# SPEC Lab REU R Resources: Data visualization with `ggplot2`: Introduction to Maps

Alix Ziff, Gaea Morales, Zachary Johnson, and Jasmine Chu
based on earlier materials by Therese Anders

Summer 2022

## ggplot2 continued

In part 3 of the walk-through-work, we introduce plotting simple maps with `ggplot2` and `maps`.

## Introduction to maps

We will use map data that is part of the `maps` package in R and does not require significant preprocessing for plotting. The [`maps` package] (https://cran.r-project.org/web/packages/maps/index.html) contains data on lines and polygons for a number of geographical units, including but not limited to, countries of the world, a database of large lakes, as well as United States federal states, counties, and cities. In this example, we retrieve data for a world map.

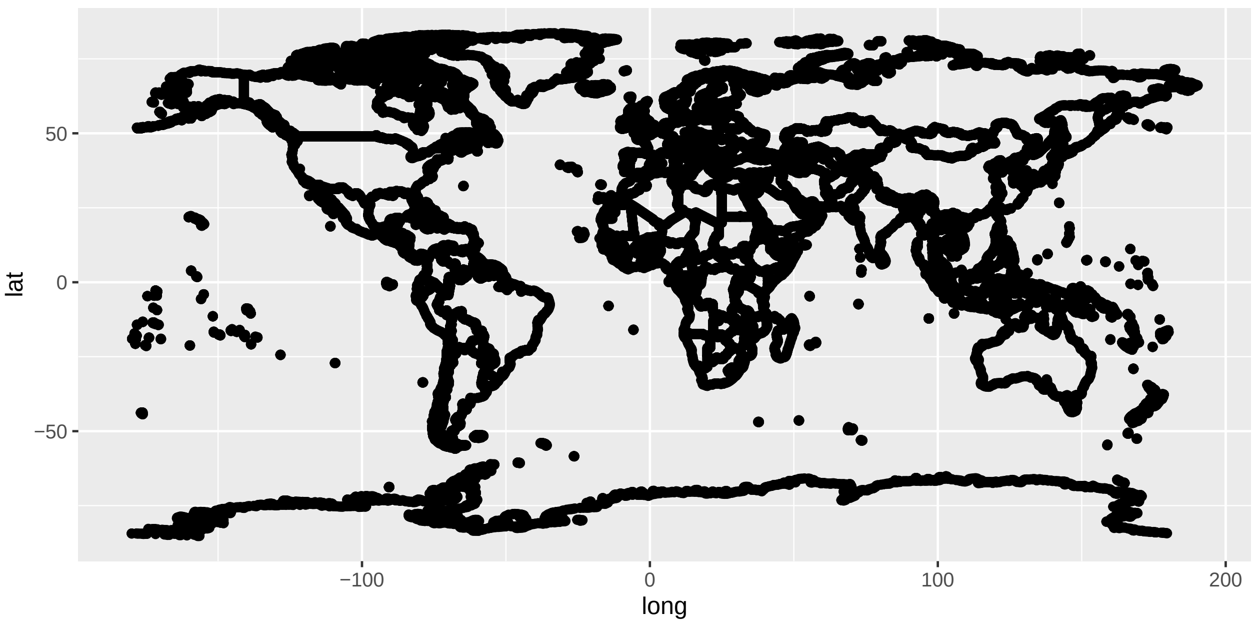
Before plotting the data, let us look at the structure of the data we drew from the `maps` package. The data are stored as a data frame and contains observations that are characterized by unique combinations of longitude and latitude values. In addition, each observation has the following attributes: group, order, region, and subregion if applicable.

```r
# rm(list=ls())
library(maps)
library(ggplot2)
library(dplyr)
library(countrycode)

dat_map <- map_data("world") # extracting the "world" data from the package
head(dat_map) # variable names of dataset 'dat_map'
```

```
##        long      lat group order region subregion
## 1 -69.89912 12.45200     1     1  Aruba      <NA>
## 2 -69.89571 12.42300     1     2  Aruba      <NA>
## 3 -69.94219 12.43853     1     3  Aruba      <NA>
## 4 -70.00415 12.50049     1     4  Aruba      <NA>
## 5 -70.06612 12.54697     1     5  Aruba      <NA>
## 6 -70.05088 12.59707     1     6  Aruba      <NA>
```

