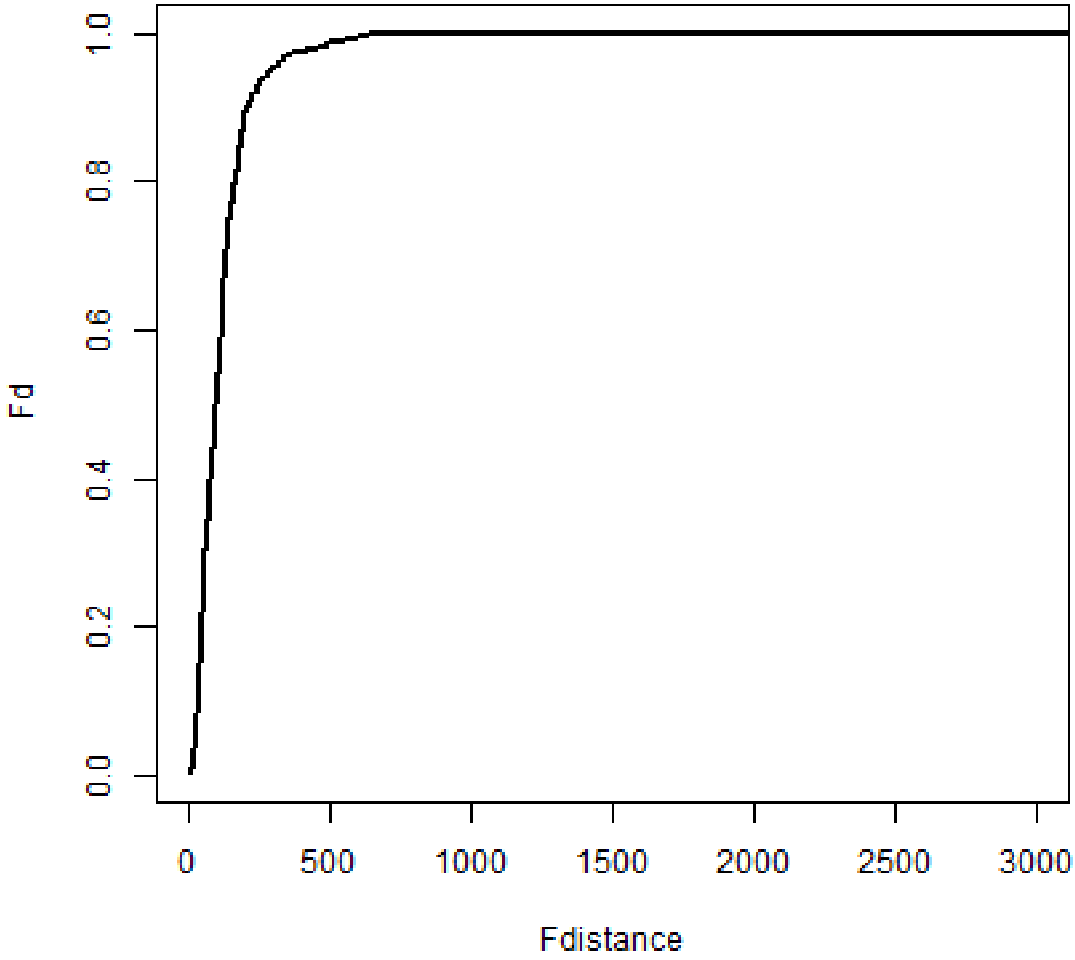
I use the centers of previously defined raster cells to compute the *F* function.

```
c# get the centers of the 'quadrats' (raster cells)
## function (...)  .Primitive("c")
p <- as.points(r)
# compute distance from all crime sites to these cell centers
d2 <- distance(p, crime)
d2 <- as.matrix(d2)
# 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)
```

(continues on next page)

(continued from previous page)

```
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)
```

