SISMID 2022 R Notes: Introduction to R

Jon Wakefield
University of Washington

2022-07-17

Preliminaries

Objectives

In this set of R notes you will:

- See the basic types of object that R uses, including spatial objects
- Learn how to install packages
- See how to read in data from a website
- Learn about basic plotting and regression models

Spatial data:

- Lancashire case-control point data
- Washington counties
- Scottish districts

Accessing the R code

To get R code alone, load the knitr library as below and then use the purl command – I comment out these lines to save time.

```
#install.packages("knitr")
#library(knitr)
#purl("2022-SISMID-Lec1-R.Rmd")
```

You need to be in the directory that contains the .Rmd file – the output file is 2022-SISMID-Lec1-R.R and is also in the current directory, which can be found by typing getwd(). To change the working directory, you can run setwd("/path/to/new/directory").

Introduction to R

This document is an R Markdown file. It is a relatively easy way to write a report (or notes!) that contains data analysis – great for reproducibility. To learn more about R Markdown, check out the online book at:

https://bookdown.org/yihui/rmarkdown/.

We describe various types of objects (vectors, matrices, arrays, data frames, lists). This can be confusing but the class command is useful.

Some functions require certain types of objects as inputs, so if a function doesn't do as expected, it can be that the wrong type of input is being used.

R's basic data types are character, numeric, integer, complex, and logical.

R's basic data structures include vector, list, matrix, data frame, and factor.

Vectors

In R, we use <- or = to assign the value on the right hand side to an object on the left hand side.

For example,

```
x1 <- "Hello"
x1
## [1] "Hello"
class(x1)
## [1] "character"
y1 <- c(1, 2, 3)
y1
## [1] 1 2 3
class(y1)
## [1] "numeric"
```

The c(...) notation defines a vector, i.e., a collection of elements of the same type. For some special vectors, there are faster ways to define them.

For example,

```
x2 <- 1:10
x2
##  [1]  1  2  3  4  5  6  7  8  9 10
class(x2)
## [1] "integer"
x2a <- seq(1, 5, 0.5)
x2a
## [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
class(x2a)
## [1] "numeric"
```

Objects of type matrix and array

Similarly, there are two-dimensional extensions of a vector (a matrix), and higher-dimensional extensions of a vector (an array). We give some examples of subsetting. When data is input the leftmost subscript moves fastest.

```
# Matrix
x3 <- matrix(1:24, nrow = 3)
x3
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    4    7   10   13   16   19   22
## [2,]    2    5    8   11   14   17   20   23
## [3,]    3    6    9   12   15   18   21   24
x3[2, 3]
## [1] 8
class(x3)
## [1] "matrix" "array"
```

```
# Array
y2 <- array(1:24, dim = c(2, 3, 4))
y2[, , 1]
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
y2[, , 4]
##      [,1] [,2] [,3]
## [1,]   19   21   23
## [2,]   20   22   24
y2[1, 2, 2]
## [1] 9
class(y2)
## [1] "array"
```

Vectorized operations

Many operators in R are vectorized, meaning that operations occur in parallel in certain R objects.

```
# Single operation
x4 <- 3
x4 <- x4^2 + 1
x4
## [1] 10

# Vectorized operation
y3 <- c(2, 3, 0)
y3 <- log(y3)
y3
## [1] 0.6931472 1.0986123      -Inf
```

Accessing elements

There are three operators that can be used to extract subsets of R objects.

The [operator always returns an object of the same class as the original. It can be used to select multiple elements of an object.

The [[operator is used to extract elements of a list or a data frame. It can only be used to extract a single element and the class of the returned object will not necessarily be a list or data frame.

The $ operator is used to extract elements of a list or data frame by literal name. Its semantics are similar to that of [[.

