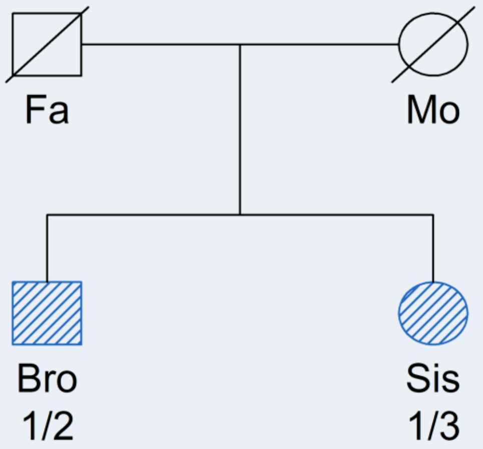


Lecture 2: Introduction to R and the ped suite

Statistical methods in genetic relatedness and pedigree analysis

NORBIS course, 13th – 17th of June 2022, Oslo

Magnus Dehli Vigeland

What is R?

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

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

Cons
- steep learning curve
- packages come and go

Time to get your hands dirty: Trying out R

Using R as a basic calculator

```
>   2 + 3
[1] 5
>   1+2      * 3
[1] 7
>   (1 + 2) * 3
[1] 9
>   4^2
[1] 16
>   exp(1)
[1] 2.718282
>   log(100)
[1] 4.60517
>   log(100, base = 10)
[1] 2
>   log10(100)
[1] 2
```

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"]]`

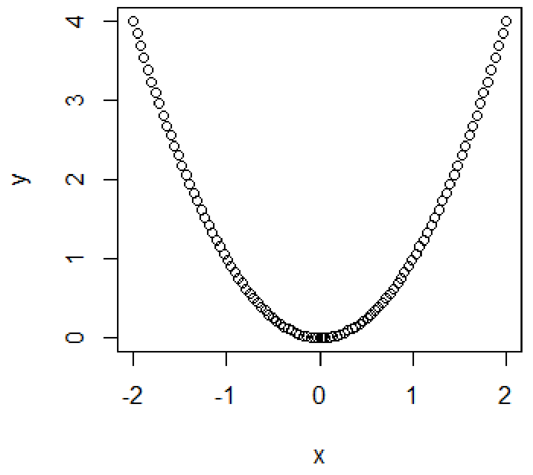
Easy to change lists:

```
>   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 plotting

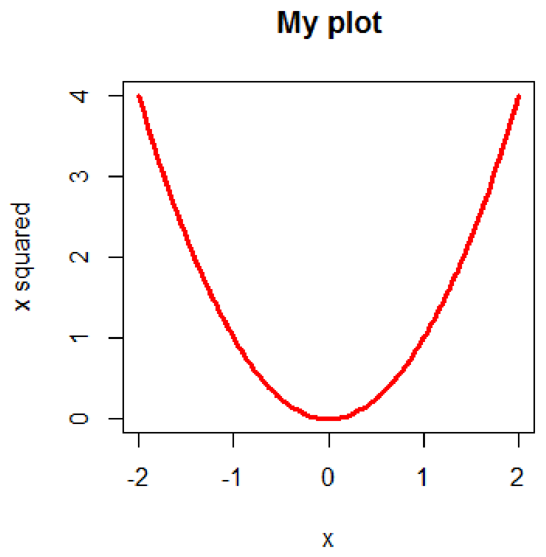
Let's plot the graph of $y = x^2$!