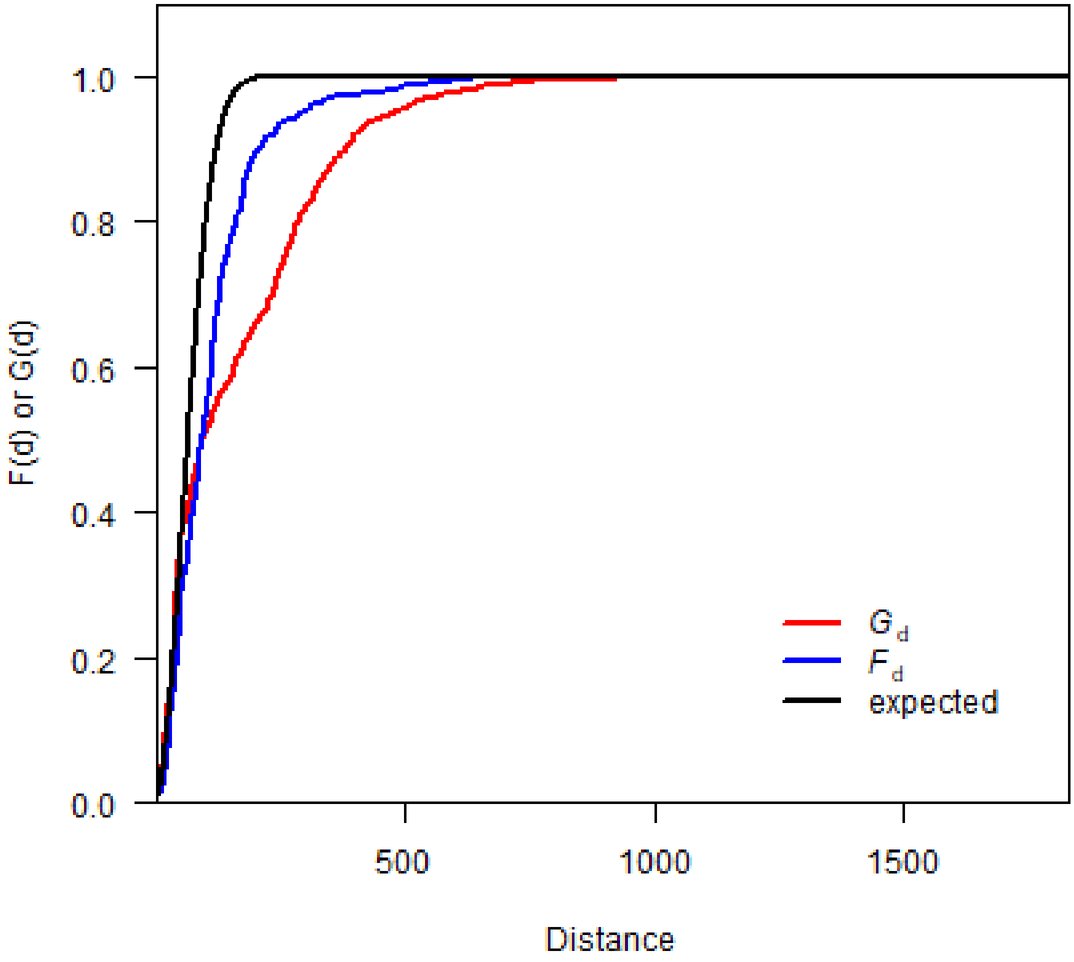
Now, let's combine F and G on one plot.

```
plot(distance, Gd, type='l', lwd=2, col='red', las=1,
    ylab='F(d) or G(d)', xlab='Distance', yaxs="i", xaxs="i", ylim=c(0,1.1))
lines(Fdistance, Fd, lwd=2, col='blue')
lines(0:2000, expected, lwd=2)
```

(continues on next page)

(continued from previous page)

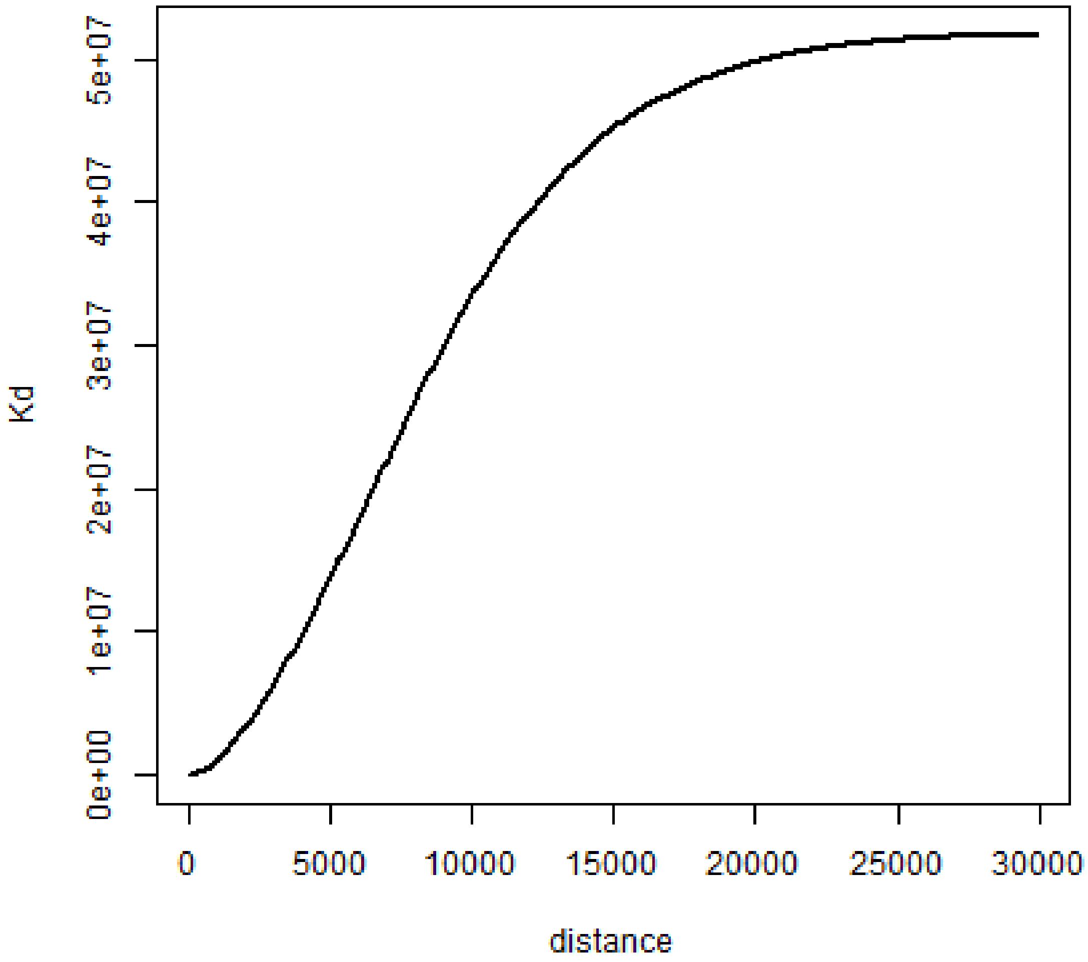
```
legend(1200, .3,
   c(expression(italic("G")["d"]), expression(italic("F")["d"]), 'expected'),
   lty=1, col=c('red', 'blue', 'black'), lwd=2, bty="n")
```



