ANT 291
Lab #2 - Introduction to the R language

Fall 2017

1. Loading a CSV file to your R workspace

- Download the **csv file** named “climate data.csv” from CANVAS.
- Open RStudio and create a new R Script. The easiest way to load the csv file is to use the “Import Dataset” menu (in RStudio only). You’ll find this menu in the “Environment” tab of one of the four RStudio windows. Once you’ve imported “climate data.csv”, you’ll see a command looking like this in your console window:

```
climate.data <- read.csv("~/Downloads/climate data.csv")
```

- Copy-paste this command into your R Script. Save your script. You can (and should) properly anotate your R Script using the “#” sign:

```
###########################################################
########################Lab #2 script#####################
###########################################################
#First, load the data:
climate.data <- read.csv("~/Downloads/climate data.csv")
```

- There are two ways to look at the data in R. You can double-click on the “climate.data” object in the “Environment” tab, or you can type:

```
View(climate.data)
##by the way, please note the R is case-sensitive:
View(climate.Data)##error message!
```

- If the data file is large, you can just look at the first few rows:

```
head(climate.data)
```

- What does the following code do?

```
class(climate.data)
?data.frame
```

Have a careful look at the different sections of the help file.

2. Use git to keep track of all the changes you make to your R Code

In Lecture, you were shown how to use the server program “git” and the client software “GitKraken” to monitor changes you make to your code.

- Open GitKraken and initialize a new repository corresponding to the folder where you saved you R Script.
- Stage all changes and commit.

3. A few simple R functions

- What does the following code do? Insert them in your R Script and add commments.

```
dim(climate.data)
nrow(climate.data)
ncol(climate.data)
dim(climate.data)[1]
dim(climate.data)[2]
names(climate.data)
colNames <- names(climate.data)
which(colNames=="Date")
```

- In Lecture, I showed you how to extract a few values from a vector. Each column of a data.frame object is a vector (a one-dimensional object and all values have the same type).

```
myVector <- climate.data$Date
fourDatesFromMyVector <- myVector[3:6]
##is identical to:
myVector <- climate.data[,2]
class(myVector)##myVector is a vector of class "factor", which means it's a list
#of character strings that also includes a list of all possible values (the levels).
#The "factor" class is what we use when we do an ANOVA (for the categorical variable)
fourDatesFromMyVector <- myVector[3:6]
```

- In the code above, I use [,2] to get the second column from my data.frame. Why this comma? This will help you understand the role of this comma:

```
climate.data[3, 2]  ##which value is extracted here?
# If I want to re-use this value later, I need to save it as a variable:
allDates <- climate.data[3, 2]
# the comma is there to indicate if we want to select rows or columns. There
# are two ways to do so:
climate.data[, 2]  ##nothing new here. It's equivalent to:
climate.data[, c(FALSE, TRUE, FALSE)]
## you can use a vector of logical values to select data.frame rows or
## columns (or select values from a vector)! Here is an application:
colNameEqualsDate <- names(climate.data) == "Date"
colNameEqualsDate
climate.data[, colNameEqualsDate]  ##or, directly:
climate.data[, names(climate.data) == "Date"]
# looks a little bit complicated? Take your time and ask if you have any
# questions.

# Now try to understand the following lines of code. Make sure you add
# comments when you add your code to your Script:
thirdRow <- climate.data[3, ]  ##the comma is now after the number
thirdRow
class(thirdRow)  ##NOT A VECTOR!
# Why is 'thirdRow' a data.frame and not a vector? I selected a single row,
# after all! Can I coerce a row to turn into a vector of character values?
# Yes, but there usually isn't any reason to do so. Here a way to do it,
```

```r
# though:
climate.dataMatrix <- as.matrix(climate.data)
thirdRowMatrix <- climate.dataMatrix[3, ]
class(thirdRowMatrix)
## thirdRowMatrix is a vector. Why? Because Matrix objects contain one type
## of vatiable only. Could be 'character' or 'numeric' or something else.
## That's THE big difference with data.frame objects.
```

