Mapping in R

Hans Gerritsen

Latest update April 2020

Some websites that give an overview of mappping in R:

https://cran.r-project.org/web/views/Spatial.html

http://spatial.ly/r/

http://geoawesomeness.com/r-goes-spatial/

Shapefiles

There are a number of packages that allow you to read shapefiles into R. The shapefiles package seems the most obvious but it doesnt offer convenient ways to plot them. The mapplots package has a function `draw.shapefile()` to deal with this, but it is easier to forget about the shapefiles packages and use the maptools package insted. The maptools function `readShapeSpatial()` reads the shapefile in as spatial object from the sp package (e.g. SpatialPolygonsDataFrame). These sp objects are widely used and can be manipulated and plotted quite easily. The rgdal package also allows you to read shapefiles in as sp objects.

```
library(sp) # for spatial objects like SpatialPolygonsDataframe
library(maptools) # for reading shapefiles
```

The function `readShapePoly()` reads the shapefile and stores it as a SpatialPointsDataframe, SpatialLines-Dataframe or SpatialPolygonsDataframe. It does not pick up the .prj file, so you have to explicitly tell it which projection you are using. If the data are in latitude and longitude you can use `CRS('+proj=longlat +ellps=WGS84 +no_defs')`. More on projections later.

```
ices <- readShapeSpatial("//Galwayfs03/Fishdata/Mapping/Shapefiles/ICES_areas_simple")
```

```
## Warning: readShapeSpatial is deprecated; use rgdal::readOGR or sf::st_read
```

```
## Warning: readShapePoly is deprecated; use rgdal::readOGR or sf::st_read
```

```
proj4string(ices) <-  CRS('+proj=longlat +ellps=WGS84 +no_defs')
class(ices)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

The rgdal function `readOGR()` does the same thing as the mapplots function `readShapePoly()` (and more). It also picks up the projection if there is a .prj file as part of the shapefile.

```
library(sp) # for spatial objects like SpatialPolygonsDataframe
library(rgdal) # for readOGR
```

```
## Warning: package 'rgdal' was built under R version 3.6.2
```

```
## rgdal: version: 1.4-8, (SVN revision 845)
##  Geospatial Data Abstraction Library extensions to R successfully loaded
##  Loaded GDAL runtime: GDAL 2.2.3, released 2017/11/20
##  Path to GDAL shared files: C:/Users/hgerritsen/Documents/R/win-library/3.6/rgdal/gdal
##  GDAL binary built with GEOS: TRUE
##  Loaded PROJ.4 runtime: Rel. 4.9.3, 15 August 2016, [PJ_VERSION: 493]
##  Path to PROJ.4 shared files: C:/Users/hgerritsen/Documents/R/win-library/3.6/rgdal/proj
```

```
##  Linking to sp version: 1.3-2
```

```r
ices <- readOGR("//Galwayfs03/Fishdata/Mapping/Shapefiles/ICES_areas_simple.shp","ICES_areas_simple")
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "\\Galwayfs03\Fishdata\Mapping\Shapefiles\ICES_areas_simple.shp", layer: "ICES_areas_simple"
## with 65 features
## It has 5 fields
```

```r
proj4string(ices) #it already knows the projection.
```

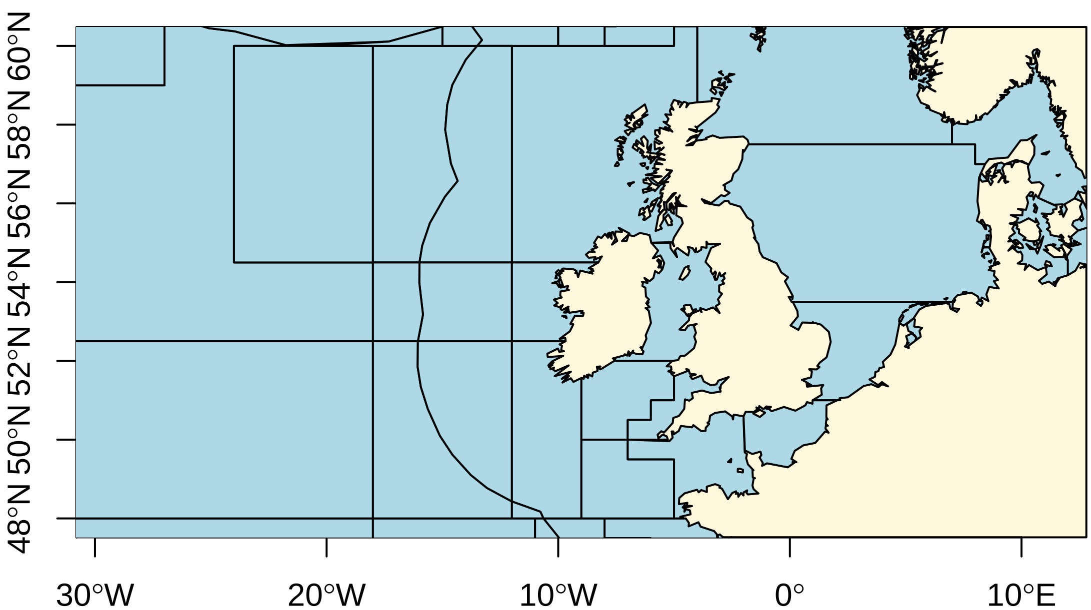
```
## [1] "+proj=longlat +ellps=WGS84 +no_defs"
```

```r
class(ices)
```

```
## [1] "SpatialPolygonsDataFrame"
## attr(,"package")
## [1] "sp"
```

You can now just plot the shapefile.

```r
plot(ices,col='lightblue',bg='cornsilk',xlim=c(-16,-2),ylim=c(48,60),axes=T)
```



Writing Shapefiles

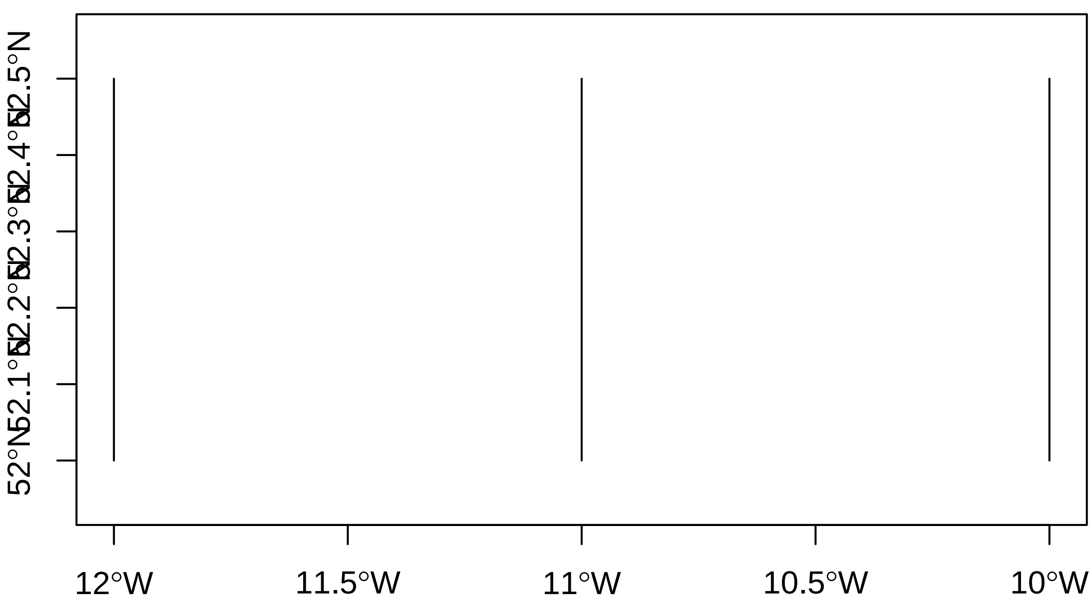
The example below shows how to make a shapefile out of a table with start and end positions of fishing trawls