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 SpatVector to an object of class "owin" (observation window) via `sf`

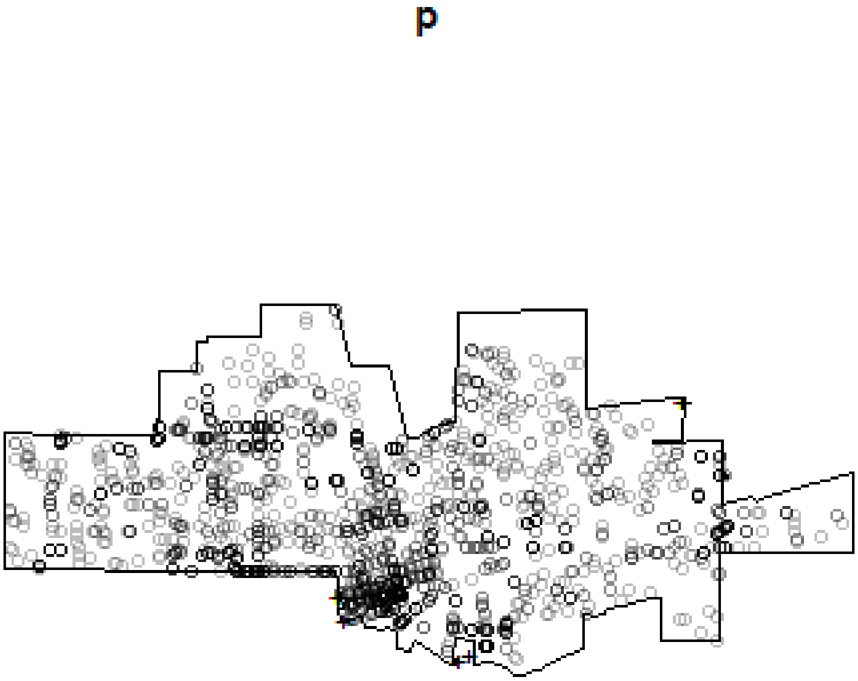
```
cityOwin <- as.owin(sf::st_as_sf(city))
class(cityOwin)
## [1] "owin"
cityOwin
## window: polygonal boundary
## enclosing rectangle: [6620591, 6654380] x [1956729.8, 1971518.9] units
```

Extract coordinates from SpatialPointsDataFrame:

```
pts <- terra::crds(crime)
head(pts)
##            x       y
## [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

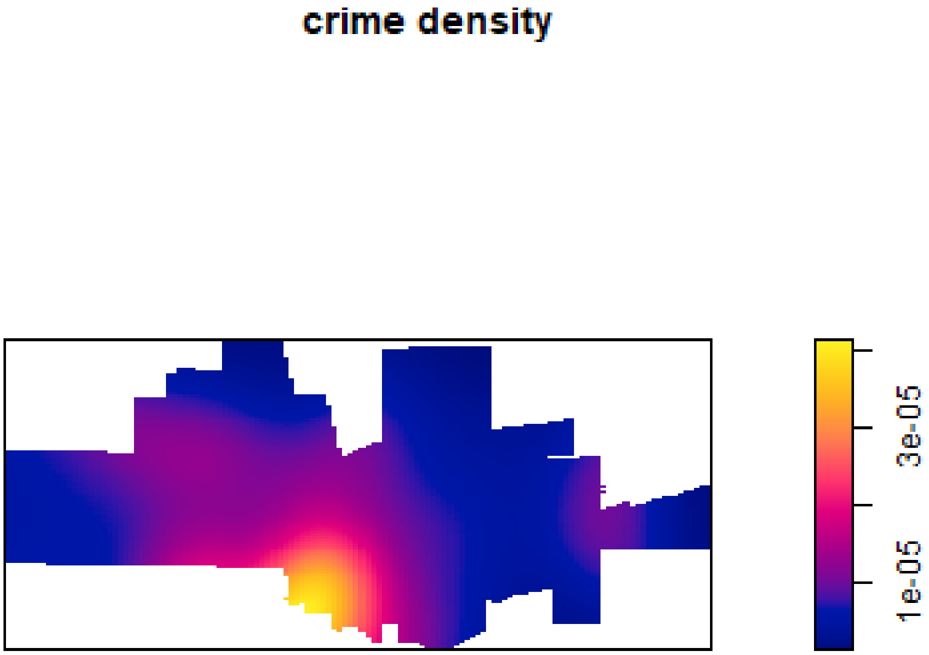
```
p <- ppp(pts[,1], pts[,2], window=cityOwin)
## Warning: 20 points were rejected as lying outside the specified window
## Warning: data contain duplicated points
class(p)
## [1] "ppp"
p
## Planar point pattern: 2641 points
## window: polygonal boundary
## enclosing rectangle: [6620591, 6654380] x [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 terra package functions for that.

```
nrow(pts)
## [1] 2661
r <- rast(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
```

(continues on next page)

(continued from previous page)

```
##  $ 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
##   ..$ 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"
##  - attr(*, "kerdata")=List of 5
##   ..$ sigma   : num 1849
##   ..$ varcov  : NULL
##   ..$ cutoff  : num 14789
##   ..$ warnings: NULL
##   ..$ 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 <- spat_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$area <- expanse(census)
census$area <- census$area/27878400
census$dens <- census$POP2000 / census$area
```

Now to get the centroids of the census blocks.

