STAT 513/413: Lecture 2
And now to computing

(starting with R)

Computing environment ($\approx$ language)

Long ago: machine code $\rightarrow$ assembly language

Programming languages: Fortran, Pascal, C(++), Java

First time a bit comfortable: Matlab (Octave?)

First dedicated for data-analysis: Lisp(-Stat)

Very fashionable now: Python

Dedicated for data-analysis: S $\rightarrow$ S-Plus $\rightarrow$ R

Our choice: R

Some reasons: R still widest scope; for Stat grad students best choice; not "industrial strength" pursued but rather brevity of code; well-matured and thus trustworthy; ...

See also (for instance)

Grolemund and Wickham `https://r4ds.had.co.nz/introduction.html`

R

R started out back then as an open source implementation of the language S (John M. Chambers and others; for a while, there was a commercial implementation called SPlus)

Now they say it is a system for statistical computation and graphics *based* on S.

(For more information, see Rizzo 1.2)

Available - for free - on any platform

Also, a HUGE number of texts, many of them freely available too

A growing list of R books
(without any preferences whatsoever)

Online

H. Wickham: Advanced R

G. Grolemund: Hands-on Programming with R

G. Grolemund and H. Wickham: R for Data Science

C. Gillespie, R. Lovelace: Efficient R Programming

N. D. Phillips: YaRrr! The Pirate's Guide to R

Printable, lecture notes, etc.

W. N. Venables, R. M. Smith and the R Core Team:
An Introduction to R

E. Paradis: R for beginners

T. Martin: The Undergraduate Guide to R

N. Matloff: The Art of R Programming

N. J. Horton, R. Pruim, D. T. Kaplan: A Student's Guide to R

Once again (repeating myself): necessities

- Access to computer(s) (where you can install all the below)

  e.g. own laptop

- Internet access (preferably with Google within reach)
- A working installation of R

  possibly with a GUI (like that for OS X on Mac)

  or within an IDE (like ESS within Emacs or RStudio)

- A *programming* editor *of your choice*:

  (once an IDE, you have it automatically there)

  otherwise Vim? Atom? Notepad? TextEdit? Gedit?

  an important feature: with a formatting extension for code
  (Emacs has ESS, and RStudio is dedicated for R)

Finally, you will also need

- A *document* editor *of your choice*:

  e.g. TeX (LaTeX)

  or Microsoft Word? Google Drive?

First session

```
retina:513 mizera$ R

R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"

...  - THIS WILL INDICATE OMITTED PART OF THE OUTPUT, I.M.

> q
function (save = "default", status = 0, runLast = TRUE)
.Internal(quit(save, status, runLast))
<bytecode: 0x7fd3cc056098>
<environment: namespace:base>
> ls()
character(0)
> m=1
> ls()
[1] "m"
> q()
Save workspace image? [y/n/c]: y
retina:513 mizera$
```

Workspace stays

```
retina:513 mizera$ open -a R .
```

In a separate window:

```
R version 4.0.3 (2020-10-10) -- "Bunny-Wunnies Freak Out"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin17.0 (64-bit)

...

[R.app GUI 1.73 (7892) x86_64-apple-darwin17.0]

[History restored from /Users/mizera/.Rhistory]

[Workspace restored from /Users/mizera/work/Lessons/513/.RData]

> ls()
[1] "m"
> m
[1] 1
```

First peculiar aspects

- Functions always must have parentheses
- Workspace stays - but not automatically, only if you wish
- Fancy assignments
- Parentheses `()` are for functions, brackets `[]` for indexing
- And braces `{}` to form groups of commands

Help!

```
> help(ls)             # usually works without quotes
> help("ls")           # those are needed only in special cases
> help('ls')           # uppercase and lowercase quotes both good
> ?ls                  # and this is the short way for all of the above
> help.search("knn") # quotes obligatory now
> help.search('knn'). # (whatever comes after # is a comment)
> ??knn                # same as help.search()

> apropos("ls")        # quotes again needed
> example(ls)          # quotes not needed
> demo()
```

Fancy assignment commands

```
> m=2      # this is what most of us do
> m = 2    # OK, at least this is
> m
[1] 2
> m <- 3   # this is what stylish people encourage
> m
[1] 3
> 3 -> m   # this is also possible, but rather discouraged
> m
[1] 3
```

Operating R: a toy task

For starters, let us consider a toy problem: preparing a fake example for a lecture on linear regression.

