# rworldmap FAQ

Andy South*

December 13, 2010

HOW DO I ...

## Contents



*Centre for Environment, Fisheries and Aquaculture Science (Cefas), Lowestoft, NR33 OHT, UK. `andy.south at cefas.co.uk; southandy at gmail.com`

# 1 find out what rworldmap is ?

rworldmap is an R package for visualising global scale data, concentrating on data referenced by country codes or gridded at half degree resolution. http://cran.r-project.org/web/packages/rworldmap/index.html

# 2 install `rworldmap` ?

To install rworldmap from R, including other required packages :
install.packages('rworldmap',dependencies=TRUE)

Alternatively download from :
http://cran.r-project.org/web/packages/rworldmap/index.html

# 3 load the package into R after installation ?

Package **`rworldmap`** must be loaded into R at the start of each session by either of the following 2 lines :

```
> require(rworldmap)
> library(rworldmap)
```

# 4 access latest version of rworldmap source code ?

http://code.google.com/p/rworld/downloads/list

# 5 access this FAQ ?

From within R :
vignette('rworldmapFAQ')
From the web :
http://cran.r-project.org/web/packages/rworldmap/rworldmapFAQ.pdf

# 6 map my own country level data ?

To map your own data you will need it in columns with one row per country, one column containing country identifiers, and other columns containing your data.

The mapping process then involves 3 steps (or 2 if your data are already in an R dataframe).

1. read data into R
2. join data to a map ( using `joinCountryData2Map()` )
3. display the map ( using `mapCountryData()` )

There is an example dataset within the package that can be accessed using the data command, and the command below shows how to display a subset of the rows and columns.

```
> data(countryExData)
> countryExData[5:10, 1:5]
```

```
   ISO3V10    Country                  EPI_regions
5      ARM    Armenia Middle East and North Africa
6      AUS  Australia    East Asia and the Pacific
7      AUT    Austria                       Europe
8      AZE Azerbaijan    Central and Eastern Europ
9      BDI    Burundi           Sub-Saharan Africa
10     BEL    Belgium                       Europe
             GEO_subregion Population2005
5           Eastern Europe         3016.3
6  Australia + New Zealand        20155.1
7           Western Europe         8189.4
8           Eastern Europe         8410.8
9           Eastern Africa         7547.5
10          Western Europe        10419.1
```

## 6.1 Reading data into R

To read in your own data from a space or comma delimited text file you will need to use : `read.csv(filename.csv)` or `read.txt(filename.txt)`, type `?read.table` from the R console to get help on this.

## 6.2 Joining data to a country map

To join the data to a map use `joinCountryData2Map`, and you will need to specify the name of column containing your country identifiers (nameJoinColumn) and the type of code used (joinCode) e.g. "ISO3" for ISO 3 letter codes or "UN" for numeric country codes. If you only have country names rather than codes use joinCode="NAME", you can expect more mismatches because there is greater variation in what a single country may be named.

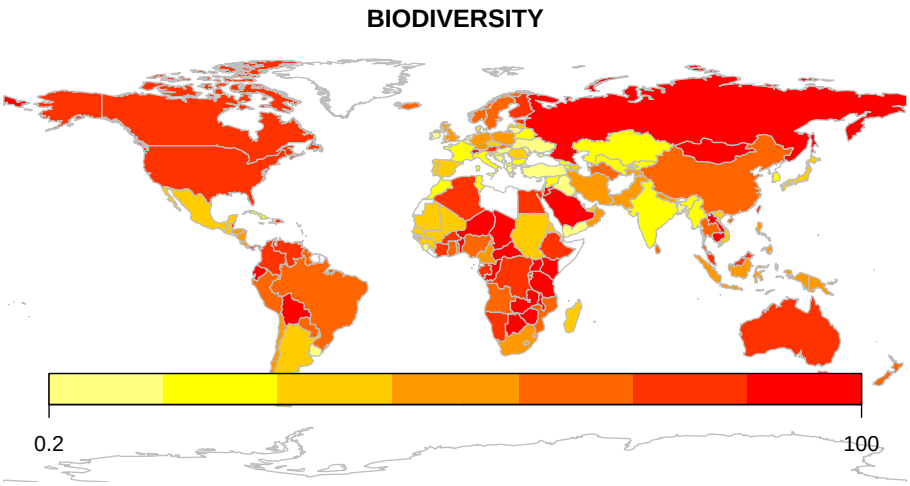
```
> data(countryExData)
> sPDF <- joinCountryData2Map(countryExData, joinCode = "ISO3",
+     nameJoinColumn = "ISO3V10")

149 codes from your data successfully matched countries in the map
0 codes from your data failed to match with a country code in the map
97 codes from the map weren't represented in your data
```