```
>   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!")
```



The pipe: |>

Introduced in R version 4.1

- Enables function chaining. Often easier to read.

Consider this code:

```
>  a = exp(2)
>  b = log(a, base = 10)
>  rep(b, times = 3)
[1] 0.868589 0.868589 0.868589
```

One-liner producing the same:

```
>  rep(log(exp(2), base = 10), times = 3)
[1] 0.868589 0.868589 0.868589
```

With piping:

```
>  exp(2) |> log(base = 10) |> rep(times = 3)
[1] 0.868589 0.868589 0.868589
```

Purists: Line break after each pipe

```
exp(2) |>
  log(base = 10) |>
  rep(times = 3)
```

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...

Installing packages

To access the functions of an external package, you must:

- install the package
  - downloads it to your computer
  - this is done only once
  - `install.packages()`
- load it into R
  - every new session
  - `library()`

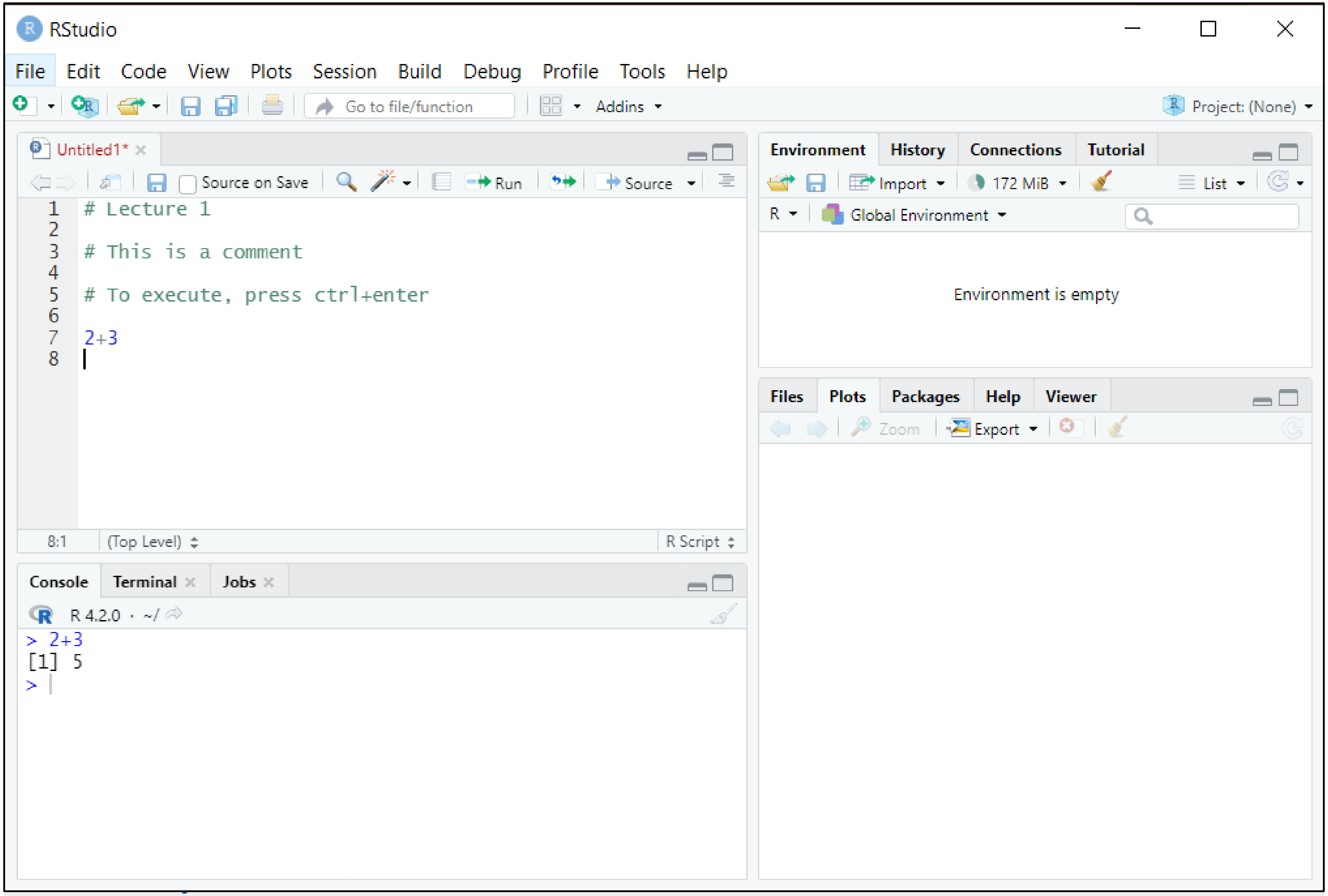
To check if a package is installed, simply try to load it:

```
>   library(pedsuite)
```

If you get an error, do:

```
>   install.packages("pedsuite")
```

RStudio and scripting

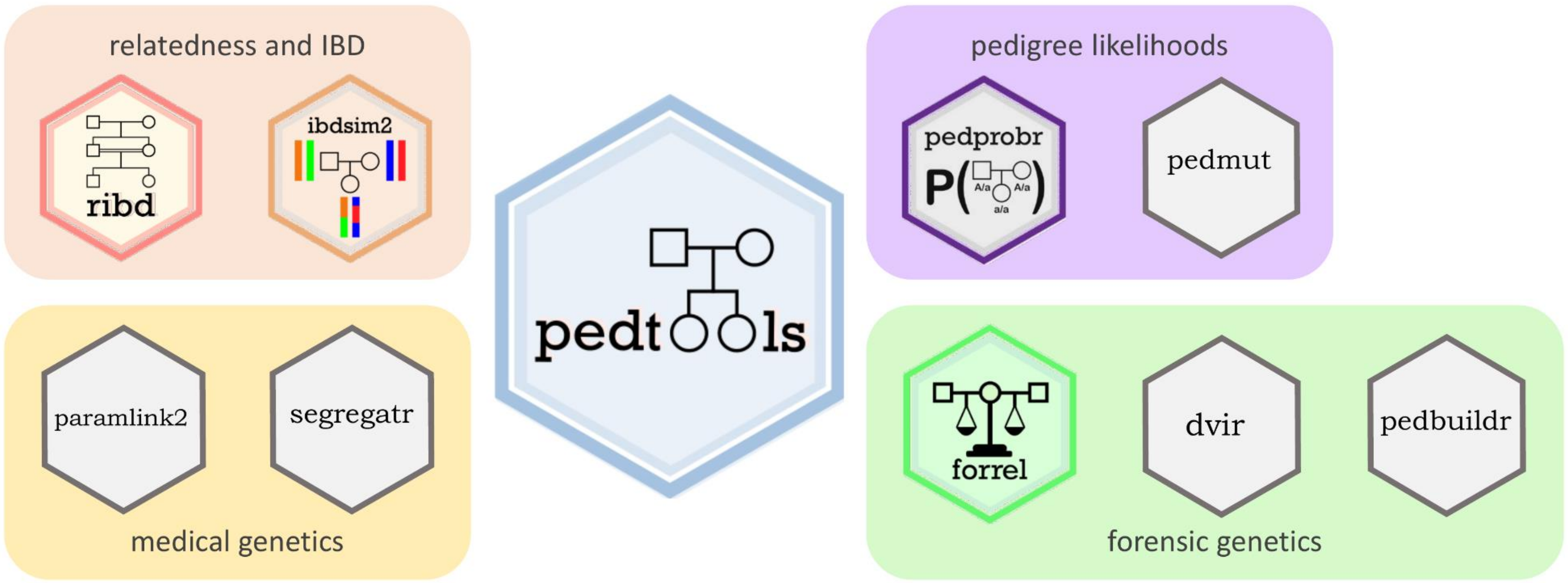


