02.221 – Lab 7: An introduction to RStudio and R

The data needed for this lab can be found on the course website or downloaded from Dropbox: https://www.dropbox.com/s/n5b51acilptrn8j/02221-Lab7.zip?dl=0. Extract the data from the zip file to an appropriate location within your documents.

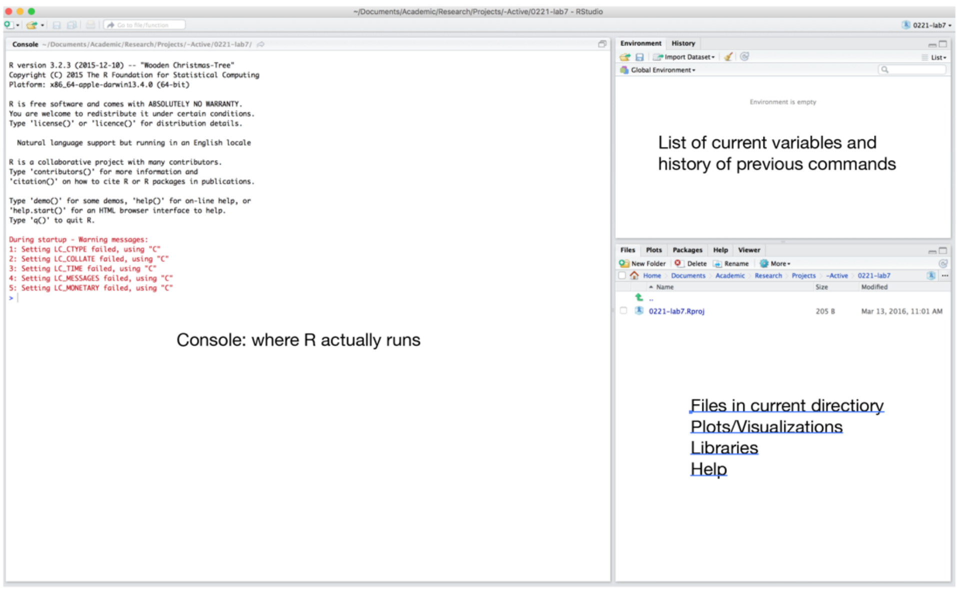
Goals

The primary goal of this lab is to get acquainted with R and the RStudio. We will do so through a preliminary analysis and visualization of the ACLED dataset that we used in the previous lab as well.

Getting to know R and RStudio

In its core, R is a programming language just like Python. However, R is specifically developed as an integrated environment that facilitates data handling, storage, analysis and visualization. Unlike most 'normal' programming languages, researchers spend a lot of time in R's *interactive mode*: running commands, inspecting output and, based on that output, running subsequent commands. Although R is perfectly capable of running independent scripts and programs without user input, especially in the early, exploratory stages of research the interactive mode is invaluable. R has a relatively steep learning curve and a set of idiosyncrasies but it is very well equipped for (big) data analysis and visualization and it's no surprise that it has quickly become one of the foundational software systems in industry and academia.

To make using R a bit more easy-going, we will use RStudio, which is an Integrated Development Environment (IDE) for R that provides a good UI and a range of useful functionality for researchers. When you open RStudio, you will see something like this:



```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

During startup - Warning messages:
1: Setting LC_CTYPE failed, using "C"
2: Setting LC_COLLATE failed, using "C"
3: Setting LC_TIME failed, using "C"
4: Setting LC_MESSAGES failed, using "C"
5: Setting LC_MONETARY failed, using "C"
> 
```

On the left you find the main R console. This is where you execute your commands and look at the output. The panels on the right is where RStudio starts to prove its worth. They're used to give you a quick overview of all the variables, plots, files etc. in your project. You will see soon enough how this comes in handy.

All commands that you need to execute within R during the remainder of this lab are set in a monospaced font. For example:

```
> summary(g)
```

Do not include the first character of the line '>'. That is just the R prompt, which is on your screen already.