You can see that a summary of how many countries are successfully joined is output to the console. You can specify verbose=TRUE to get a full list of countries. The object returned (named sPDF in this case) is of type `SpatialPolygonsDataFrame` from the package `sp`. This object is required for the next step, displaying the map.

## 6.3 Displaying a countries map

`mapCountryData` requires as a minimum a `SpatialPolygonsDataFrame` object and a specification of the name of the column containing the data to plot. The first line starting par ... below and in subsequent plots simply ensures the plot fills the available space on the page.

```
> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> mapCountryData(sPDF, nameColumnToPlot = "BIODIVERSITY")
```




In this small map the default legend is rather large. This could be fixed by calling the addMapLegend function as in the code below.

```
> mapParams <- mapCountryData(sPDF, nameColumnToPlot = "BIODIVERSITY",
+     addLegend = FALSE)
> do.call(addMapLegend, c(mapParams, legendWidth = 0.5,
+     legendMar = 2))
```

Using `do.call` allows the output from mapCountryData to be used in addMapLegend to ensure the legend matches the map while also allowing easy modification of extra parameters such as `legendWidth`.

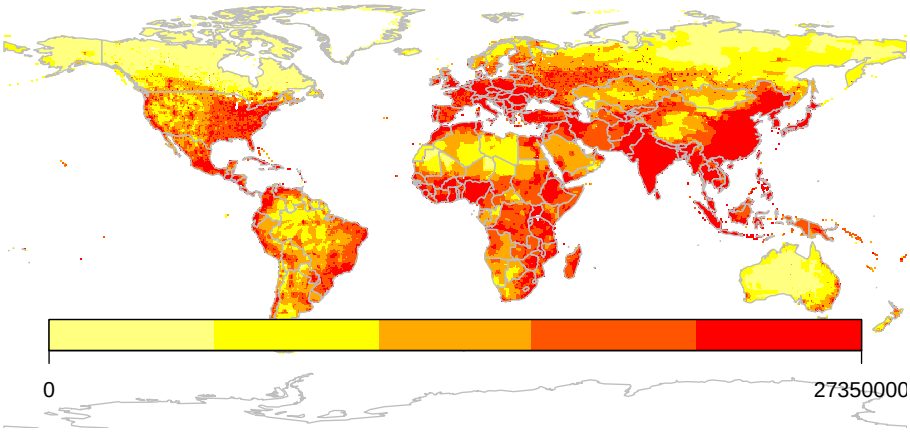
# 7 map my own half degree gridded data ?

The `mapGriddedData` function can accept either

1. an object of type `SpatialGridDataFrame`, as defined in the package `sp`
2. the name of an ESRI gridAscii file as a character string

rworldmap contains an example SpatialGridDataFrame that can be accessed and printed as shown in the code below.

