Spatial graphics

Rolf Bänziger, r.banziger@westminster.ac.uk

24 November 2022

Spatial (or geospatial) data is data related to specific locations. We often want to display this kind of data overlaid on a map. This tutorial demonstrates producing geospatial graphics with *ggplot2*.

Libraries

We need to install a number of libraries first. *sf* provides functions for working with geospatial data. See Simple Feature for R for more details. *rnaturalearth* and *rnaturalearthdata* provide a world map with some demographics data, and *eurostat* provides a number of statistics from the European Union. We also need *tidyverse*, which we use to manipulate data.

```
install.packages(c("sf", "rnaturalearth", "rnaturalearthdata", "rgeos",
                   "eurostat", "tidyverse", "cartogram", "s2"))
```

```
library(ggplot2)
library(sf)
```

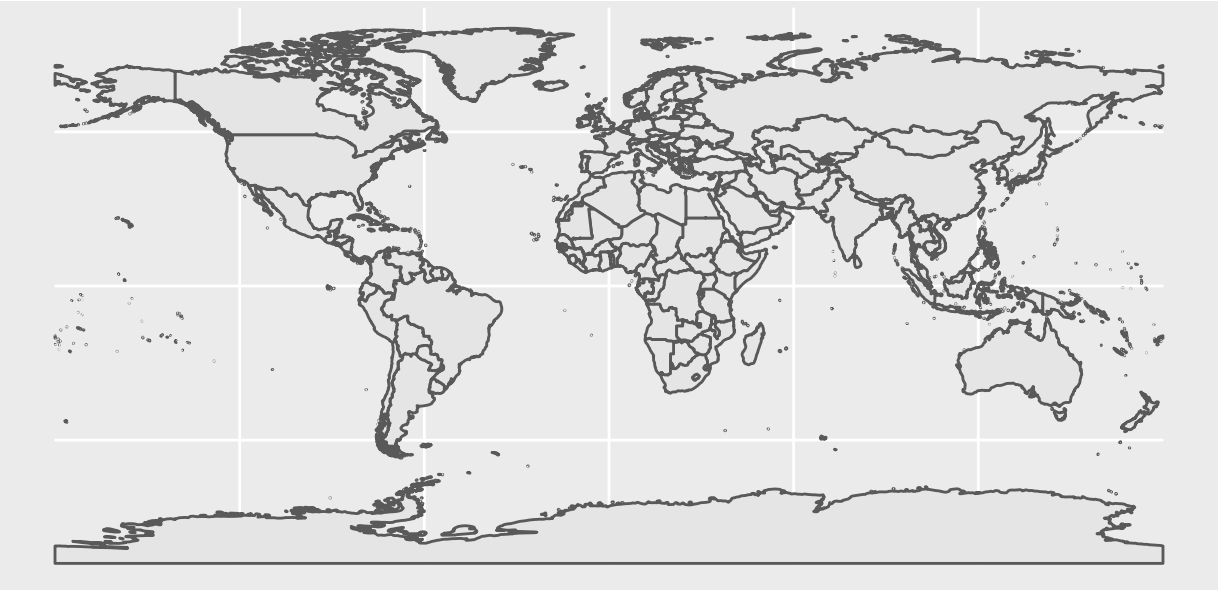
```
## Warning: package 'sf' was built under R version 4.2.2
```

```
library(rnaturalearth)
```

```
## Warning: package 'rnaturalearth' was built under R version 4.2.2
```

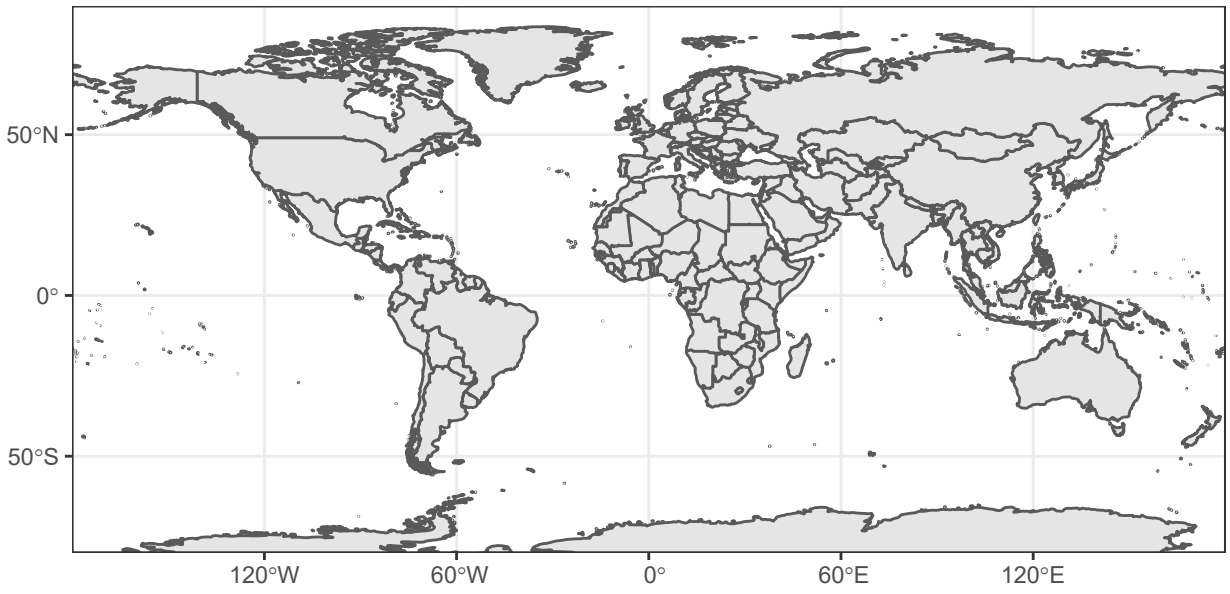
Let's get the world map and save it in the variable `world`. Now we can plot it using `geom_sf()`. Note that we don't need to pass any aesthetics mappings just yet.

```
world <- ne_countries(returnclass = "sf", scale="medium")
ggplot(world) + geom_sf()
```



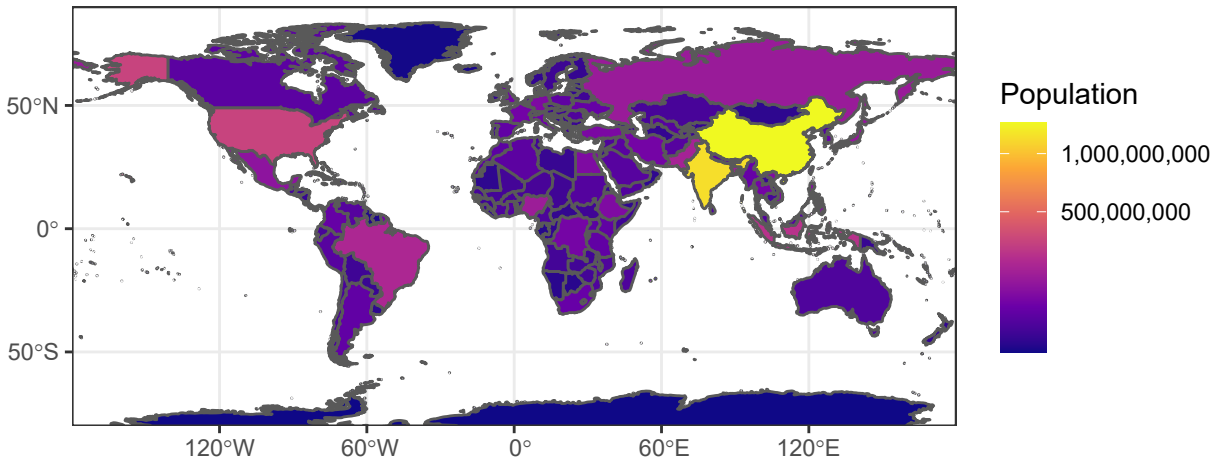
Let's make it a bit nicer. `theme_bw()` works well with maps and `coord_sf(expand = F)` ensures the latitude and longitude are added. We also specify northern and southern limits.

```
ggplot(world) + geom_sf() +
  theme_bw() + coord_sf(ylim = c(-80, 90), expand = F)
```



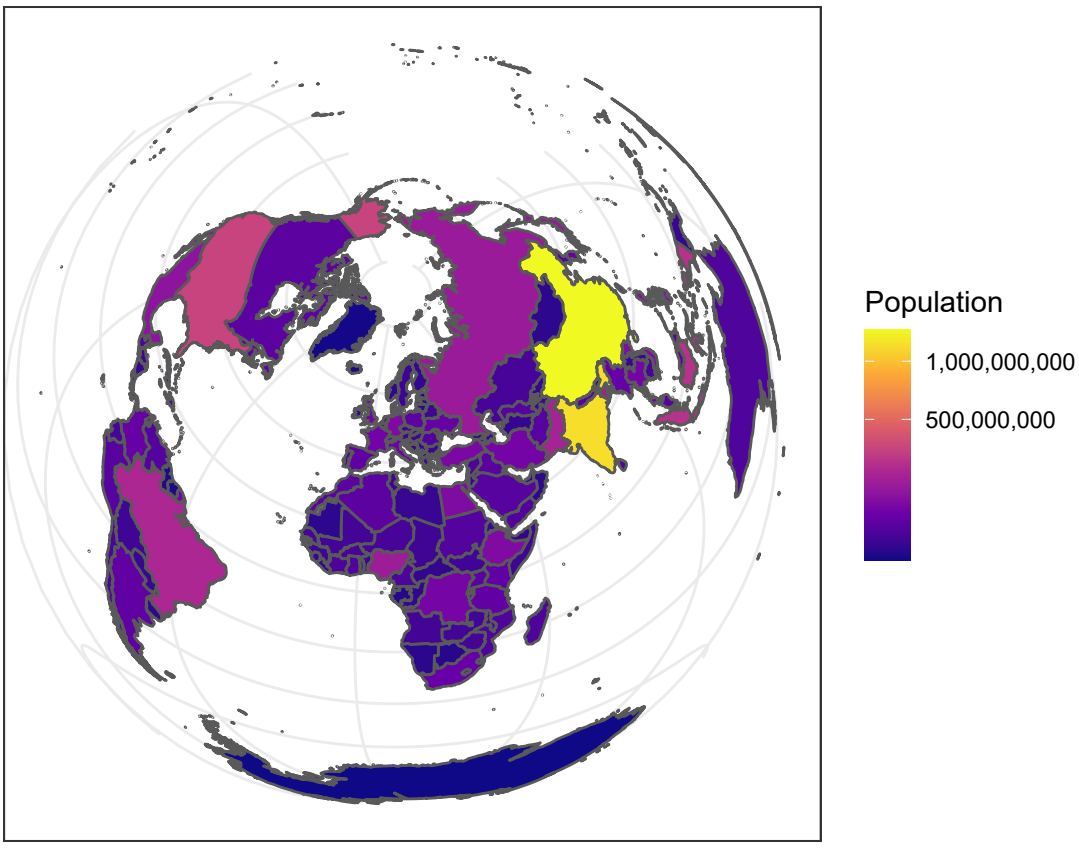
`geom_sf()` supports mapping data to the *fill* aesthetics. Let's use this to display the estimated population. Note that we transform the scale into an exponential scale by adding the `trans = "sqrt"` option. Otherwise the scale would be completely dominated by China and India.

```
ggplot(world) + geom_sf() + aes(fill = pop_est) +
  scale_fill_viridis_c(option = "plasma", trans = "sqrt", label = scales::comma) +
  labs(fill = "Population") +
  theme_bw() + coord_sf(ylim = c(-80, 90), expand = F)
```