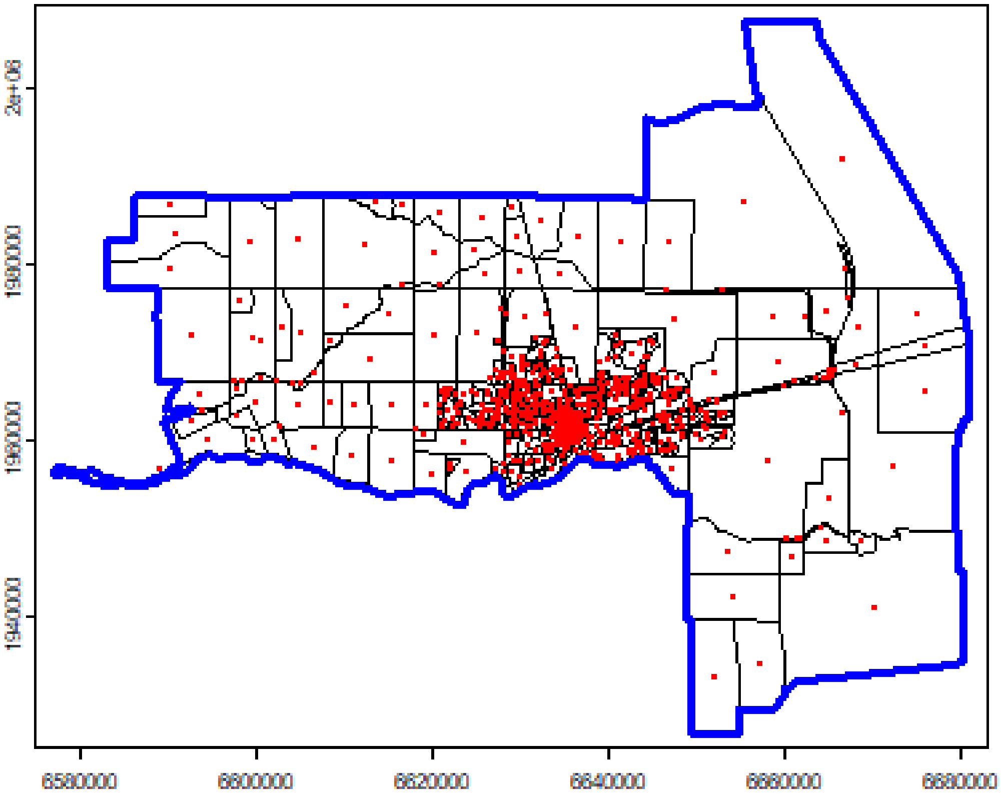
```
p <- terra::crds(centroids(census))
head(p)
##            x       y
## [1,] 6666671 1991720
## [2,] 6655379 1986903
## [3,] 6604777 1982474
## [4,] 6612242 1981881
## [5,] 6613488 1986776
## [6,] 6616743 1986446
```

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(sf::st_as_sf(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'
```

(continues on next page)

(continued from previous page)

```
## window: polygonal boundary
## enclosing rectangle: [6576938, 6680926] x [1926586.1, 2007558.2] units
## *** 1 illegal point stored in attr(,"rejects") ***
```

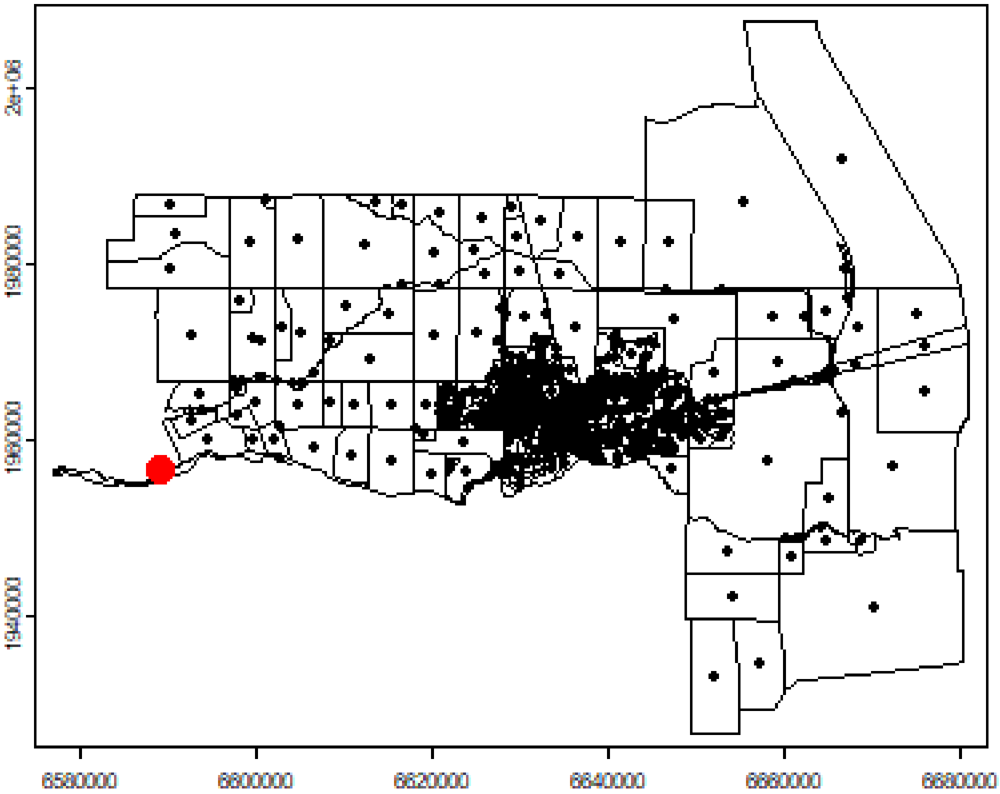
Note the warning message: “1 point was rejected as lying outside the specified window”. That is odd, there is a polygon that has a centroid that is outside of the polygon. This can happen with, e.g., kidney shaped polygons.

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

