Technical Guideline Series

Contents

Part 2 - Preparing and mapping data

to IPBES Regions and Sub-regions

Prepared by Joy Kumagai - Technical Support Unit of Knowledge and Data
Reviewed by Aidin Niamir - Head of the Technical Support Unit of Knowledge and Data
For any inquires please contact tsu.data@ipbes.net

Version: 2.3
Last Updated: 15 August 2022

DOI: 10.5281/zenodo.6992546

The guide will show how to aggregate and map FAO data according to the IPBES Regions and Sub-Regions polygons using R. For this exercise, we chose the FAO population data but any FAOSTAT dataset can be used.

Let's begin by loading the following packages.

```r
library(sf)
library(dplyr)
library(magrittr)
library(FAOSTAT)
library(httr) # to download data off of Zenodo
library(rnaturalearth) # download ocean data from natural earth
library(graticule) # for mapping
```

I. Downloading Necessary Data

A. Downloading FAO data The first step is to download the FAO data using the FAOSTAT package. The url can be found in FAO STAT's data description file here.

```
FAOSTAT::download_faostat_bulk("http://fenixservices.fao.org/faostat/static/bulkdownloads/Population_E_
```

B. Downloading IPBES regions and subregions Now we will download the shapefile of the IPBES Regions and Sub-regions off of Zenodo. This can be accomplished manually or through a few lines of code.

To download the shapefile manually, please go to the IPBES Regions and Sub-Regions Zenodo entry.

To do this through a script, first identify the record ID of the Zenodo entry, which is the numbers following "*zenodo.*" at the end of the URL. We then create a URL with the record ID and query the API for information about the record.

```
recordID <- "3923633"
url_record <- paste0("https://zenodo.org/api/records/", recordID)
record <- httr::GET(url_record)
record # Status 200 indicates a successful download
## Response [https://zenodo.org/api/records/5719431]
##   Date: 2022-08-15 15:23
##   Status: 200
##   Content-Type: application/json
##   Size: 6.46 kB
```

Now, we can inspect the contents downloaded with the function content()

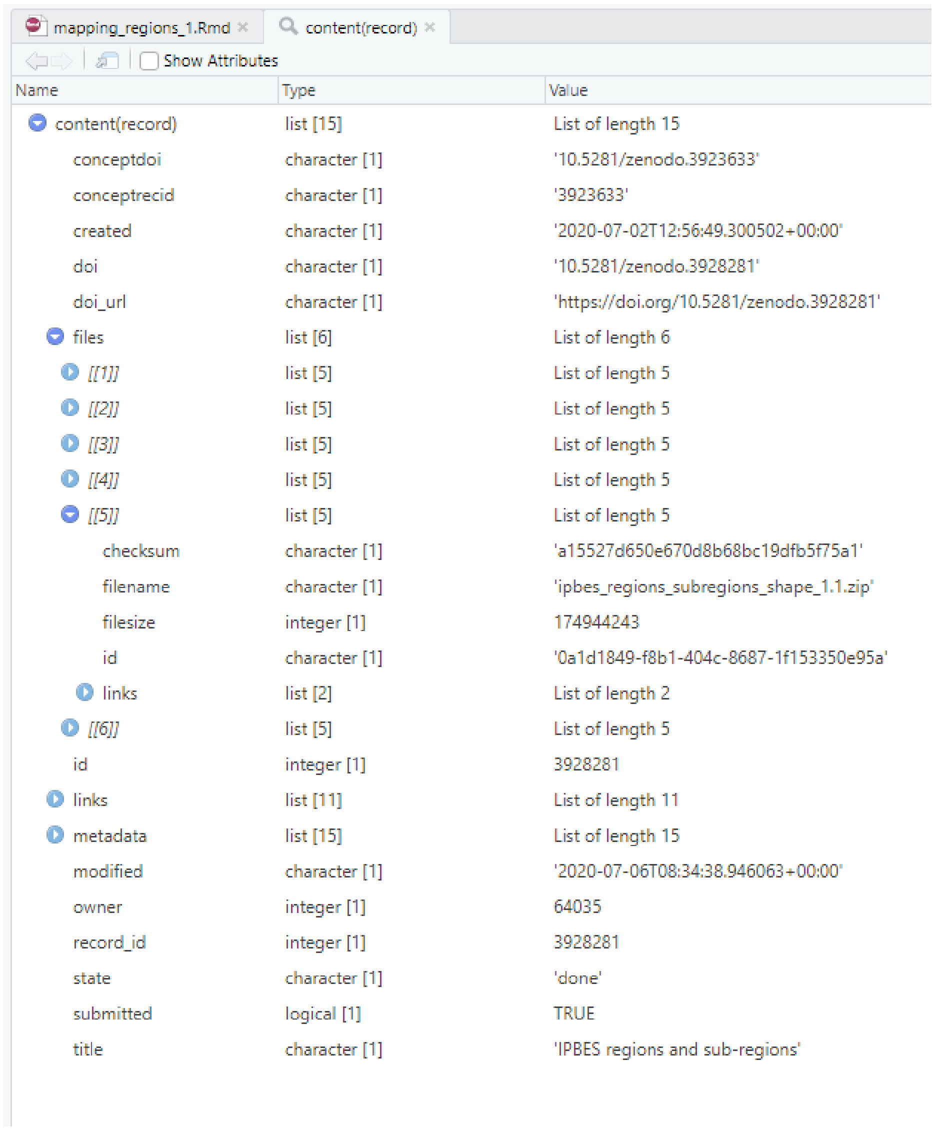
```
View(content(record)) # view displays the output in a human readable form within R Studio
```