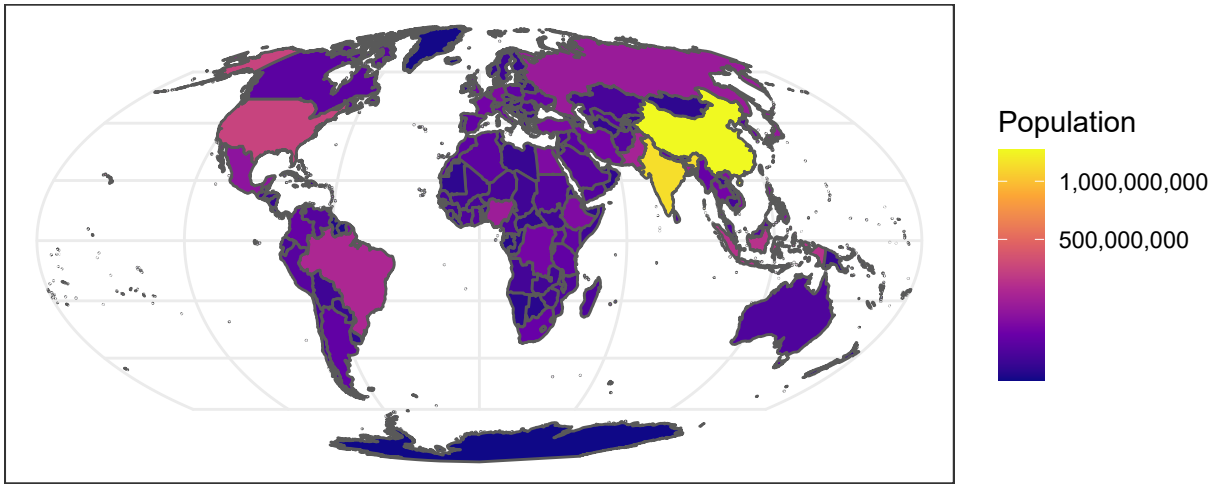
The function `coord_sf` allows setting the coordinate reference system, which includes both projection and extent of the map. By default, the map will use the coordinate system that is defined within the map data or, if none is defined, WGS84 (latitude/longitude, the reference system used in GPS). Use the argument `crs` to override this setting. Here, we set it the EPSG (European Petroleum Survey Group) code 3035, which is the European-centric ETRS89 (European Terrestrial Reference System 1989) Lambert Azimuthal Equal-Area projection. There are many other EPSG codes, see EPSG.org for a list of all available codes.

```
ggplot(world) + geom_sf() + aes(fill = pop_est) +
  scale_fill_viridis_c(option = "plasma", trans = "sqrt", label = scales::comma) +
  labs(fill = "Population") +
  theme_bw() +
  coord_sf(crs = st_crs(3035))
```



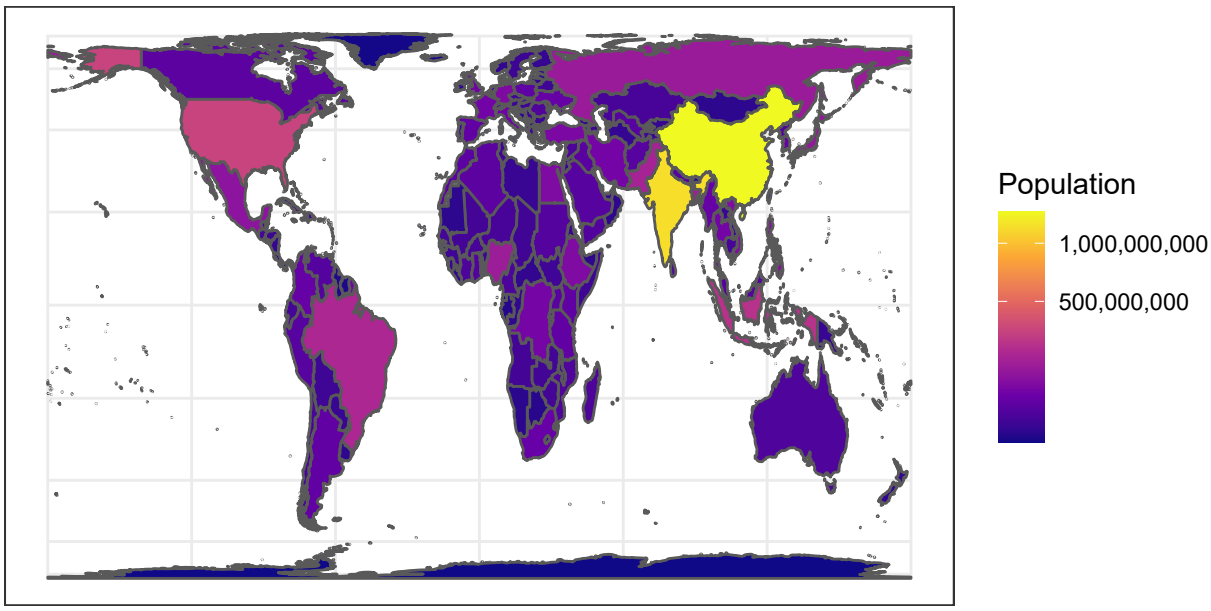
Alternativley, you can directly specify the projection using a *proj4string*. See Proj4 for a list of available projections. The example below shows the *Mollweide* projection:

```
ggplot(world) + geom_sf() + aes(fill = pop_est) +
  scale_fill_viridis_c(option = "plasma", trans = "sqrt", label = scales::comma) +
  labs(fill = "Population") +
  theme_bw() +
  coord_sf(crs="+proj=moll")
```



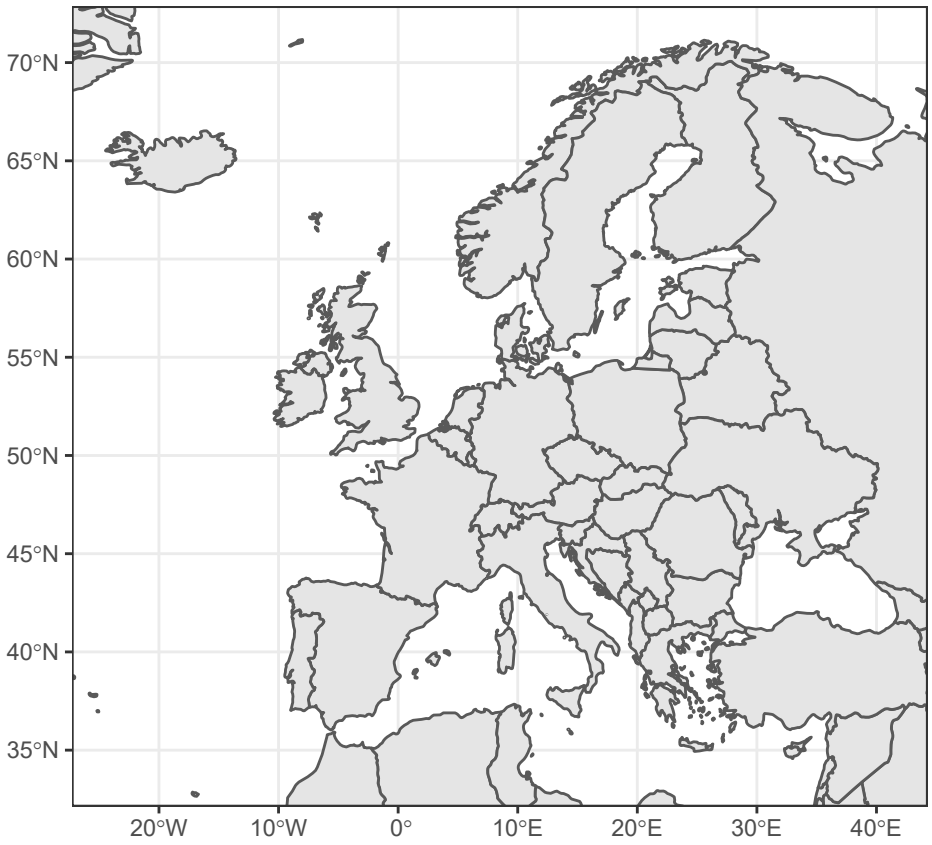
And this map uses the Gall-Peters projection:

```
ggplot(world) + geom_sf() + aes(fill = pop_est) +
  scale_fill_viridis_c(option = "plasma", trans = "sqrt", label = scales::comma) +
  labs(fill = "Population") +
  theme_bw() +
  coord_sf(crs="+proj=cea +lon_0=0 +lat_ts=45")
```



We can also use the `coord_sf` function to zoom into a specific region, by specifying latitude and longitude ranges. Note that the latitude relates to the y -axis and the longitude to the x-axis. Negative values refer to southern or western coordinates, whereas positive values refer to northern or eastern coordinates.

```
ggplot(world) + geom_sf() + theme_bw() +
  coord_sf(ylim = c(34, 71), xlim = c(-24, 41))
```