```
# Lecture 1

# This is a comment

# To execute, press ctrl+enter

2+3
```

Environment is empty

```
> 2+3
[1] 5
>
```

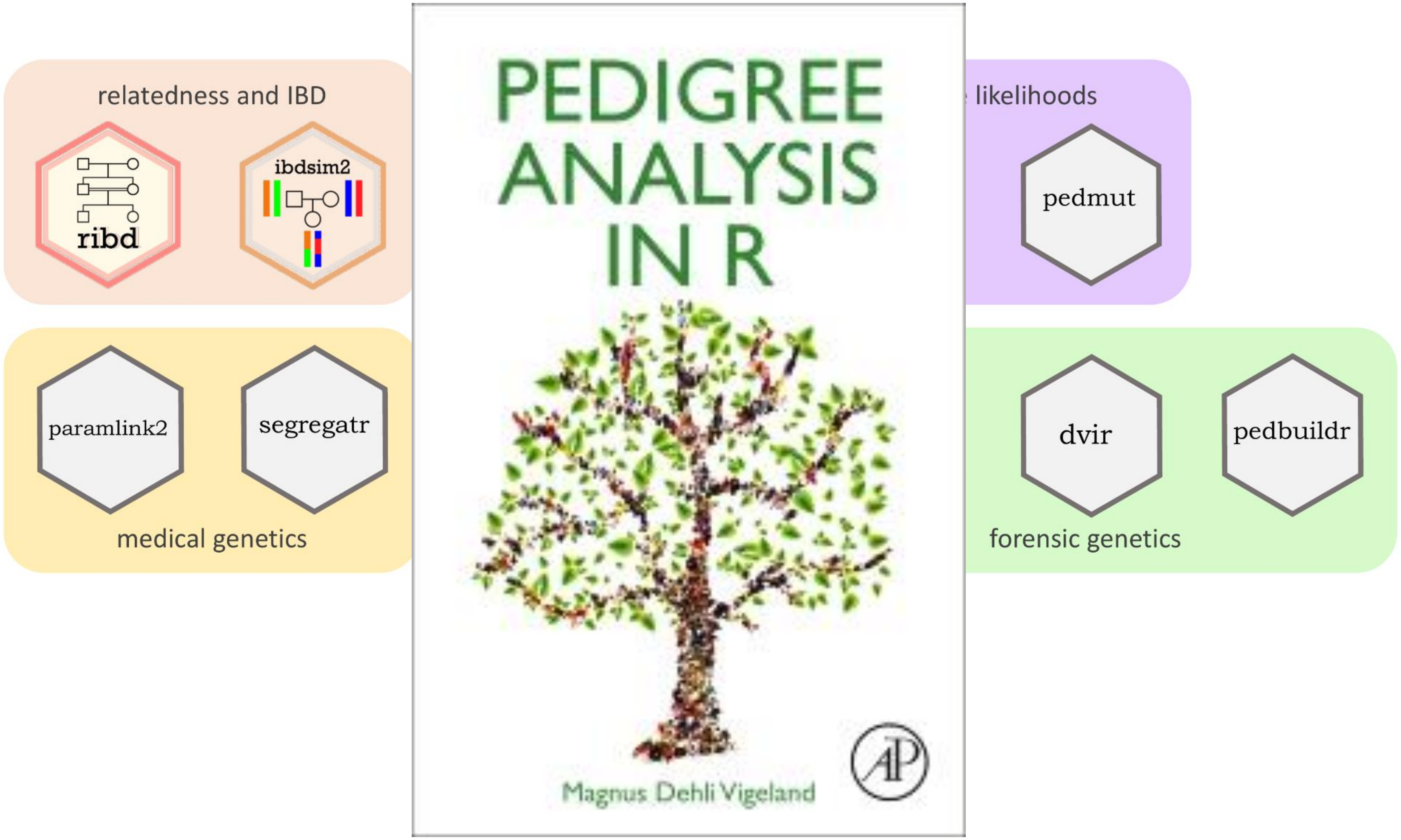
The ped suite: A collection of packages for pedigree analysis in R



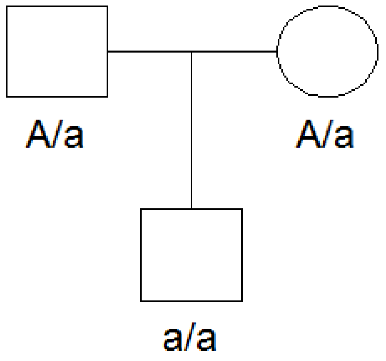
Home page:
https://magnusdv.github.io/pedsuite

Source code available on GitHub:
https://github.com/magnusdv

The ped suite: A collection of packages for pedigree analysis in R



pedtools: Tools for working with pedigrees in R



What it contains

pedigrees

markers

What it does

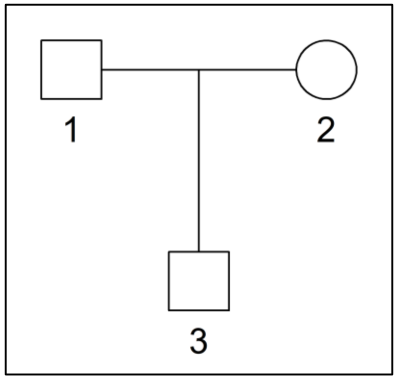
- Create
- Manipulate
- Investigate
- Plot ← kinship2

Either:

- ✓ Build with **pedtools**
- ✓ Read from file:
  - `readPed()`
  - `readFam()`

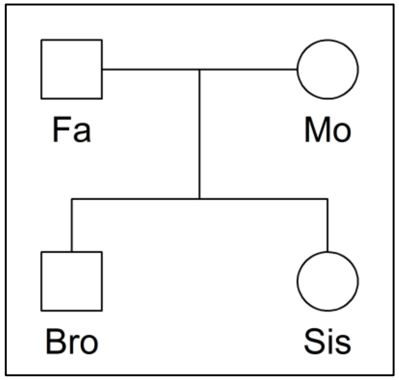
Building pedigrees