```r
# generate some data
stat <- data.frame(Station=c('FG1','FG2','GD1'),ShootLon=c(-10,-11,-12),ShootLat=c(52,52,52),HaulLon=c(
```

```r
# stack it
stat$id <- 1:nrow(stat)
stat1 <- with(stat,data.frame(id,lon=ShootLon,lat=ShootLat))
stat1 <- with(stat,rbind(stat1,data.frame(id,lon=HaulLon,lat=HaulLat)))
stat1 <- stat1[order(stat1$id),]

# attribute table
data <- stat
row.names(data) <- data$id

# trickery to get a spatial lines dataframe
l <- split(stat1[,c('lon','lat')],stat1$id)
ll <- sapply(l,Line) #use sapply to retain names
lll <- lapply(seq_along(ll),function(i) Lines(ll[[i]],ID=names(ll)[i]))
crs <- CRS("+proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0")
sl <- SpatialLines(lll,crs)
sldf <- SpatialLinesDataFrame(sl, data)
plot(sldf,axes=T)
```




```r
sldf@data
```

```
##   Station ShootLon ShootLat HaulLon HaulLat Comment id
## 1     FG1      -10       52     -10    52.5    Good  1
## 2     FG2      -11       52     -11    52.5     Bad  2
## 3     GD1      -12       52     -12    52.5    Ugly  3
```

```r
# wite the shapefile
writeOGR(sldf, ".", "TestShapefile", "ESRI Shapefile",overwrite_layer=T)
```

Manipulating spatial dataframes

Some of the commands for 'normal' dataframes work on spatial dataframes (but not all because they are S4 objects, which are tricky). For example `names()` will give you the names of the columns in the attribute table, which is the data slot.

```r
names(ices)
```

```
## [1] "OBJECTID"   "ICES_area"  "Area_km2"   "Shape_Leng" "Shape_Area"
```

```r
head(ices@data)
```

```
##   OBJECTID ICES_area   Area_km2 Shape_Leng Shape_Area
## 0        1       Xa2  673583.30   42.05683   71.35407
## 1        2       Xa1 1055135.68   78.14629   96.31485
## 2        3      IXb1  302623.77   23.83313   31.71605
## 3        4      IXb2  161014.14   29.43313   17.28395
## 4        5       IXa  165211.80   30.56079   17.42807
## 5        6     VIIIc   95178.08   23.01613   10.05655
```

```r
ices@data$ICES_area
```

```
##  [1] Xa2    Xa1    IXb1   IXb2   IXa    VIIIc  VIIIb  Xb     VIIIe1 VIIId1
## [11] VIIId2 VIIIa  VIIj1  VIIh   VIIf   VIIg   XIIc   VIIk1  VIIk2  VIIj2
## [21] IVc    VIIc1  VIIc2  VIIb   VIIa   24     22     23     25     26
## [31] IVb    28-2   28-1   27     IIIa   XIIb   VIb1   VIb2   VIa    29
## [41] Vb1a   32     XIIa1  XIIa4  IVa    Va1    Vb1b   XIVb1  30     31
## [51] Va2    XIVb2  IIa1   IIa2   IIb1   Ia     XIVa   IIb2   Ib     VIIIe2
## [61] Vb2    XIIa3  XIIa2  VIIe   VIId
## 65 Levels: 22 23 24 25 26 27 28-1 28-2 29 30 31 32 Ia Ib IIa1 ... XIVb2
```

```r
ices$ICES_area
```

```
##  [1] Xa2    Xa1    IXb1   IXb2   IXa    VIIIc  VIIIb  Xb     VIIIe1 VIIId1
## [11] VIIId2 VIIIa  VIIj1  VIIh   VIIf   VIIg   XIIc   VIIk1  VIIk2  VIIj2
## [21] IVc    VIIc1  VIIc2  VIIb   VIIa   24     22     23     25     26
## [31] IVb    28-2   28-1   27     IIIa   XIIb   VIb1   VIb2   VIa    29
## [41] Vb1a   32     XIIa1  XIIa4  IVa    Va1    Vb1b   XIVb1  30     31
## [51] Va2    XIVb2  IIa1   IIa2   IIb1   Ia     XIVa   IIb2   Ib     VIIIe2
## [61] Vb2    XIIa3  XIIa2  VIIe   VIId
## 65 Levels: 22 23 24 25 26 27 28-1 28-2 29 30 31 32 Ia Ib IIa1 ... XIVb2
```

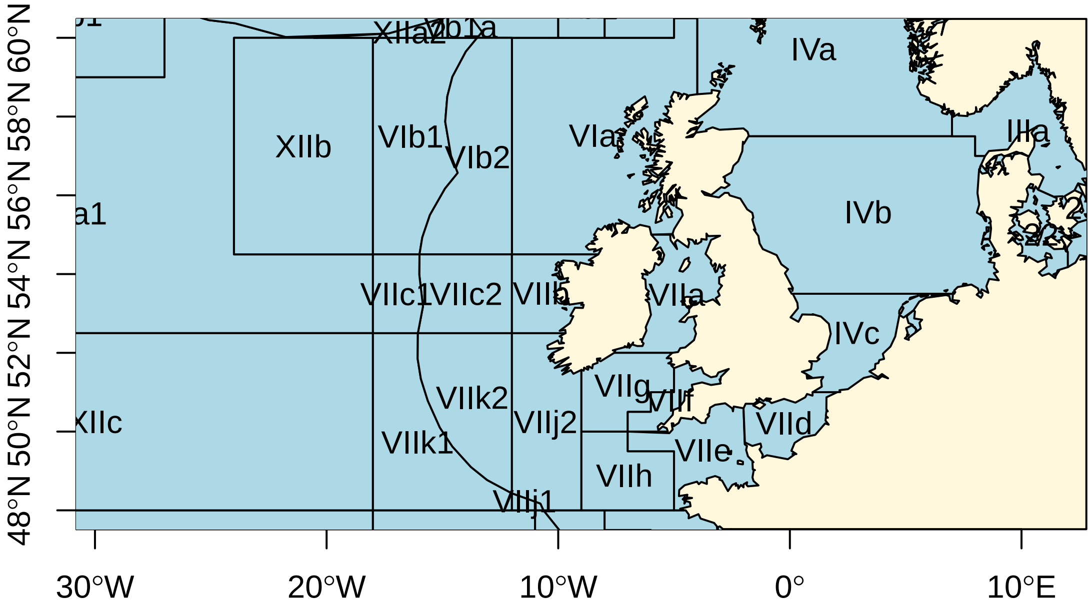
You can use this to subset the dataframe