Europe

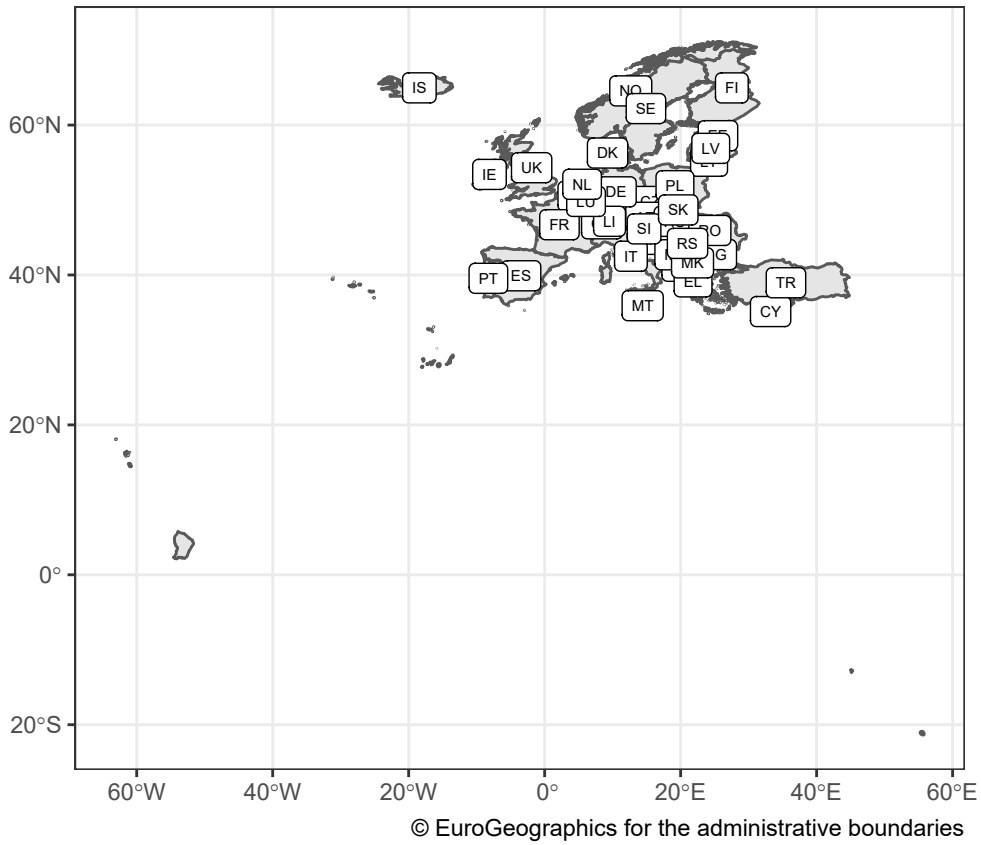
Let's draw some maps with European data. The package *eurostat* provides functions to get maps and data from Eurostat.

```
library(eurostat)
```

```
## Warning: package 'eurostat' was built under R version 4.2.2
```

We can download the map with national boundaries using `get_eurostat_geospatial(nuts = 0)`. See NUTS for details about the regional levels. As usual, we use `ggplot()` and `geom_sf()` to draw the map. We also draw labels for each country with `geom_sf_text()`. Because the map data is provided by Eurostat, we need to add a copyright note.

```
eurostat.map <- get_eurostat_geospatial(nuts = 0)
ggplot(eurostat.map) + geom_sf() +
  geom_sf_label(aes(label = id), size = 2) +
  theme_bw() +
  labs(caption = "© EuroGeographics for the administrative boundaries",
       x = NULL, y = NULL)
```



Oops, looks like the map includes some colonies, let's crop it to only show continental europe...

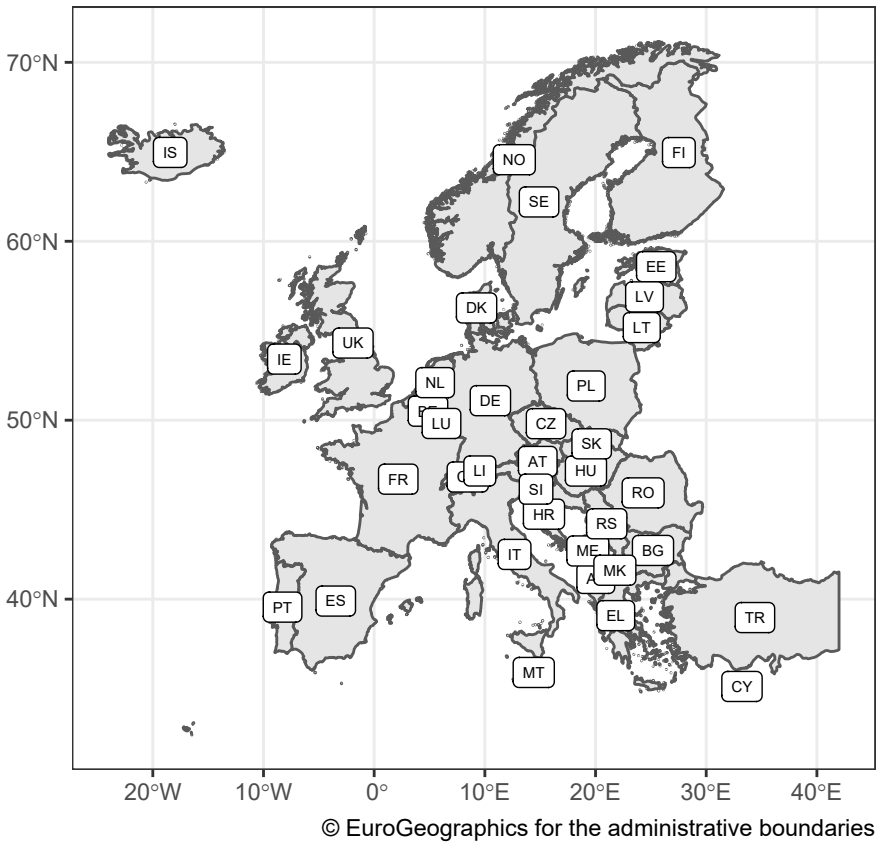
```
eurostat.map <- st_crop(eurostat.map, xmin = -24, xmax = 42,                    ymin = 3
```

```
## Warning: attribute variables are assumed to be spatially constant throughout all
## geometries
```

```
ggplot(eurostat.map) +geom_sf() +
  geom_sf_label(aes(label = id), size = 2) +
  theme_bw() +

  labs(caption = "© EuroGeographics for the administrative boundaries",
       x = NULL, y = NULL)
```

```
## Warning in st_point_on_surface.sfc(sf::st_zm(x)): st_point_on_surface may not
## give correct results for longitude/latitude data
```



Let us load the demographic at national level (*demo_gind*). Consult the Eurostat Database for a list of all available data sets.

```
demographics <- get_eurostat("demo_gind")
head(demographics)
```

```
## # A tibble: 6 x 4
##   indic_de geo   time         values
##   <chr>    <chr> <date>        <dbl>
## 1 JAN      AD    2022-01-01    79535
## 2 JAN      AL    2022-01-01  2793592
## 3 JAN      AT    2022-01-01  8978929
## 4 JAN      AZ    2022-01-01 10156366
## 5 JAN      BE    2022-01-01 11631136
## 6 JAN      BG    2022-01-01  6838937
```

We see that the data set above contains one row per country, statistical index and year. We'd rather have only the data from 2020, and only one row per country containing all indices. Let's use `tidyverse` functions to filter and pivot the data.

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.2.2
```

```
demo2 <-
  demographics %>%
  filter(time == as.Date("2020-01-01")) %>%
  pivot_wider(names_from = indic_de, values_from = values)
head(demo2)
```

```
## # A tibble: 6 x 30
##   geo   time            AVG CNMIGRAT CNMIGRATRT  DEATH    FAVG FDEATH    FJAN
##   <chr> <date>        <dbl>    <dbl>      <dbl>  <dbl>   <dbl>  <dbl>   <dbl>
## 1 AL    2020-01-01  2837848   -16684       -5.9  27605 1422551  12101 1425342
## 2 AM    2020-01-01  2961473     3374        1.1  36170 1563917  16139 1562689
## 3 AT    2020-01-01  8916864    39596        4.4  91599 4529002  46227 4522292
## 4 AZ    2020-01-01 10093121     1101        0.1  75647 5052194  33566 5039100
## 5 BE    2020-01-01 11538604    44873        3.9 126896 5847754  64688 5841215
## 6 BG    2020-01-01  6934015    30715        4.4 124735 3574335  58785 3581836
## # ... with 21 more variables: FLBIRTH <dbl>, GBIRTHRT <dbl>, GDEATHRT <dbl>,
## #   GROW <dbl>, GROWRT <dbl>, JAN <dbl>, LBIRTH <dbl>, MAVG <dbl>,
## #   MDEATH <dbl>, MIGT <dbl>, MIGTRT <dbl>, MJAN <dbl>, MLBIRTH <dbl>,
## #   NATGROW <dbl>, NATGROWRT <dbl>, NATT <dbl>, NATTRT <dbl>, POPSHARE <dbl>,
## #   POPSHARE_EU27_2020 <dbl>, POPT <dbl>, POPTRT <dbl>
```

That's looks much better. To use the data in our geo-plot, we need to join the map and the demo2 data frames. We use the `left_join` function from tidyverse, which will retain all values from `eurostat.map` and only use matching records from `demo2`.

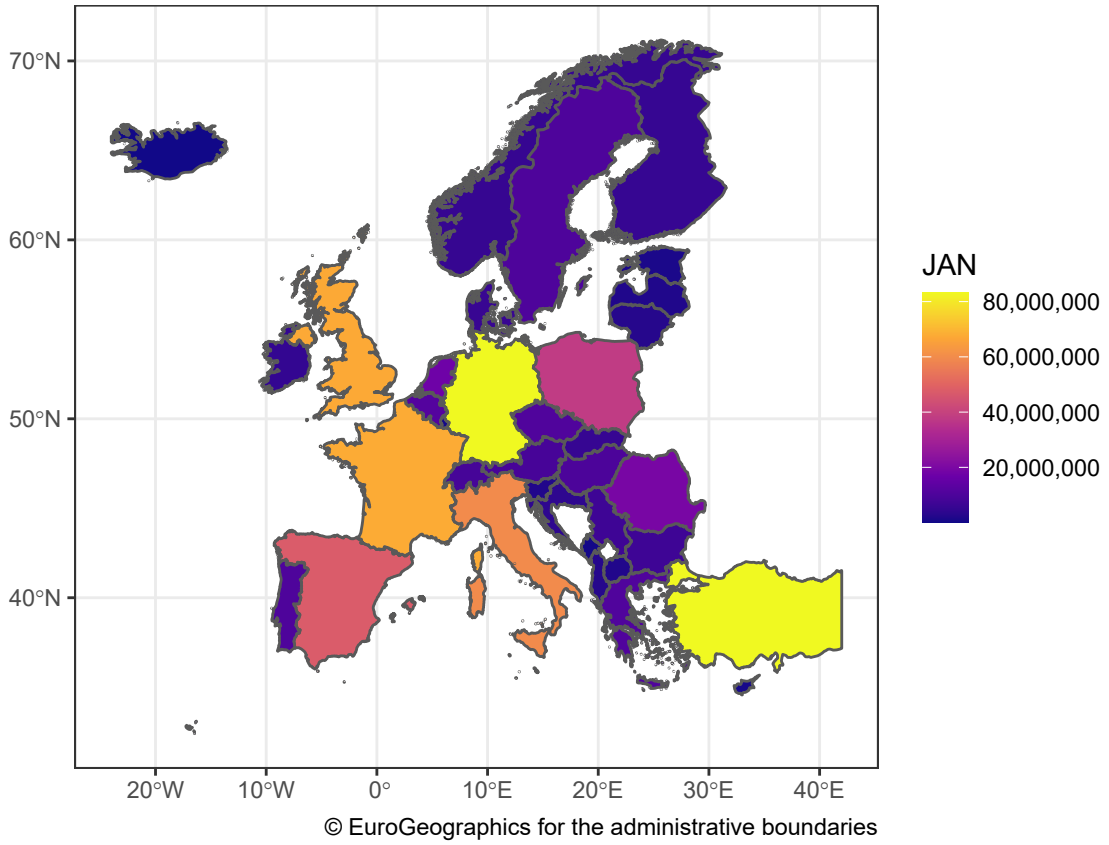
```
demomap <- left_join(eurostat.map, demo2, by = "geo")
head(as.data.frame(demomap))
```