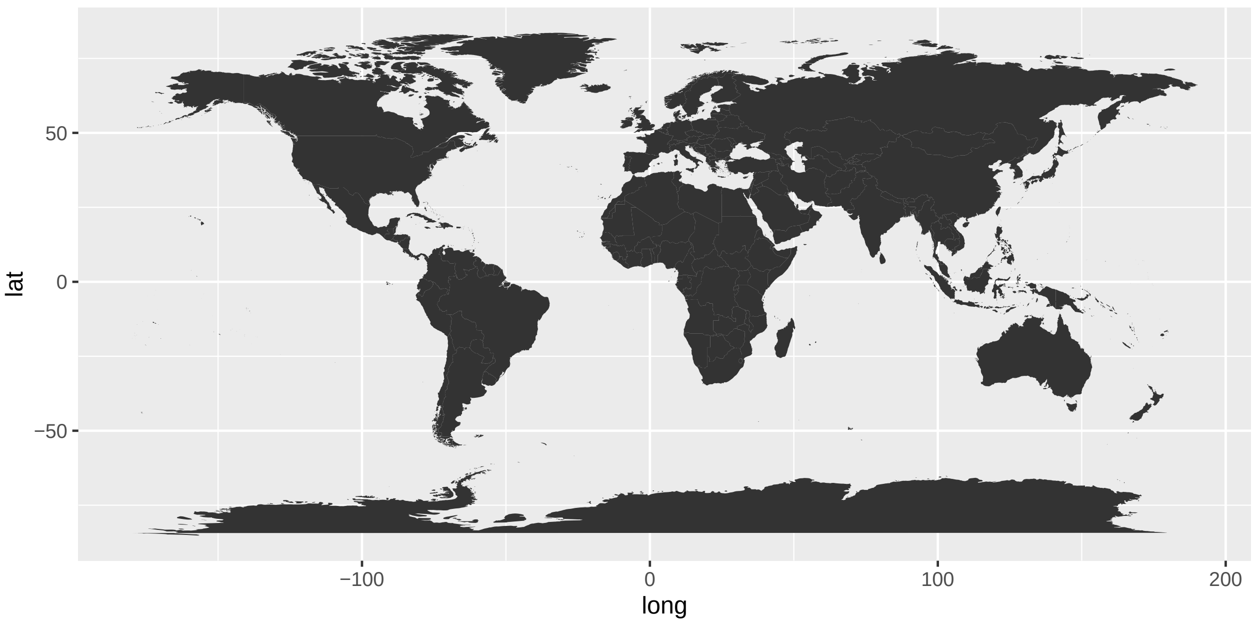
The longitude and latitude information denotes points along the borders of countries and geographical regions. We can represent them as points in the x-y-coordinate system, plotting the longitude along the x-axis and latitude along the y-axis.

```r
ggplot(dat_map, aes(x = long, y = lat)) + #setting longitude and latitude
  # data points along the x and y axis
  geom_point()
```



We can use `ggplot2`'s `geom_polygon()` function to plot the observations as polygons, rather than points. In order to do that, we need to specify the grouping parameter. The `geom_polygon()` function connects the points for each group in the order of the data set (think of connecting dots on a sheet of paper without lifting your pen). This means that if the data are not already in the appropriate order, we need to sort it before plotting.

```r
ggplot(dat_map, aes(x = long, y = lat, group = group)) +
  geom_polygon()
```



