A crash course in R + pedsuite

For self study!

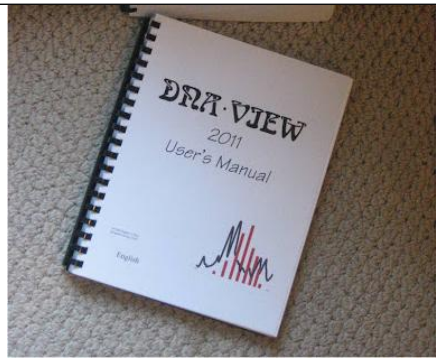
What is R? (And why should you care?)

- A framework for statistical computing
  - calculator
  - data handling and numerical analysis
  - flexible plotting
  - programming language
  - external packages
    - anyone can make one
    - thousands!

Familias.no

BONAPARTE

Designed, built and proven for real world case work



Pros
- free!
- very widely used
- anything is possible (but not always easy)
- scripting --> reproducibility

Cons
- learning curve
- packages come and go

Oh boy, that sounds great!

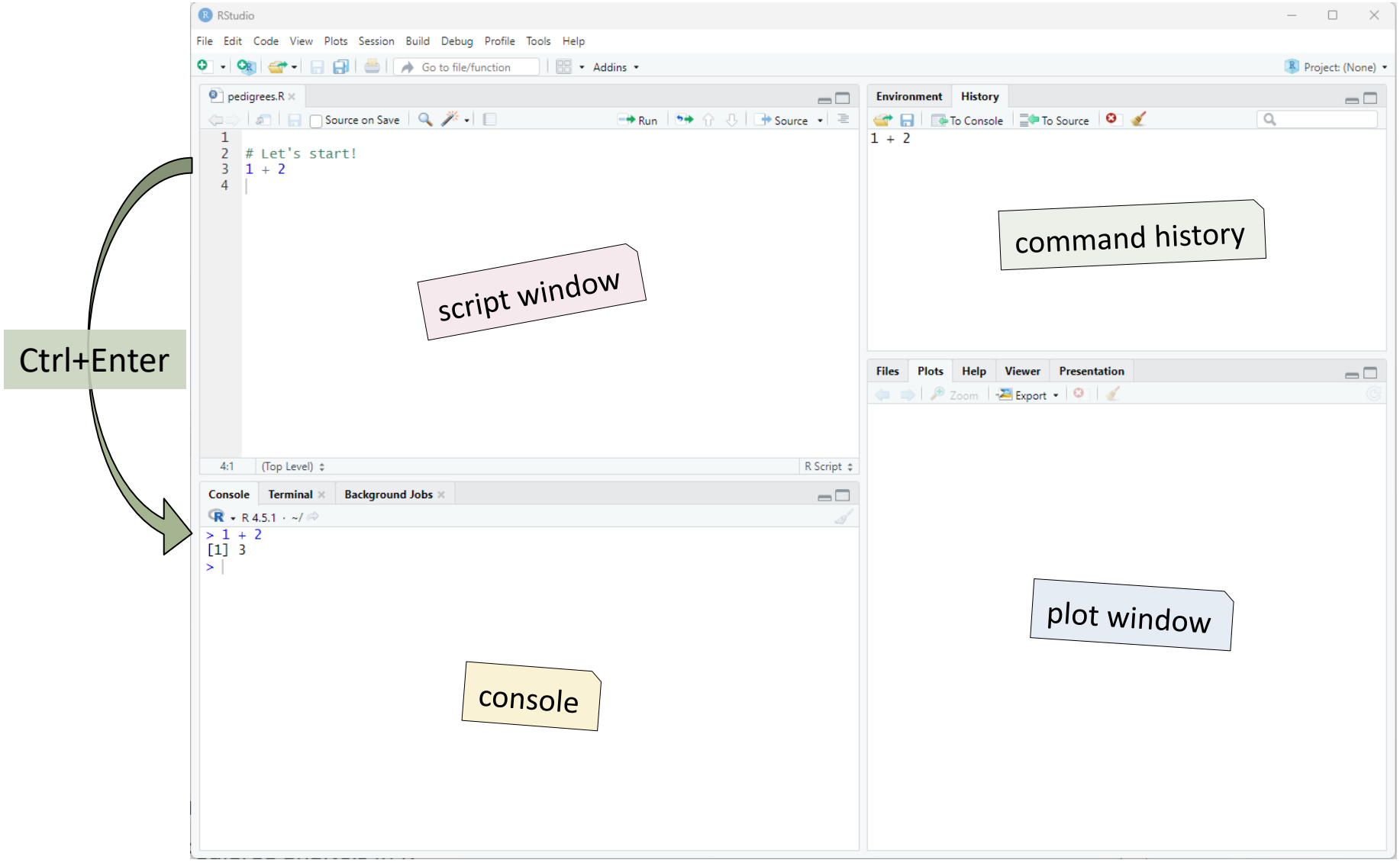
I which I knew R ...