- Now let’s sort climate.data by date (i.e., using the second column, from least to greatest).

```r
## slow way:
sort(allDates)  ##does the job but only for the second column and that's not what we want.
dateIndices <- order(climate.data$Date)
dateIndices  ##what are these integers?
climate.dataSortedByDate <- climate.data[dateIndices, ]
head(climate.dataSortedByDate)  ##nice!
## fast way:
climate.dataSortedByDate <- climate.data[order(climate.data[, 2]), ]
## We were able to sort the table, despite the fact that the dates were not
## coded as 'date' objects! R just considered them as character strings.
## Still, R was able to sort the dates properly. Why is that? Well, because
## the dates were formatted as Year-Month-Day instead of Month-Day-Year. It
## is good practic to ALWAYS format dates this way: Year-Month-Day, e.g.
## 2017-10-04. It's the international standard (ISO 8601). Change your system
## locale settings and/or your Excel preferences if needed!
```

- It’s time for a new Git stage/commit!
- A few more useful functions to insert in your R script (don’t forget to add your own comments):

```r
summary(climate.dataSortedByDate)
##column 'source' only has 1 possible value. Let's delete this column. I'm also renaming
#my data.frame because I'd rather use a shorter name.
dat <- climate.dataSortedByDate[,-1]##intuituve
head(dat)
#Let's also display the "mean" column first, and then the Date column
dat2 <- dat[,c(2,1)]
head(dat2)
##the numbers in the "Mean"" column are the average global temperature anomalies in
#degrees Celsius relative to base period 1951-1980.
##Let's rename this column
names(dat2)[1] <- "mean.Temperature.Deviations"##avoid spaces when possible
##Let's plot mean.Temperature.Deviations=f(Date)
plot(dat2$Date, dat2$mean.Temperature.Deviations)
#or
plot(dat2$mean.Temperature.Deviations~dat2$Date)
#or
plot(mean.Temperature.Deviations~Date, data=dat2)
##the x-axis looks a little funny. Let's tell R that dates aren't simple character
#strings of class factor
class(dat2$Date)##just checking
dat2$Date <- as.Date(dat2$Date)
```

```r
class(dat2$Date)##Now R knows that Dates are dates...
range(dat2$Date)
##The plot() function is pretty smart and attempts to make nice plots if it can
plot(mean.Temperature.Deviations~Date, data=dat2)##nicer!
?par ##most possible arguments are listed in this help page. See also:
?plot.window
?plot
?plot.default

plot(mean.Temperature.Deviations~Date, data=dat2, type="l", col="red", bty="l", las=1,
     xlab="", ylab="Deviation from 1951-1980 average (Celsius)",
     main="Global surface temperature: 1880-2016")##nicer!
###add a grey line:
abline(0,0, lty=2, col=grey(level = 0.5))
###add a smoothed line. Hang on tight, we're going to put all the code in one line.
?loess
lines(dat2$Date, predict(loess(mean.Temperature.Deviations~as.numeric(Date), span=.5,
                               data=dat2)), lwd=2)
lines(dat2$Date, predict(loess(mean.Temperature.Deviations~as.numeric(Date), span=.2,
                               data=dat2)), lwd=2, lty=3)
lines(dat2$Date, predict(loess(mean.Temperature.Deviations~as.numeric(Date), span=2,
                               data=dat2)), lwd=2, lty=5)
```

4. Homework (very short this week)

Due on Monday!

1. Use the following functions to simulate 100 data points according to the following linear model:

$$y_i = -3x_i + 2 + \epsilon_i \tag{1}$$

$$\epsilon_i \sim \mathcal{N}(\mu = 0,\, \sigma = 0.2) \tag{2}$$

$$x_i \sim \mathcal{U}(min = 0, max = 1) \tag{3}$$

2. Let's say your variables are named x and y. Plot x vs y. Then, fit the following models to your data:

```r
model1 <- lm(y ~ x)
model2 <- glm(y ~ x)
```

3. Look at the model outputs and try to interprete as many elements as you can:

```r
summary(model1)
summary(model2)
```