```
sp <- vect(p, crs=crs(win))
i <- relate(sp, win, "intersects")
i <- which(!i)
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:

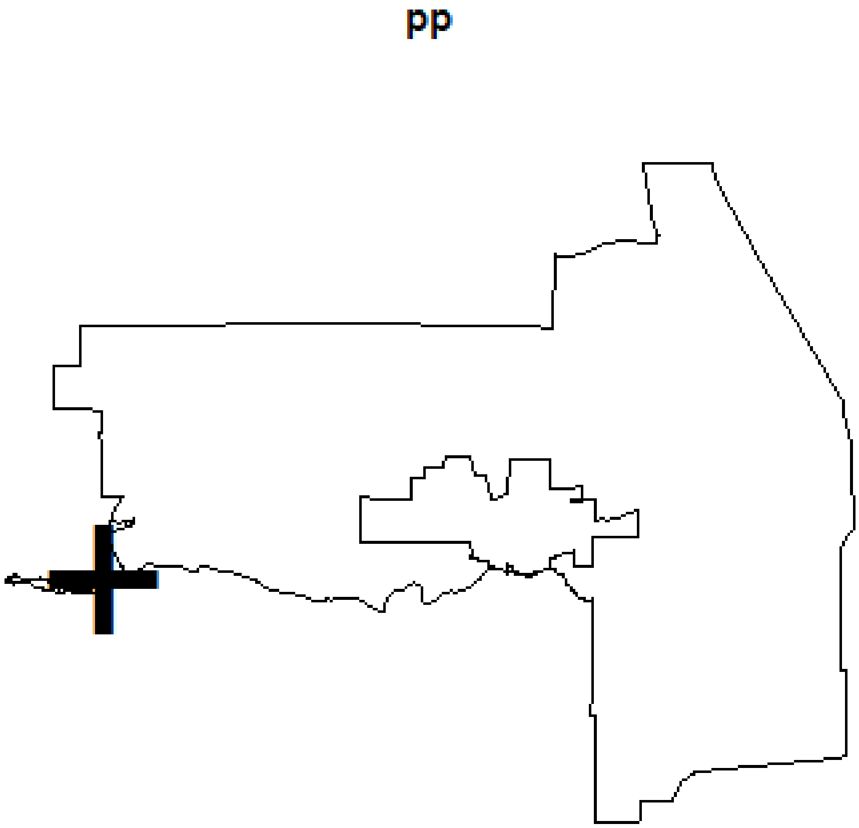
```
zoom(census)
```

And add the red points again

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

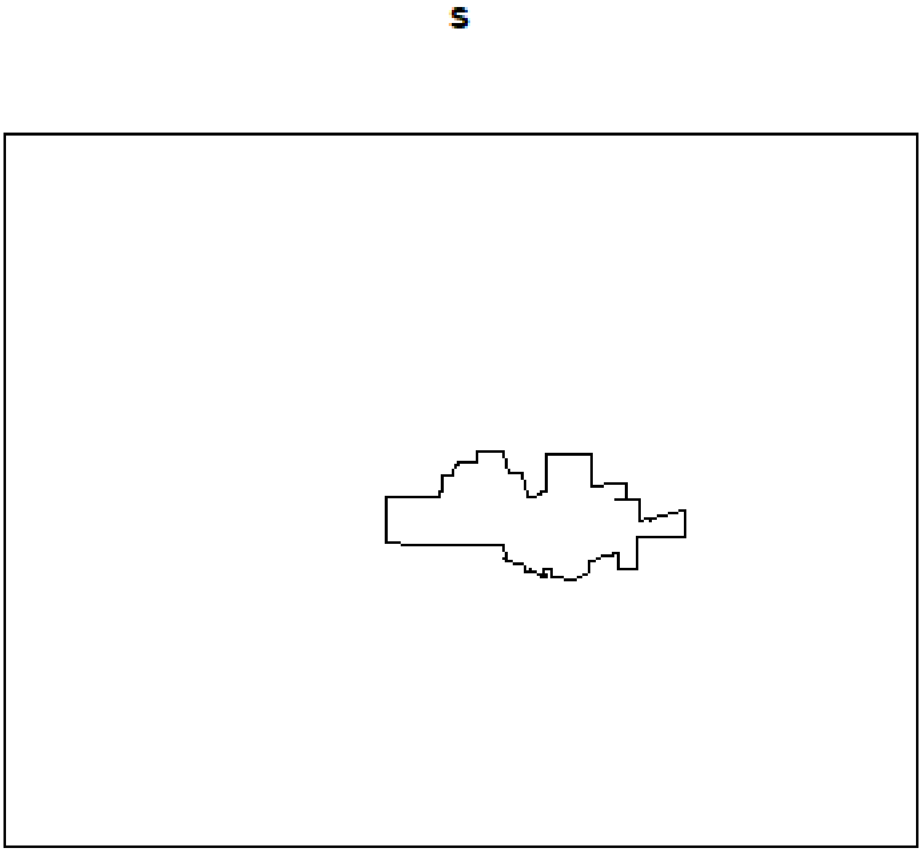
To only use points that intersect with the window polygon, that is, where 'i == TRUE':

```
pp <- ppp(p[i,1], p[i,2], window=owin, marks=census$dens[i])
## Warning: 1 point was rejected as lying outside the specified window
plot(pp)
## Warning in plot.ppp(pp): 1 illegal points also plotted
plot(city, add=TRUE)
```