In essence, R is just a fancy calculator. So let's get started on trying it out. It's not surprising you can do simple arithmetic:

```
> 2+6*5
```

But you can also do more advanced calculations:

```
> log(10)
```

Notice that R prints [1] in front of the result. This has to do with the way that R prints numbers and vectors. The number in brackets is the index of the first number on that line. Let's see how that works when there is more than one number to print:

```
> sample(100, 10, replace = TRUE)
```

What do you see now? sample(), just like log(), is a *function*. In R, the name of the function is always directly followed by parentheses. Within the parentheses you provide the arguments that you want to feed into the functions. Can you guess what sample() does, and what each argument is for? *Write your answer down.* If you are unsure what a specific command does, or what kind of input it expects, you can always check the help:

```
> help(sample)
```

This will open the documentation for that specific function in the lower right panel. It is nice to be able to calculate things, but calculations often yield values that you want to use as input for even more calculations. We need a way to store R's intermediate results! We can easily do so by assigning values to *variables* or *objects*. To assign the outcome of our very first calculation to an object, enter:

```
> x <- 2+6*5
```

‘<-‘ should be read as a single symbol. It is an arrow pointing to the variable to which the value is assigned. It is similar to how other programming languages use the ‘=’ sign for assignment. We don't see any output in R this time but the output of your calculation is stored. If you just type the name of the variable, R will print its value:

```
> x
[1] 32
```

We can now use this variable in subsequent calculations:

```
 > x + x + 6
```

Using just a single number is pretty limiting. R lets you store entire vectors in one variable. A vector is nothing more than an array of elements, in this case numbers:

```
> distance <- c(10,15,20)
> distance
[1] 10 15 20
```

Remember that you can use help(c) to find out what the c() function exactly does. Basically it combines all of its arguments together into a vector. Note that you can get individual elements within the vector by using square brackets.

```
> distance[1]
```

Contrary to most other languages, R starts indexing at 1 instead of 0. Keep that in the back of your mind because it will cause you problems sooner or later. We can do calculations on the distance object just like we did on single numbers. Let assume our distance object is in kilometers but we would like to know meters:

```
> distance*1000
```

If we know the time it took to cover this distance, we can also calculate average speed:

```
> time <- c(2,2.25,2.5)
> speed <- distance/time
```

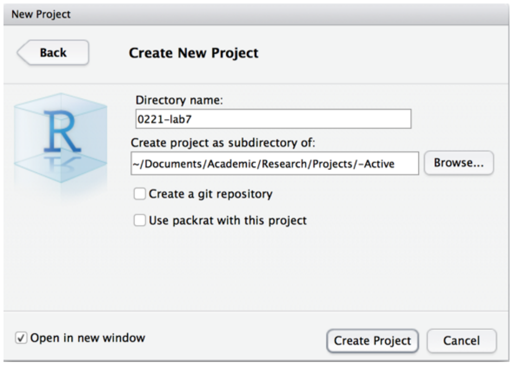
You can easily visualize your data as well by plotting your variables:

```
> plot(distance,speed)
```

Notice the plot appearing in the lower-right panel, while you will see an overview of your objects, as well as your command history, in the upper-right panel.

Reading and exploring data

Before we start reading in and analyzing some 'real' data, it is good practice to start a new RStudio project first. Go to File | New Project and give your project and appropriate name and location. Afterwards, make sure you move/copy the data you downloaded for this lab into the same folder.



New Project

Back — Create New Project

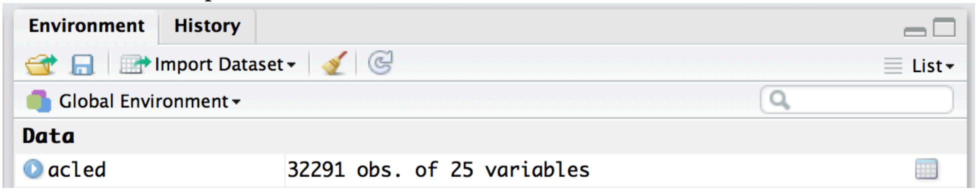
Directory name: 0221-lab7

Create project as subdirectory of: ~/Documents/Academic/Research/Projects/-Active

To read in the dataset, we can use the read.csv() function. This is one of the strengths of the R environment: there is a whole family of read.* functions that allow you to read in almost any data format you can think of. Let's assign the result of the function to an object so we can interact with it later.