```r
plot(subset(ices,ICES_area=='VIa'))
```

Or to add labels to a plot using the function `getSpatialPolygonsLabelPoints()`

```
plot(ices,col='lightblue',bg='cornsilk',xlim=c(-16,-2),ylim=c(48,60),axes=T)
text(data.frame(getSpatialPolygonsLabelPoints(ices)),labels=ices$ICES_area)
```

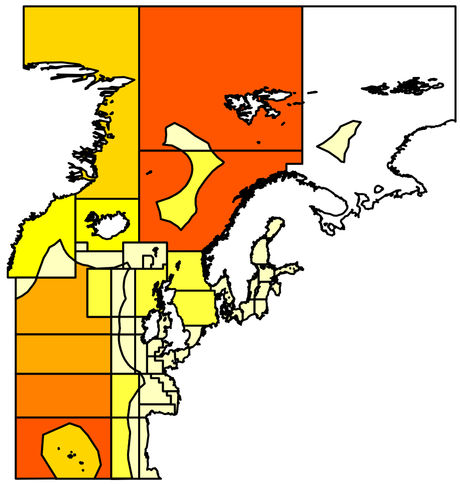
```
## Warning: use *apply and slot directly, or coordinates method
```



Colour by numbers

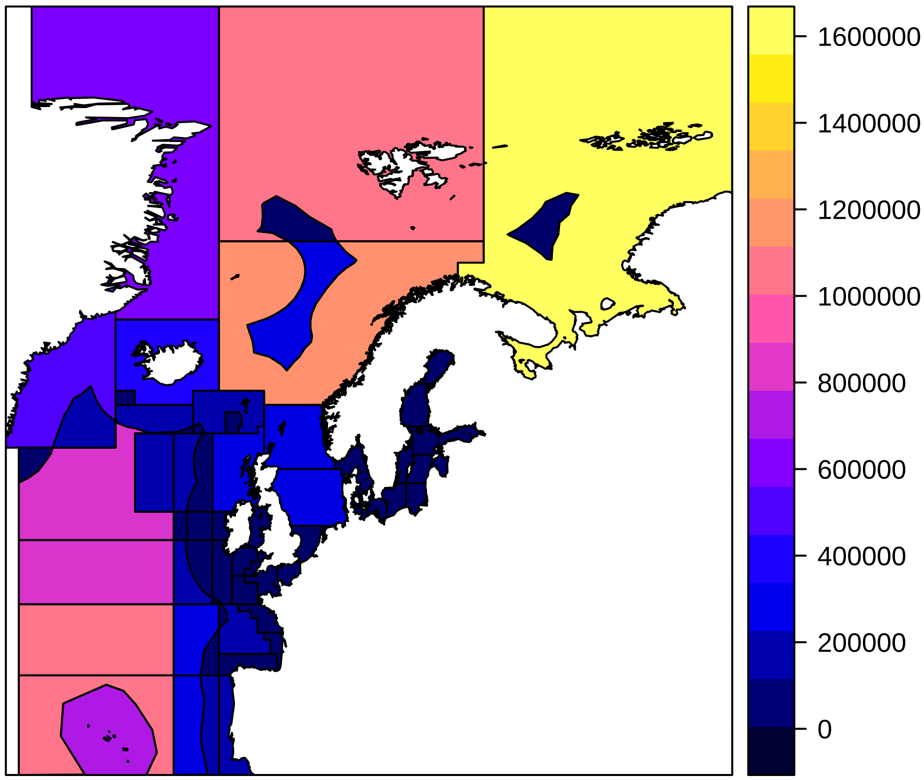
You can also use the attributes to assign colours to each polyogn. In this example we colour the ICES division according to their size. Not very useful but it illustrates the point: you need a vector of colours in the same order as the polygons in the SpatialPolygonsDataframe

```
breaks <- seq(0,max(ices$Area_km2),len=10) # breakpoints for colour scale
cols <- rev(heat.colors(9)) # one more breakpoint than colour
i <- findInterval(ices$Area_km2,breaks)
plot(ices,col=cols[i])
```



You can get something similar with spplot().

```
spplot(ices['Area_km2'])
```



Geoprocessing

With the rgeos package we can do some geoprocessing. Note that you need to have installed GEOS software on your pc for this package to work.

```
library(rgeos)
```

You may want to combine some ICES divisions to show the stock area and TAC area of cod in VIIb-k.

First specify the stock area and the tac area.

```
stockarea <- c('VIIe','VIIf','VIIg','VIIh','VIIj1','VIIj2','VIIk1','VIIk2')
tacarea <- c('VIIb','VIIc1','VIIc2','VIIe','VIIf','VIIg','VIIh','VIIj1','VIIj2','VIIk1','VIIk2','VIIIa'
```

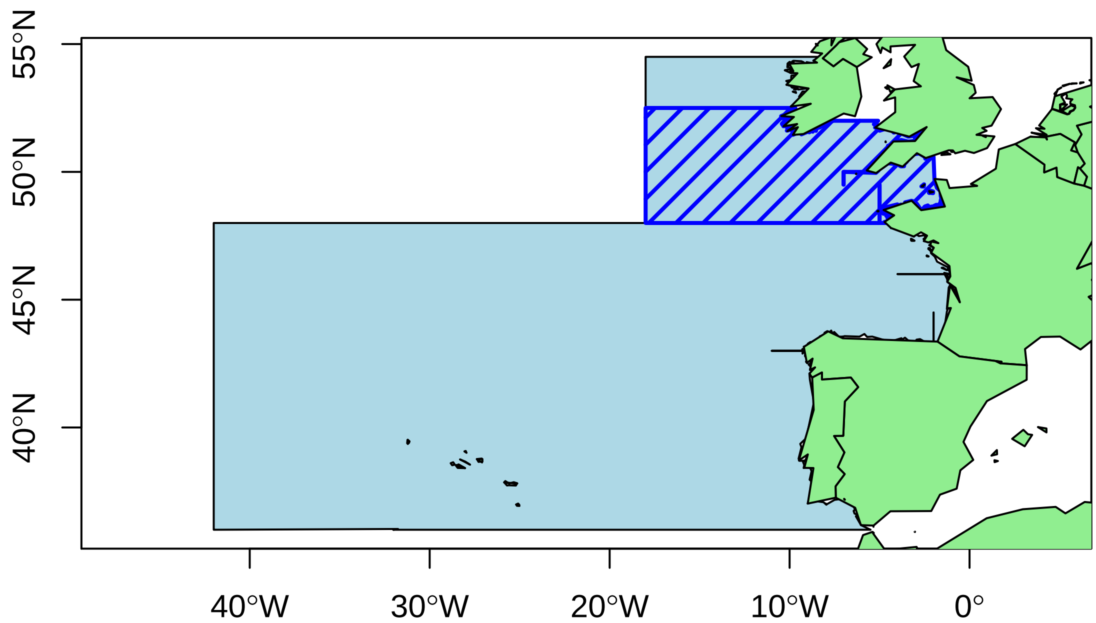
Now make a vector that indicates which polygons we want to combine.

