Lab 1 - Stat 411/511

Introduction

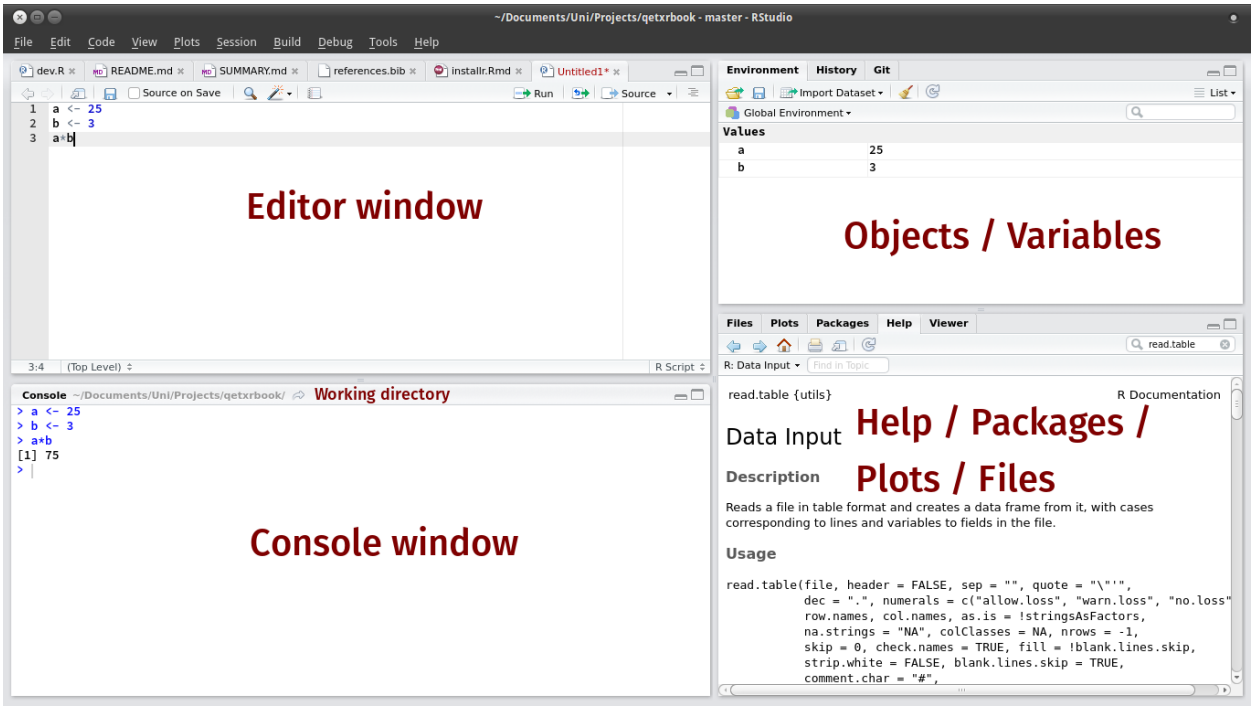
The term R is used to refer to both the programming language as well as the software that interprets the scripts written using it. The learning curve may be steeper than with other statistical software, but with R the results of your analysis or your plot does not rely on remembering what order you clicked on things, but instead on the written commands you generated. In R you will work in scripts or with dynamic documents with scripts within them (Rmd or Rnw files). Scripts may feel strange at first, but they make the steps you used in your analysis clear for both you and for someone who wants to give you feedback.

RStudio is a free computer application that allows you access to the resources of R, while providing you with a comfortable environment working environment. There are many ways you can interact with R, but for many reasons RStudio has become the most popular. To function correctly, RStudio uses R behind the scenes, and therefore both need to be installed on your computer.

For our use of RStudio the desktop version will suffice. There are many videos and guides for installation. See the Stat 217 textbook (pages 12 - 14) here: https://scholarworks.montana.edu/xmlui/bitstream/handle/1/2999/Greenwood_Book.pdf?sequence=3&isAllowed=y_

RStudio has a panel of 4 windows, where each can be viewed at the same time and have multiple tabs available.

- the **Editor** for your scripts and documents (top-left)
- the R **Console** (bottom-left)
- your **Environment (Objects/Variables)/History** (top-right)
- and your **Files/Plots/Packages/Help/Viewer** (bottom-right).



Work Flow in R

It is good code writing practice to keep a set of all related data, analysis, plots, documents, etc. in the same folder. When all the pieces are in the same folder, it allows for a clean workflow and working directory. When you are executing code for a document/script R will search for things (such as data) in the same folder as the document/script, which is called a *relative path*. If you are having troubles loading your data into RStudio and you have saved your files in this way, it is possible that R is searching in the wrong location and you need to change your working directory.