Don't worry!

It's not that hard.

Here is a quick intro to R that contains most of what you need

To get started, open RStudio!


Ctrl+Enter
RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Go to file/function
Addins
Project: (None)
pedigrees.R
Source on Save
Run
Source
1
2 # Let's start!
3 1 + 2
4
4:1
(Top Level)
R Script
script window
Environment
History
To Console
To Source
1 + 2
command history
Files
Plots
Help
Viewer
Presentation
Zoom
Export
plot window
Console
Terminal
Background Jobs
R 4.5.1 · ~/
> 1 + 2
[1] 3
>
console

Basic calculations

```
>   2 + 3
[1] 5
>   2+      3
[1] 5
>   (1 + 2) * 3
[1] 9
>   4^2
[1] 16
>   log(100)
[1] 4.60517
>   log(100, base = 10)
[1] 2
>   log10(100)
[1] 2
```

Spaces don't matter

$e^{4.60517} \approx 100$

Variables

I use this

Two (mostly synonymous) ways to assign values: `=` or `<-`

```
>   a = 5      or   a <- 5
>   b = 2      or   b <- 2
>   a
[1] 5
>   a - 2*b
[1] 1
```

Changing a variable:

```
>   a = a+1
>   a
[1] 6
```

Common beginners' mistake:
forgetting to assign after change

Creating new variables from old:

```
>   newVar = a^b
>   newVar
[1] 36
```

Most programmers stick to either
camelCase or **snake_case**
when naming their variables

Vectors

```
>   c(3, 2, 6, -1)
[1]  3  2  6 -1
>   4:20
[1]  4  5  6  7  8  9 10 11 12
[10] 13 14 15 16 17 18 19 20
>   5:7 - 4
[1] 1  2  3
>   c(10,20,30,40) + c(1,3,8,0)
[1] 11 23 38 40
>   seq(from = 2, to = 15, by = 3)
[1]  2  5  8 11 14
```

The `c()` operator!

The `':'` operator
(shortcut for consecutive numbers)

There is a help page
for every function!
`> ?seq`

Character vectors:

```
>   c("Alice", "Bob")
```

Logical vectors:

```
>   c(TRUE, FALSE, T, F)
[1] TRUE FALSE TRUE FALSE
```

Built-in logcial constants:
TRUE short form: **T**
FALSE short form: **F**

Matrix-like containers

Data frames: Collects vectors of the same length

```
>   x = data.frame(Name   = c("Ali", "Bob", "Joe"),
                   Weight = c(75, 81, 70))
>   x
  Name Weight
1  Ali     75
2  Bob     81
3  Joe     70
```

Use **$** to refer to columns: **x$Name**

Matrices:

```
>   x = matrix(1:12, nrow = 3, ncol = 4)
>   x
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Note: No **$** for matrices!

First column: **x[, 1]**
First row: **x[1,]**

Faster, but less flexible. Good for all-numeric (or all-character) data

Lists

```
>   a = list(good = 1:3, bad = 0)
>   a
$good
[1] 1  2  3

$bad
[1] 0
>   a$good
[1] 1  2  3
```

Alternative to **$**:
a[["good"]]

Easy to change lists:

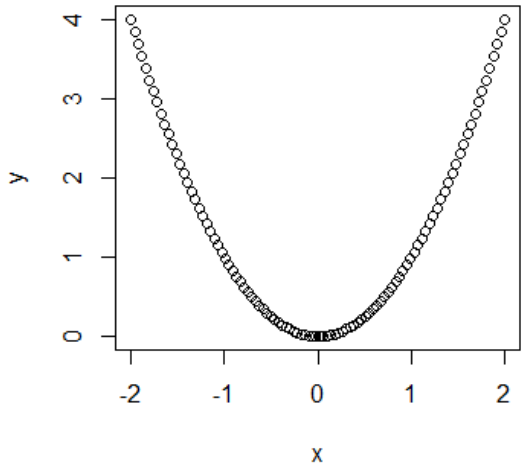
```
>   a$bad = NULL              (delete item)
>   a$ok = -1                 (add new item)
>   a$good = c(a$good, 10)    (modify item)
>   a
$good
[1] 1  2  3  10