```
stockid <- ifelse(ices$ICES_area %in% stockarea, 1, NA)
tacid <- ifelse(ices$ICES_area %in% tacarea, 1, NA)
```

And finally combine the polygons (Divisions) of the stock area into one, same for tac area

```
stock <- gUnaryUnion(ices,stockid)
tac <- gUnaryUnion(ices,tacid)

plot(tac,axes=T,col='lightblue')
plot(stock,col='blue',border='blue',lwd=2,density=10,add=T)
data(wrld_simpl)
plot(wrld_simpl,col='lightgreen',add=T)
```



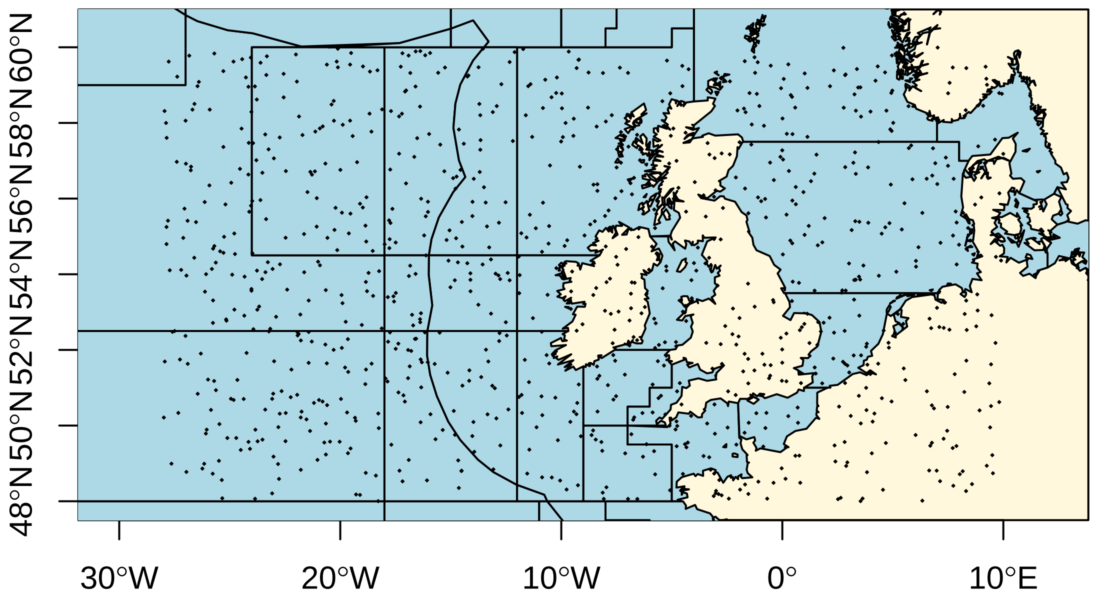
You can do other things with rgeos like intersect, merge, touches, etc, but you can do all of these things in ArcGis as well. Also note that [rgeos] can be quite fussy and might refuse to do certain things because it thinks geometries are invalid.

Points in polygons

You might have survey stations and want to know which ICES division, functional unit or spatial stratum do they fall in? You can use the `over()` function from the sp package to do a spatial overlay for points, grids and polygons.

Let's generate some random point data, this could be your survey stations for example. First we make a dataframe and then convert this into a SpatialPointsDataframe by telling it which columns correspond to spatial coordinates, notice the tilde ~ in the second line of code. We also specify the projection.

```
points <- data.frame(lon=runif(1000,-28,10),lat=runif(1000,48,60),value=1)
coordinates(points) <- ~ lon + lat
proj4string(points) <- CRS('+proj=longlat +ellps=WGS84 +no_defs')
plot(ices,col='lightblue',bg='cornsilk',axes=T,xlim=c(-18,0),ylim=c(48,60.5))
plot(points,add=T,cex=0.1)
```



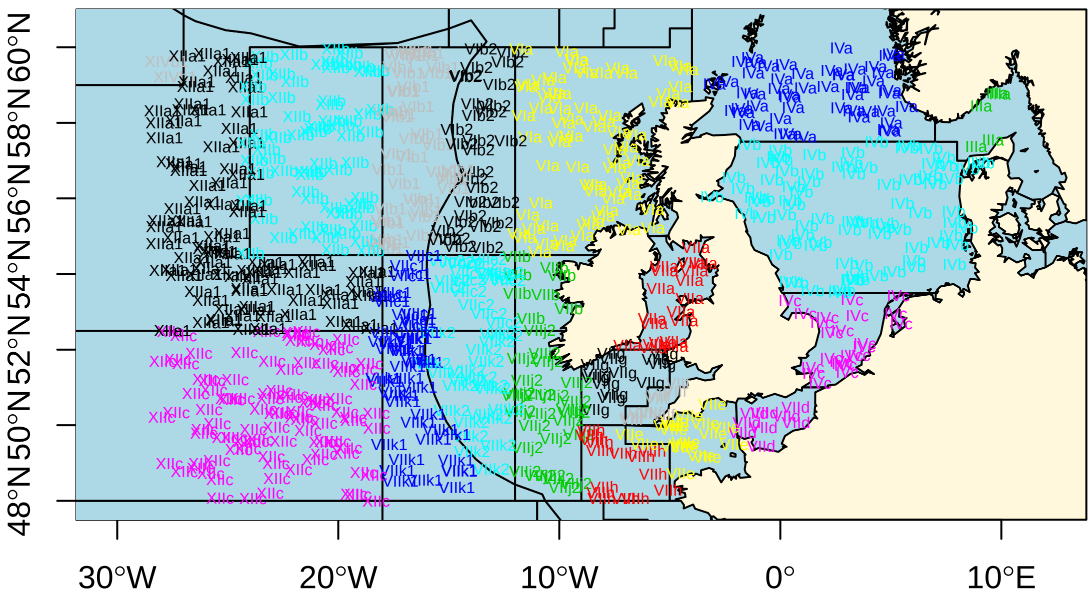
Now we can do do the spatial overlay. Each point will get the attributes from the polygon it falls in.

```
points1 <- data.frame(points,over(points,ices))
head(points1)
```

```
##           lon      lat value optional OBJECTID ICES_area   Area_km2
## 1   7.842464 55.65949     1     TRUE       31       IVb 281660.09
## 2  -8.766457 59.33249     1     TRUE       39       VIa 236928.84
## 3  -0.764981 49.93544     1     TRUE       65      VIId  33303.03
## 4 -22.210295 51.48058     1     TRUE       17      XIIc 831766.00
## 5   3.990084 58.29997     1     TRUE       45       IVa 265533.28
## 6 -14.519108 48.79839     1     TRUE       18     VIIk1 115964.77
##   Shape_Leng Shape_Area
## 1   48.85989  39.875805
## 2   77.36546  35.817634
## 3   13.41155   4.172082
## 4   57.00000 108.000000
## 5  111.96588  42.925913
## 6   19.11353  14.767250
```

We can check that this works by plotting the ICES area assigned to each point as text on the map.