The easiest way to do this is,

- click on the **Session** drop-down from the top of the screen,
- select the **Set Working Directory** tab,
- select **To Source File Location**.

After this process R will be searching for objects (such as data) in the same folder that the document/script you're working on is saved in.

Working in R

RStudio allows for you to execute commands directly from the document/script editor by using the **Ctrl + Enter** (on Macs, **Cmd + Return**) shortcut. If you place your cursor on the line in the document/script that you would like to run and hit this shortcut, R will execute that line(s) of code for you.

If R is ready to accept commands, the R console (in the bottom-left) will show a > prompt. When R receives a command (by typing, copy-pasting, or using the shortcut) it will execute it, and when finished will show the results and display the > symbol once again.

Calculator

Practice: Enter each of the following commands and confirm that the response is the correct answer.

```
1 + 2

16*9

sqrt(2)

20/5

18.5 - 7.21

3 %% 2 ## what is this doing?
```

Importing Data

Entering Data into R by Hand

For small data sets, one option is to enter the data by hand. To enter data by hand, we will use the `c()` command. This is a default function in `R` which combines the elements you provide it (arguments) to form a vector, where elements are separated with a comma.

Example: A random sample of 16 wolf dens was obtained and the number of wolf pups per den was recorded. We can enter the number of wolf pups for each den:

```
pups <- c(5, 8, 7, 5, 3, 4, 3, 9, 5, 8, 5, 6, 5, 6, 4, 7)
```

Remarks:

- In the above code `<-` is the assignment operator. It assigns values on the right to objects on the left. So, after executing `year <- 3`, the value of `year` is `3`. The arrow can be read as 3 goes into year. For historical reasons, you can also use `=` for assignments, but not in every context. Because of the slight differences in syntax, it is good practice to always use `<-` for assignments.
  - In RStudio, typing Alt at the same time as the - key will write `<-` in a single keystroke. Neat!

1. If you type in `Pups` into the console, what error does `R` output?

Importing Data from a Text File into R

Lactococcus lactis and *Leuconostoc citrovorum* are two common bacteria used for making cottage cheese. While developing a new type of cottage cheese, a large dairy producer has added the fungus *Penicillium candidiuma* (PC), typically used to make Brie, to their cottage cheese recipe. One part of the process used to make cottage cheese involves cooking curdled milk for about an hour. A researcher is interested in determining whether adding PC increases the cooking time. Seven dairy facilities (referred to as A, B, . . . , G) make two batches of cottage cheese, one with and one without the fungus PC. The cooking time in minutes was recorded for each batch. The results of the experiment are in a text file called `dairy.txt` that you can download from the course website.

Text data files that are tab or space delimited can be read into `R` ! Because this data storage file uses spaces to separate the variables and the data values, the variable names cannot have spaces in them (e.g. don't use `Cook Time`). To get `dairy.txt` into `R`,

- download it from the course website,
- save it into the *same* file as this **Lab 1** document, and
- execute the following command:

```
dairy <- read.table("dairy.txt", header = TRUE)
```

The `read.table` is a *function* that takes many different inputs (*arguments*). The first argument is the name of the data file, as it is stored in your file directory, in quotations with a `.txt` extension. The `header = TRUE` argument tells `R` that the first line of the text file contains the variable names of each of the columns of data.

Recall: When you are executing code for a document/script `R` will search for things (such as data) in the same folder as the document/script. If you received the following error when executing the line above,

```
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") :
  cannot open file 'dairy.txt': No such file or directory,
```

then there could be a few potential reasons.

- The `dairy.txt` data file could be saved in a *different* file than your **Lab 1** document is saved, or
- `R` is searching in the wrong location and you need to change your working directory.

To troubleshoot this issue, you should first verify that the text file is stored in the *same* file as your **Lab 1**. Next, you should change the working directory `R` is looking in. The easiest way to do this is,

- click on the **Session** drop-down from the top of the screen,
- select the **Set Working Directory** tab,
- select **To Source File Location**.

After this process `R` will be searching for objects (such as data) in the same folder that the document/script you're working on is saved in. Now, run the previous line of code to verify that the `dairy.txt` file has been imported.

Importing Data from a Course Website into `R`

For data that you need to download from the course website, you do not need to worrying about where to save the file. Instead, the data are imported by directly telling `R` the http(s) address to use and the name of the file stored there.

```
dairy2 <- read.table("http://www.math.montana.edu/parker/courses/STAT411Fall2017/dairy.txt", header = T
```

Remark: Notice the above file was stored into a object named `dairy2` **not** `dairy`. This is because we had previously saved *the same* data set into an object named `dairy`. If we wished to overwrite the contents stored in the `dairy` object, we could have run the above code and saved the data as `dairy`.