This information we received contains metadata for the record, and within this we can find the specific URL to download the IPBES regions and sub-regions shapefile. We then use this URL and the function GET() to download the shapefile.

```
# Contains the url to download the shapefile
url_shape <- content(record)$files[[5]]$links$download

httr::GET(url_shape, write_disk("ipbes_regions_subregions.zip", overwrite = T)) # Downloads shapefile
## Response [https://zenodo.org/api/files/5c418a2d-55bb-47e1-b206-279ee278397c/ipbes_regions_subregions
##   Date: 2022-08-15 15:23
##   Status: 200
##   Content-Type: application/octet-stream
##   Size: 175 MB
## <ON DISK>  C:\Users\jkumagai\Documents\IPBES\R\Geoinformatics\Technical Guidelines Series\Mapping Gu
unzip("ipbes_regions_subregions.zip") # unzips shapefile
```

II. Uploading data into R Studio

Now that our data is on our computer, we need to upload the data into R studio and project the spatial data.



```
Name                  Type            Value
content(record)       list [15]       List of length 15
  conceptdoi          character [1]   '10.5281/zenodo.3923633'
  conceptrecid        character [1]   '3923633'
  created             character [1]   '2020-07-02T12:56:49.300502+00:00'
  doi                 character [1]   '10.5281/zenodo.3928281'
  doi_url             character [1]   'https://doi.org/10.5281/zenodo.3928281'
  files               list [6]        List of length 6
    [[1]]             list [5]        List of length 5
    [[2]]             list [5]        List of length 5
    [[3]]             list [5]        List of length 5
    [[4]]             list [5]        List of length 5
    [[5]]             list [5]        List of length 5
      checksum        character [1]   'a15527d650e670d8b68bc19dfb5f75a1'
      filename        character [1]   'ipbes_regions_subregions_shape_1.1.zip'
      filesize        integer [1]     174944243
      id              character [1]   '0a1d1849-f8b1-404c-8687-1f153350e95a'
      links           list [2]        List of length 2
    [[6]]             list [5]        List of length 5
  id                  integer [1]     3928281
  links               list [11]       List of length 11
  metadata            list [15]       List of length 15
  modified            character [1]   '2020-07-06T08:34:38.946063+00:00'
  owner               integer [1]     64035
  record_id           integer [1]     3928281
  state               character [1]   'done'
  submitted           logical [1]     TRUE
  title               character [1]   'IPBES regions and sub-regions'
```

