rworldmap vignette

Andy South*

November 12, 2014

rworldmap is a package for visualising global data, concentrating on data referenced by country codes or gridded at half degree resolution.

Funded by :

Contents

*Norwich, UK. southandy at gmail.com

1 Introduction

Package `rworldmap` is loaded by

```
> library(rworldmap)
```

This vignette shows a few examples of the main rworldmap functions to get you started. To access the full help system, type ?rworldmap in the R console. The functions are designed to operate with few specified parameters in which case default values are used, but can also accept user input to allow flexibility e.g. in size, data categorisation, and colour schemes.

```
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
95 codes from the map weren't represented in your data
```

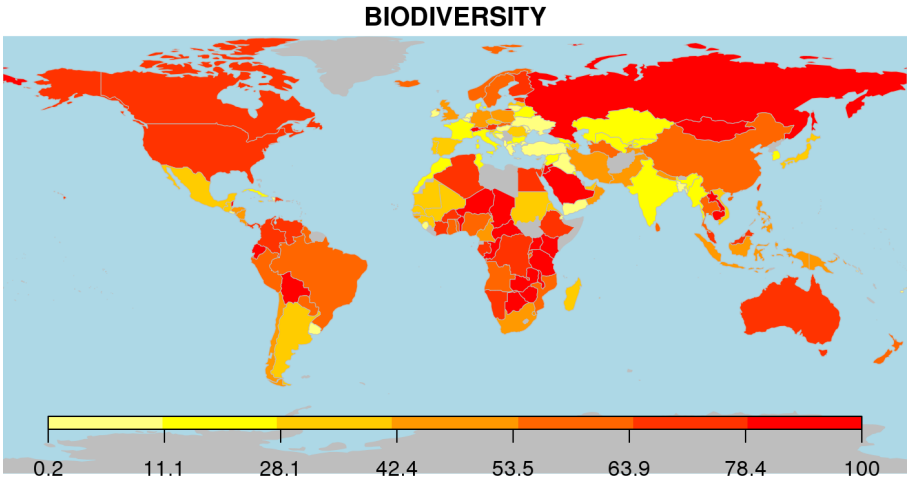


Figure 1: An example from mapCountryData

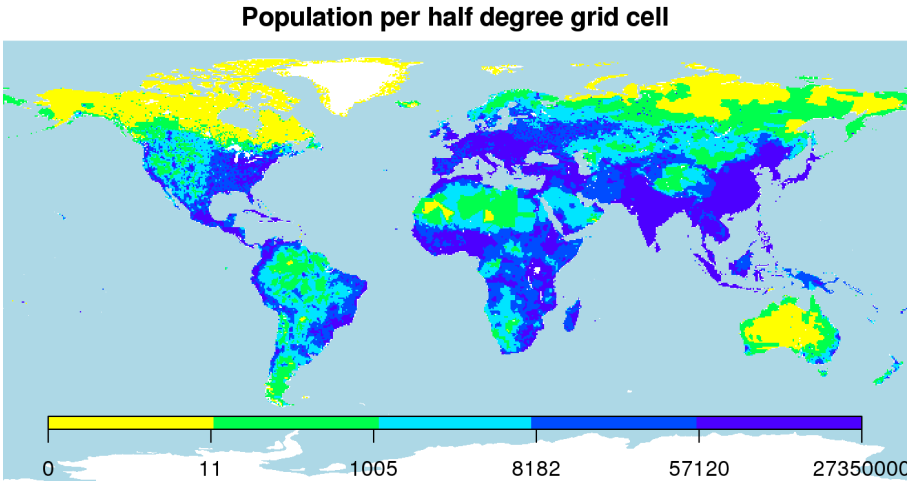


Figure 2: An example from mapGriddedData