Data frame

A tabular form of data where each column may contain different types of values (i.e., similar to a spreadsheet), is called a data frame in R.

```
x5 <- data.frame(ID = 1:4,
                 name = c("A", "B", "C", "D"),
                 color = factor(c("red", "blue", "blue", "green")))
class(x5)
## [1] "data.frame"
sapply(x5, class)
##          ID        name       color
##   "integer" "character"    "factor"
x5$ID
## [1] 1 2 3 4
```

```
x5[,1]
## [1] 1 2 3 4
x5[1,]
##   ID name color
## 1  1    A   red
```

```
x5[, 2]
## [1] "A" "B" "C" "D"
x5$color
## [1] red   blue  blue  green
## Levels: blue green red
x5[1:2, ]
##   ID name color
## 1  1    A   red
## 2  2    B  blue
x5[1:2, 2:3]
##   name color
## 1    A   red
## 2    B  blue
x5[, c("ID", "name")]
##   ID name
## 1  1    A
## 2  2    B
## 3  3    C
## 4  4    D
```

Factors

Notice that although factors look similar to characters, they can behave very differently in action.

We encourage new users of `R` to read the section on 'R Nuts and Bolts' of this online book R Programming for Data Science by Roger Peng, (if not the whole book) for more details.

Lists

Another commonly used data structure is 'list'. It creates a list of elements that are not necessarily of the same type, dimension, or have the same attributes. For example,

```
y4 <- list(apple = 1:20, orange = x5)
class(y4)
## [1] "list"
y4
## $apple
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
##
## $orange
##   ID name color
## 1  1    A   red
## 2  2    B  blue
## 3  3    C  blue
## 4  4    D green
```

To subset a list:

```
y4[[1]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

```
y4[["apple"]]
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
y4$apple
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20

y4[["orange"]]
##   ID name color
## 1  1    A   red
## 2  2    B  blue
## 3  3    C  blue
## 4  4    D green
y4$orange
##   ID name color
## 1  1    A   red
## 2  2    B  blue
## 3  3    C  blue
## 4  4    D green
```

Coercion

Objects can be explicitly coerced from one class to another using the `as.*` functions, if available.

```
x <- 0:6
class(x)
## [1] "integer"
as.numeric(x)
## [1] 0 1 2 3 4 5 6
as.logical(x)
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
as.character(x)
## [1] "0" "1" "2" "3" "4" "5" "6"
```

R packages: Installation

To install packages that are hosted on CRAN, we only need to use the `install.packages` function.

```
#install.packages("ggplot2", dep = TRUE)
#install.packages("rgdal", dep = TRUE)
```

Some packages not hosted on CRAN. For example, the INLA package that we will be using for space-time smoothing is hosted on its own website.

```
# Commented out to save time
# install.packages("INLA",
#                  repos=c(getOption("repos"),
#                          INLA="https://inla.r-inla-download.org/R/stable"))
```

Load R packages

Once a package is installed, you can load it into your workspace. Then you can use all the functions and data provided in that package.

```
library(ggplot2)
library(rgdal)
library(SUMMER)
```

You can check your loaded packages in the workspace by (I haven't evaluated here, as I have lots of packages installed!).

```
# sessionInfo()
```

Spatial R

R for Spatial Analysis

R has extensive spatial capabilities, the Spatial task view is here.

Bivand et al (2013), is **the** reference book for GIS in R, though unfortunately it's quite hard to follow if you're an R novice.

Representing Spatial Data

Spatial classes were defined to represent and handle spatial data, so that data can be exchanged between different classes.

Class definitions are objects that contain the formal definition of a class of R objects, usually referred to as an S4 class.

Just as components of a list are accessed using `$`, slots of an object are accessed using `@`.

The `sp` library is the workhorse for representing spatial data.

The most basic spatial object is a 2d or 3d point: a set of coordinates may be used to define a `SpatialPoints` object.

From the help function:

`SpatialPoints(coords, proj4string=CRS(as.character(NA)),bbox = NULL)`

- PROJ.4 is a library (not an R package) for performing conversions between cartographic projections.
- The points in a `SpatialPoints` object may be associated with a set of attributes to give a `SpatialPointsDataFrame` object.

Creating a Spatial Object

As an example, the `splancs` library was pre-`sp` and so does not use spatial objects.

`splancs` contains a number of useful functions for analyzing spatial referenced point data.