Figure 1: The picture above shows the resulting R Studio window which displays what was downloaded in a human readable form.

```
pop_raw <- FAOSTAT::read_faostat_bulk("Population_E_All_Data_(Normalized).zip") # load the population d
shape <- sf::st_read("IPBES_Regions_Subregions2.shp") # shapefile
```

```
## Reading layer `IPBES_Regions_Subregions2' from data source
##   `C:\Users\jkumagai\Documents\IPBES\R\Geoinformatics\Technical Guidelines Series\Mapping Guidelines
##   using driver `ESRI Shapefile'
## Simple feature collection with 257 features and 5 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -180 ymin: -90 xmax: 180 ymax: 83.65833
## Geodetic CRS:  WGS 84
```

We chose to project the data into the Robinson projection as it minimizes distortions in both area and distance. To find the proj4 notation please visit this link.

```
crs_robin <-  "+proj=robin +lon_0=0 +x_0=0 +y_0=0 +datum=WGS84 +units=m +no_defs"
shape <- sf::st_transform(shape, crs_robin)
```

To plot the ocean in our maps, we will also download ocean data from the rnaturalearth package and project it

```
ocean <- rnaturalearth::ne_download(scale = 10, type = 'ocean', category = 'physical', returnclass = "s
```

```
## OGR data source with driver: ESRI Shapefile
## Source: "C:\Users\jkumagai\AppData\Local\Temp\RtmpOAuXVr", layer: "ne_10m_ocean"
## with 1 features
## It has 3 fields
```

```
ocean <- sf::st_transform(ocean, crs = crs_robin) # changes the projection
ocean <- ocean[,1]
```

III. Cleaning the data

The next important step is to clean the data to ensure it can be joined and mapped easily. For this example, we will filter to only include the total population for each country in 2018.

```
pop_2018 <- pop_raw %>%
  dplyr::filter(element == "Total Population - Both sexes" &
           year == 2018) %>%
  dplyr::select(area_code, # these columns are selected from the original data
         area,
         element,
         year,
         unit,
         value)
```

By examining the Area names within the dataset, one will notice that every name after Zimbabwe refers to aggregated data, therefore we will remove these from our analysis.

```
tail(pop_2018$area, 34)
```

```
##  [1] "World"
##  [2] "Africa"
##  [3] "Eastern Africa"
##  [4] "Middle Africa"
##  [5] "Northern Africa"
##  [6] "Southern Africa"
##  [7] "Western Africa"
##  [8] "Americas"
##  [9] "Northern America"
## [10] "Central America"
## [11] "Caribbean"
## [12] "South America"
## [13] "Asia"
## [14] "Central Asia"
## [15] "Eastern Asia"
## [16] "Southern Asia"
## [17] "South-eastern Asia"
## [18] "Western Asia"
## [19] "Europe"
## [20] "Eastern Europe"
## [21] "Northern Europe"
## [22] "Southern Europe"
## [23] "Western Europe"
## [24] "Oceania"
## [25] "Australia and New Zealand"
## [26] "Melanesia"
## [27] "Micronesia"
## [28] "Polynesia"
## [29] "European Union (27)"
## [30] "Least Developed Countries"
## [31] "Land Locked Developing Countries"
## [32] "Small Island Developing States"
## [33] "Low Income Food Deficit Countries"
## [34] "Net Food Importing Developing Countries"
```

```
pop_2018 <- pop_2018[1:237, ] # Selects the first 237 records, thus removing the last 34 which are aggr
```

Finally, we need to add the ISO3 codes onto the dataframe, so we can easily join it to the IPEBS Regions and Sub-Regions data. The translateCountryCode() function provided by the FAOSTAT package allows us to easily do this.

```
pop_2018 <- FAOSTAT::translateCountryCode(data = pop_2018, from = "FAOST_CODE", to = "ISO3_CODE", "area
##
## NOTE: Please make sure that the country are matched according to their definition
## Warning in FAOSTAT::translateCountryCode(data = pop_2018, from = "FAOST_CODE", : The following entri
##     FAOST_CODE ISO3_CODE
## 53         351      <NA>
## 234        277      <NA>
##                                            OFFICIAL_FAO_NAME
## 53  China (China mainland, Hong Kong SAR, Macao SAR, Taiwan)
## 234                                the Republic of South Sudan
```