2 Mapping your 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
```

2.1 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.

2.2 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.

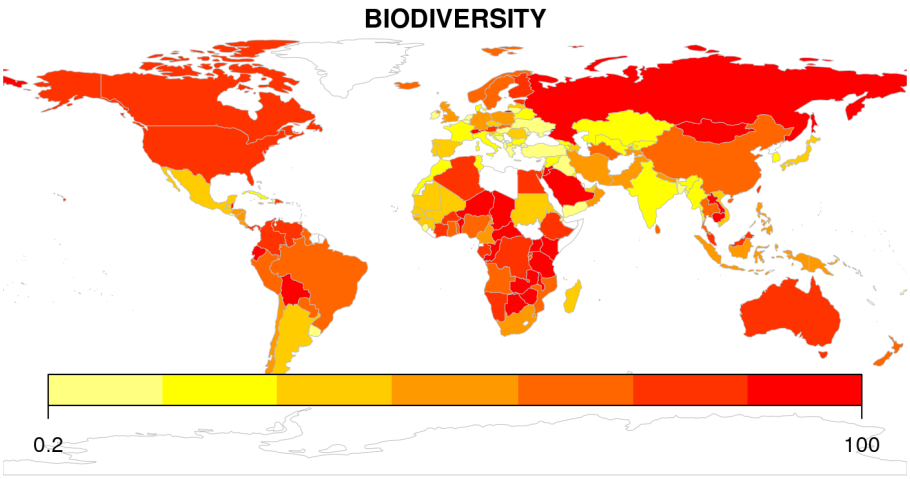
```
> data(countryExData)
> sPDF <- joinCountryData2Map( countryExData
+                            , joinCode = "ISO3"
+                            , nameJoinColumn = "ISO3V10")

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
95 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.

2.3 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`.

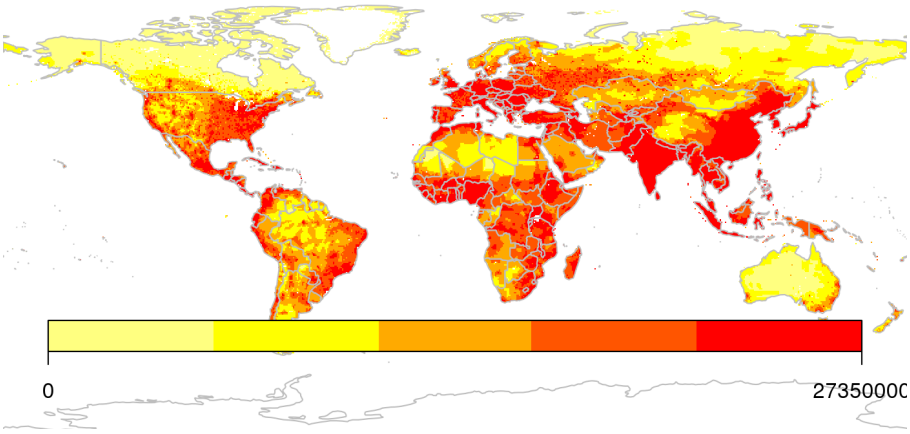
3 Mapping your own half degree gridded data

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
3. a 2D R matrix or array (rows by columns)

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

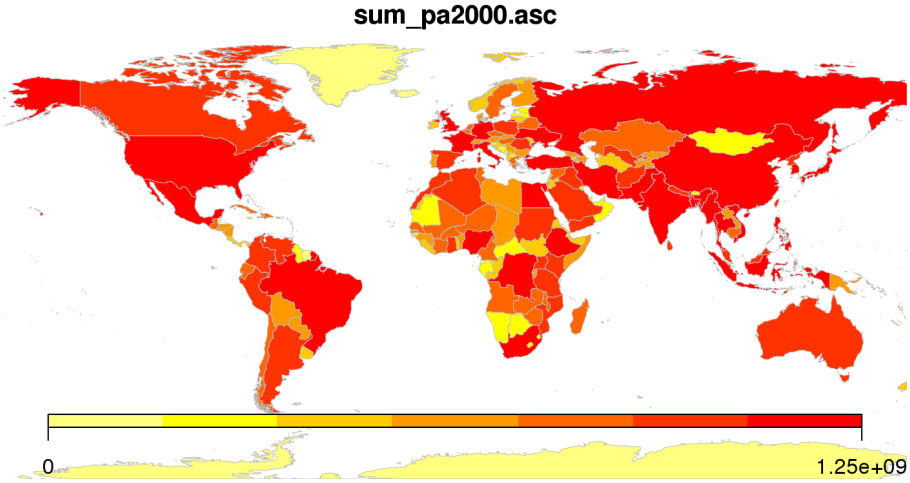
```
> par(mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
> data(gridExData)
> mapGriddedData(gridExData)
```