```
library(sp)
library(splancs)
data(southlancs)  # case control data
summary(southlancs)
##        x               y               cc
##  Min.   :346475   Min.   :412437   Min.   :0.00000
##  1st Qu.:353031   1st Qu.:417358   1st Qu.:0.00000
##  Median :355870   Median :421900   Median :0.00000
##  Mean   :355526   Mean   :421518   Mean   :0.05852
##  3rd Qu.:358202   3rd Qu.:425984   3rd Qu.:0.00000
##  Max.   :364435   Max.   :428987   Max.   :1.00000
```

Let's look at the case control points as regular R data.

```
class(southlancs)
## [1] "data.frame"
```

```
names(southlancs)
## [1] "x"  "y"  "cc"
points <- southlancs[, c("x", "y")]
summary(points)
##        x               y
##  Min.   :346475   Min.   :412437
##  1st Qu.:353031   1st Qu.:417358
##  Median :355870   Median :421900
##  Mean   :355526   Mean   :421518
##  3rd Qu.:358202   3rd Qu.:425984
##  Max.   :364435   Max.   :428987
```

We convert into a `SpatialPoints` object and then create a `SpatialPointsDataFrame` data frame.

```
casepoints <- SpatialPoints(southlancs[, c("x", "y")])
summary(casepoints)
## Object of class SpatialPoints
## Coordinates:
##      min    max
## x 346475 364435
## y 412437 428987
## Is projected: NA
## proj4string : [NA]
## Number of points: 974
```

```
class(casepoints)
## [1] "SpatialPoints"
## attr(,"package")
## [1] "sp"
proj4string(casepoints)
## [1] NA
```

No info on where these points are on planet Earth...

Examining a Spatial Object

Try out some commands

```
head(coordinates(casepoints), 3)
##        x      y
## 1 359014 416976
## 2 352909 426935
## 3 353848 422172
bbox(casepoints)
##      min    max
## x 346475 364435
## y 412437 428987
casepoints@coords[1:3, ]
##        x      y
## 1 359014 416976
## 2 352909 426935
## 3 353848 422172
```

```
str(casepoints)
## Formal class 'SpatialPoints' [package "sp"] with 3 slots
##   ..@ coords     : num [1:974, 1:2] 359014 352909 353848 359202 357795 ...
```

```
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:974] "1" "2" "3" "4" ...
##   .. .. ..$ : chr [1:2] "x" "y"
##   ..@ bbox       : num [1:2, 1:2] 346475 412437 364435 428987
##   .. ..- attr(*, "dimnames")=List of 2
##   .. .. ..$ : chr [1:2] "x" "y"
##   .. .. ..$ : chr [1:2] "min" "max"
##   ..@ proj4string:Formal class 'CRS' [package "sp"] with 1 slot
##   .. .. ..@ projargs: chr NA
```

Creating a Spatial Object (continued)

Now create a `SpatialPointsDataFrame` data frame.

```
caseDF <- SpatialPointsDataFrame(coords = casepoints, data = as.data.frame(southlancs$cc))
class(caseDF)
## [1] "SpatialPointsDataFrame"
## attr(,"package")
## [1] "sp"
names(caseDF)
## [1] "southlancs$cc"
```

Now examine the `SpatialPointsDataFrame` data frame.

```
summary(caseDF)
## Object of class SpatialPointsDataFrame
## Coordinates:
##      min    max
## x 346475 364435
## y 412437 428987
## Is projected: NA
## proj4string : [NA]
## Number of points: 974
## Data attributes:
##  southlancs$cc
##  Min.   :0.00000
##  1st Qu.:0.00000
##  Median :0.00000
##  Mean   :0.05852
##  3rd Qu.:0.00000
##  Max.   :1.00000
```

Visualizaing Spatial Data

Plotting spatial data can be provided in a variety of ways, see Chapter 3 of Bivand et al. (2013).

The most obvious is to use the regular plotting functions, by converting `Spatial` dataframes to regular dataframes, for example using `as.data.frame`.

Reading Shapefiles

ESRI (a company one of whose products is ArcGIS) shapefiles consist of three files, and this is a common form.

- The first file (*.shp) contains the geography of each shape.

- The second file (*.shx) is an index file which contains record offsets.
- The third file (*.dbf) contains feature attributes with one record per feature.

Reading Shapefiles

Many repositories host data. Globally, the Database of Global Administrative Areas (GADM) here provides maps and spatial data for all countries and their subdivisions (admin-0, admin-1, admin-2,...).

Locally, the UW Geospatial Data Resources: here.

As an example, consider Washington county data that was downloaded from this site. The data consists of the three files: wacounty.shp, wacounty.shx, wacounty.dbf.

The following code reads in these data and then draws a county level map of 1990 populations.

First load the libraries.