Importing Data from a Comma Separated Value (CSV) File into `R`

Outside of classroom examples, data are rarely read into `R` using a text file. Most people you will meet or collaborate with will store their data with variables names that include spaces or other characters that would cause `read.table` to throw an error. It is way more convenient to keep your data in a *csv* (comma separated value) file.

Two major advantages of keeping your data in csv files are that (1) you can use a spreadsheet processor (like Microsoft Excel) to manage the data and (2) `read.csv` (or `read_csv`) are sophisticated enough to deal with the variable names in the first row that have spaces and/or other characters in them.

Download the `dairy.csv` file from the course website and save it into the **same** file as this **Lab 1** document.

Now, import the data file into `R` by running the following code:

```
dairy_csv <- read.csv("dairy.csv")
```

Remark: We could have instead downloaded the data, in the csv format, directly from the course website with the following line of code:

```
dairy_csv <- read.csv("http://www.math.montana.edu/parker/courses/STAT411Fall2017/dairy.csv")
```

Structure of Data

The data set we will use is organized into data tables. When you imported the dairy data into RStudio was saved as an object. You are able to inspect the structure of the `dairy` object using functions built in to `R` (no packages necessary).

2. Run the following code. What is output from each of the following commands?

```
class(dairy_csv) ## output?
```

```
## [1] "data.frame"
```

```
names(dairy_csv) ## output?
```

```
## [1] "Dairy"     "Treatment" "Time"
```

```
dim(dairy_csv) ## output?
```

```
## [1] 14  3
```

```
str(dairy_csv) ## output?
```

```
## 'data.frame':    14 obs. of  3 variables:
##  $ Dairy    : Factor w/ 7 levels "A","B","C","D",..: 1 1 2 2 3 3 4 4 5 5 ...
##  $ Treatment: Factor w/ 2 levels "withoutPC","withPC": 2 1 2 1 2 1 2 1 2 1 ...
##  $ Time     : int  68 61 75 69 62 64 86 76 52 52 ...
```

```
summary(dairy_csv) ## output?
```

```
##  Dairy     Treatment      Time
##  A:2   withoutPC:7   Min.   :38.00
##  B:2   withPC   :7   1st Qu.:54.25
##  C:2                Median :66.00
##  D:2                Mean   :63.50
##  E:2                3rd Qu.:71.25
##  F:2                Max.   :86.00
##  G:2
```

3. What class is each variable in the `dairy` dataset?

Extracting Data

If we were interested in accessing a specific variable in our data set, we use the `$` command. This command extracts the specified variable (on the right of the `$` sign) from the data set (on the left of the `$` sign). When this is extracted, `R` views the variable as a vector of entries, which is what the [1:14] refers to.

```
times <- dairy_csv$Time
## extracts the Time column from the dataset and saves it into a new variable
## named times

str(times)  ## using the new variable (remember case matters!)
```

```
##  int [1:14] 68 61 75 69 62 64 86 76 52 52 ...
```

Remarks:

- There are a few simple rules that apply when creating the same of a new object (like we did above):
  - the name cannot start with a digit (`1times` is not allowed, but `times1` is)
  - the name cannot contain any punctuation symbols, except for `.` and `_` (`.` is not recommended)
  - you should not name your object the same as any common functions you may use (`mean`, `sd`, etc.).

Another method for accessing data in the data set is using the bracket notation ([row, column]). If you look to your right in the **Environment** window, you notice that RStudio tells you the dimensions of the `dairy` data (observations are rows and variables are columns).

You can (roughly) view the data set as a matrix of entries, with variable names for each of the columns. I could instead use bracket notation to perform the same task as above, using the following code.

```
times <- dairy_csv[ , 3] ## This takes ALL rows of data but only the third column

str(times) ## times inherits the same structure as before when we use this method
```

```
##  int [1:14] 68 61 75 69 62 64 86 76 52 52 ...
```

Packages

As we mentioned previously, `R` has many packages, which people around the world work on to provide and maintain new software and new capabilities for `R`. You will slowly accumulate a number of packages that you use often for a variety of purposes. In order to use the elements (data, functions) of the packages, you have to first install the package (only once) and then load the package (every time).

We're going to install a few packages that are often used.