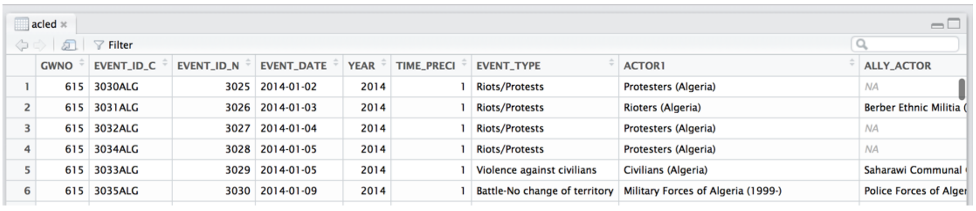
```
> acled <- read.csv("acled-2014-2015.csv",
stringsAsFactors = F)
```

The result of the read.csv() function is an object of the data.frame class (you can always check the class of an object through `class(object)`). A data.frame is similar to the attribute table in QGIS: it is basically a collection of variables or vectors (the columns), all of equal length (the rows). The difference with an ordinary matrix is that the type or class of each column can be different (i.e. you can mix integers with strings etc.). There are a number of ways to inspect the resulting object. First, RStudio provides a graphical way to look at the data.frame by simply clicking on it in the Environment pane.



acled 32291 obs. of 25 variables

You will now see an interactive table appear that you can browse, filter and search.



acled

	GWNO	EVENT_ID_C	EVENT_ID_N	EVENT_DATE	YEAR	TIME_PRECI	EVENT_TYPE	ACTOR1	ALLY_ACTOR
1	615	3030ALG	3025	2014-01-02	2014	1	Riots/Protests	Protesters (Algeria)	NA
2	615	3031ALG	3026	2014-01-03	2014	1	Riots/Protests	Rioters (Algeria)	Berber Ethnic Militia (
3	615	3032ALG	3027	2014-01-04	2014	1	Riots/Protests	Protesters (Algeria)	NA
4	615	3034ALG	3028	2014-01-05	2014	1	Riots/Protests	Protesters (Algeria)	NA
5	615	3033ALG	3029	2014-01-05	2014	1	Violence against civilians	Civilians (Algeria)	Saharawi Communal
6	615	3035ALG	3030	2014-01-09	2014	1	Battle-No change of territory	Military Forces of Algeria (1999-)	Police Forces of Alger

R itself also provides a number of functions that allow you to get a quick sense of a dataset and its structure. Execute each one of the below functions and use the output of the function as well as the help functionality to understand what each function does. *Write your answer down.*

```
> str(acled)
> summary(acled)
> head(acled)
> tail(acled)
```

Just like the vectors earlier, we can use the square brackets '[]' to select specific elements within the data.frame. As we now have two dimensions (rows and columns) the syntax is as follows: df[row, column]. Try to understand the difference between the following commands:

```
> acled[1,1]
> acled[,1]
> acled[1,]
```

You can use the same technique to select a series of consecutive rows:

```
> acled[1:3,]
```

Or based on another vector:

```
> acled[c(1,5,7,9),]
```

Or combined:

```
> acled[c(1,5,7,9,10:14),]
```

Of course, referring to columns by their index is not so intuitive. They do have names after all! You can do so by using the dollar-sign after the object name:

```
> acled$FATALITIES
```

This will print only the FATALITIES variable of the data.frame. With that simple building block you can do some pretty advanced data crunching. For example, try executing:

```
> acled$FATALITIES > 400
```

This will evaluate the statement for every element in the vector and return true/false. You will see a long vector with true/false as a result. You can use this as your row index to subset your data.frame:

```
> acled[acled$FATALITIES > 400,]
```