```
##   id CNTR_CODE               NUTS_NAME LEVL_CODE FID NUTS_ID geo       time
## 1 BG        BG                                0  BG      BG  BG 2020-01-01
## 2 CH        CH SCHWEIZ/SUISSE/SVIZZERA         0  CH      CH  CH 2020-01-01
## 3 CY        CY                   ΚΥΠΡΟΣ         0  CY      CY  CY 2020-01-01
## 4 AL        AL                 SHQIPËRIA         0  AL      AL  AL 2020-01-01
## 5 CZ        CZ          ČESKÁ REPUBLIKA         0  CZ      CZ  CZ 2020-01-01
## 6 BE        BE         BELGIQUE-BELGIË          0  BE      BE  BE 2020-01-01
##        AVG CNMIGRAT CNMIGRATRT  DEATH    FAVG FDEATH    FJAN FLBIRTH GBIRTHRT
## 1  6934015    30715        4.4 124735 3574335  58785 3581836   28714      8.5
## 2  8638167    54548        6.3  76195 4352436  38570 4337170   41615      9.9
## 3   892006     4494        5.0   6422  455946   3038  453534    4740     11.1
## 4  2837848   -16684       -5.9  27605 1422551  12101 1425342   13589      9.9
## 5 10697858    26927        2.5 129289 5424309  62690 5421943   53924     10.3
## 6 11538604    44873        3.9 126896 5847754  64688 5841215   55843      9.9
##   GDEATHRT   GROW GROWRT      JAN LBIRTH    MAVG MDEATH   MIGT MIGTRT    MJAN
## 1     18.0 -34934   -5.0  6951482  59086 3359681  65950  44013    6.3 3369646
## 2      8.8  64267    7.4  8606033  85914 4285731  37625 267528   31.0 4268863
## 3      7.2   8002    9.0   888005   9930  436061   3384  47230   52.9  434471
## 4      9.7 -16214   -5.7  2845955  28075 1415298  15504  31024   10.9 1420613
## 5     12.1   7838    0.7 10693939 110200 5273550  66599  98635    9.2 5271996
## 6     11.0  32327    2.8 11522440 114350 5690850  62208 182727   15.8 5681225
##   MLBIRTH NATGROW NATGROWRT   NATT NATTRT POPSHARE POPSHARE_EU27_2020   POPT
## 1   30372  -65649      -9.5 183821   26.5      1.4                1.6 227834
## 2   44299    9719       1.1 162109   18.8       NA                 NA 429637
```

```
## 3    5190    3508       3.9  16352   18.3      0.2                0.2  63582
## 4   14486     470       0.2  55680   19.6       NA                 NA  86704
## 5   56276  -19089      -1.8 239489   22.4      2.1                2.4 338124
## 6   58507  -12546      -1.1 241246   20.9      2.2                2.6 423973
##   POPTRT                          geometry
## 1   32.9 MULTIPOLYGON (((22.97854 43...
## 2   49.7 MULTIPOLYGON (((8.56741 47....
## 3   71.3 MULTIPOLYGON (((33.77666 34...
## 4   30.6 MULTIPOLYGON (((19.77166 42...
## 5   31.6 MULTIPOLYGON (((14.48801 51...
## 6   36.7 MULTIPOLYGON (((5.0811 51.4...
```

And now we can use the data in the aesthetics mapping to colour the countries.

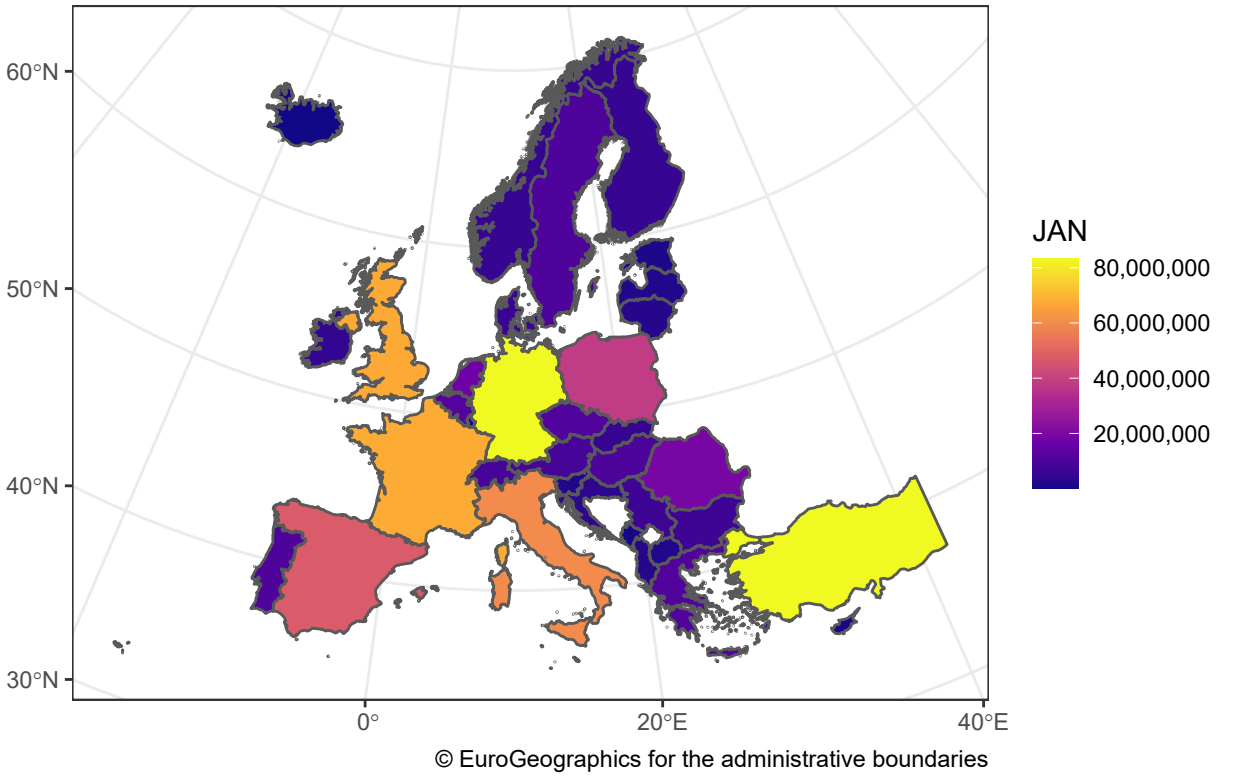
```
ggplot(demomap) + aes(fill = JAN) + geom_sf() +
  theme_bw() +
  scale_fill_viridis_c(option = "plasma", label = scales::comma) +
  labs(caption = "© EuroGeographics for the administrative boundaries",
       x = NULL, y = NULL)
```



Remember, we can improve the map display by using appropriate map projections. Here, we use the Lambert Conformal Conic Europe projection (EPSG 3034).

```
ggplot(demomap) + aes(fill = JAN) + geom_sf() +
  theme_bw() +
  coord_sf(crs = st_crs(3034)) +
```

```
  scale_fill_viridis_c(option = "plasma", label = scales::comma) +
  labs(caption = "© EuroGeographics for the administrative boundaries",
       x = NULL, y = NULL)
```



The map above is a choropleh, which means we can also plot it as a distorted cartogram. Note that we first need to transform the map to use a Europe-specific CRS, in this case EPSG:23038.