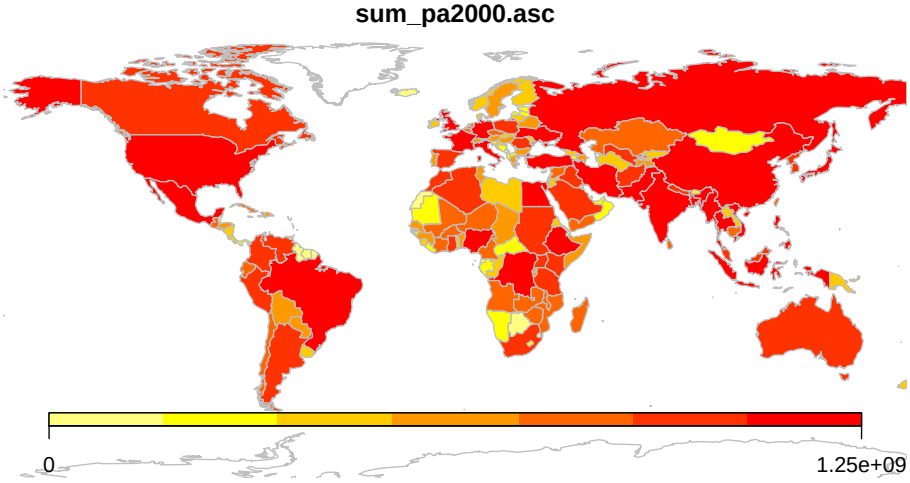
```
> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> data(gridExData)
> mapGriddedData(gridExData)
```




# 8 aggregate half degree gridded data to countries ?

mapHalfDegreeGridToCountries() takes a gridded input file, and aggregates, to a country level and plots the map, it accepts most of the same arguments as `mapCountryData()`. In the example below the trick from above of modifying the legend using addMapLegend() is repeated.

```
> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> mapParams <- mapHalfDegreeGridToCountries(gridExData,
+     addLegend = FALSE)
> do.call(addMapLegend, c(mapParams, legendWidth = 0.5,
+     legendMar = 2))
```

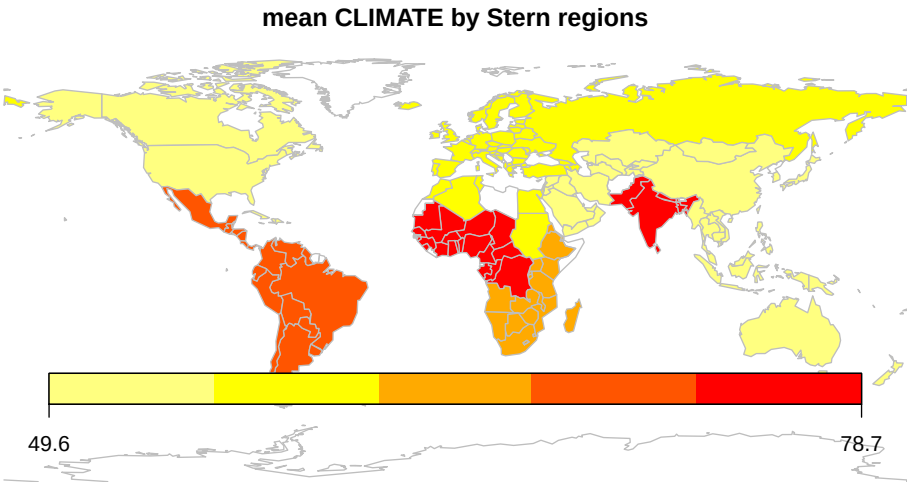



# 9 aggregate country level data to global regions ?

Country level data can be aggregated to global regions specified by `regionType` in `country2Region` which outputs as text, and `mapByRegion` which produces a map plot. The regional classifications available include SRES, GEO3, Stern and GBD.

```
> sternEnvHealth <- country2Region(inFile = countryExData,
+     nameDataColumn = "ENVHEALTH", joinCode = "ISO3",
+     nameJoinColumn = "ISO3V10", regionType = "Stern",
```

```
+     FUN = "mean")
> print(sternEnvHealth)

                meanENVHEALTHbyStern
Australasia                 78.86000
Caribbean                   82.18000
Central America             82.78750
Central Asia                77.24000
East Asia                   75.52308
Europe                      95.19762
North Africa                77.38000
North America               98.70000
South America               83.62727
South Asia                  61.96000
South+E Africa              49.06316
West Africa                 36.99474
West Asia                   82.78000

> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> mapByRegion(countryExData, nameDataColumn = "CLIMATE",
+     joinCode = "ISO3", nameJoinColumn = "ISO3V10", regionType = "Stern",
+     FUN = "mean")
```




# 10 alter the appearance of my maps ?

The following arguments can be specified to alter the appearance of your plots.