```
plot(ices,col='lightblue',bg='cornsilk',axes=T,xlim=c(-18,0),ylim=c(48,60.5))
with(points1,text(lon,lat,ICES_area,col=as.numeric(as.factor(ICES_area)),cex=0.5))
```



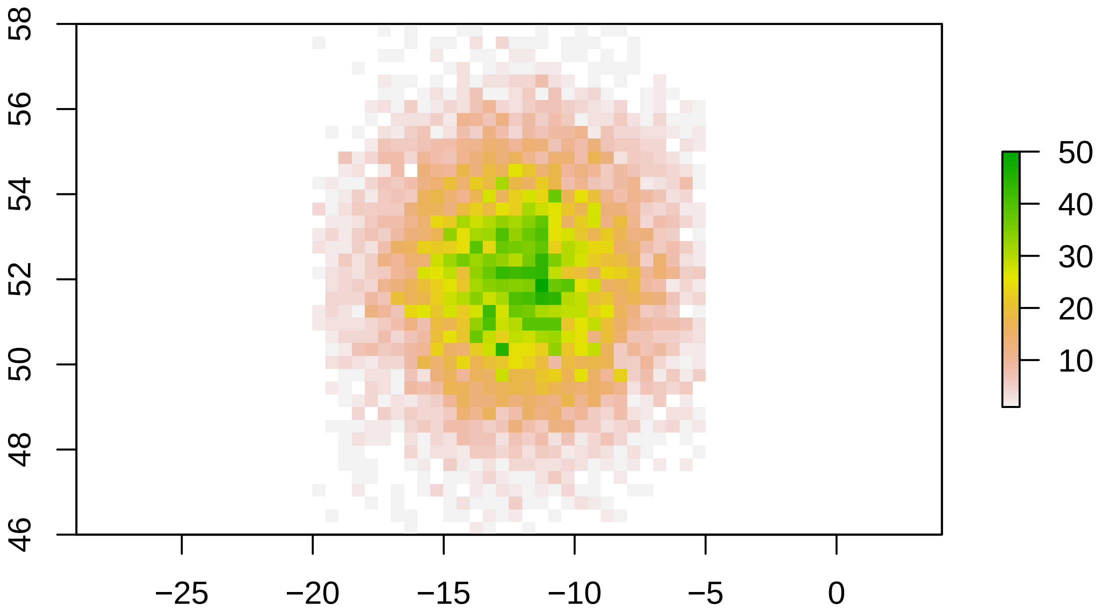
Rasters

VMS data are often aggregated in a raster, you can also treat rectangle data as a raster. The mapplots package has a way to deal with this (make.grid) but it is probably tidier to the raster package. The package deals particularly well with large files that would be unmanageble if you treat them as a matrix.

```
library(raster)

points <- data.frame(lon=rnorm(10000,-12,3),lat=rnorm(10000,52,2),value=1)
coordinates(points) <-~ lon + lat
proj4string(points) <- CRS('+proj=longlat +ellps=WGS84 +no_defs')

raster1 <- raster()
extent(raster1) <- c(-20,-5,46,58)
res(raster1) <- c(0.5,0.3)
projection(raster1) <- CRS('+proj=longlat +ellps=WGS84 +no_defs')
raster2 <- rasterize(points,raster1,field='value',sum)
plot(raster2)
```



The rasterVis package has some nice methods for visualising raster data. Its worth checking out these examples: http://oscarperpinan.github.io/rastervis/

Polygons from raster

see example to code for the 2019 Atlas: F:\ResourceBook\2018\Cover

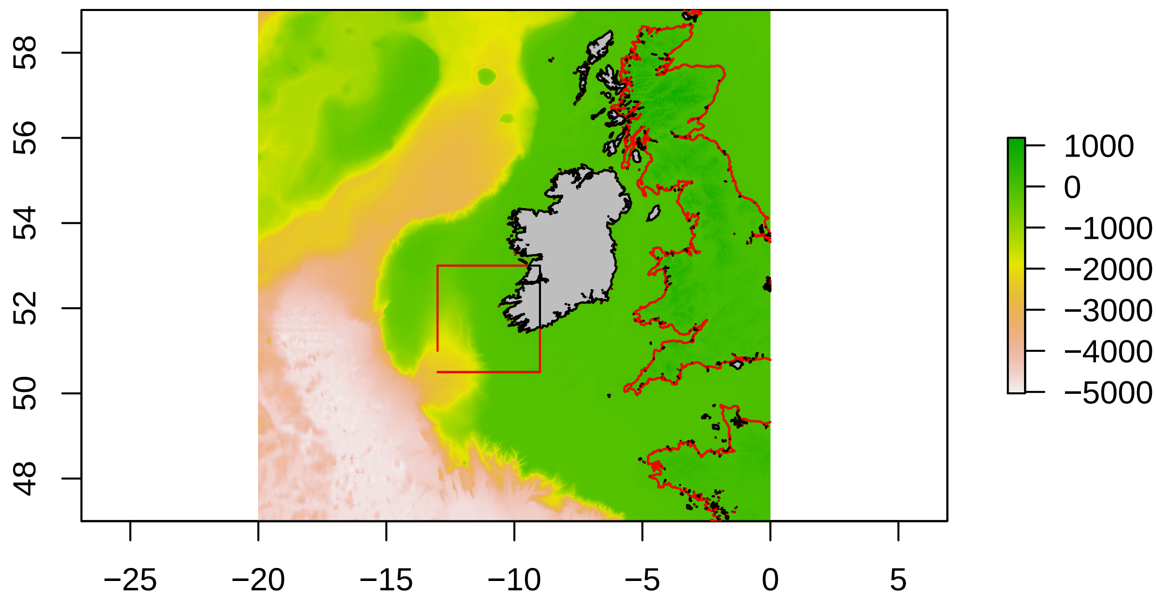
```
library(sp)
library(rgdal)
library(raster)
library(rgeos)
library(rgdal)

depth <- raster('F:\\ResourceBook\\2018\\Bathymetry\\GEBCO_2019_-20.0_59.0_0.0_47.0.nc')
plot(depth)

# make a rectangle with the satial extent of your polygons, note that you should leave a small gap! (he
l1 <- cbind(depth@extent[c(1,2,2,1,1)]+c(.1,-.1,-.1,.1,.1),depth@extent[c(3,3,4,4,3)]+c(.1,.1,-.1,-.1,.
#or
l1 <- cbind(c(-13,-13,-9,-9,-13),c(51,53,53,50.5,50.5))
Sl1 <- Line(l1)
S1 <- Lines(list(Sl1), ID=1)
Sl <- SpatialLines(list(S1))
Sl@proj4string <- depth@crs
plot(Sl,add=T)

a <- rasterToContour(depth,levels=0,maxpixels=1e9)
```

```r
plot(a,add=T)
b <- gUnion(a,Sl)
plot(b,add=T,col=2)
c <- gPolygonize(b)
plot(c,col='grey',add=T)
```




