# 02.221 – Lab 8: Data visualization with R

The data needed for this lab is the same as for Lab 7. It is recommended to continue your work within the same RStudio project.

## Goals

The primary goal of this lab is to get familiar with different data visualization approaches in R, in particular with ggplot's approach to the 'grammar of graphics' and the different elements that constitute a 'gg' plot.

## Grammar of Graphics

The underlying approach that the *ggplot* visualization library is using to build graphs and visualizations is based on the 'grammar of graphics'. The central idea behind this is that we don't refer to a single graph as 'a bar chart' or 'a histogram' but rather dissect these visualizations into their smaller building blocks. Just like verbs, nouns and adverbs are some of the building blocks of languages so too are geometric objects, scales, axes and coordinate systems building blocks of every graph. This allows us to build visualizations in a flexible manner that go much beyond the default graph types available in software like, for example, Microsoft Excel.

In the previous lab assignment, you had your first exposure to ggplot's logic. Make sure you continue with the complete 1997-2015 dataset in the rest of this lab. For ease of use, we will assume you have the data stored in the 'acled' object. Take for example the following command:

```
ggplot(acled, aes(EVENT_TYPE)) + geom_bar()
```

Although that is a decent first plot for a simple bar chart, it is immediately clear that the x-axis labels are much too long to be plotted horizontally. As every element (axis, labels, scales etc.) is an individual, and changeable, building block in ggplot graphs, we can easily adjust this by adding additional parameters with the '+' operator.

```
ggplot(acled, aes(EVENT_TYPE)) + geom_bar() +
theme(axis.text.x = element_text(angle = 45, hjust
= 1))
```

Remember, whatever is last in the sequence will overrule previous parameter definitions. For example, the following will plot the labels at a 90 degree angle:

```
ggplot(acled, aes(EVENT_TYPE)) + geom_bar() +
theme(axis.text.x = element_text(angle = 45, hjust
= 1)) + theme(axis.text.x = element_text(angle =
90, hjust = 1))
```

If we wanted to include our graph in a report, we probably wanted to make sure the reader understood exactly what variable was plotted on each axis. The default variable names are often shortened for ease of use but for final graphs it is better to be explicit. Try to understand what each of the functions in the following statement do:

```
ggplot(acled, aes(EVENT_TYPE)) + geom_bar() +
theme(axis.text.x = element_text(angle = 45, hjust
= 1)) + xlab("Type of violent event") +
ylab("Number of events") + ggtitle("Violent events
in Africa 1997–2015")
```

Another element that often needs some fine-tuning is the tick marks used by default. For example, consider the following graph:

```
ggplot(acled, aes(YEAR)) + geom_bar()
```

By default, ggplot determines it will place a tick mark every 5 years (2000, 2010 etc.). Depending on your question, you might want to have a tick mark at a smaller interval. We can set the tick marks by supplying a parameter to the x axis scale:

```
ggplot(acled, aes(YEAR)) + geom_bar() +
scale_x_continuous(breaks = 1997:2015)
```

Or, to have a tick mark every other year:

```
ggplot(acled, aes(YEAR)) + geom_bar() +
scale_x_continuous(breaks = pretty(acled$YEAR, n =
8))
```

The same way, we can also 'zoom in' or limit the extent of an axis to a certain range. For example, if we are only interested in the years 2005-2015, we can do the following:

```
ggplot(acled, aes(YEAR)) + geom_bar() +
scale_x_continuous(breaks = 2005:2015, limits =
c(2004, 2016))
```

One of the key benefits of data visualization, especially in the initial stages of a research project, is that it allows us to get a quick sense of the distribution and dimensions of an entire dataset. This initial stage is often referred to as 'exploratory data analysis'. For example, instead of only visualizing the total number of events each year, we can subdivide per event type:

```
ggplot(acled, aes(YEAR, fill=EVENT_TYPE)) +
geom_bar()
```

By default ggplot uses a sequential color scheme for each event type. We know that event types are nominal variables so it would be better to use a qualitative scheme here. Luckily, ggplot has built-in support for a number of color schemes, including color brewer:

```
ggplot(acled, aes(YEAR, fill=EVENT_TYPE)) +
geom_bar() + scale_fill_brewer(type="qual",
palette = 3)
```