```
library(maps)
library(maptools)
```

The three required data files (wacounty.*) can be downloaded from here: http://faculty.washington.edu/jonno/SISMIDmaterial/, and can be put in a directory `R-examples` (which you'll need to create).

You can put in any old directory, including the current directory, so long as you use the correct name when you call. The following code does this for you:

```
# Create directory
if (!("R-examples" %in% list.files())) {
    dir.create("R-examples")
}

# Download files using urls If this throws an error, comment out and manually
# download the .shp, .shx and .dbf files from the website.
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/wacounty.dbf",
    destfile = "R-examples/wacounty.dbf")
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/wacounty.shp",
    destfile = "R-examples/wacounty.shp")
download.file("http://faculty.washington.edu/jonno/SISMIDmaterial/wacounty.shx",
    destfile = "R-examples/wacounty.shx")

# Now read all the files into a single data frame
wacounty <- rgdal::readOGR(dsn = "R-examples", layer = "wacounty")
## OGR data source with driver: ESRI Shapefile
## Source: "/Users/kpaulson/Documents/SISMID-spatial/R-examples", layer: "wacounty"
## with 39 features
## It has 6 fields
class(wacounty)
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

Looking at the data

```
names(wacounty)
## [1] "AreaName" "AreaKey"  "INTPTLAT" "INTPTLNG" "TotPop90" "CNTY"

# Let's see what these variables look like
```

```
wacounty$AreaName[1:3]  # county names
## [1] "WA, Adams County"  "WA, Asotin County" "WA, Benton County"
wacounty$AreaKey[1:3]  # FIPS codes
## [1] "53001" "53003" "53005"
```

Examining SpatialPolygonDataFrames

```
proj4string(wacounty)
## [1] NA
head(wacounty)
## class       : SpatialPolygonsDataFrame
## features    : 6
## extent      : -124.7312, -116.915, 45.5434, 48.5504  (xmin, xmax, ymin, ymax)
## crs         : NA
## variables   : 6
## names       :         AreaName, AreaKey,  INTPTLAT,    INTPTLNG, TotPop90, CNTY
## min values  : WA, Adams County,   53001, 45.773673, -123.931203,    13603,    1
## max values  : WA, Clark County,   53011, 48.109502,  -117.18502,   238053,    9
```

Examining the data

Let's look in the data slot

```
head(wacounty@data)
##             AreaName AreaKey INTPTLAT  INTPTLNG TotPop90 CNTY
## 0   WA, Adams County   53001 46.98899 -118.5569    13603    1
## 1  WA, Asotin County   53003 46.18248 -117.1850    17605    3
## 2  WA, Benton County   53005 46.24764 -119.5015   112560    5
## 3  WA, Chelan County   53007 47.87696 -120.6414    52250    7
## 4 WA, Clallam County   53009 48.10950 -123.9312    56464    9
## 5   WA, Clark County   53011 45.77367 -122.4843   238053   11
```

Drawing a map

We look at some variables.

```
wacounty$INTPTLAT[1:3]  # latitude
## [1] 46.98899 46.18248 46.24764
wacounty$INTPTLNG[1:3]  # longitude
## [1] -118.5569 -117.1850 -119.5015
wacounty$CNTY[1:3]
## [1] "1" "3" "5"
wacounty$TotPop90[1:3]
## [1]  13603  17605 112560
```