There are two records where no ISO3 Code was assigned: China (including mainland, Hong Kong SAR, Macao SAR, and Taiwan) and South Sudan. “China mainland” refers to the same area as “China” in our dataset, so we are safe to exclude the China (including mainland, Hong Kong SAR, Macao SAR, and Taiwan) from our analysis.

For South Sudan, we will add the same ISO-3 Code we have in the IPBES Regions and Sub-regions dataset.

```r
pop_2018[230,2] <- "SSD" # South Sudan
pop_2018 <- na.omit(pop_2018) # removes China (including other areas)
```

IV: Joining and Aggregating

We have all of our data downloaded locally, uploaded into R, and formatted properly. The last step is to join and aggregate the data to the IPBES regions and sub-regions shapefile.

First, we join the IPBES regions and sub-regions attributes to our data table. I drop the spatial attributes of the IPBES regions and sub-regions dataset to speed up the process.

```r
colnames(shape)[2] <- "ISO3_CODE"
regions <- shape %>%
  as.data.frame() %>% # drops the spatial attributes
  dplyr::select(ISO3_CODE, Region, Sub_Region)  # filters the columns

pop_2018 <- dplyr::left_join(x = pop_2018, y = regions, by = "ISO3_CODE") %>% # Joins data
 tidyr::drop_na() # conveenient function from tidyr package
```

Secondly, we aggregate the data per IPBES regions and sub-regions. In our example, I calculate the total population per region and per sub-region using the group_by() function.

```r
pop_2018 <- pop_2018 %>%
  dplyr::group_by(Region) %>% # Grouping by regions
  dplyr::mutate(region_pop = sum(value)/1000) %>% # calculates total population (millions) per region
  dplyr::ungroup() %>%
  dplyr::group_by(Sub_Region) %>% # Grouping by sub-region
  dplyr::mutate(sub_region_pop = sum(value)/1000) %>% # calculates total population (millions) per sub-
  dplyr::ungroup()
pop_2018
```

```
## # A tibble: 234 x 11
##    FAOST_CODE ISO3_CODE area        element  year unit   value Region Sub_Region
##         <int> <chr>     <chr>       <chr>   <int> <chr>  <dbl> <chr>  <chr>
##  1          1 ARM       Armenia     Total ~  2018 1000~ 2.95e3 Europ~ Eastern E~
##  2          2 AFG       Afghanistan Total ~  2018 1000~ 3.72e4 Asia ~ South Asia
##  3          3 ALB       Albania     Total ~  2018 1000~ 2.88e3 Europ~ Central a~
##  4          4 DZA       Algeria     Total ~  2018 1000~ 4.22e4 Africa North Afr~
##  5          5 ASM       American S~ Total ~  2018 1000~ 5.55e1 Asia ~ Oceania
##  6          6 AND       Andorra     Total ~  2018 1000~ 7.70e1 Europ~ Central a~
##  7          7 AGO       Angola      Total ~  2018 1000~ 3.08e4 Africa Southern ~
##  8          8 ATG       Antigua an~ Total ~  2018 1000~ 9.63e1 Ameri~ Caribbean
##  9          9 ARG       Argentina   Total ~  2018 1000~ 4.44e4 Ameri~ South Ame~
## 10         10 AUS       Australia   Total ~  2018 1000~ 2.49e4 Asia ~ Oceania
## # ... with 224 more rows, and 2 more variables: region_pop <dbl>,
## #   sub_region_pop <dbl>
```

Finally, we join the formatted FAO data to the spatial data we originally had so we can create maps.

```r
data <- dplyr::full_join(x = shape, y = pop_2018, by = "ISO3_CODE")
```