```r
bathy0 <- c

a <- rasterToContour(depth,levels=-50,maxpixels=1e9)
b <- gUnion(a,Sl)
c <- gPolygonize(b)
bathy1 <- c

a <- rasterToContour(depth,levels=-100,maxpixels=1e9)
b <- gUnion(a,Sl)
c <- gPolygonize(b)
bathy2 <- c

col <- c("#FFFFFF","#C5EBF4","#3CBED8","#F5C86D")

xlim <- c(-11.5,-10)
ylim <- c(51,52.5)
  par(mar=rep(0,4))
  mapplots::basemap(xlim=xlim,ylim=ylim,bg=NA)
  plot(bathy2,col=col[2],border=F,add=T)
  plot(bathy1,col=col[3],border=F,add=T)
```

```r
plot(bathy0,col=col[4],border=1,lwd=0.5,add=T)
```

Projections

You can use projections in R, for example you can transform your basic longlat WGS84 coordinate reference system to UTM zone 30N. Check http://spatialreference.org/ to find the relevant proj4 string

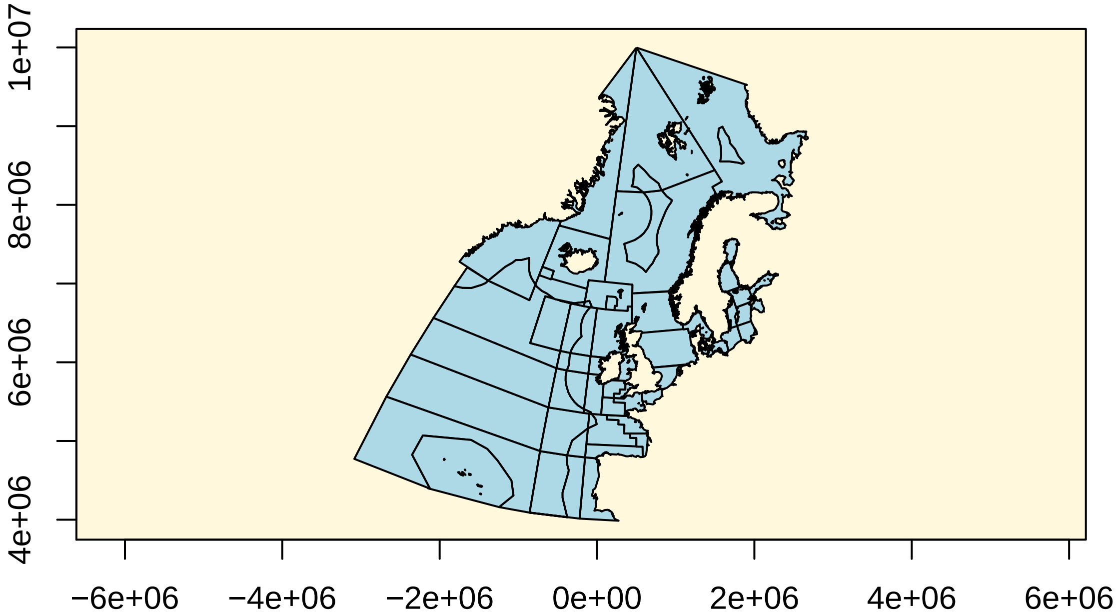
```r
library(rgdal) # for spTransform
library(raster) # for crop
```

First we need to crop the data as some ICES divisions go all the way to the north pole, the projection algoritm doesnt know what to do with that, so we crop the top a bit.

```r
ices1 <- crop(ices,extent(-180,180,0,89.9))
```

Now we transform the projection to UTM zone 30N.

```r
crs <- CRS("+proj=utm +zone=30 +ellps=WGS84 +datum=WGS84 +units=m +no_defs ")
ices2 <- spTransform(ices1, crs)
plot(ices2,col='lightblue',bg='cornsilk',axes=T)
```



Notice that straight lines along the same latitude (the parrallels) are plotted as straight lines while they should be curved in this projection. This is not an R problem, the same happens in ArcGis.

Dealing with ICES rectangles, divisions etc

I wrote the mapplots package to deal with VMS data and to plot things like pie plots in maps. Some of this is obsolete now but there are a few handy functions in the package. One of these deals with ICES rectangles. You can get the midpoint from a vector of rectangles:

```
library(mapplots)
ices.rect(c('29E1','30E1','29E2'))
```

```
##    lon   lat
## 1 -8.5 50.25
## 2 -8.5 50.75
## 3 -7.5 50.25
```

Even when the rectangles have been messed up by Bill Gates who likes to turn rectangles with ‘E’ in them into numbers:

```
ices.rect(c(290,300,2900))
```

```
##    lon   lat
## 1 -8.5 50.25
## 2 -8.5 50.75
## 3 -7.5 50.25
```

It works the other way around as well, for any given position it will tell you the rectangle (but note that it will happily go ‘off the chart’).

```
ices.rect2(lon=c(-12.1,-11.9,170),lat=c(54.3,48.1,-47.6))
```

```
## [1] "37D7" "25D8" "67W0"
```

One more thing when dealing with ICES areas that took me a while to figure out. You may want to find the sub-area that an ICES division belongs to (e.g. Division VIIa is in sub-area VII). So essentially you want to tease out the roman numeral and remove everithing else. This is where regular expressions come in (look for help on the `regexp()` function for more info). Anyway here is the trick: replace every character that is not 'I' or 'V' or 'X' with an empty character ('').

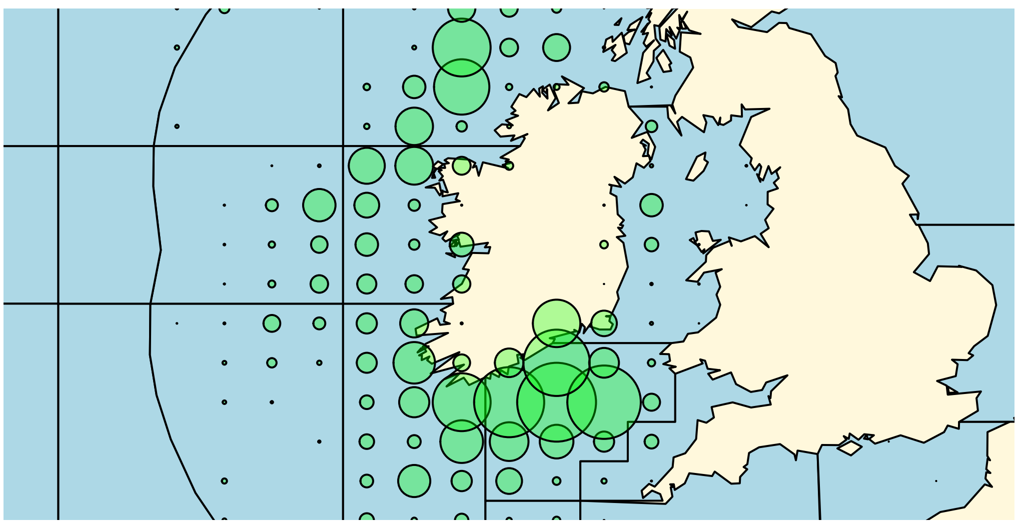
```
gsub('[^IVX]','',c('VIIa','VIIb1','IXb1','28-2'))
```