We set up the colors to map.

```
library(RColorBrewer)
plotvar <- wacounty$TotPop90  # variable we want to map
summary(plotvar)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2248   16863   38558  123102   81478 1507319
nclr <- 8  # next few lines set up the color scheme for plotting
plotclr <- brewer.pal(nclr, "Oranges")
```

```
brks <- round(quantile(plotvar, probs = seq(0, 1, 1/(nclr))), digits = 1)
colornum <- findInterval(plotvar, brks, all.inside = T)
colcode <- plotclr[colornum]
plot(wacounty, col = colcode)
legend(-119, 46, legend = leglabs(round(brks, digits = 1)), fill = plotclr, cex = 0.35,
    bty = "n")
```

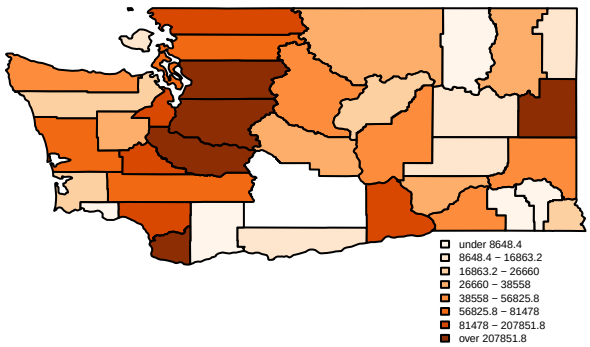


Figure 1: 1990 Washington population counts by census tracts

As an alternative we can use the `spplot` function, which uses lattice (trellis) plot methods for spatial data with attributes.

```
spplot(wacounty, zcol = "TotPop90", col.regions = brewer.pal(9, "Purples"), cuts = 8)
```

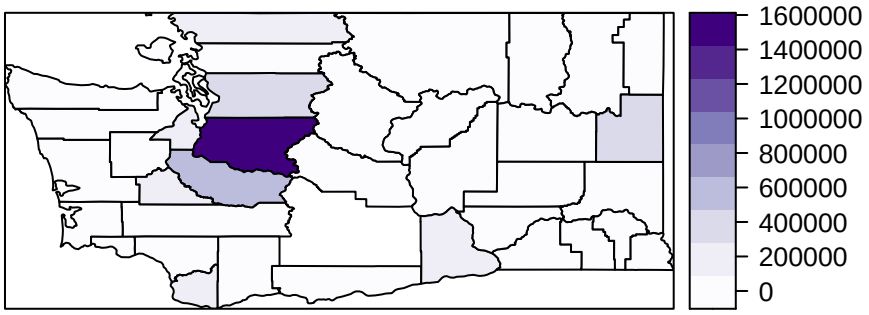
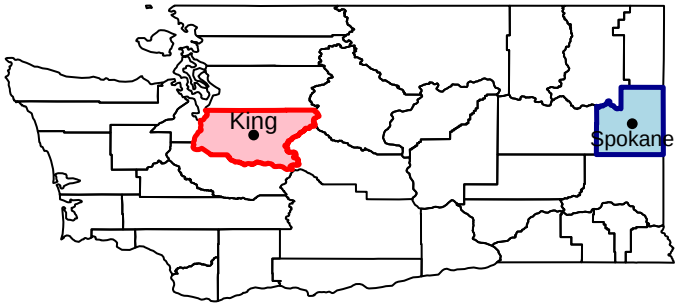


Figure 2: 1990 Washington population counts by county

Highlight a county

```
# identify counties of interest
xx = which(wacounty$CNTY == 33)
xx2 = which(wacounty$CNTY == 63)
# plot the whole state
plot(wacounty)
# highlight counties of interest
plot(wacounty[xx, ], col = "pink", border = "red", add = T, lwd = 2.5)
plot(wacounty[xx2, ], col = "lightblue", border = "darkblue", add = T, lwd = 2.5)
# Add some labels
text(coordinates(wacounty[xx, ]), "King", cex = 0.75, pos = 3, offset = 0.15)
text(coordinates(wacounty[xx2, ]), "Spokane", cex = 0.7, pos = 1, offset = 0.25)
points(coordinates(wacounty[c(xx, xx2), ]), pch = 16, cex = 0.75)
```