And to get a smooth interpolation of population density.

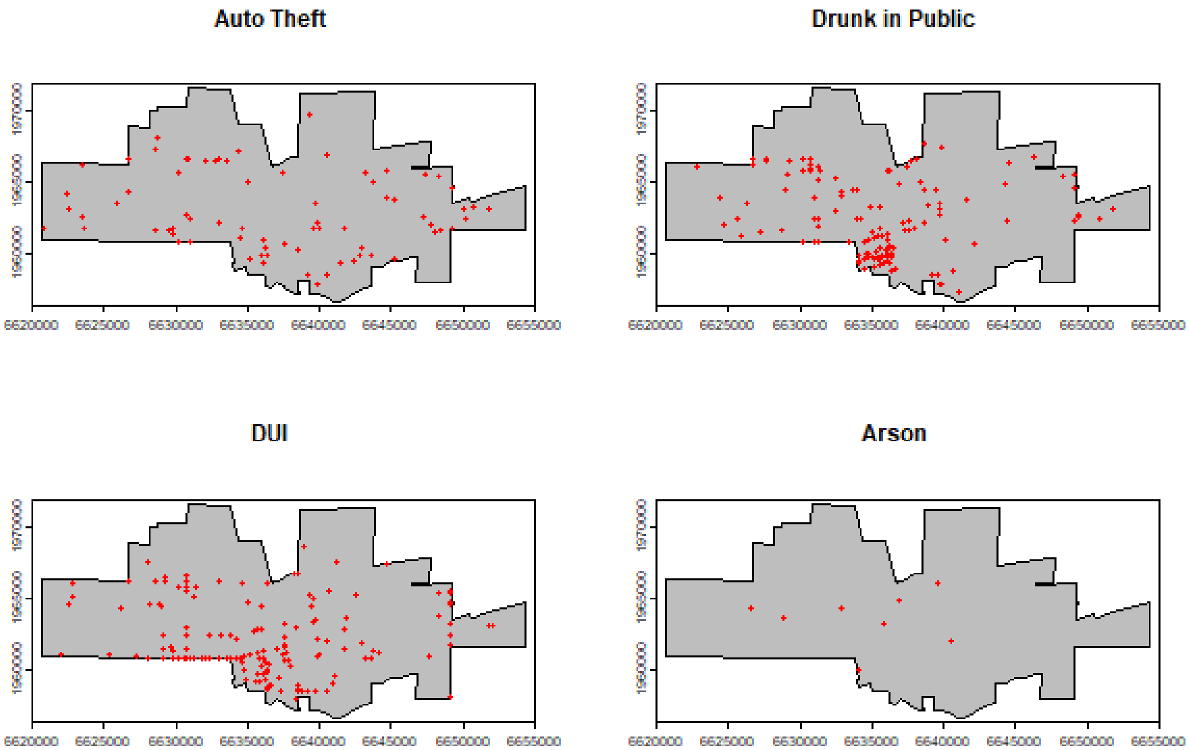
```
s <- Smooth.ppp(pp)
plot(s)
## Warning: All pixel values are NA
## Warning: Cannot determine range of values for colour map
plot(city, add=TRUE)
```