$ok
[1] -1
```

Basic plots

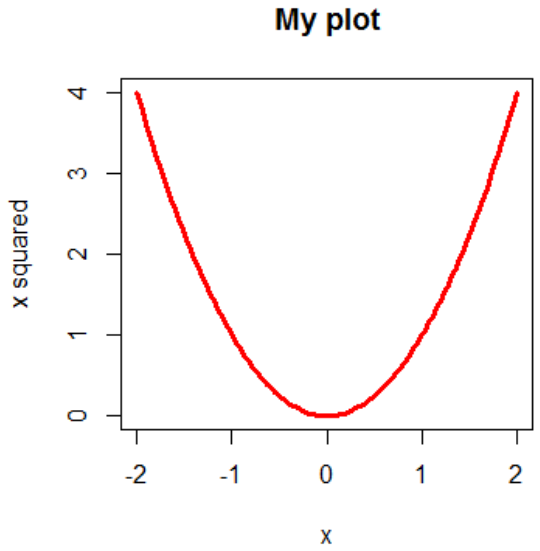
Let's plot the graph of $y = x^2$!

```
>   x = seq(-2, 2, length = 100)
>   y = x^2
>   plot(x, y)
```



Many options to play with...

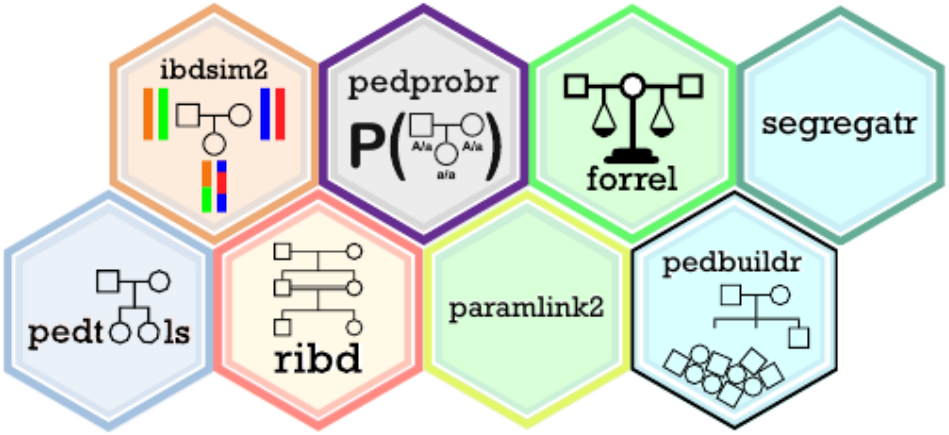
```
>   plot(x, y, type="l", lwd = 3, col = "red",
         ylab = "x squared", main = "My plot!")
```



R stuff skipped in this brief introduction

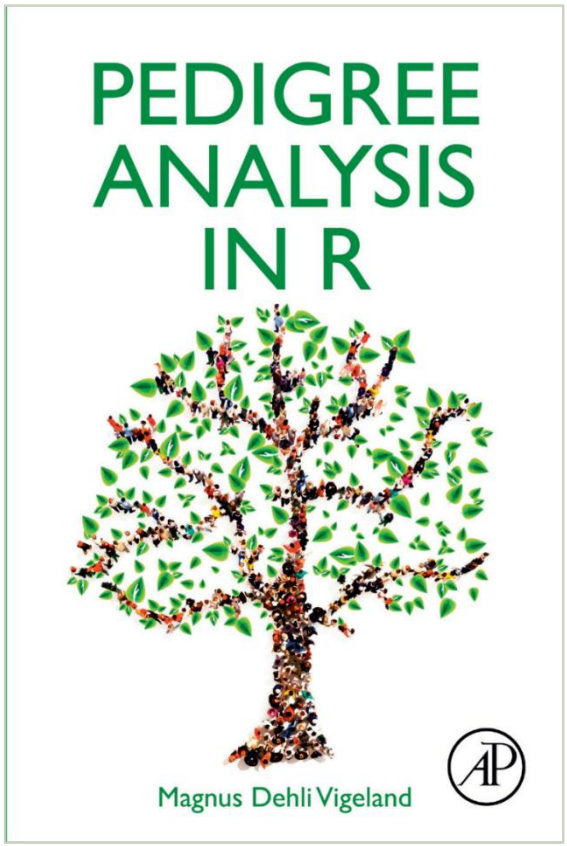
- User-defined functions
- Loops, `apply()`, `lapply()`, etc.
- Basic statistics, linear models + +
- Random numbers
- The "tidyverse" for data science
- ... and LOTS of other things...

pedsuite:
A collection of packages for pedigree analysis in R



Home page:
https://magnusdv.github.io/pedsuite

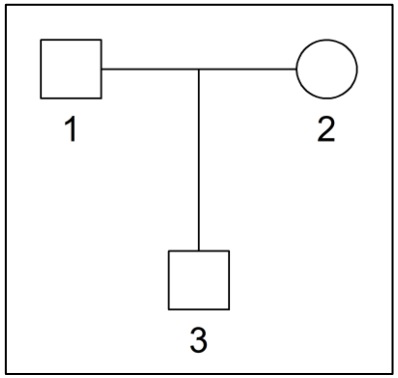
Source code + documentation:
https://github.com/magnusdv



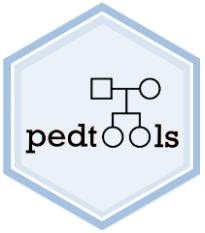
Your first pedigree