- Use the **Install** button in the **Packages** tab
- Type in `car, Sleuth3,` and `mosaic` into the blank line (separated by commas)
- Check the **Install dependencies** box
- Click on the **Import** button

There will be a large amount of output coming out of the console. This output is `R` trying to download the package(s) you requested by contacting the mirror you chose for it to use when downloading (I chose Northern Michigan University). Once the computer has downloaded the packages, it will tell you that "`The downloaded binary packages are in`", followed by the location of the files.

Now that the files are downloaded, we need to load them in order to use them. The following code will load each package, please run it!

```
library(car)
library(Sleuth3)
library(mosaic)
```

```
## Loading required package: dplyr
##
## Attaching package: 'dplyr'
## The following object is masked from 'package:car':
##
##     recode
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
## Loading required package: lattice
## Loading required package: ggformula
## Loading required package: ggplot2
##
## New to ggformula?  Try the tutorials:
##  learnr::run_tutorial("introduction", package = "ggformula")
##  learnr::run_tutorial("refining", package = "ggformula")
## Loading required package: mosaicData
## Loading required package: Matrix
```

```
##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by this.
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.

##
## Attaching package: 'mosaic'

## The following object is masked from 'package:Matrix':
##
##     mean

## The following objects are masked from 'package:dplyr':
##
##     count, do, tally

## The following objects are masked from 'package:car':
##
##     deltaMethod, logit

## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median,
##     prop.test, quantile, sd, t.test, var

## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

Notice that when loading the `mosaic` package that there is a large amount of output. This output is telling you all of the other packages that are loaded with the `mosaic` package, because `mosaic` is dependent on them (`dplyr`, `lattice`, `ggformula`, `ggplot2`, etc.).

Importing a Dataset from The Statistical Sleuth

For most of this course we will be using data sets from the package developed for The Statistical Sleuth (`Sleuth3`). Fortunately for us, these data sets can be directly accessed in `R`.

First, let's load in the data set from the example *Sex Discrimination in Employment – An Observational Study* from Chapter 1. Since we have already downloaded the `Sleuth3` package and sourced it in (`library(Sleuth3)`), we just need to extract the `case0102` data set from the package.

There are a few different ways to load these data into `R`, (1) source in the file directly *or* (2) source in the file and assign it to a new variable. Both methods are shown below.

```
data(case0102)  ## sources in the data set directly, but is named case0102

sex_descrimination <- case0102  ## sources in the data set and gives it a more intutitive name
```

You could look at the data frame by typing `case0102` or `sex_descrimination`. Because the data set has 93 observations, we might just want to look at the first 6 rows, using the `head` function.

```
head(case0102)
```

```
##   Salary    Sex
## 1   3900 Female
## 2   4020 Female
## 3   4290 Female
## 4   4380 Female
## 5   4380 Female
## 6   4380 Female
```

```
head(sex_descrimination)
```

```
##   Salary    Sex
## 1   3900 Female
## 2   4020 Female
## 3   4290 Female
## 4   4380 Female
## 5   4380 Female
## 6   4380 Female
```

4. Explain how these data are organized. (What does each column represent? Each row?)

5. Write the `R` code to show the names of the variables in the data set.

```
## code here!
```

6. Write the `R` code to show the dimensions of the data set.

```
## code here!
```

Functions

In R there are both functions that are built in (require no package to be loaded), as well as functions that are housed within specific packages. You have already used a few built in functions to inspect the structure of the dairy data (str, class, summary, dim). As we know, a function transforms an input into an output. You have to provide R with the inputs (arguments) required for the function to generate an output. The argument(s) inside a function happen after the (symbol. You know an object is a function when it is immediately followed by a (and the corresponding closing) comes after the arguments are complete (str()).

Arguments describe the details of what a function is asked to do. Many functions take named arguments where the name of the argument is followed by an = sign and then the value of the argument. Here are some examples.

```
sqrt(2)  ## 2 is the argument
```

```
## [1] 1.414214
```

```
mean(dairy_csv$Time)  ## takes a numerical input
```

```
## [1] 63.5
```

```
mean(dairy_csv$Treatment)  ## gives an error because the input is not the correct data type
```

```
## Warning in mean.default(x, ..., na.rm = na.rm): argument is not numeric or
## logical: returning NA
```

```
## [1] NA
```

```
mean(Time ~ Treatment, data = dairy_csv)
```

```
## withoutPC    withPC
##  61.14286  65.85714
```

```
## from the mosaic package: format is (response ~ explanatory, data =
## dataset)
```

Using the sex descrimination data:

7. Write the R code to output the average salaries for all individuals (ignoring sex).

```
## code here!
```

8. Write the R code to output the average salaries separately for Males and Females.

```
## code here!
```