## Choropleth maps

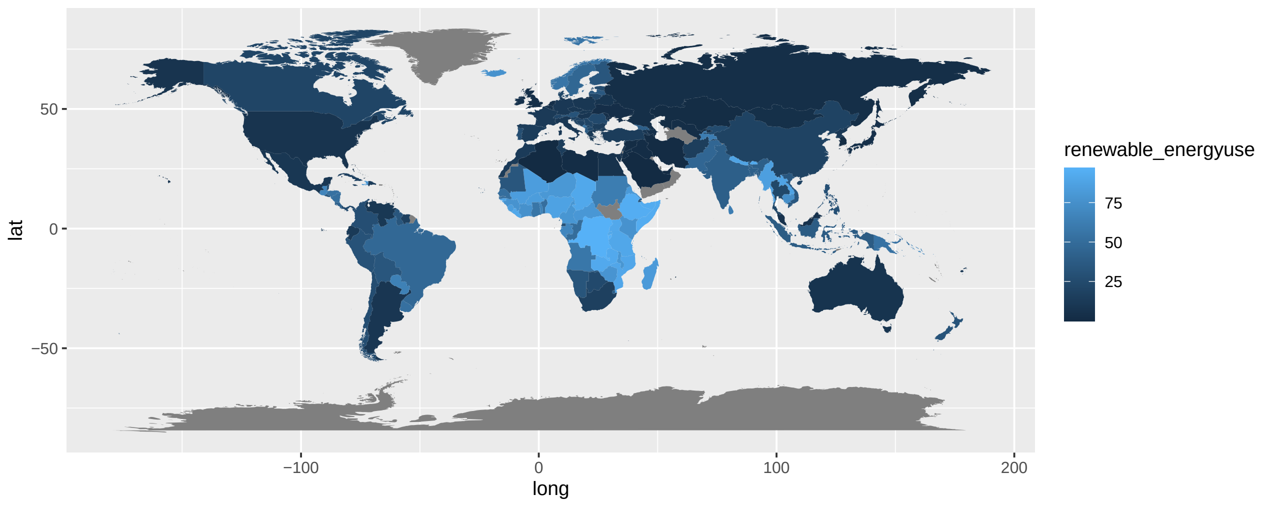
Choropleth maps use differences in shading of specific geographic regions to visualize aggregate or summary data for those regions. Here, we will plot a map of the world where the shading denotes differences in the use of renewable energy across countries.

To plot the choropleth map, we will first need to **merge** our world map data with the data from the World Development Indicators. Unfortunately, country names differ between the two data sets. Therefore, we need to use the `countrycode()` function from the `countrycode` package to create vector of matching country codes for the two data sets before merging. We will use the `full_join()` function from the `dplyr` package for merging to make sure that we do not drop missing values from either dataset to be merged.

```
dat_map$gwno <- countrycode(dat_map$region,
                            origin = "country.name",
                            destination = "gwn") # using the Gleditsch & Ward
                               # countrycode

dat <- read.csv("wdi_cleaned_part2.csv")

dat$gwno <- countrycode(dat$country,
                            origin = "country.name",
                            destination = "gwn") # # using the Gleditsch & Ward
                               # countrycode

merged <- full_join(dat_map, dat, by = "gwno") #merge by using full_join()
```

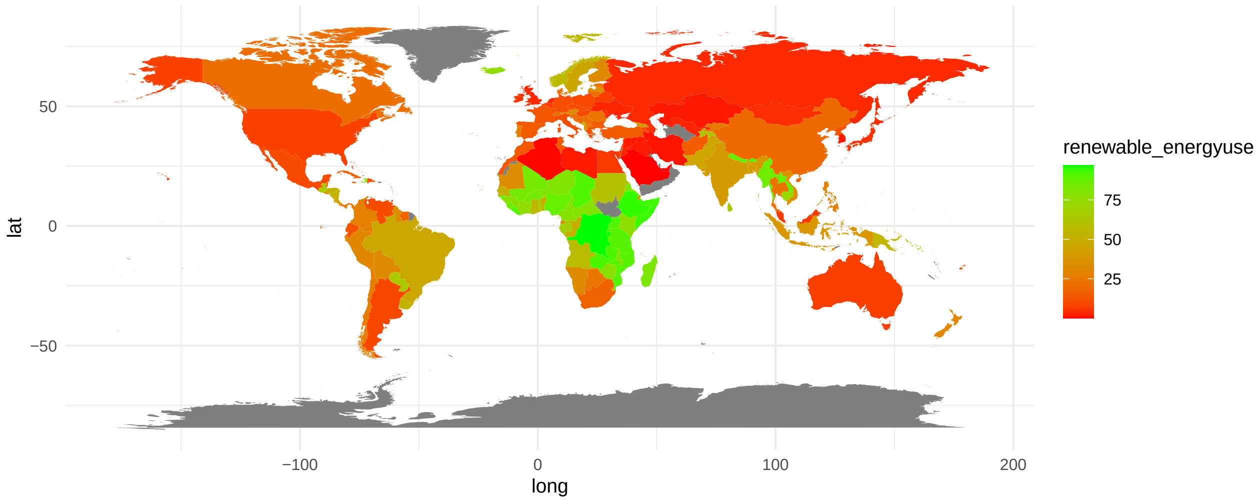
Now, we can plot our data. Suppose we want to illustrate the spatial variation of the usage of renewables in 2010. We plot the map as before, but pass the `renewable_energyuse` to the `fill` parameter inside the aesthetics argument. In addition, we specify that we only want to use data from 2010 using the `subset()` command. Note that we use our newly created `merged` data frame, rather than `dat_map`.

```
ggplot(subset(merged, year == 2010),
       aes(x = long, y = lat, group = group, fill = renewable_energyuse)) +
  geom_polygon()
```