```
library(cartogram)
```

```
## Warning: package 'cartogram' was built under R version 4.2.2
```

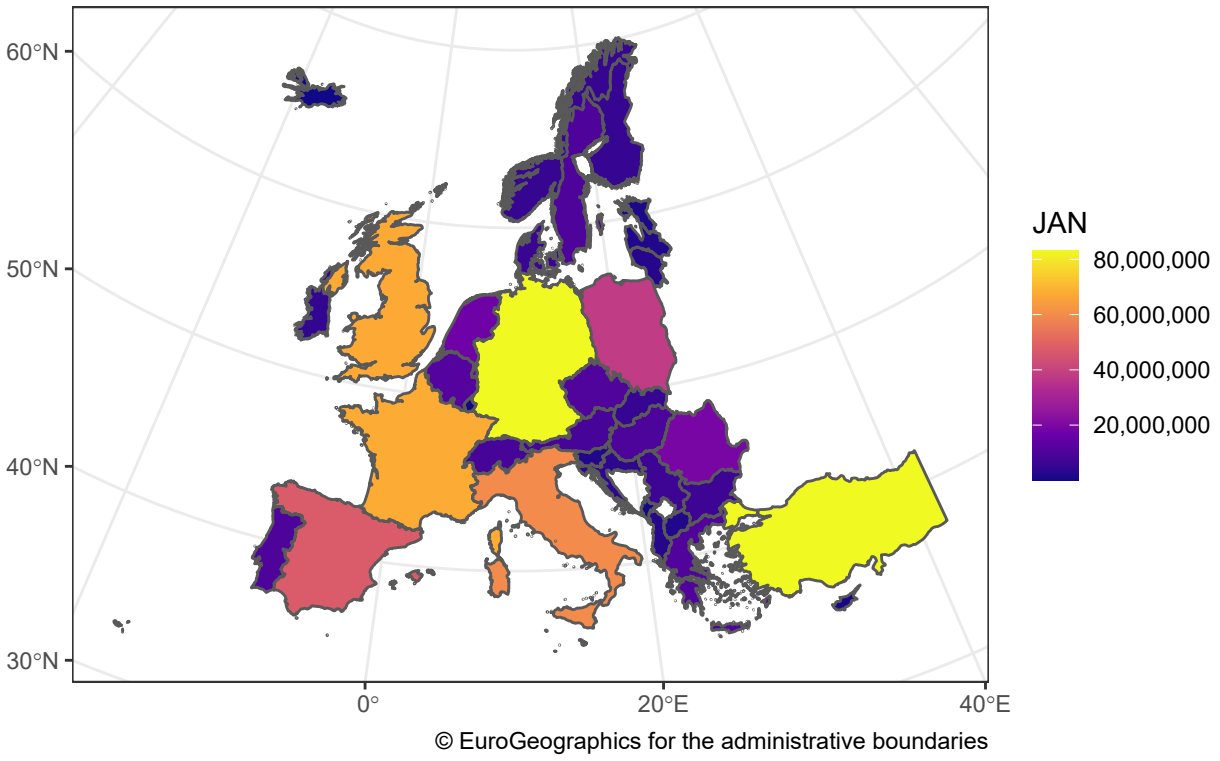
```
europeCartogram <- cartogram_cont(
  st_transform(demomap, crs = st_crs(3034)),
  weight = "JAN",
  itermax = 3 #limit to 3 iterations, so it's a bit faster
  )
```

```
## Mean size error for iteration 1: 2.32998444936206
```

```
## Mean size error for iteration 2: 1.90279687313565
```

```
## Mean size error for iteration 3: 1.61516171385541
```

```
ggplot(europeCartogram) + aes(fill = JAN) + geom_sf() +
  theme_bw() +
  coord_sf(crs = st_crs(3034)) +
  scale_fill_viridis_c(option = "plasma", label = scales::comma) +
  labs(caption = "© EuroGeographics for the administrative boundaries",
       x = NULL, y = NULL)
```



United Kingdom

You will often need to import maps other sources. Mapping data is often provided in so-called *Shapefiles*. United Kingdom upper tier local authority (i.e. counties, unitary authorities, London boroughs, etc.) boundaries are available to download from geoportal.statistics.gov.uk. Note that there are different resolutions of the data. It's often enough to use generalised or even ultra generalised data, as these are much smaller and graphics render much quicker.

The *sf* package can read data in the *Shapefile* format, using the `st_read` function.

```
utla.map <- st_read(
  "UTLA2019/Counties_and_Unitary_Authorities_(December_2019)_Boundaries_UK_BUC.shp")
```

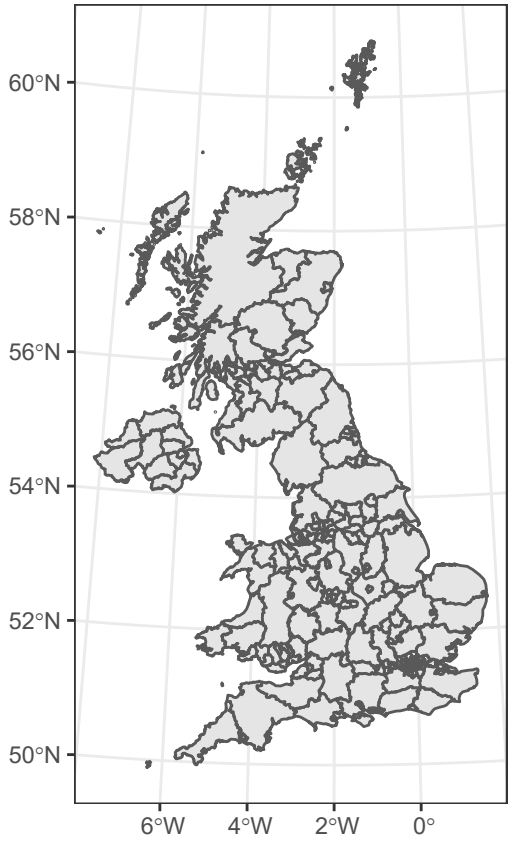
```
## Reading layer `Counties_and_Unitary_Authorities_(December_2019)_Boundaries_UK_BUC' from data source
##   using driver `ESRI Shapefile'
## Simple feature collection with 216 features and 10 fields
```

```
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -116.1928 ymin: 7054.1 xmax: 655653.8 ymax: 1218618
## Projected CRS: OSGB36 / British National Grid
```

```
head(as.data.frame(utla.map))
```

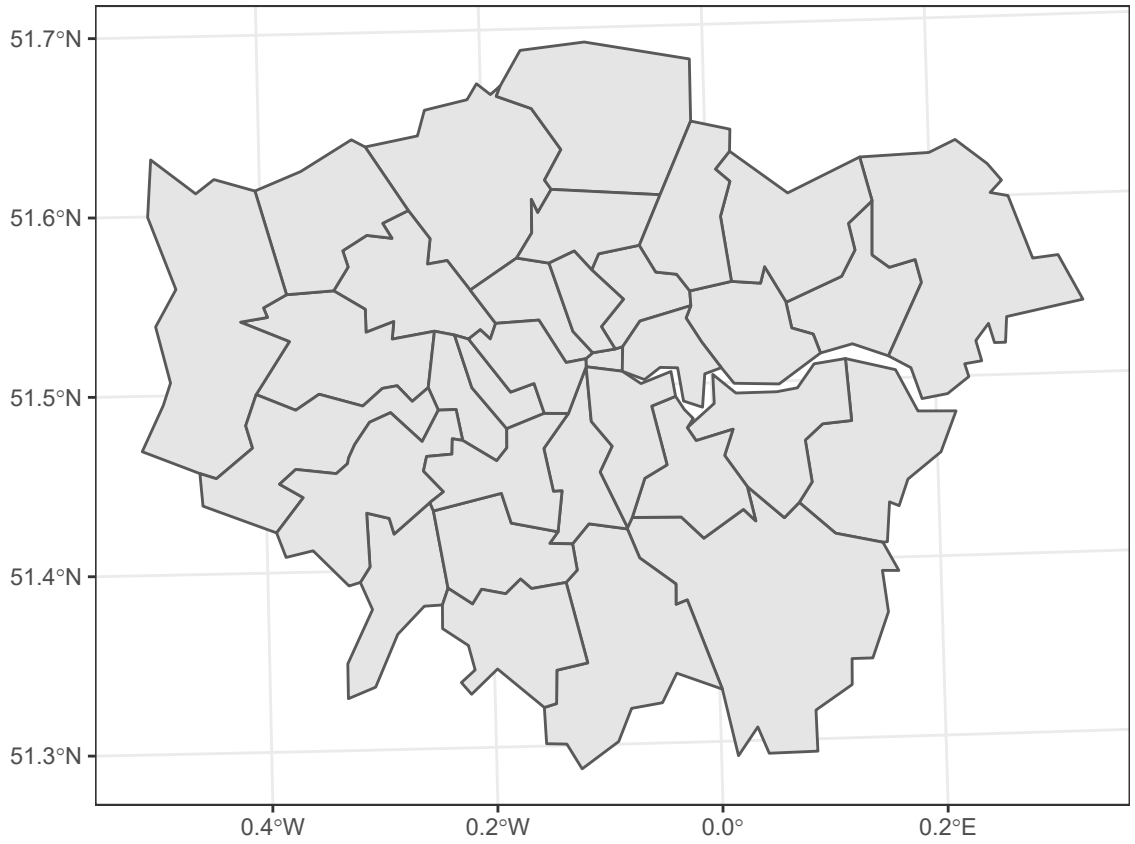
```
##   objectid ctyua19cd            ctyua19nm ctyua19nmw  bng_e  bng_n     long
## 1        1 E06000001           Hartlepool       <NA> 447160 531474 -1.27018
## 2        2 E06000002        Middlesbrough       <NA> 451141 516887 -1.21099
## 3        3 E06000003 Redcar and Cleveland       <NA> 464361 519597 -1.00608
## 4        4 E06000004     Stockton-on-Tees       <NA> 444940 518183 -1.30664
## 5        5 E06000005           Darlington       <NA> 428029 515648 -1.56835
## 6        6 E06000006               Halton       <NA> 354246 382146 -2.68853
##        lat st_areasha st_lengths                       geometry
## 1 54.67614   96845510   50305.33 MULTIPOLYGON (((448986 5367...
## 2 54.54467   52908459   34964.41 MULTIPOLYGON (((451752.7 52...
## 3 54.56752  248679106   83939.75 MULTIPOLYGON (((451965.6 52...
## 4 54.55691  207159134   87075.86 MULTIPOLYGON (((451965.6 52...
## 5 54.53534  198812771   91926.84 MULTIPOLYGON (((419709.3 51...
## 6 53.33424   80285488   66618.91 MULTIPOLYGON (((355156.2 38...
```

```
ggplot(utla.map) + geom_sf() + theme_bw()
```



If we only want to display a part of the map we need to filter the data. Here, we create a map of the London boroughs (all London boroughs start with E09).