Instead of stacked bars, we can also place the bars next to each other or create a line chart instead. However, with a large set of categories, capturing too much information in a single graph quickly becomes messy:

```
ggplot(acled, aes(YEAR, fill=EVENT_TYPE)) +
geom_bar(position="dodge") +
scale_fill_brewer(type="qual", palette = 3)
ggplot(acled, aes(YEAR)) + geom_freqpoly(aes(group
= EVENT_TYPE, colour = EVENT_TYPE), stat =
"count") + scale_color_brewer(type="qual", palette
= 3)
```

This is why it is often so useful to generate small multiples for the various variables in your dataset. For example, instead of capturing everything in one graph, we can use ggplot's facet_grid() function to do this:

```
ggplot(acled, aes(YEAR)) + geom_bar() +
facet_grid(EVENT_TYPE ~ .)
```

You immediately see that the label for each 'strip' is cut short. As we are working within ggplot's grammar of graphics system, each element can be individually adjusted, including those labels. For example:

```
ggplot(acled, aes(YEAR)) + geom_bar() +
facet_grid(EVENT_TYPE ~ .) + theme(strip.text.y =
element_text(angle = 0))
```

Similarly, if we wanted to compare the temporal differences between different event types, it might be better to not have each strip use the exact same y-axis scale. We can set this accordingly:

```
ggplot(acled, aes(YEAR)) + geom_bar() +
facet_grid(EVENT_TYPE ~ ., scales = "free_y") +
theme(strip.text.y = element_text(angle = 0))
```

You should start to see how this same approach can be used to look at other variable and plot types as well. In the previous lab, you made two maps for different years of the data set. With the facet approach, you can make a set of small multiples for each and every year within the dataset:

```
ggplot(acled, aes(y = LATITUDE, x = LONGITUDE)) +
geom_point(alpha = 0.4, aes(colour = EVENT_TYPE,
size = FATALITIES)) + scale_size_area(max_size =
10) + facet_wrap(~ YEAR)
```

Now that you are somewhat familiar with ggplot's basic design philosophy, use its documentation (http://docs.ggplot2.org/current/), StackOverflow and Google to try and achieve the following:

1. Create a faceted plot that shows the temporal distribution (e.g. bar chart) of the number of fatalities (not events) per country. Discuss the advantages and disadvantages of such an approach vis-à-vis a set of small multiple maps where each map represents a single year.
2. Create a bar chart, pie chart, coxcomb and bullseye chart of the number of events per event type. Discuss the similarities and differences between each chart type in a short paragraph.

**Make sure that your charts have appropriate labels, titles, scales and** colors and embed each as within your homework submission.

## A first look at interactive graphics in R

So far we have used ggplot to create static graphs. Of course, many of the visualizations that we make today are specifically designed for use on the web and thus can use some interactivity as well. There are a number of libraries available that make it relatively easy to bring ggplot-like graphs to the web. The easiest to use with our limited experience so far is 'plotly' (https://plot.ly/r/). Install and load the plotly library (reference the previous lab if you're unsure on the procedure).

After loading plotly, creating interactive graphs is really as simple as feeding a ggplot object to the ggplotly() function. For example:

```
gg <- ggplot(acled, aes(YEAR, fill=EVENT_TYPE)) +
geom_bar() + scale_fill_brewer(type="qual",
palette = 3)
ggplotly(gg)
```

Behind the scenes, the ggplotly() function is translating ggplot's syntax into plotly's native syntax and subsequently into a html/js visualization. This works well for standard graph types but does not scale to more advanced visualization types, for which it is better to use plotly's own graph syntax.

Use the documentation available at [https://plot.ly/r/#basic-charts](https://plot.ly/r/#basic-charts) to select two chart types of your own choosing that will help visualize a certain aspect of the ACLED data. Use plotly's native function plot_ly() instead of the conversion function used above to build your visualization. When you are done, please create an account at rpubs.com. Rpubs is a platform for the publication of documents and analyses created in RStudio (more on that next week!). You can also use it to host your interactive visualizations, which it what we will use it for now. Send your visualization to rpubs by simply clicking the 'publish' button Publish next to your graph in RStudio. Publish both of your graphs to rpubs.com.

Again, **make sure that your charts have appropriate labels, titles, scales and colors**. In your homework, include a short paragraph on why you chose the two graphs types and, of course, include a link to both graphs.

## Assignment

On the class website, you will find the assignment for this lab. It consists of the ggplot and plotly graphs you were asked to make above. The assignment needs to be submitted as a single PDF file. Please make sure you submit the assignment by **March 29.**