```
## [1] "VII" "VII" "IX"  ""
```

Other mapplots functions

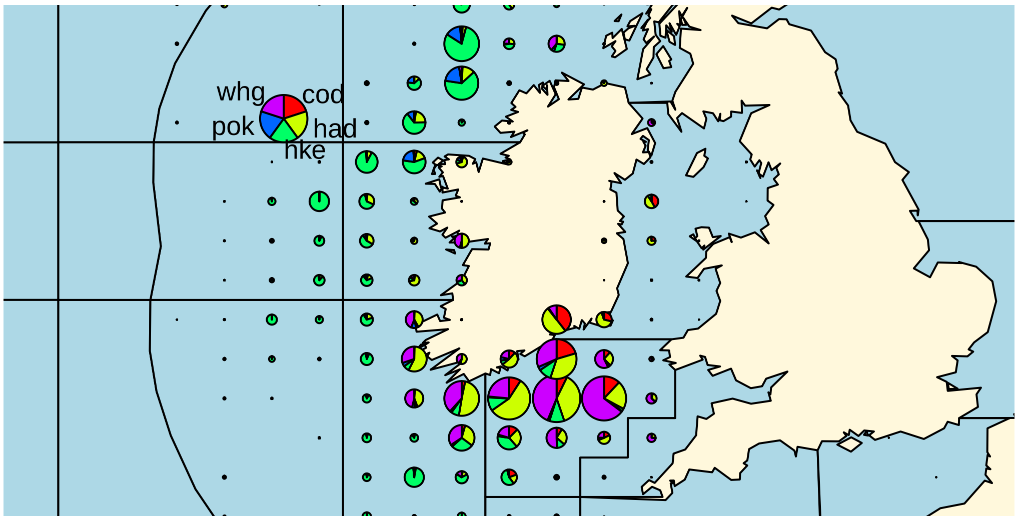
Bubble plots

```
library(mapplots)
data(landings)
xlim <- c(-12,-5)
ylim <- c(50,56)
plot(ices,xlim=xlim,ylim=ylim,col='lightblue',bg='cornsilk')
agg <- aggregate(list(z=landings$LiveWeight),list(x=landings$Lon,y=landings$Lat),sum)
draw.bubble(agg$x, agg$y, agg$z, maxradius=0.5, pch=21, bg="#00FF0050")
```



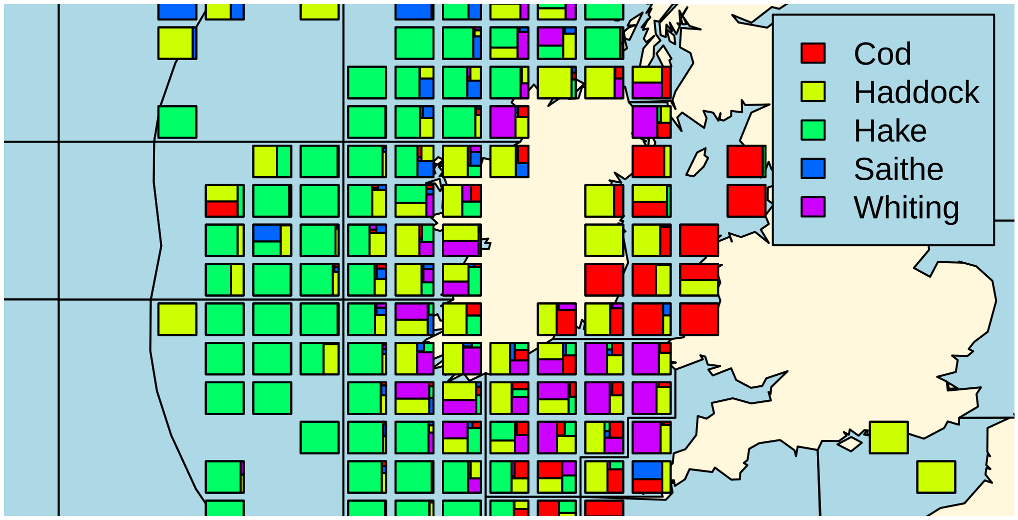
Pie plots

```
plot(ices,xlim=xlim,ylim=ylim,col='lightblue',bg='cornsilk')
xyz <- make.xyz(landings$Lon,landings$Lat,landings$LiveWeight,landings$Species)
col <- rainbow(5)
draw.pie(xyz$x, xyz$y, xyz$z, radius = 0.3, col=col)
legend.pie(-13.25,54.8,labels=c("cod","had","hke","pok","whg"), radius=0.3, bty="n", col=col,
 cex=0.8, label.dist=1.3)
```



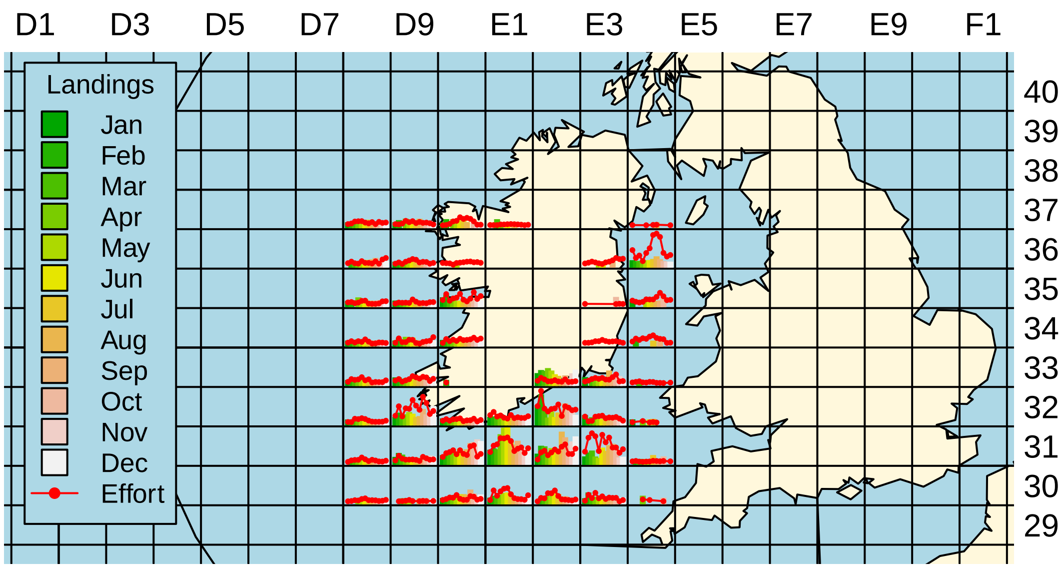
2D barplots

