Lab 1: Getting started with R and RStudio

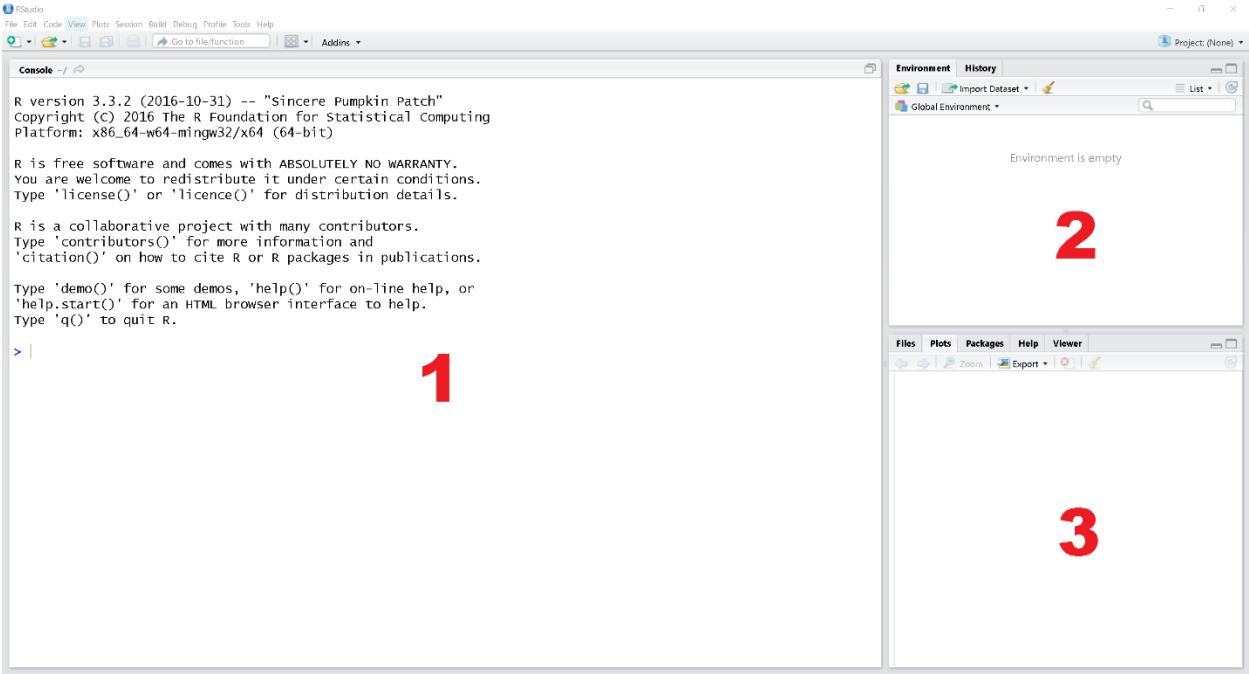
Questions? david.montwe@ualberta.ca or isaacren@ualberta.ca

1. Installing R and RStudio

To install R, go to https://cran.r-project.org/ and click on the 'Download R for Windows' link. On the next page click on 'install R for the first time' and finally 'Download R for Windows'. After downloading, click on the .exe file and install R by accepting all defaults.

RStudio is a so-called 'integrated development environment' or IDE, that allows for more efficient work flows and management of R. It can be downloaded from http://www.rstudio.com/download. Please install RStudio and launch the program.

2. Become familiar with the Rstudio environment



You should see three main panels:

1. The R-console, where you can type commands that are then interpreted by R.
2. The Global environment. All files you import, and all objects you create will be listed here, but we will come back to this later.
3. This panel has multiple tabs. For now, let's focus on three of them. The Plots tab is where the graphics are displayed. Packages contains a list of all the statistical and graphical 'add-ins' (called packages in R) that are currently installed. This is also where you can install new packages and update existing ones. The Help tab gives you access to the documentation of R packages.

3. Let's get started with R

R can be used like a calculator. Click into the console panel and type:

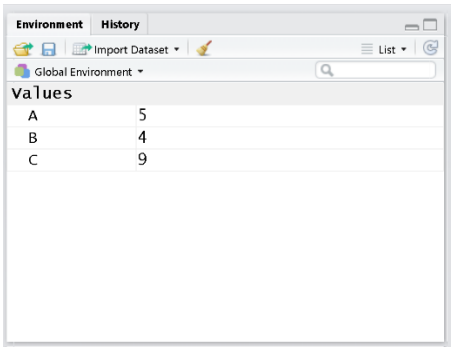
```
2+3
```
(Now press "Enter" on your keyboard – do this to run lines of code in the R-console)

R is an object-oriented programming language. This means that you can store data to objects and "call" them up again:

```
A=2+3
A
a
```

Typing "a" will produce an error message because R is case-sensitive.

```
B=4
A+B
C=A+B
C
```



Notice that the environment panel is now populated with the objects A, B &C.