```
>   library(pedtools)

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



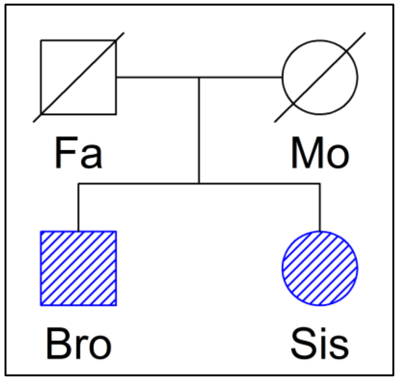
Names and sex:

```
>   y = nuclearPed(father = "Fa",
                   mother = "Mo",
                   child = c("Bro", "Sis"),
                   sex = 1:2)
>   plot(y)
```



Many ways to tweak the plot!

```
>   plot(y,
         deceased = c("Fa", "Mo"),
         hatched = c("Bro", "Sis"),
         col = list(blue = c("Bro", "Sis")),
         cex = 1.5)
```



Some useful functions

Create: basic

- singleton
- nuclearPed
- linearPed
- halfSibPed
- avuncularPed
- cousinPed

Create: complex

- ancestralPed
- doubleCousins
- quadHalfFirstCousins
- fullSibMating
- randomPed

Manipulate

- addSon
- addDaugher
- addParents
- addChildren

- swapSex
- relabel
- removeIndividuals

- branch
- subset

- mergePed
- breakLoops

Relatives

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

- ancestors
- descendants
- unrelated

Member subsets

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

Another example

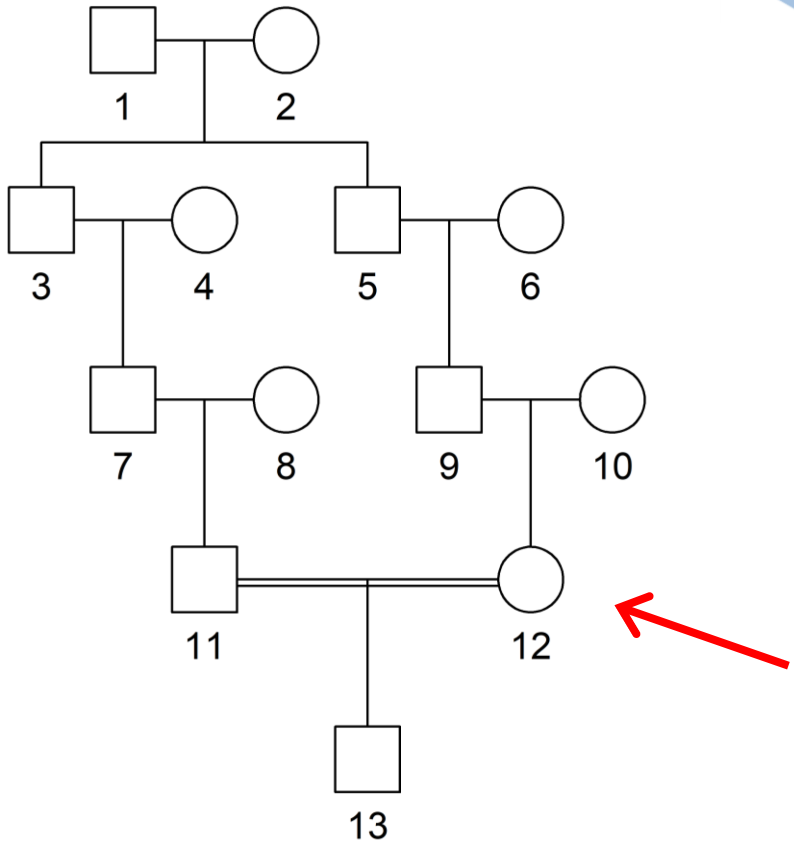
```
>   x = cousinPed(2)
>   plot(x)
```