- `catMethod` method for categorisation of data ”pretty”, ”fixedWidth”, ”diverging”,”logfixedWidth”,”quantiles”,”cat
  or a numeric vector defining breaks.
- `numCats` number of categories to classify the data into, may be modified if that exact number
  is not possible for the chosen catMethod.
- `colourPalette` a string describing the colour palette to use, choice of :
  1. ”palette” for the current palette
  2. a vector of valid colours, e.g. c(”red”,”white”,”blue”) or output from RColourBrewer
  3. one of ”heat”, ”diverging”, ”white2Black”, ”black2White”, ”topo”, ”rainbow”, ”terrain”,
     ”negpos8”, ”negpos9”

- `addLegend` set to TRUE for a default legend, if set to FALSE the function addMapLegend() or addMapLegendBoxes() can be used to create a more flexible legend.
- `mapRegion` a region to zoom in on, can be set to a country name from getMap()$NAME or one of ”eurasia”,”africa”,”latin america”,”uk”,”oceania”,”asia”

# 11 create my own colour palette ?

```
> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> sPDF <- joinCountryData2Map(countryExData, joinCode = "ISO3",
+     nameJoinColumn = "ISO3V10", projection = "none",
+     )
> op <- palette(c("green", "yellow", "orange", "red"))
> cutVector <- quantile(sPDF@data[["BIODIVERSITY"]], na.rm = TRUE)
> sPDF@data[["BIOcategories"]] <- cut(sPDF@data[["BIODIVERSITY"]],
+     cutVector, include.lowest = TRUE)
> levels(sPDF@data[["BIOcategories"]]) <- c("low", "med",
+     "high", "vhigh")
> mapCountryData(sPDF, nameColumnToPlot = "BIOcategories",
+     catMethod = "categorical", mapTitle = "Biodiversity categories",
+     colourPalette = "palette", oceanCol = "lightblue",
+     missingCountryCol = "white")
```