Next we will apply some mathematical functions:

```
sqrt(C)
C^3
log(C)
log(C,10)
abs(-5)
```

Every function in R comes with a documentation and examples. Try:

```
?log()
?abs()
```

Multiple values of the same data type are called vectors. We can create some example using the c() function.

```
X=c(1,4,3,5,7)
Y=c(5,7,9,4,8)
```

Check the environment panel. R automatically recognized that the objects contain numeric values. Now let's calculate the arithmetic mean and standard error of X and do some math:

```
mean(X)
sd(X)
X*10
Z=Y+3
Z
boxplot(X,Y,Z)
```

Note that the plot is displayed in the plot window. We can even already apply a statistical test:

```
t.test(X,Y)
t.test(X,Z)
```

Vectors can be transformed into matrices with the cbind() function (c stands for column here). The function as.data.frame() transforms the resulting matrix into a data frame.

```
K=as.data.frame(cbind(X,Y,Z))
```

Check the environment panel. K is now listed under 'Data'. Click on the  icon next to K to get a list of columns of K.

```
X=X*10
K
```

Note that the first command did not change the X-Column in K. Specific columns of a data frame can be accessed using the $-sign:

```
K$X=K$X*10
t(K)
plot(K)
```

4. Working efficiently with RStudio

Writing all commands directly into the console is not an efficient way to work with R. RStudio contains a script editor that can be used to create and save scripts of code. This creates a reproducible archive of your analysis. Before we create the first script, please create a new folder at a place you like, for example, on your desktop: *Right click – New – Folder – Lab1*

Now create a new script in RStudio:

In RStudio, go to "File" > "New file" > "New R Script"

Now the "Console" window is split in half and we have a new panel where we can enter text.

In the new panel, write a few lines of code, for example:

```
X=c(1,4,3,5,7)
mean(X)
```

Now save this script by hitting the save button 

Navigate to your "Lab1" folder and enter "Lab1_script" as a file name.

If you close RStudio now (RStudio will ask if the workspace should be saved; select "No"), a double click on the "Lab1_script.R" file in your Lab1-folder will open RStudio and contains your commands.

Click on 

to run the entire script in the R-console, or select individual lines of code and hit "Run".

5. Importing and analysing data

There are several ways to import data to R, but I highly recommend using Excel-generated CSV files (Comma Separated Values). There are several advantages to CSV files: (1) they are plain text files, which are good for long-term data archiving, (2) almost any software package (including R) can import them error-free, and (3) you can double-click CSV files to quickly open them in Excel for editing.

Create a CSV file in Excel:

- Open Excel and enter the following dataset:

X	Y	Z
1	5	8
4	7	10
3	9	12
5	4	7
7	8	11

- Save it as an Excel spreadsheet in your folder "C:\Lab1\data.xls"
- To save as a CSV file, chose "Save as"
- From the drop-down box "Save as type" choose "CSV (Comma delimited) (.csv)"
- Put it in the same location "C:\Lab1\data.csv"
- Dismiss all warning dialogues and close Excel

There are several ways to import data into RStudio, in almost all you have to deal with file paths. RStudio can only import files that are in the current working directory or if you specify a file path. To know where the current working directory is, type:

```
getwd()
```

It is not a big issue to specify file paths if you work with only one Computer. But if you want to use your script on multiple computers and if you want to send me scripts for checking, this quickly becomes chaotic.

A neat way to avoid this is to set the R workspace to your Lab1 folder by going to "Session" > "Set Working Directory" > "To Source File Location". The source file is your script, and therefore we can check if it worked:

```
getwd()
```

Now, create a new script and save it as "correlation" in your folder. In the script we can type:

```
dat=read.csv("data.csv")
head(dat)
str(dat)

cor(dat$X,dat$Y)
lm(dat$Y~dat$X)
plot(dat$Y~ dat$X)
abline(lm(dat$Y~ dat$X))
plot(dat)
```

- Execute your code line by line with Ctrl-R:

- head()and str() are alternate ways to check your import.

- head() allows you to check the first few rows, while str() gives you more information about the imported variable types, hinting at potential errors.

- cor() returns the Pearson correlation coefficient for two variables.

- lm() returns the regression equation. The symbol ~ always means "as a function of".

- plot(X~Y) give us a scatter plot of Y as a function of X.

- abline() adds a function to the scatter plot.

- plot(dat) creates a scatter plot for all variables in the data table "dat".

6. Working with you own data

We will normally use CSV file formats in this course, and I recommend that you do the same for your own datasets. Below are some useful rules that help with file management for this course, and for your own projects.

- Always keep your original spreadsheet (typically an Excel file) and include documentation in this file (where did the data come from? when was it collected? what are the rows, columns, variables?)

- Create a simplified CSV for analysis, which just has a single header row with simple variable names followed by data rows (no documentation or comments, no blank rows or columns). Save this CSV file with the same name in the same folder.

- In the CSV file, variable names can only contain letters, numbers, and underscores (A-Z, a-z, 0-9, _). Don't use spaces or symbols. Variable names should be no more than 8 characters long and must start with a letter. For clarity, I find it useful to choose variable names that have ALL UPPER CASE letters and use lower case letters for other R code.

- R and SAS may have trouble reading CSV files that were generated by other programs. In this case open and re-save the CSV files in Excel

- Always put all the files you need for a particular analysis into one folder with a good descriptive name. Don't accumulate too many files in a folder, but rather make new folders for different purposes (a folder may for example contain the files: data.xls, data.csv, anova.r, summary.r, graphics.r).