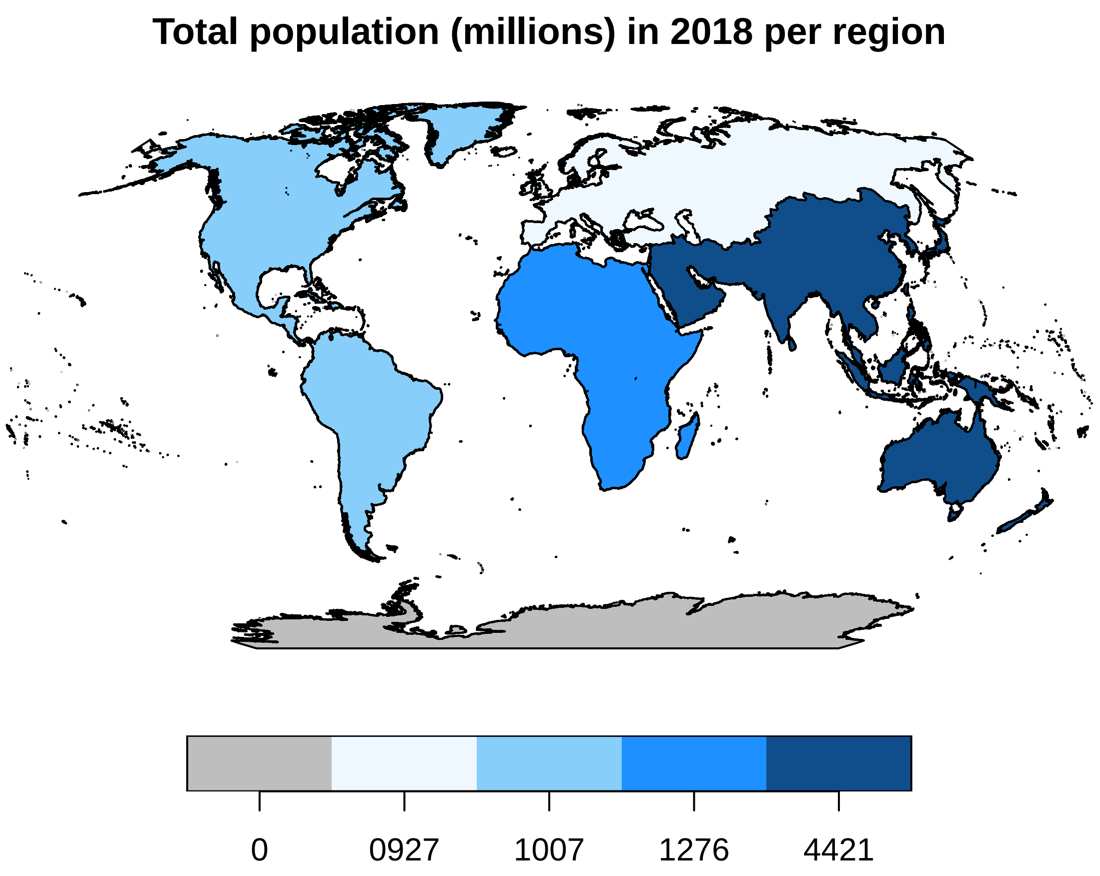
V. Mapping

All that is left to do is to map the data per region and sub-region. We begin by dissolving the spatial data per region and subregion so country borders are not included.

```r
data_region <- data %>% # this dissolves the data by region
  dplyr::group_by(Region.x) %>%
  dplyr::summarise(region_pop2 = sum(value, na.rm = T)/1000) %>%
  sf::st_cast()

data_subregion <- data %>% # this dissolves the data by subregion
  dplyr::group_by(Sub_Region.x) %>%
  dplyr::summarise(sub_region_pop2 = sum(value, na.rm=T)/1000) %>%
  sf::st_cast()
```