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)
}
```

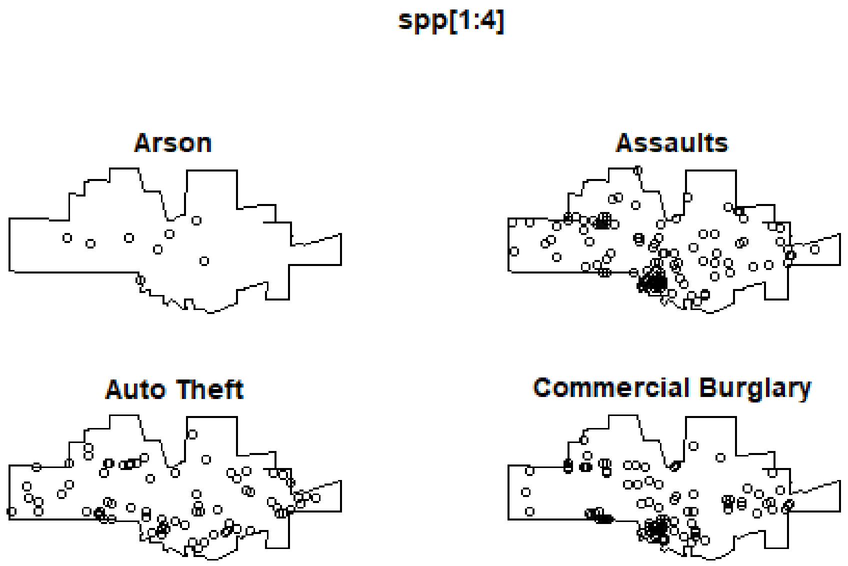


Create a marked point pattern object (ppp) for all crimes. It is important to coerce the marks to a factor variable.

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

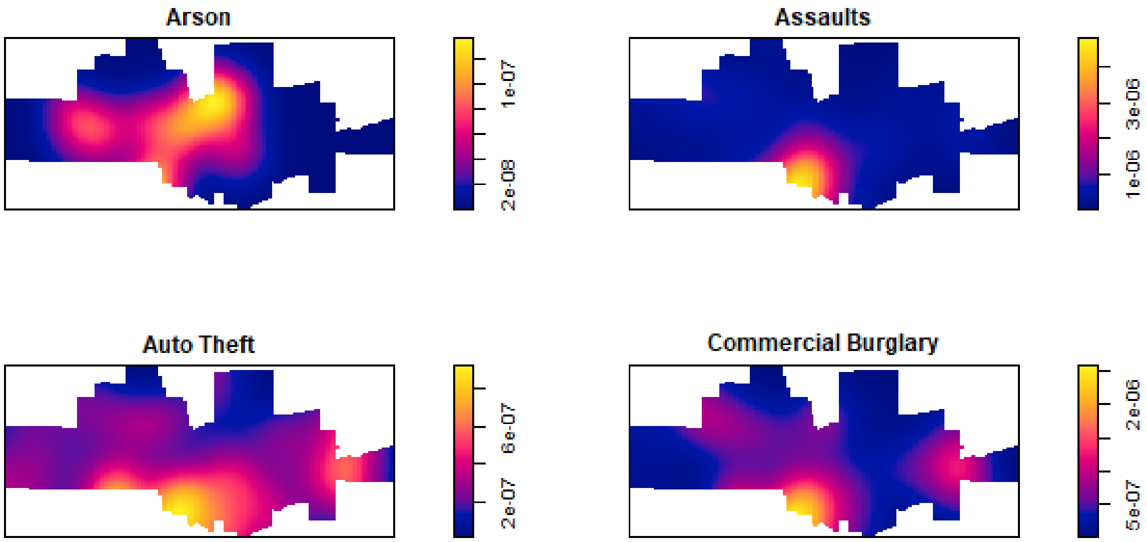
We can split the mpp object by category (crime)

```
spp <- split(mpp)

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



The crime density by category:

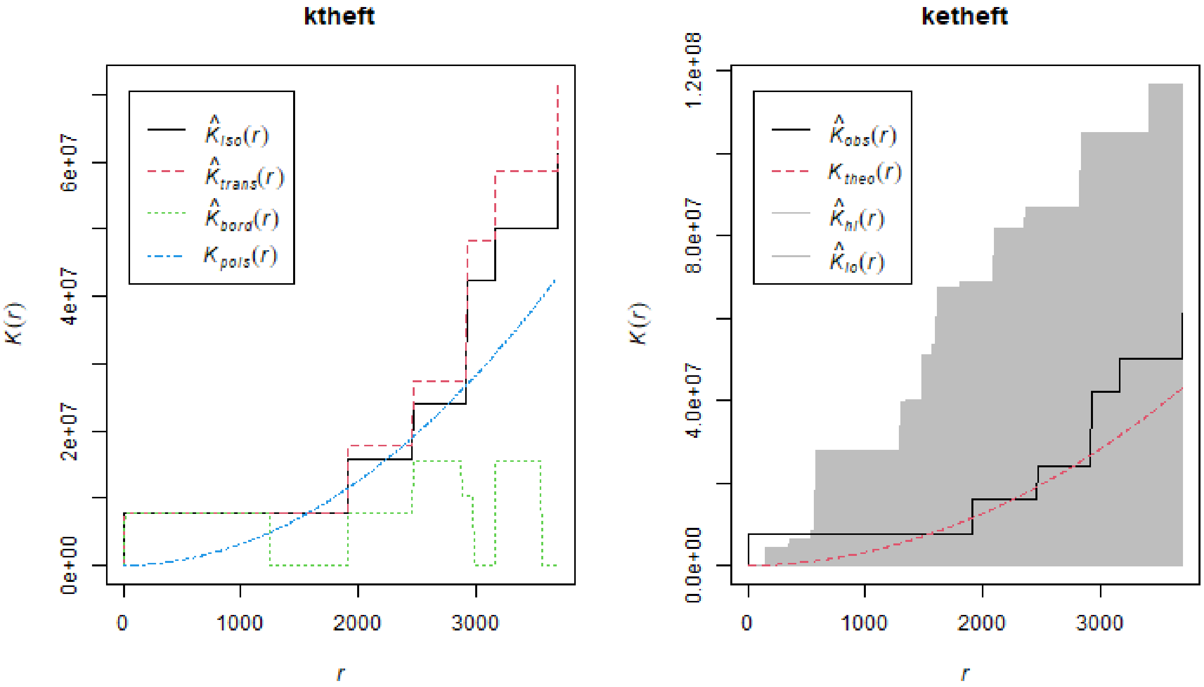
```
plot(density(spp[1:4]), main='')
```