This will return only the rows with more than 400 fatalities. How many events were there in 2014-2015 that satisfy that criterion? *Write your answer down.*

You can also use the dollar-sign notation to do some quick tabulation, for example in combination with the table() function.

```
> table(acled$EVENT_TYPE)
> sort(table(acled$EVENT_TYPE))
```

Try to use the same technique to find out the COUNTRY with the largest number of events in the database. *Write your answer down.*

Finally, we can use the dollar-notation to create new columns as well. For example, let's assume we want to find out whether there's a specific day of the week that sees more events than other days. To do so, we would need to convert the date variable and calculate the weekday for each individual date. We can do so in one command like this:

```
> acled$weekday <- wday(as.Date(acled$EVENT_DATE))
```

During which day of the week do most events happen? *Write your answer down.*

Visualization

You have now completed some initial exploration of your dataset and, admittedly, some of the functions used so far can be pretty useful. However, instead of looking at the output of several summary and tabulation functions, it might be better to provide a visual summary of the data as well. To do so, we'll extend the functionality of base R with that of a visualization library. Libraries are a very important part of R's ecosystem. Base R is kept relatively lean, with all extra functionality contained in additional libraries or packages. We can install and load libraries from within R. The installation only has to be done once, but you will need to load specific libraries that you need for every new project or analysis.

To install the visualization library that we will use, execute the following command:

```
> install.packages('ggplot2')
```

After installation, you can load the library:

```
> library(ggplot2)
```

We can now use the ggplot() function to start building plots. For example, execute the following:

```
> ggplot(acled, aes(FATALITIES)) +
geom_histogram(binwidth = 5)
```

The syntax of the ggplot function is as follows: ggplot(dataset, aes(x-variable, y-variable)). This initial function builds up the data used for the plot. Subsequently you have complete freedom to create different plot types based on that data. geom_histogram() creates a histogram but we can just as easily create an ordinary bar chart:

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

Or a bar chart that doesn't use the count of cases but rather a specific y-axis variable:

```
> ggplot(acled, aes(EVENT_TYPE, FATALITIES)) +
geom_bar(stat = "identity")
```

With the same techniques, we create a temporal line chart that shows the number of fatalities per month. For ease of use we first create a new variable containing the year-month combination:

```
> acled$month <- substr(acled$EVENT_DATE, 1, 7)
```

Then we plot the result, aggregated by month:

```
> ggplot(aggregate(FATALITIES ~ month, acled,
sum), aes(month, FATALITIES, group = 1)) +
geom_line()
```

Do you understand what the substr() and aggregate() functions do? Try to find out and *write your answer down*.

Finally, we can also quickly create plots of our geographic variables with the geom_point() function:

```
> ggplot(acled, aes(y = LATITUDE, x = LONGITUDE))
+ geom_point()
```

These are all very basic plots but you can start to see how they are the foundation under more advanced visualization. For example, with relative ease we can create a categorized proportional symbol map:

```
> ggplot(acled, aes(y = LATITUDE, x = LONGITUDE))
+ geom_point(alpha = 0.4, aes(colour = EVENT_TYPE,
size = FATALITIES)) + scale_size_area()
```

To finish your assignment, download the complete 1997-2015 dataset for Africa from http://www.acleddata.com/data/version-6-data-1997-2015/. Read the data into R and use it to answer the following questions:

- What was the most frequent event type in 2004? (Try answering through a bar graph)
- Which year had the most fatalities due to violent events against civilians? (Try answering through a line graph)
- Create two proportional symbol maps showing the location and number of fatalities of events in 1999 and 2004. Save your maps as .png (by clicking on the Export ▾ button) and include them in your final document.

Assignment

On the class website, you will find the assignment for this lab. It consists of the answers to the questions asked through the lab as well as the final questions using the entire 1997-2015 dataset. The assignment needs to be submitted as a single PDF file. Please make sure you submit the assignment by **March 22.**