```
ldn.map <- utla.map[substr(utla.map$ctyua19cd, 1, 3) == "E09",]
ggplot(ldn.map) + geom_sf() +
  theme_bw()
```



You might notice that the map is somewhat tilted. This is because the imported shapefile uses the British National Grid coordinate reference system (CRS).

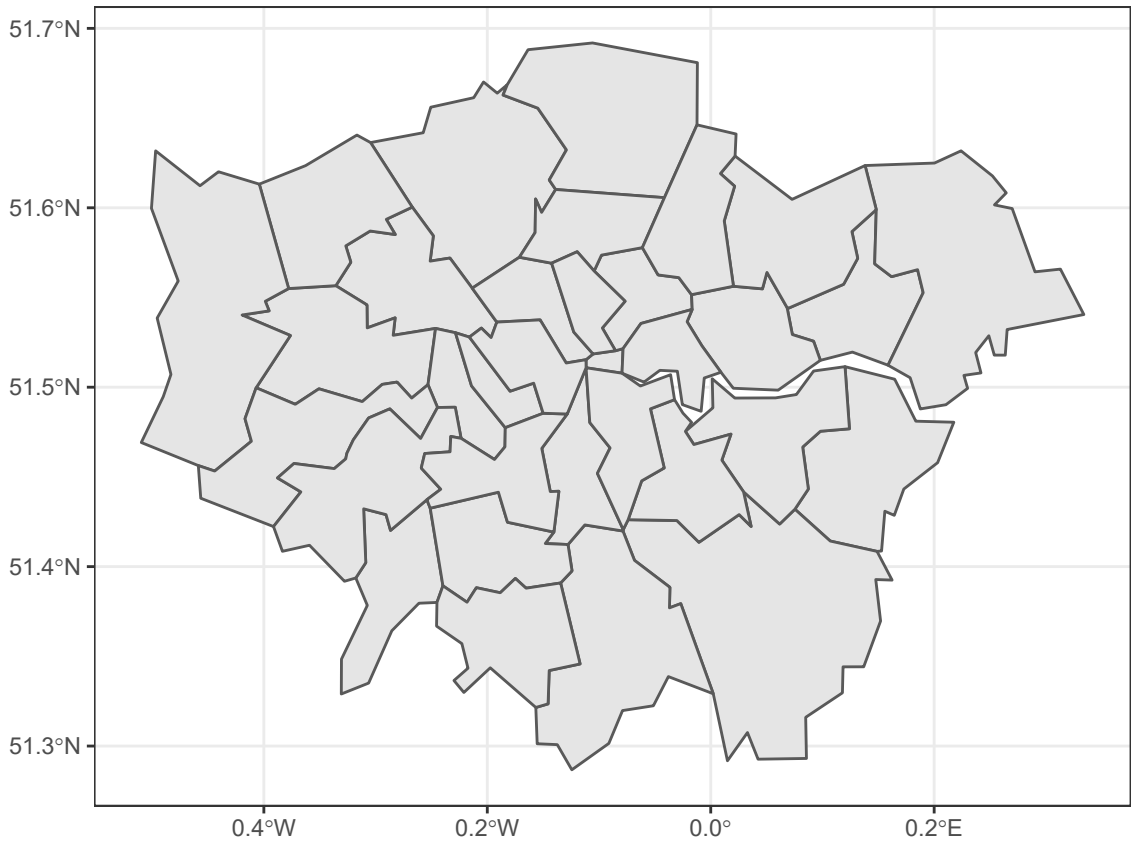
```
st_crs(ldn.map)
```

```
## Coordinate Reference System:
##   User input: OSGB36 / British National Grid
##   wkt:
## PROJCRS["OSGB36 / British National Grid",
##     BASEGEOGCRS["OSGB36",
##         DATUM["Ordnance Survey of Great Britain 1936",
##             ELLIPSOID["Airy 1830",6377563.396,299.3249646,
##                 LENGTHUNIT["metre",1]]],
##         PRIMEM["Greenwich",0,
##             ANGLEUNIT["degree",0.0174532925199433]],
##         ID["EPSG",4277]],
##     CONVERSION["British National Grid",
##         METHOD["Transverse Mercator",
##             ID["EPSG",9807]],
##         PARAMETER["Latitude of natural origin",49,
##             ANGLEUNIT["degree",0.0174532925199433],
```

```
##             ID["EPSG",8801]],
##         PARAMETER["Longitude of natural origin",-2,
##             ANGLEUNIT["degree",0.0174532925199433],
##             ID["EPSG",8802]],
##         PARAMETER["Scale factor at natural origin",0.9996012717,
##             SCALEUNIT["unity",1],
##             ID["EPSG",8805]],
##         PARAMETER["False easting",400000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8806]],
##         PARAMETER["False northing",-100000,
##             LENGTHUNIT["metre",1],
##             ID["EPSG",8807]]],
##     CS[Cartesian,2],
##         AXIS["(E)",east,
##             ORDER[1],
##             LENGTHUNIT["metre",1]],
##         AXIS["(N)",north,
##             ORDER[2],
##             LENGTHUNIT["metre",1]],
##     USAGE[
##         SCOPE["Engineering survey, topographic mapping."],
##         AREA["United Kingdom (UK) - offshore to boundary of UKCS within 49°45'N to 61°N and 9°W to 2
##         BBOX[49.75,-9,61.01,2.01]],
##     ID["EPSG",27700]]
```

We can transform the map to use *WGS84* CRS.

```
ggplot(ldn.map) + geom_sf() +
  theme_bw() +
  coord_sf(crs = st_crs(4326))
```



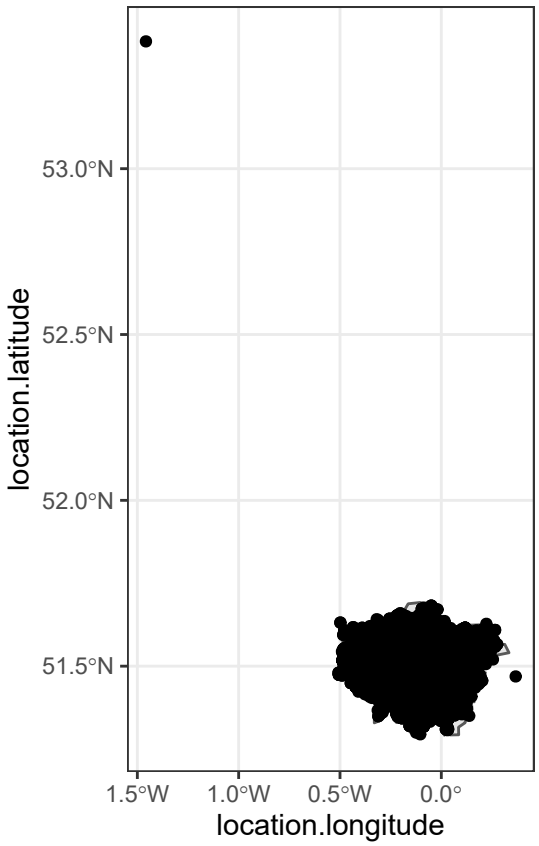
Now let's add some data to the map. Let's display the locations of police stop and searches on the map.

We can load the data from data.police.uk, as a JSON object with `readJSON`. The `flatten` method converts this into a usable data frame. Then we need to make sure that the coordinates are actually stored as numbers, and finally we remove any records without coordinates.

```
library(jsonlite)
#stop_search <- fromJSON("https://data.police.uk/api/stops-force?force=metropolitan&date")
stop_search <- fromJSON("stops-force.json")
stop_search <- flatten(stop_search)
stop_search$location.latitude <-
  as.numeric(stop_search$location.latitude)
stop_search$location.longitude <-
  as.numeric(stop_search$location.longitude)
stop_search <- stop_search[!is.na(stop_search$location.latitude),]
```

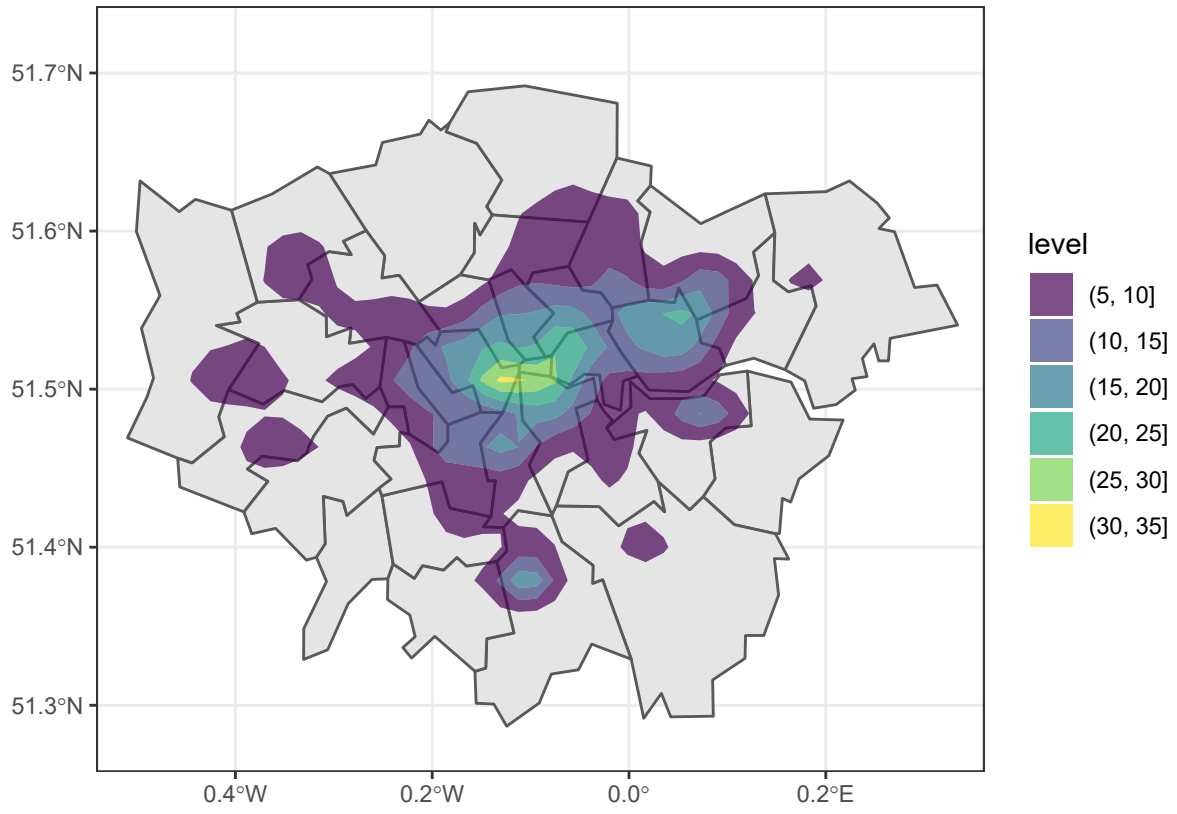
We use `geom_point()` to overlay the stop and searches on the map.