```
> library(pedsuite)

> x = nuclearPed()
> plot(x)
```



Some useful functions



Create: basic

- singleton
- nuclearPed
- linearPed
- halfSibPed
- cousinPed
- halfCousinPed

Create: complex

- ancestralPed
- doubleCousins
- quadHalfFirstCousins
- fullSibMating
- randomPed

Manipulate

- addSon
- addDaugher
- addParents
- addChildren

- swapSex
- relabel
- removeIndividuals

- branch
- subset

- mergePed
- breakLoops

Member subsets

- founders
- nonfounders
- leaves
- males
- females
- typedMembers
- untypedMembers

Relatives

- father
- mother
- children
- siblings
- grandparents
- spouses

- ancestors
- descendants
- unrelated

Another example

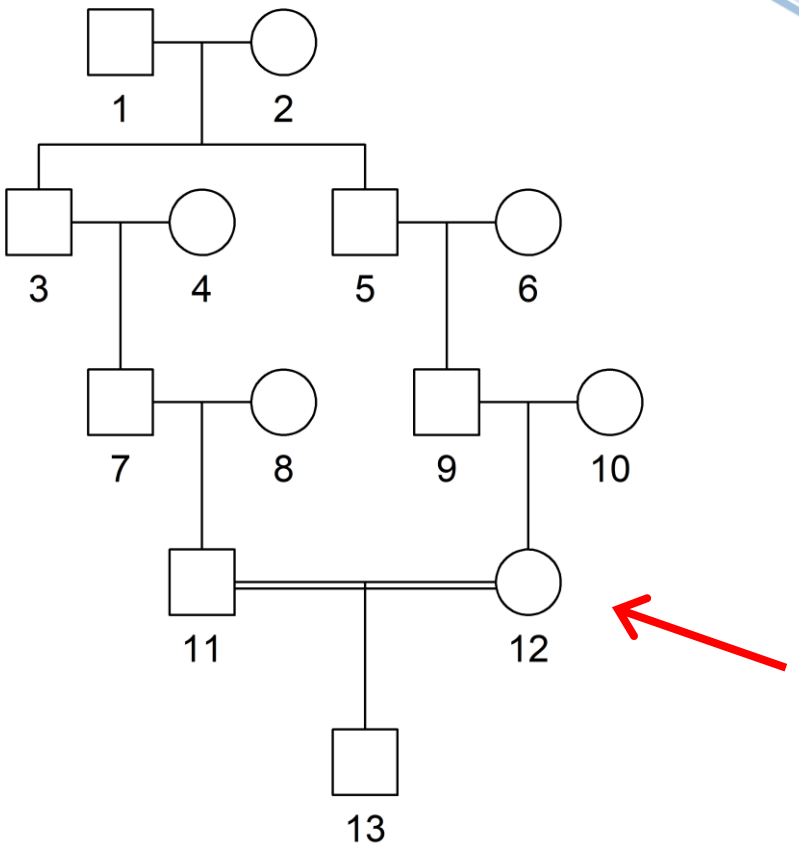
```
> x = cousinPed(2)
> plot(x)
```

Change gender:

```
> x = swapSex(x, 12)
> plot(x)
```

Add inbred child

```
> x = addSon(x, parents = 11:12)
> plot(x)
```



Remember

- Store the result after each change!
- It is OK to use the same name
  (if you don't need the previous object)

Shortcut command for this pedigree

```
> x = cousinPed(2, child = TRUE)
```

The pipe |>

Introduced in R 4.1

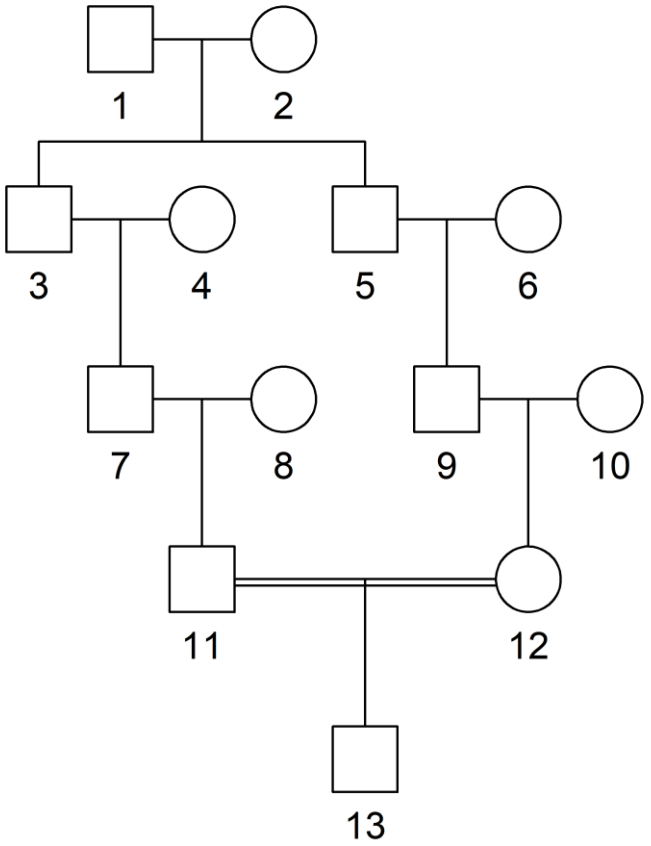
Instead of ...

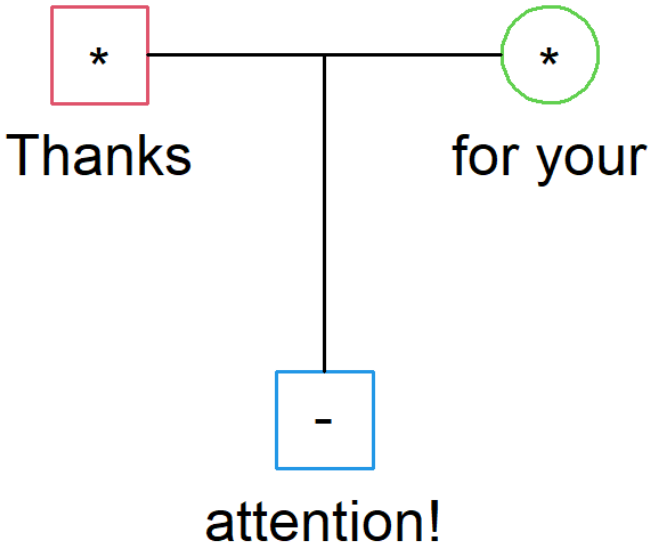
```
>   x = cousinPed(2)
>   x = swapSex(x, 12)
>   x = addSon(x, parents = 11:12)
```

... we can write

```
>   x = cousinPed(2) |>
      swapSex(12) |>
      addSon(parents = 11:12)
```

Feeds the previous result into the next function!






```
plot(nuclearPed(), col = 2:4, textInside = c("*", "*", "-"),
     labs = c("Thanks" = 1, "for your" = 2, "attention!" = 3))
```