4 Aggregating half degree gridded data to a country level

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))
```

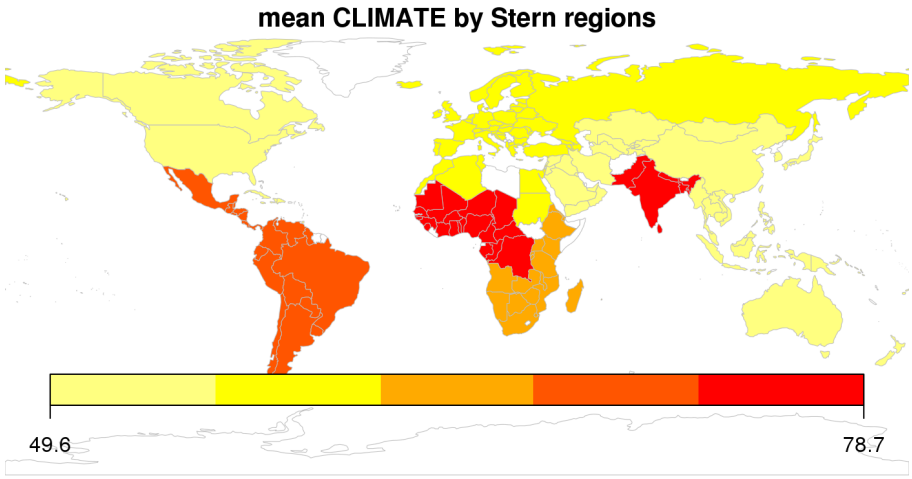


5 Aggregating country level data to global regions

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.

```
> #Using country2Region to calculate mean ENVHEALTH in Stern regions.
> 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"
+             )
```



6 Map display options common across the plotting methods

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

- `catMethod` method for categorisation of data ”pretty”, ”fixedWidth”, ”diverging”,”logFixedWidth”,”quantiles”,”categorical”, 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”

7 Example maps with settings modified

This demonstrates how continuous data can be put into categories outside of the rworldmap functions and how a user defined colour palette can be used. Because the catMethod="categorical" was used a legend with separate boxes rather than a colour bar is added.