Scottish lip cancer data

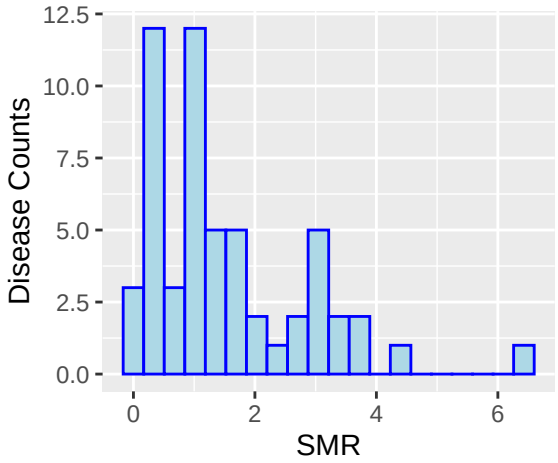
We will first fit a number of models to the famous Scottish lip cancer data.

We have counts of disease, expected numbers and an area-based covariate (proportion in agriculture, fishing and farming) in each of 56 areas.

```
library(SpatialEpi)
data(scotland)
Y <- scotland$data$cases
X <- scotland$data$AFF
E <- scotland$data$expected
# Relative risk estimates
smr <- Y/E
summary(E)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.100   4.050   6.300   9.575  10.125  88.700
summary(smr)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.000   0.496   1.111   1.522   2.241   6.429
```

Distribution of SMRs

```
ggplot(data.frame(smr), aes(x = smr)) + geom_histogram(color = "blue",
    fill = "lightblue", bins = 20) + labs(x = "SMR",
    y = "Disease Counts")
```

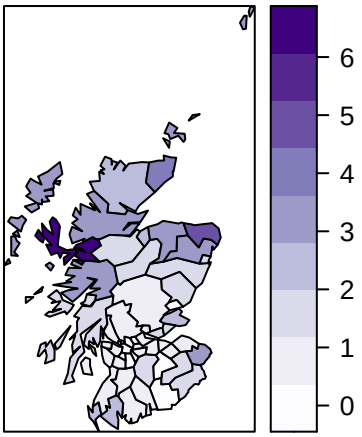


Mapping

```
smap <- scotland$spatial.polygon
scotd <- scotland$data[, c("county.names", "cases", "expected", "AFF")]
scotd$SIR <- scotd$cases/scotd$expected
```

```
# set the polygons ID slot values to correspond to county names
sapply(slot(smap, "polygons"), function(x) {
    slot(x, "ID")
})
##  [1] "skye-lochalsh" "banff-buchan"  "caithness"     "berwickshire"
##  [5] "ross-cromarty" "orkney"        "moray"         "shetland"
##  [9] "lochaber"      "gordon"        "western.isles" "sutherland"
## [13] "nairn"         "wigtown"       "NE.fife"       "kincardine"
## [17] "badenoch"      "ettrick"       "inverness"     "roxburgh"
## [21] "angus"         "aberdeen"      "argyll-bute"   "clydesdale"
## [25] "kirkcaldy"     "dunfermline"   "nithsdale"     "east.lothian"
## [29] "perth-kinross" "west.lothian"  "cumnock-doon"  "stewartry"
## [33] "midlothian"    "stirling"      "kyle-carrick"  "inverclyde"
## [37] "cunninghame"   "monklands"     "dumbarton"     "clydebank"
## [41] "renfrew"       "falkirk"       "clackmannan"   "motherwell"
## [45] "edinburgh"     "kilmarnock"    "east.kilbride" "hamilton"
## [49] "glasgow"       "dundee"        "cumbernauld"   "bearsden"
## [53] "eastwood"      "strathkelvin"  "tweeddale"     "annandale"
rownames(scotd) <- scotd$county
```

```
smap <- SpatialPolygonsDataFrame(smap, scotd, match.ID = TRUE)
spplot(smap, zcol = "SIR", col.regions = brewer.pal(9, "Purples"), cuts = 8)
```



Exercises

- Experiment with the `spplot` command, changing colors, for example
- Map the expected numbers from the Scottish lip cancer example