```
ggplot(ldn.map) + geom_sf() +
  geom_point(data = stop_search,
             aes(location.longitude, location.latitude)) +
  theme_bw() +
  coord_sf(crs = st_crs(4326))
```



That is clearly not quite right! There are way too many points on the map and two incidents are far outside of London. By creating a density map we show the density of incidents at a certain location instead of the actual dots and by specifying `xlim` and `ylim` we make sure to only show London.

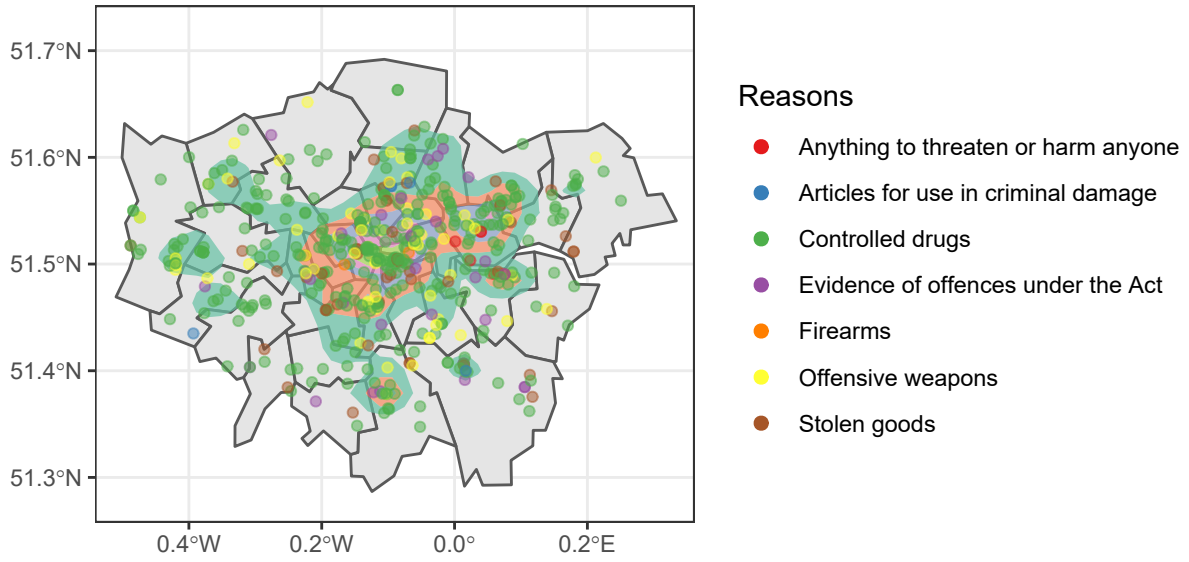
```
ggplot(ldn.map) + geom_sf() +
  geom_density2d_filled(data = stop_search, alpha = 0.7,
                        breaks = seq(5, 35, 5),
                        aes(location.longitude, location.latitude)) +
  theme_bw() +
  coord_sf(crs = st_crs(4326), xlim = c(-0.5, 0.32), ylim = c(51.28, 51.72)) +
  labs(x= "", y ="")
```



Finally, let's also display a sample of the stop and searches to get an idea for the reasons of the stop and searches.

```
stop_search_sample <- stop_search[sample(nrow(stop_search), 500),]
stop_search_sample <-
  stop_search_sample[!is.na(stop_search_sample$object_of_search),]
ggplot(ldn.map) + geom_sf() +
  geom_density2d_filled(data = stop_search, alpha = 0.7,
                        breaks = seq(5, 35, 5),
                        aes(location.longitude, location.latitude)) +
  geom_point(data = stop_search_sample, alpha = 0.5,
             aes(location.longitude, location.latitude,
                 colour = object_of_search)) +
  theme_bw() +
  scale_fill_brewer(palette = "Set2") +
  scale_color_brewer(palette = "Set1") +
  coord_sf(crs = st_crs(4326), xlim = c(-0.5, 0.32), ylim = c(51.28, 51.72)) +
  guides(fill = "none",
         colour = guide_legend(
           title = "Reasons",
           override.aes = list(
             alpha = 1,
             size = 2,
             fill = NA,
             linetype = 0
             ))) +
```

```
labs(x= "", y ="")
```


51.7°N
51.6°N
51.5°N
51.4°N
51.3°N
0.4°W
0.2°W
0.0°
0.2°E
Reasons
Anything to threaten or harm anyone
Articles for use in criminal damage
Controlled drugs
Evidence of offences under the Act
Firearms
Offensive weapons
Stolen goods

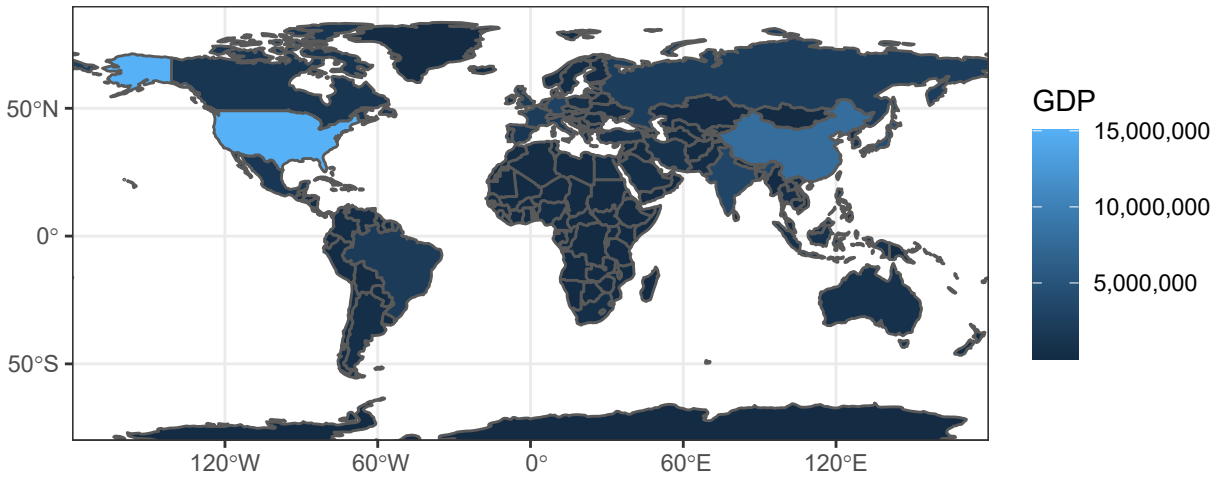
Exercises

1. Create a world map showing the GDP per country. Note, the object returned by `ne_countries()` contains the estimated GDP in the column `gdp_md_est`. Should you use a linear or an exponential scale?
2. Create a world map showing which economic group each country belongs to. Use the `economy` column.
3. Using the `ne_countries()` function, create a map showing only Africa. Show the population of each country.
4. The eurostat statistic tgs00047 contains the percentage of households that have internet access at home by NUTS level 2 regions. Visualise the newest available data.
5. The file `ukcities.csv` contains a list of all cities in the UK, their coordinates and population. Draw a map of the UK and add the cities as dots with the size proportional to the cities' population. Bonus: Add labels for the five largest cities.

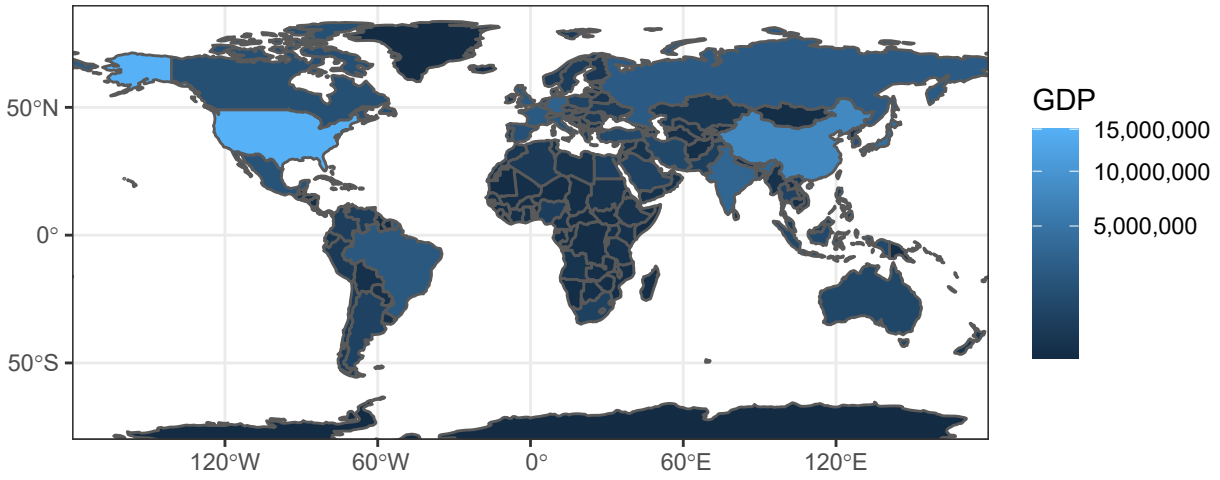
Solutions

1. Create a world map showing the GDP per country. Note, the object returned by `ne_countries()` contains the estimated GDP in the column `gdp_md_est`. Should you use a linear or an exponential scale?