```
> par(mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
> #joining the data to a map
> sPDF <- joinCountryData2Map(countryExData,
+               joinCode = "ISO3",
+               nameJoinColumn = "ISO3V10"
+               )
> #creating a user defined colour palette
> op <- palette(c('green','yellow','orange','red'))
> #find quartile breaks
> cutVector <- quantile(sPDF@data[["BIODIVERSITY"]],na.rm=TRUE)
> #classify the data to a factor
> sPDF@data[["BIOcategories"]] <- cut(sPDF@data[["BIODIVERSITY"]]
+                                     , cutVector
+                                     , include.lowest=TRUE )
> #rename the categories
> levels(sPDF@data[["BIOcategories"]]) <- c('low', 'med', 'high', 'vhigh')
> #mapping
> mapCountryData( sPDF
+               , nameColumnToPlot='BIOcategories'
+               , catMethod='categorical'
+               , mapTitle='Biodiversity categories'
+               , colourPalette='palette'
+               , oceanCol='lightblue'
+               , missingCountryCol='white'
+               )
```

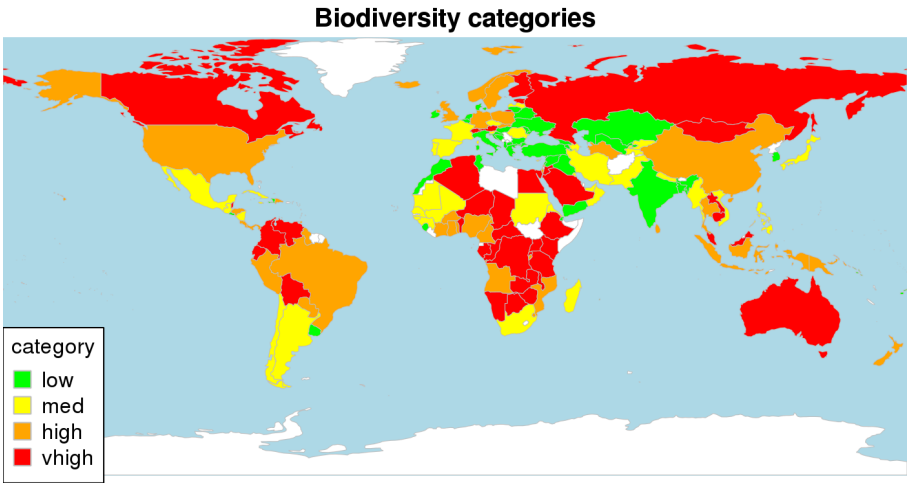
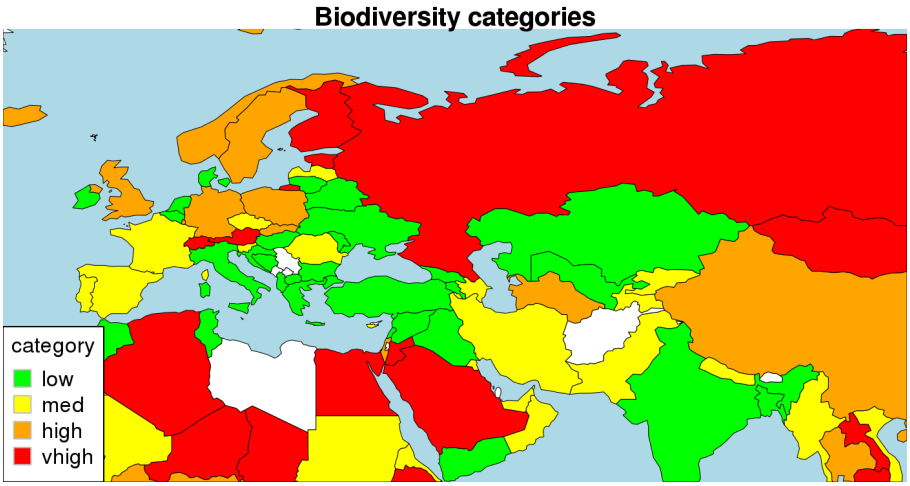


Figure 3: An example of a categorical map produced from mapCountryData

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”.

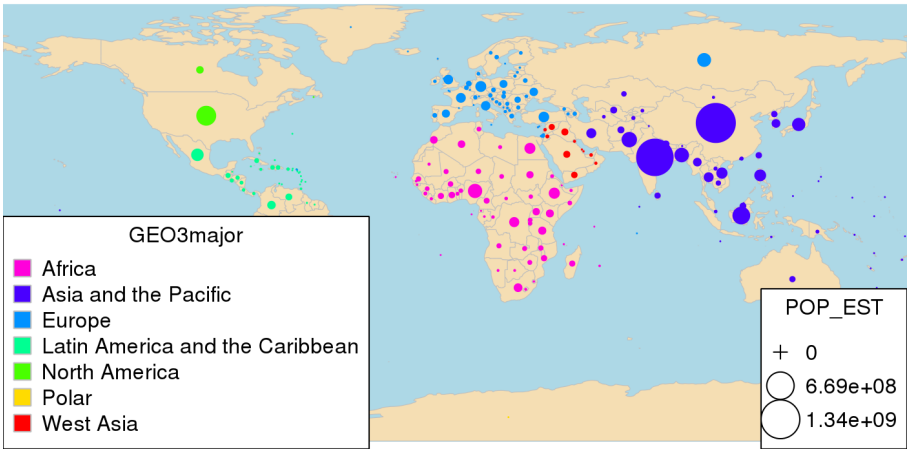
```
> par(mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
> mapCountryData( sPDF
+                , nameColumnToPlot='BIOcategories'
+                , catMethod='categorical'
+                , mapTitle='Biodiversity categories'
+                , colourPalette='palette'
+                , oceanCol='lightblue'
+                , missingCountryCol='white'
+                , mapRegion='Eurasia'
+                , borderCol='black'
+                )
> ## At end of plotting, reset palette to previous settings:
> palette(op)
>
```



8 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.

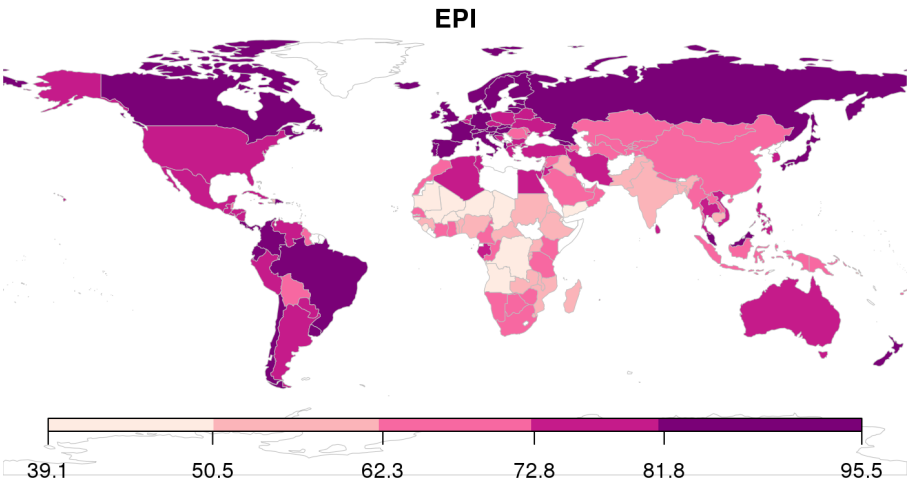
```
> par(mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
> mapBubbles( dF=getMap()
+           , nameZSize="POP_EST"
+           , nameZColour="GEO3major"