We would like to plot 20 points. Their first coordinates, let us denote them by letter $\mathtt{m}$ (you know, $\mathtt{x}$ is soooo overused) are to be equidistantly spaced (say, $\mathtt{m}$ is to contain $1, 2, 3, \ldots, 20$)

while their second coordinates (how about $\mathtt{n}$, just for alphabetic sequence? $\mathtt{y}$ is overused, too) are to lie on a line, say, $\mathtt{n} = 2 + 3\mathtt{m}$, *but not exactly*: they are to be a bit wiggled up and down, in a "random fashion". That is, a formula from regression courses

$$\mathtt{n}_k = 2 + 3\mathtt{m}_k + e_k$$

where e_k is the "wiggle" for the k-th point; doing it by hand for each single point is tedious, we better employ (a bit predating the curriculum...) - a function `rnorm()` that generates pseudorandom numbers with normal distribution. To learn more about it, run

```
> ?rnorm
```

The simplest alternative: `rnorm(x)` returns $\mathtt{x}$ numbers with standard normal distribution

Something like this

Modi operandi in R: command line

- **command line**: you type commands which are then immediately executed; if there is an error, you simply retype the last command, or very few preceding ones

```
> for (k in 1:20) m[k]=k
Error: object 'm' not found
> m=1
> for (k in 1:20) m[k]=k
```


Is that it? We can check immediately


```
> m
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```


And now

```
> n=0
> for (k in 1:20)
+ n[k]=2+3*m[k]+rnorm(1)
```

What's that? In the interactive, command line session, continuation sign + appears always if the command is syntactically not finished

Is it really it?

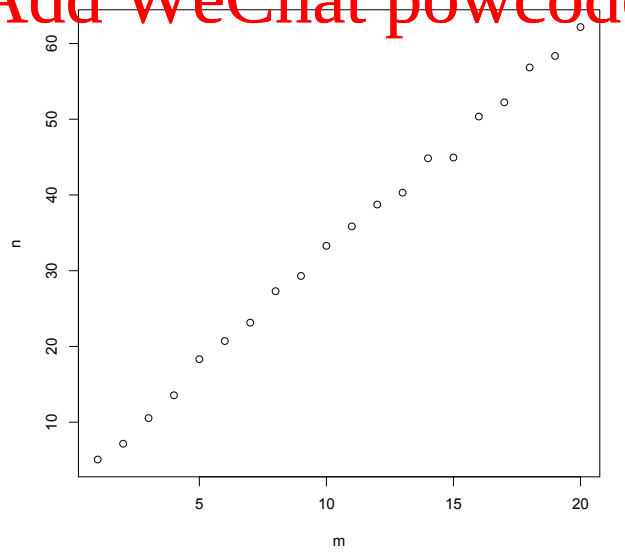
Well, we can check

```
> n
 [1]  3.895190  8.108667 10.792542 13.202841 16.097341 20.454508
 [7] 23.205155 26.961782 28.712749 31.998655 34.046058 38.807411
[13] 40.414637 45.686550 46.802203 49.454804 52.687893 56.788084
[19] 58.775084 61.673681
```

but it may not tell that much; perhaps better is to make a picture


```
> plot(m,n)
```


...which gives something like this:



Modi operandi in R continued: script

- **script**: already a "program", consisting of several commands in a file - one may edit by a programming editor, and then execute its contents as a whole; if there is an error, you have to correct it via editing the file, and then execute again

Executing can be done by a command `source()` in R

or, IDE, Integrated Development Environment, may streamline this

A convenient (at least for some) way of creating scripts is to take a successful command line session, collect commands, and edit them. The result of this, the file `my.R`, may look as follows:


```
m=1
> n=0
for (k in 1:20) m[k]=k
for (k in 1:20)
+ n[k]=2+3*m[k]+rnorm(1)
plot(m,n)
```

One then does the following in R...

```
> source("my.R")
```

... and see what happens

```
> source("my.R")
Error in file(filename, "r", encoding = encoding) :
  cannot open the connection
In addition: Warning message:
In file(filename, "r", encoding = encoding) :
  cannot open file 'my.R': No such file or directory
```

Oops! has to be corrected... the file `my.R` has to be placed right


```
> source("my.R")
Error in source("my.R") : my.R:2:1: unexpected '>'
1: m=1
2: >
   ^
```


Errors, errors... but after a while, one gets to something like this

```
m=1
n=0
for (k in 1:20)
m[k]=k
n[k]=2+3*m[k]+rnorm(1)
plot(m,n)
```

But

It will run through, but it will not work (try yourself to see why)

After another while, you finally come to

```
m=1
n=0
for (k in 1:20) {
m[k]=k
n[k]=2+3*m[k]+rnorm(1) }
plot(m,n)
```


which means you are ready to submit, aren't you?

More about what to submit later. Now an IMPORTANT NOTE:

In this course, scripts will be fine

(That means: despite the existence of *functions* as another modus operandi, and despite their apparent advantages, we will be happy with scripts... if they work as desired, of course. But functions will be accepted as well without any penalty.)

So, are we done...?