```
plot(ices,xlim=xlim,ylim=ylim,col='lightblue',bg='cornsilk')
xyz <- make.xyz(landings$Lon,landings$Lat,landings$LiveWeight,landings$Species)
draw.barplot2D(xyz$x, xyz$y, xyz$z, width = 0.8, height = 0.4, col=col)
legend("topright", legend=colnames(xyz$z), fill=col, bg="lightblue", inset=0.02)
```



Other plots

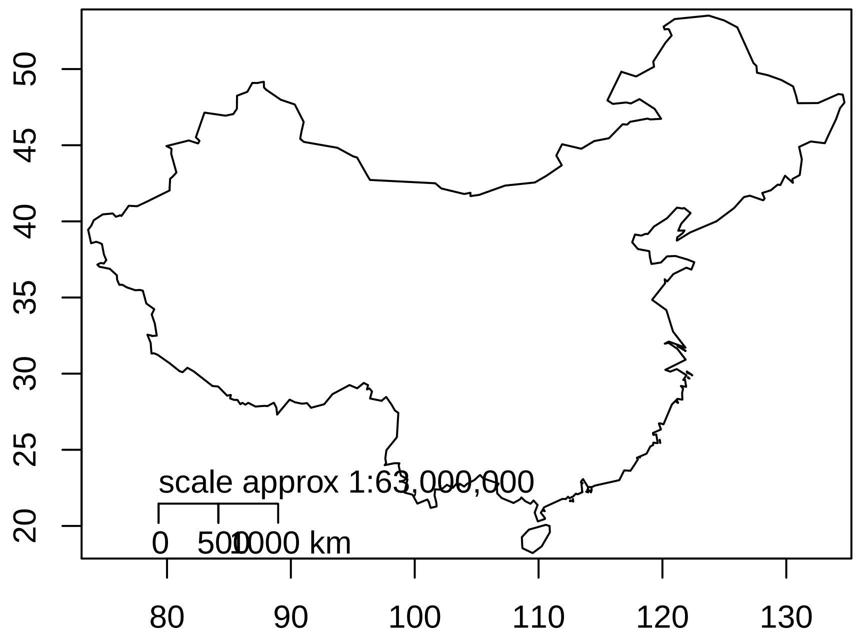
```
data(effort)
plot(ices,xlim=xlim,ylim=ylim,col='lightblue',bg='cornsilk')
col <- terrain.colors(12)
effort$col <- col[match(effort$Month,1:12)]
draw.rect(lty=1, col=1)
draw.xy(effort$Lon, effort$Lat, effort$Month, effort$LiveWeight, width=1, height=0.5,
 col=effort$col, type="h",lwd=3, border=NA)
draw.xy(effort$Lon, effort$Lat, effort$Month, effort$Effort, width=1, height=0.5, col="red",
 type="l", border=NA)
draw.xy(effort$Lon, effort$Lat, effort$Month, effort$Effort, width=1, height=0.5, col="red",
 type="p",cex=0.4,pch=16, border=NA)
legend("topleft", c(month.abb,"Effort"), pch=c(rep(22,12),16), pt.bg=c(col,NA),
 pt.cex=c(rep(2,12),0.8),col=c(rep(1,12),2), lty=c(rep(NA,12),1), bg="lightblue",
 inset=0.02, title="Landings", cex=0.8)
```



maps package

The maps package has a number of built-in maps of countries and cities and a function to plot these. mapdata has additional maps (also higher resolution maps)

```
library(maps)
map("world", "China")
map.scale()
map.axes()
```



ggmap

The package ggmap has some nice features like importing google satellite images. Have a look at these sites for some more examples: https://www.nceas.ucsb.edu/~frazier/RSpatialGuides/ggmap/ggmapCheatsheet.pdf and https://journal.r-project.org/archive/2013-1/kahle-wickham.pdf

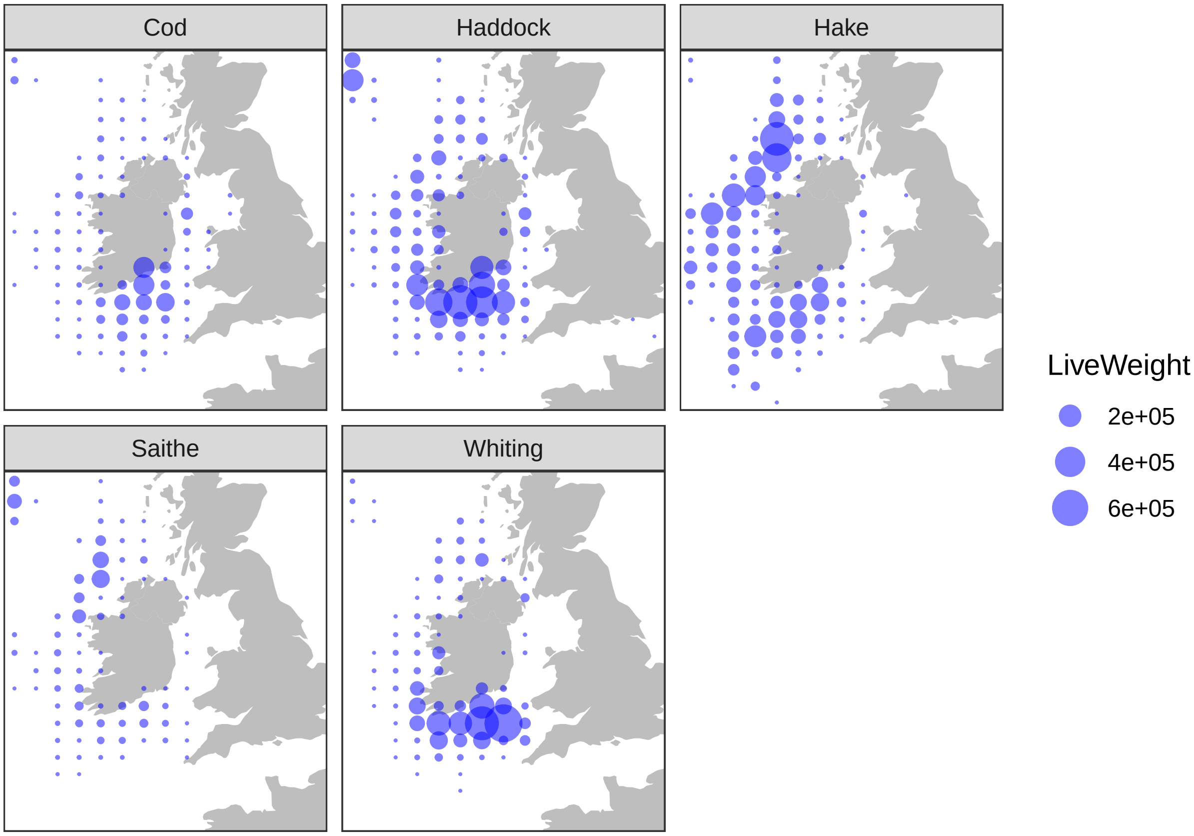
ggplot

You can also plot maps directly in ggplot

```
library(ggplot2)
world <- map_data('world')
world <- subset(world,region%in%c('Ireland','UK','France','Spain'))

maptheme <- theme_bw() +
  theme(line=element_blank(), axis.text=element_blank(), axis.title=element_blank())

ggplot(landings,aes(Longitude,Latitude)) +
  geom_polygon(aes(long,lat,group=group),world,fill='grey') +
  coord_map('mercator',xlim=c(-14,1),ylim=c(48,58)) +
  geom_point(aes(size=LiveWeight),col='blue',alpha=0.5) +
  scale_size_area() +
  facet_wrap(~Species) +
  maptheme
```



kml

<http://gsif.isric.org/doku.php?id=wiki:tutorial_plotkml?