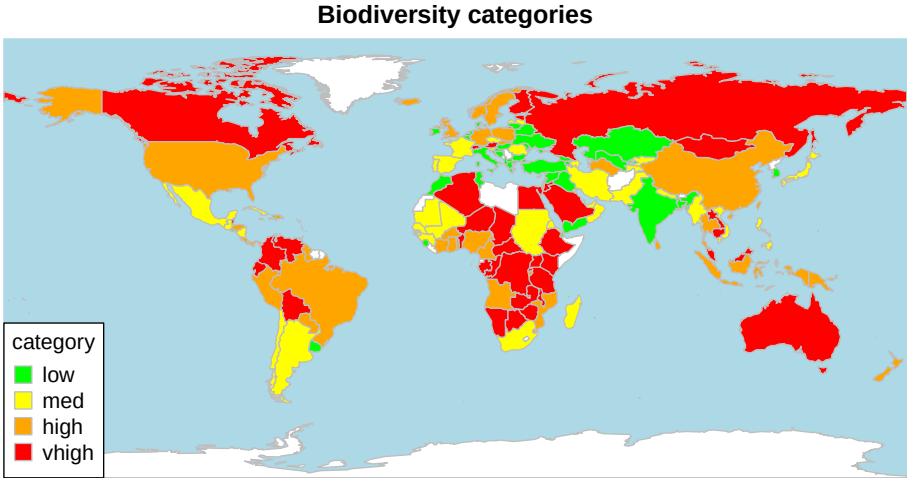


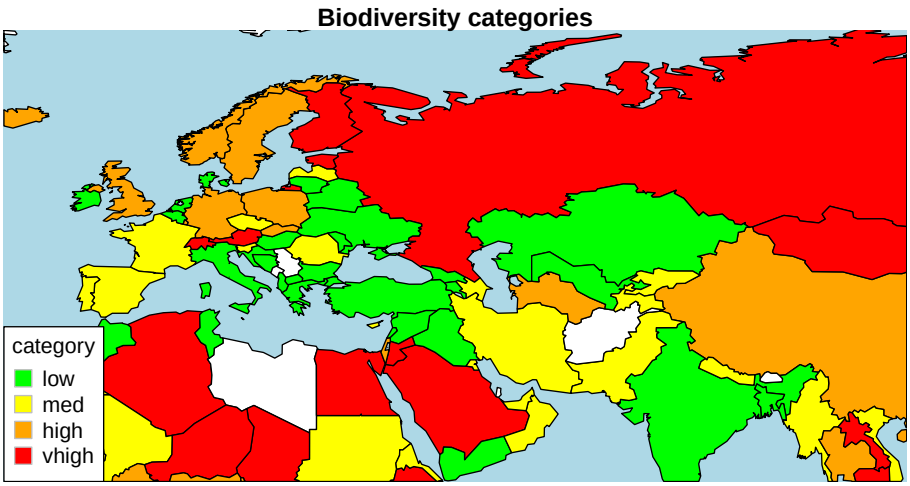

Figure 1: An example of a categorical map produced from mapCountryData

# 12 zoom in on defined regions ?

You can zoom in on a map by specifying mapRegion=”Eurasia” (or by specifiying xlim and ylim) and the country outlines can be changed by borderCol=”black”.

```
> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> mapCountryData(sPDF, nameColumnToPlot = "BIOcategories",
+     catMethod = "categorical", mapTitle = "Biodiversity categories",
+     colourPalette = "palette", oceanCol = "lightblue",
+     missingCountryCol = "white", mapRegion = "Eurasia",
```

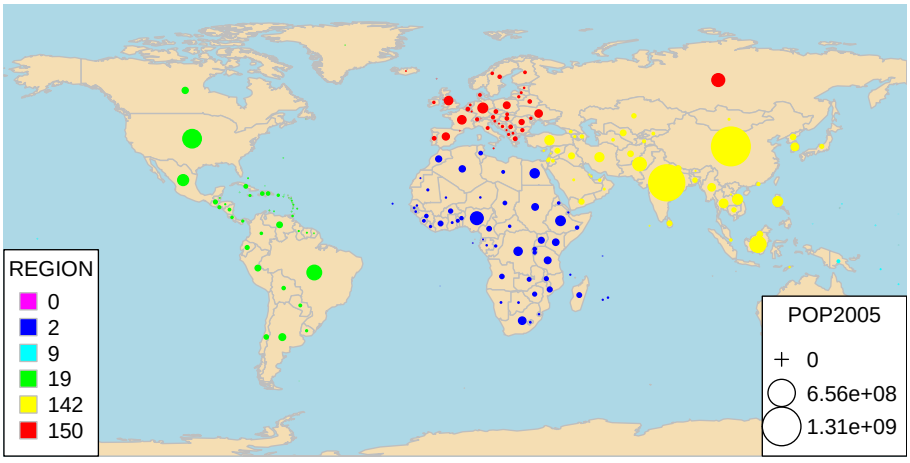
```
+      borderCol = "black")
> palette(op)
```




## 13 create map bubble plots ?

The `mapBubbles` function allows flexible creation of bubble plots on global maps. You can specifiy data columns that will determine the sizing and colouring of the bubbles (using `nameZsize` and `nameZColour` ). The function also accepts other spatialDataFrame objects or data frames as long as they contain columns specifiying the x and y coordinates. The interactive function `identifyCountries` allows the user to click on bubbles and the country name and optionally an attribute variable will be printed on the map.