```
world <- ne_countries(returnclass = "sf")
ggplot(world) + geom_sf() + aes(fill = gdp_md_est) +
  scale_fill_continuous(label = scales::comma) +
  labs(fill = "GDP") +
  theme_bw() + coord_sf(ylim = c(-80, 90), expand = F)
```

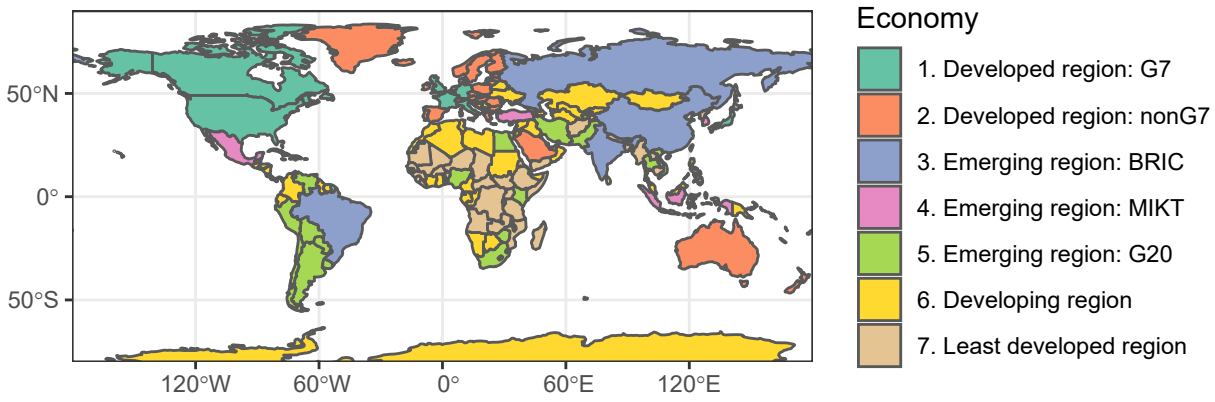



```
ggplot(world) + geom_sf() + aes(fill = gdp_md_est) +
  scale_fill_continuous(label = scales::comma, trans = "sqrt") +
  labs(fill = "GDP") +
  theme_bw() + coord_sf(ylim = c(-80, 90), expand = F)
```



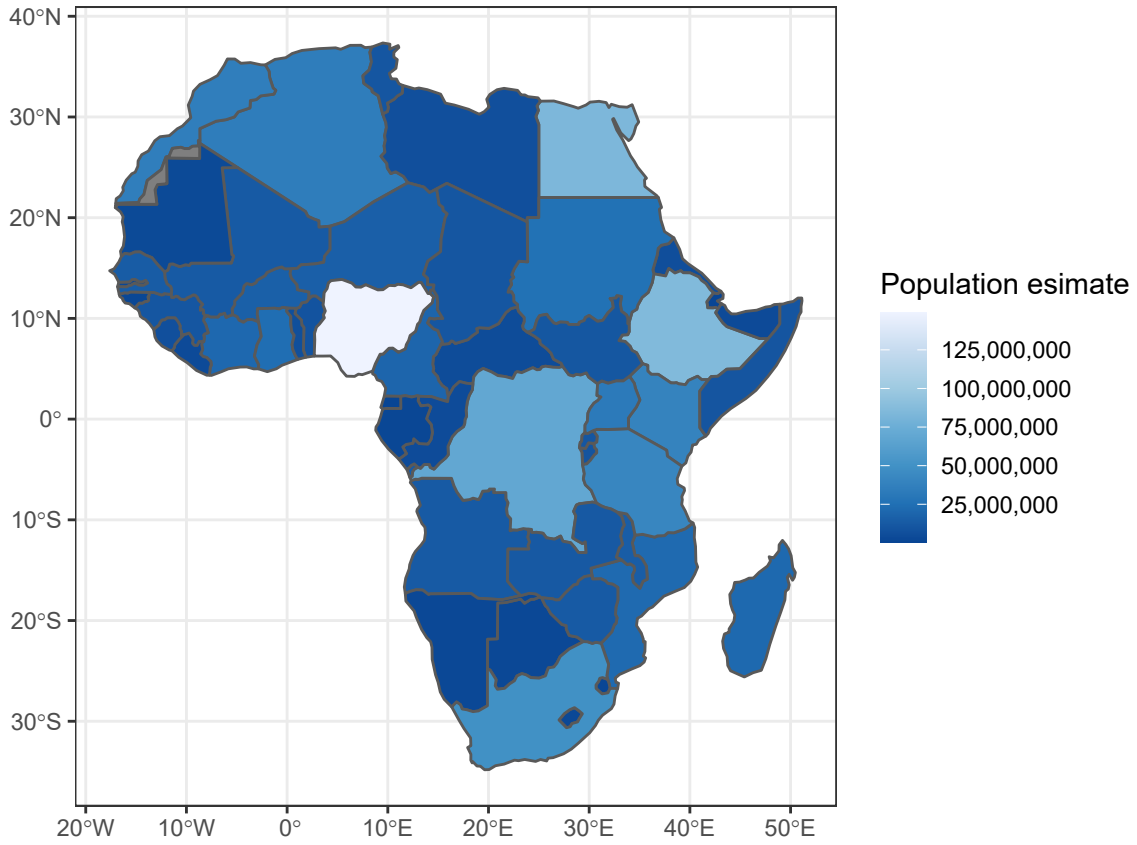
2. Create a world map showing which economic group each county belongs to. Use the `economy` column.

```
world <- ne_countries(returnclass = "sf")
ggplot(world) + geom_sf() + aes(fill = economy) +
  scale_fill_brewer(palette = "Set2") +
  labs(fill = "Economy") +
  theme_bw() + coord_sf(ylim = c(-80, 90), expand = F)
```



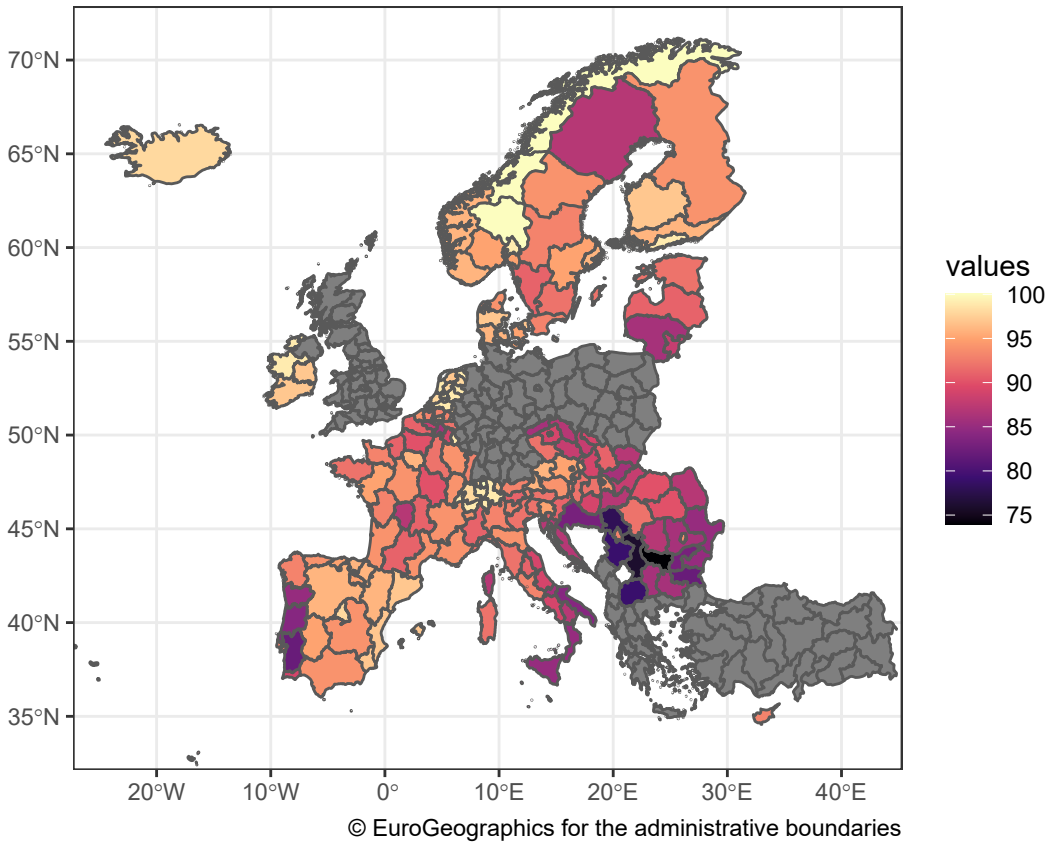
3. Using the `ne_countries()` function, create a map showing only Africa. Show the population of each country.

```
africa <- ne_countries(returnclass = "sf", continent = "Africa")
ggplot(africa) + geom_sf() + aes(fill = pop_est) +
  scale_fill_distiller(label = scales::comma, breaks = seq(0, 200e+6, 25e+6)) +
  labs(fill = "Population esimate") +
  theme_bw()
```



4. The eurostat statistic tgs00047 contains the percentage of households that have internet access at home by NUTS level 2 regions. Visualise the newest available data.

```
euro.map.regional <- get_eurostat_geospatial(nuts = 2)
data <- get_eurostat("tgs00047")
data <-
  data %>%
  group_by(geo) %>%
  slice_max(order_by = time, n = 1)
data.map <- left_join(euro.map.regional, data, by = c("id" = "geo"))
ggplot(data.map) + geom_sf() + aes(fill = values) +
  theme_bw() +
  scale_fill_viridis_c(option = "magma") +
  coord_sf(xlim=c(-24,42), ylim=c(34, 71)) +
  labs(caption = "© EuroGeographics for the administrative boundaries",
       x = NULL, y = NULL)
```



5. The file `ukcities.csv` contains a list of all cities in the UK, their coordinates and population. Draw a map of the UK and add the cities as dots with the size proportional to the cities' population. Bonus: Add labels for the five largest cities.

```
library(readr)
ukcities <- read_csv("ukcities.csv")
```

```
## Rows: 51 Columns: 4
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (1): city
## dbl (3): lat, lng, population
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
utla.map <- st_read("UTLA2019/Counties_and_Unitary_Authorities_(December_2019)_Boundaries_UK_BUC.shp")
```

```
## Reading layer `Counties_and_Unitary_Authorities_(December_2019)_Boundaries_UK_BUC' from data source
##   using driver `ESRI Shapefile'
## Simple feature collection with 216 features and 10 fields
## Geometry type: MULTIPOLYGON
## Dimension:     XY
## Bounding box:  xmin: -116.1928 ymin: 7054.1 xmax: 655653.8 ymax: 1218618
## Projected CRS: OSGB36 / British National Grid
```

```
largest <- slice_max(ukcities, order_by = population, n = 5)

ggplot(utla.map) + geom_sf() +
  geom_point(data = ukcities, aes(lng, lat, size = population), colour = "orchid3") +
  geom_text(data = largest, aes(lng, lat, label = city),
            size = 3, fontface = "bold") +
  scale_size_continuous(label = scales::comma,
                        guide = guide_legend(reverse=TRUE)) +
  theme_bw() +
  coord_sf(crs = st_crs(4326)) +
  labs(x = "", y = "")
```