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)
## 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)
```



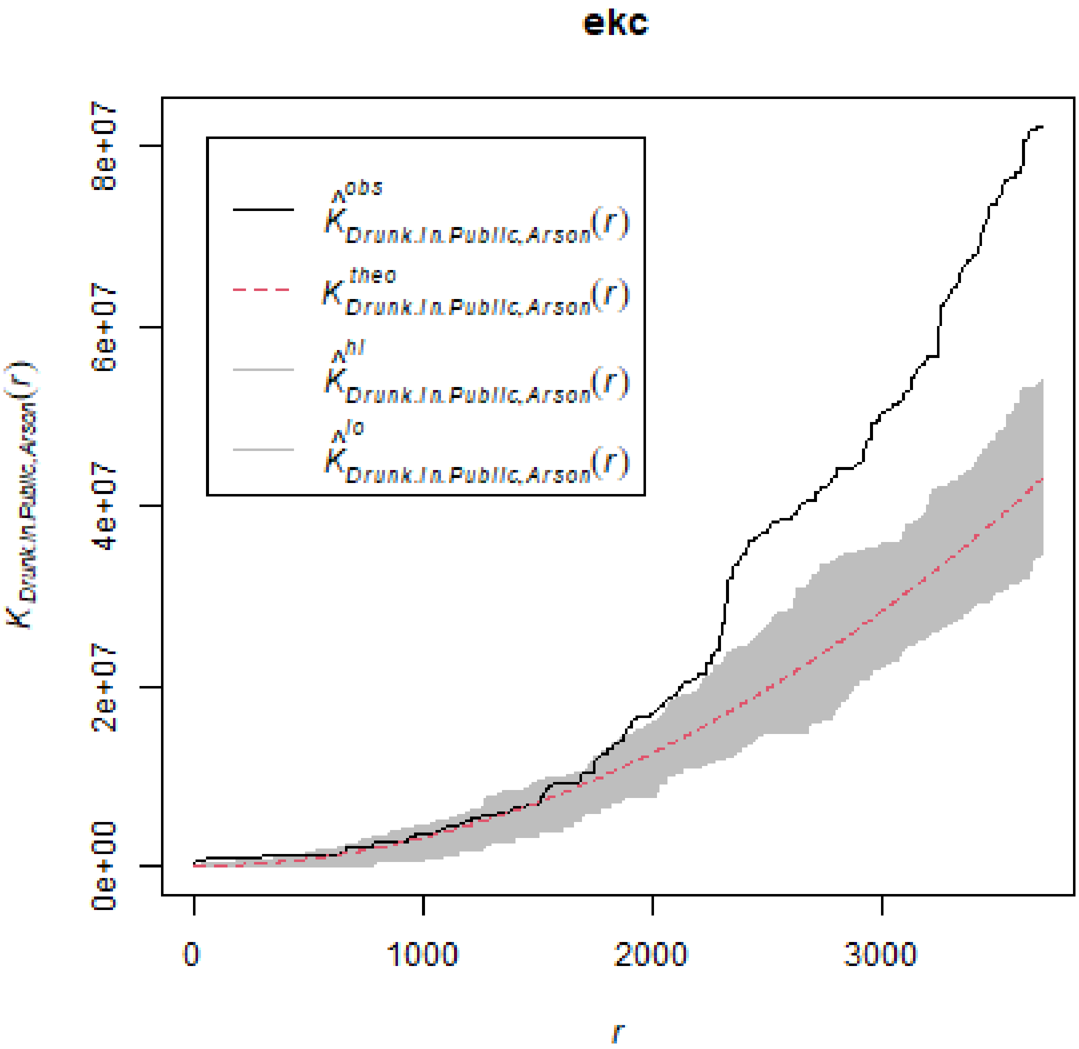
Let's try to answer the question you have been wanting to answer all along. Is population density a good predictor of being (booked for) "drunk in public" and for "Arson"? One approach is to do a Kolmogorov-Smirnov ('kstest') on 'Drunk in Public' and 'Arson', using population density as a covariate:

```
KS.arson <- cdf.test(spp$Arson, ds)
KS.arson
##
##  Spatial Kolmogorov-Smirnov test of CSR in two dimensions
##
## data:  covariate 'ds' evaluated at points of 'spp$Arson'
##      and transformed to uniform distribution under CSR
## D = 0.50693, p-value = 0.01145
## alternative hypothesis: two-sided
KS.drunk <- cdf.test(spp$'Drunk in Public', ds)
KS.drunk
##
##  Spatial Kolmogorov-Smirnov test of CSR in two dimensions
##
## data:  covariate 'ds' evaluated at points of 'spp$"Drunk in Public"'
##      and transformed to uniform distribution under CSR
## D = 0.54008, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

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