+           , colourPalette='rainbow'
+           , oceanCol='lightblue'
+           , landCol='wheat'
+           )
>
```



9 Combining 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(RColorBrewer)
> library(classInt)
> #getting example data and joining to a map
> data("countryExData",envir=environment(),package="rworldmap")
> sPDF <- joinCountryData2Map( countryExData
+                            , joinCode = "ISO3"
+                            , nameJoinColumn = "ISO3V10"
+                            , mapResolution='coarse'
+                            )
> #getting class intervals using a 'jenks' classification in classInt package
> classInt <- classIntervals( sPDF[["EPI"]], n=5, style="jenks")
> catMethod = classInt[["brks"]]
> #getting a colour scheme from the RColorBrewer package
> colourPalette <- brewer.pal(5,'RdPu')
> #calling mapCountryData with the parameters from classInt and RColorBrewer
> 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 ) )
```


EPI
39.1
50.5
62.3
72.8
81.8
95.5

```
> #uses sPDF from the previous chunk
>
> #10 frames July 2013 started getting unableto allocate bitmap error
> #op <- par(fin=c(7,9),mfcol=c(5,2),mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
> #reducing to 8 frames to try to avoid memory error
> op <- par(fin=c(7,9),mfcol=c(4,2),mai=c(0,0,0.2,0),xaxs="i",yaxs="i")
> brewerList <- c("Greens","Greys","Oranges","OrRd"
+                 ,"PuBuGn","Purples","YlGn","YlGnBu","YlOrBr","YlOrRd")
> #for(i in 1:10)
> for(i in 1:8)
+    {
+     #getting a colour scheme from the RColorBrewer package
+     colourPalette <- brewer.pal(7,brewerList[i])
+
+     #calling mapCountryData with the parameters from RColorBrewer
+     mapParams <- mapCountryData( sPDF
+                                , nameColumnToPlot="CLIMATE"
+                                , addLegend=FALSE
+                                , colourPalette=colourPalette
+                                , mapTitle=brewerList[i] )
+     do.call( addMapLegend
+            , c(mapParams,horizontal=FALSE,legendLabels="none",legendWidth=0.7))
+    }
> par(op)
```