Now, we choose the palette and plot by region.

```r
data_region$region_pop2 <- as.character(round(data_region$region_pop2 )) # Treats the values as groups
data_region$region_pop2[5] <- "0927" # Ensures the legend displays correctly

# Plotting by regions
palette <- c("grey","aliceblue", "lightskyblue", "dodgerblue", "dodgerblue4") # colors


plot(data_region[,2], pal = palette, main = "Total population (millions) in 2018 per region")
```



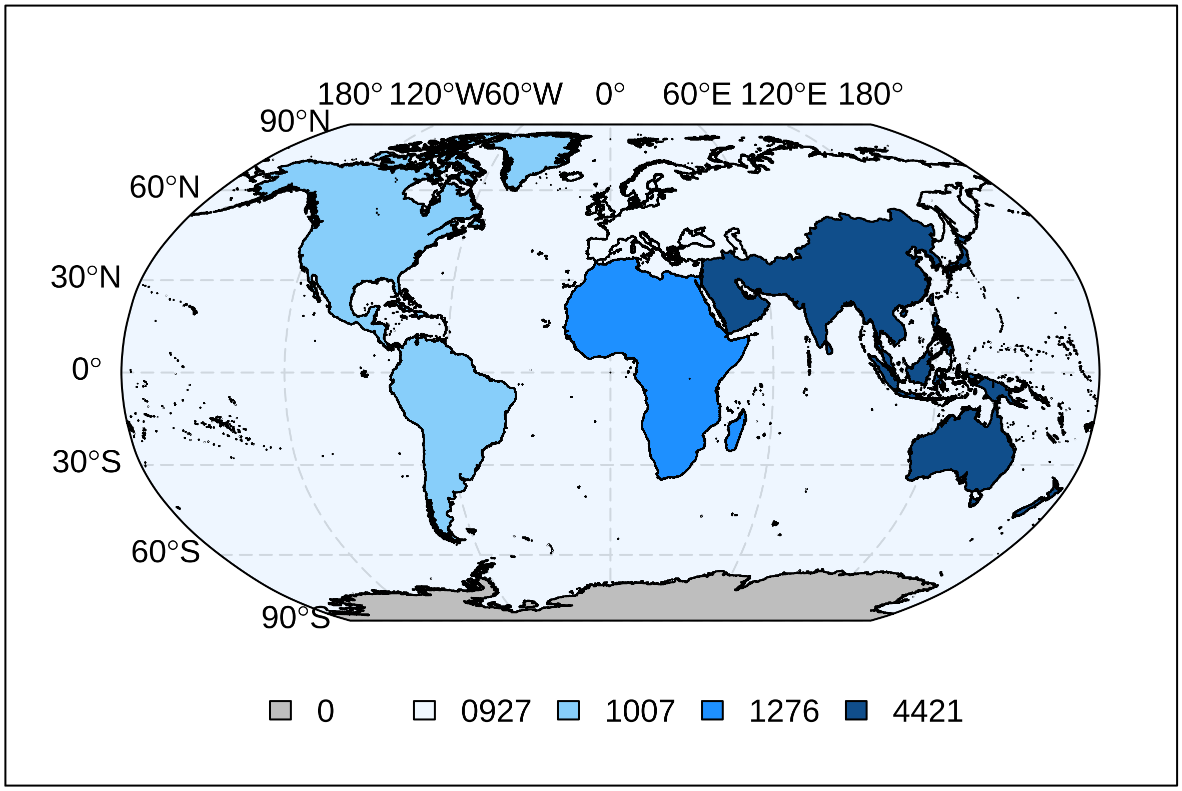
If you would like to add graticules and an ocean background, follow this example. First, we will set up the graticules we will plot

```
# Creates latitude and longitude labels and graticules
lat <- c(-90, -60, -30, 0, 30, 60, 90)
long <- c(-180, -120, -60, 0, 60, 120, 180)
labs <- graticule::graticule_labels(lons = long, lats = lat, xline = -180, yline = 90, proj = crs_robin
lines <- graticule::graticule(lons = long, lats = lat, proj = crs_robin) # graticules
```