The `ggplot2` default color scheme does not illustrate differences in the use of renewable energy particularly well. We can customize the color scheme to make differences between countries more apparent.

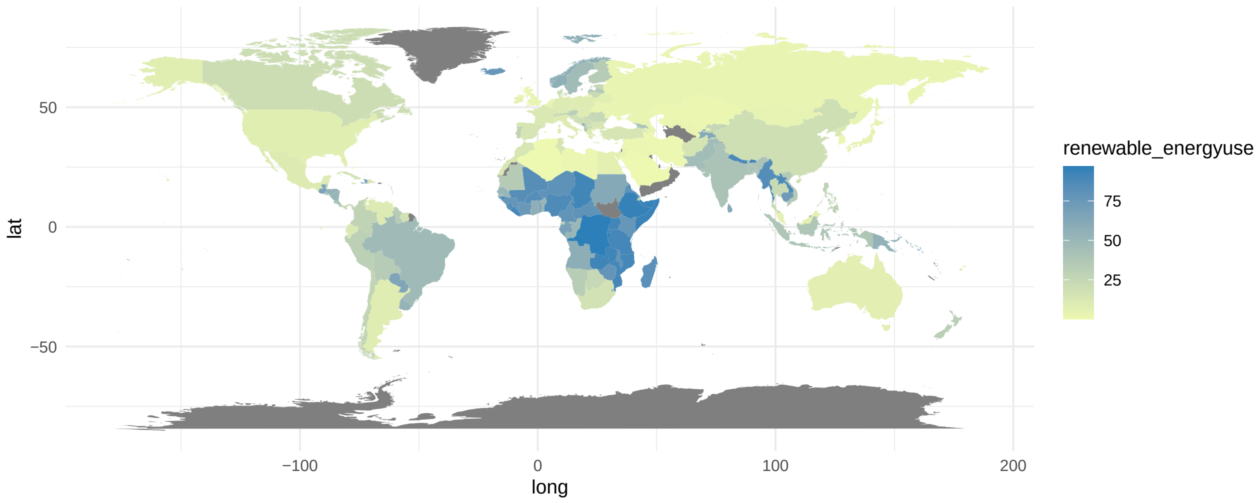
```r
ggplot(subset(merged, year == 2010),
       aes(x = long, y = lat, group = group, fill = renewable_energyuse)) +
  geom_polygon() +
  scale_fill_gradient(low = "red", high = "green") +
  theme_minimal()
```



**Helpful hint:** As we mentioned in prior data visualization parts, we can make an effort to differentiate colors in ways that are color blind safe (for instance, the above map is not because of the red/green colors). One of the most useful resources for maps is Color Brewer 2.0, where we can specify the type of values you are working with (sequential, diverging, or qualitative) and the number of values you need (up to 12).

You can then simply choose the color scheme, and copy in the generated HEX codes.

```r
ggplot(subset(merged, year == 2010),
       aes(x = long, y = lat, group = group, fill = renewable_energyuse)) +
  geom_polygon() +
  scale_fill_gradient(low = "#edf8b1", high = "#2c7fb8") +
  theme_minimal()
```


renewable_energyuse
75
50
25
50
0
−50
lat
−100
0
100
200
long