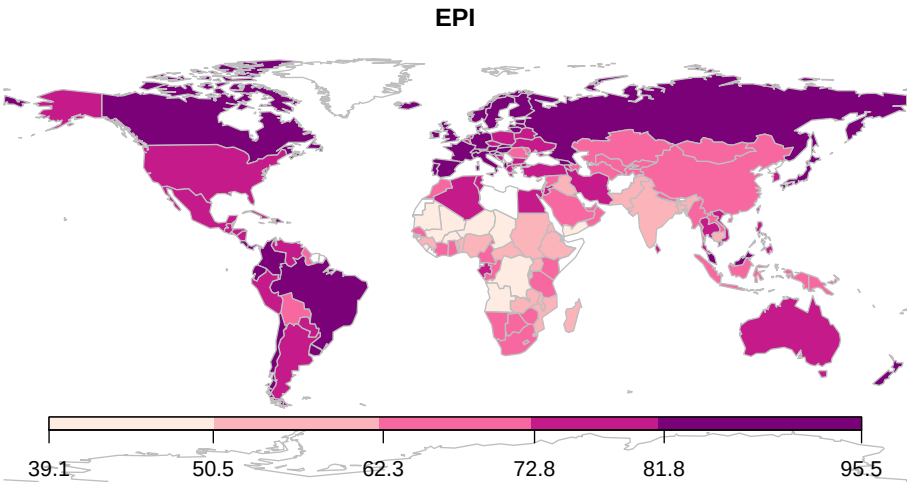
```
> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> mapBubbles(dF = getMap(), nameZSize = "POP2005", nameZColour = "REGION",
+      colourPalette = "rainbow", oceanCol = "lightblue",
+      landCol = "wheat")
```



# 14 combine rworldmap with other packages classInt and RColorBrewer ?

Whilst rworldmap sets many defaults internally there is also an option to use other packages to have greater flexibility. In this example the package classInt is used to create the classification and RColorBrewer to specify the colours. The following page demonstrates how multiple maps can be generated in the same figure and shows a selection of different RColorBrewer palettes.

```
> par(mai = c(0, 0, 0.2, 0), xaxs = "i", yaxs = "i")
> library(classInt)
> library(RColorBrewer)
> data("countryExData", envir = environment(), package = "rworldmap")
> sPDF <- joinCountryData2Map(countryExData, joinCode = "ISO3",
+     nameJoinColumn = "ISO3V10", mapResolution = "coarse")
> classInt <- classIntervals(sPDF[["EPI"]], n = 5, style = "jenks")
> catMethod = classInt[["brks"]]
> colourPalette <- brewer.pal(5, "RdPu")
> mapParams <- mapCountryData(sPDF, nameColumnToPlot = "EPI",
+     addLegend = FALSE, catMethod = catMethod, colourPalette = colourPalette)
> do.call(addMapLegend, c(mapParams, legendLabels = "all",
+     legendWidth = 0.5, legendIntervals = "data", legendMar = 2))
```




# 15 ensure plots fill the panel space available?

Use par(mar=c(bottom,top,left,right)) to set margins. This returns the previous settings so you can use oldPar <- par(...) then par(oldPar) to reset.