Then, we will plot the graticules, ocean data, then region data, and finally the text for latitude and longitude lines, legend, and surrounding box.

```
par(mar = c(2,3,1,2)) # Adjusts the edges of the frame
plot(lines, lty = 5, col = "lightgrey",  main = "Total population (millions) in 2018 per region") # plo
plot(ocean, col = ggplot2::alpha("slategray1", 0.3), add = TRUE)
plot(data_region[,2], pal = palette, add = TRUE)
text(subset(labs, labs$islon), lab = parse(text = labs$lab[labs$islon]), pos = 3, xpd = NA) # plots lon
text(subset(labs, !labs$islon), lab = parse(text = labs$lab[!labs$islon]), pos = 2, xpd = NA) # plots l
legend("bottom", # adding the legend last
       legend = (data_region %>% pull(region_pop2) %>% sort()),
       fill = palette,
       horiz = TRUE, bty = "n")
box(which = "plot", lty = "solid") # Map frame
```

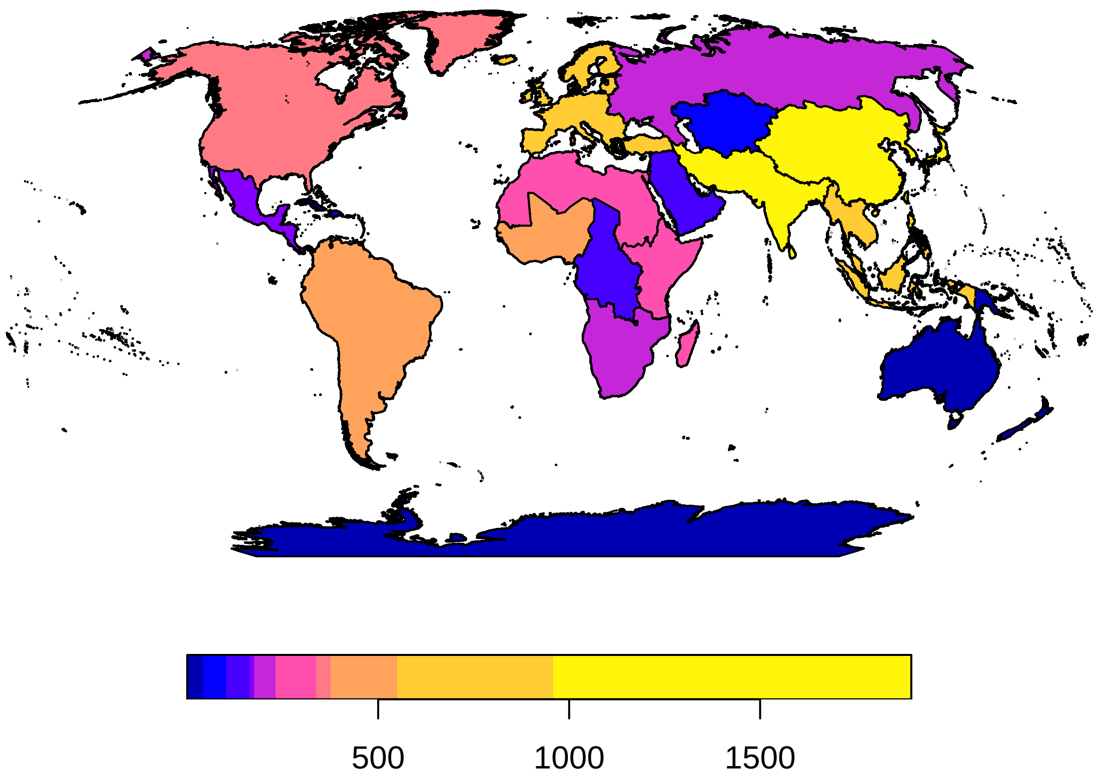
Total population (millions) in 2018 per region



We can also plot by subregion.

```
plot(data_subregion[,2], main = "Total population (millions) in 2018 per sub-region", breaks = "quantil
```

Total population (millions) in 2018 per sub−region



Your feedback on this content is welcome. Let us know what other useful material would you like to see here by emailing tsu.data@ipbes.net