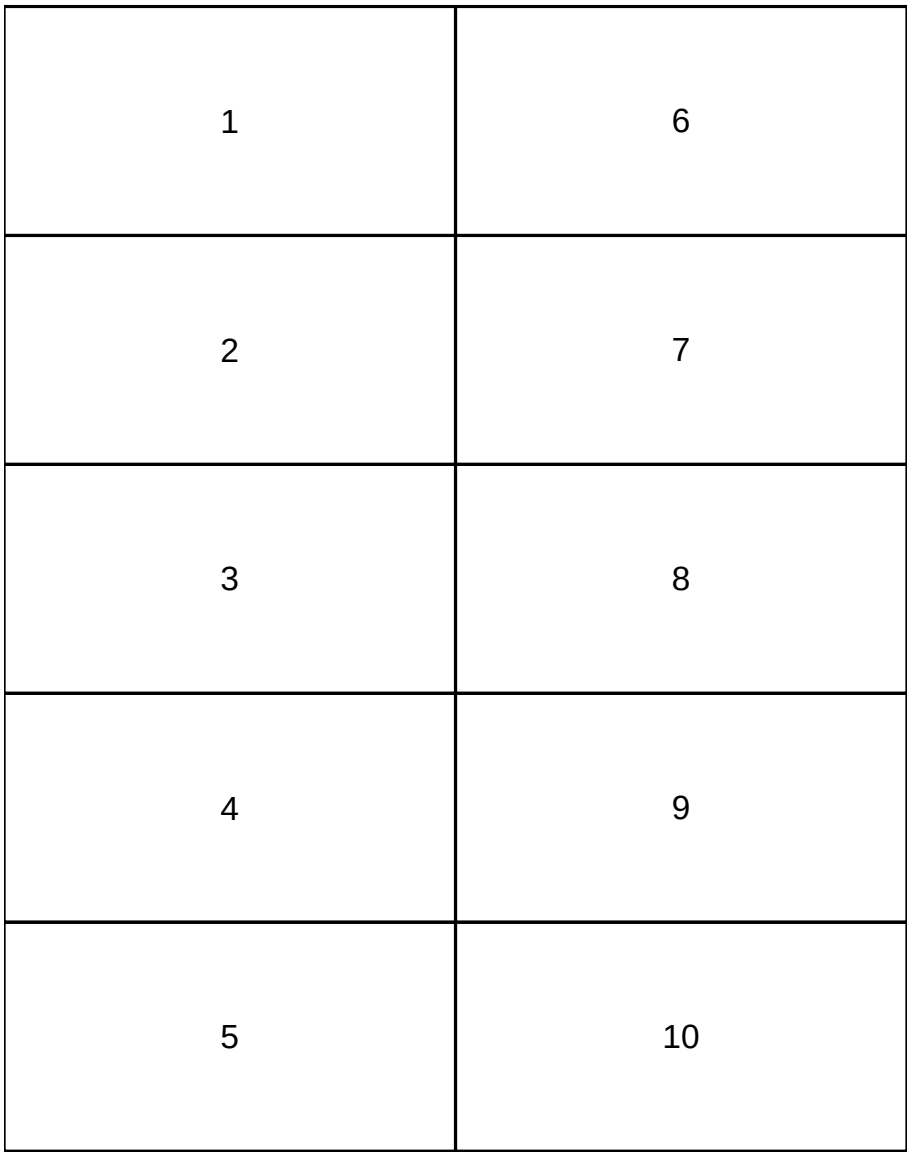
```
> oldPar <- par(mar = c(0, 0, 0, 0))
> par(oldPar)
```

# 16 create multi-panel plots ?

using the layout() command as shown below, layout.show() indicates how the panels are arranged Beware that the colour bar legends used when addLegend=TRUE can interfere with this ordering (addLegend=FALSE or addMapLegendBoxes() are OK)

Creating 2 columns 5 rows with a 0.5cm gap at the top

```
> oldPar <- par(mar = c(0, 0, 0, 0))
> nPanels <- layout(cbind(c(0, 1, 2, 3, 4, 5), c(0, 6,
+     7, 8, 9, 10)), heights = c(lcm(0.5), 1, 1, 1, 1,
+     1), respect = F)
> layout.show(nPanels)
> par(oldPar)
```




Creating 3 columns 4 rows (with a gap at the top) appropriate for showing monthly data

```
> oldPar <- par(mar = c(0, 0, 0, 0))
> nPanels <- layout(rbind(c(0, 0, 0), c(1, 2, 3), c(4,
+     5, 6), c(7, 8, 9), c(10, 11, 12)), heights = c(lcm(0.5),
+     1, 1, 1, 1), respect = F)
> layout.show(nPanels)
> par(oldPar)
```

| | | |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | 6 |
| 7 | 8 | 9 |
| 10 | 11 | 12 |

## 17 add lines of latitude and longitude to a map ?

For the latitude longitude projection used in most rworldmap maps the following adds respectively : 1) Equator 2) Greenwich meridian 3) Tropics of capricorn and cancer as dashed grey lines

```
> abline(h = 0)
> abline(v = 0)
> abline(h = c(-20, 20), lty = 2, col = "grey")
```