Change gender:

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

Add inbred child

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



Remember or pipe!

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

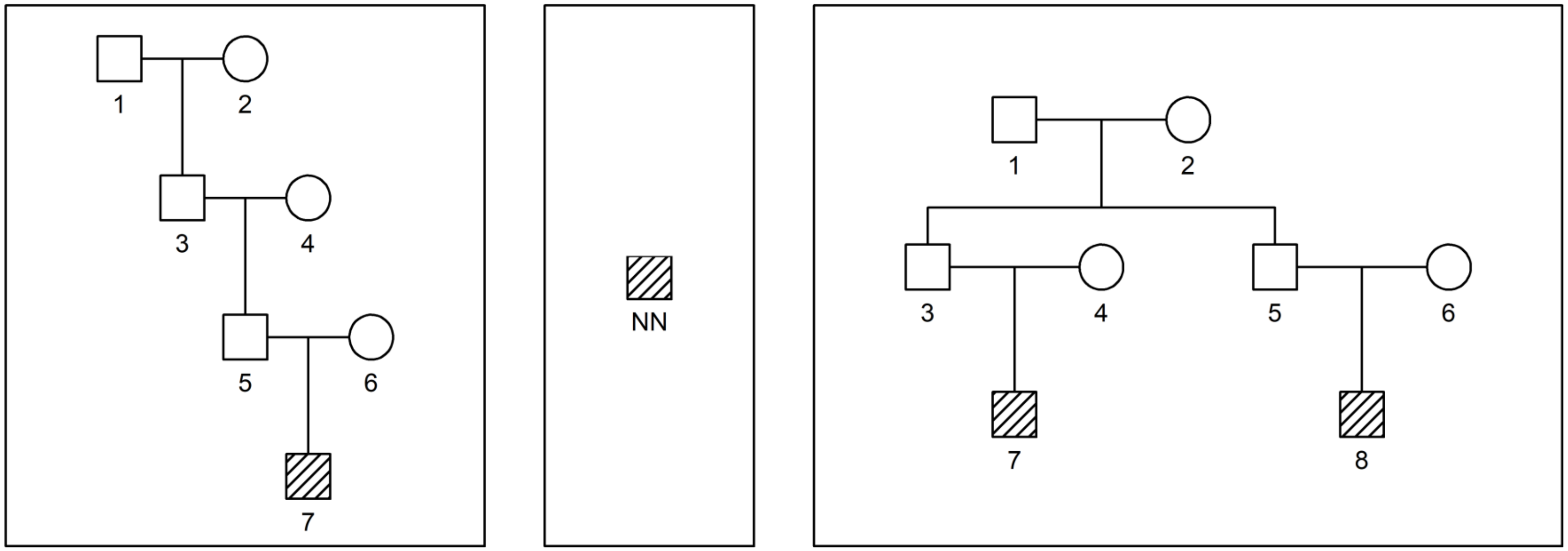
Shortcut command for this pedigree

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

List of pedigrees

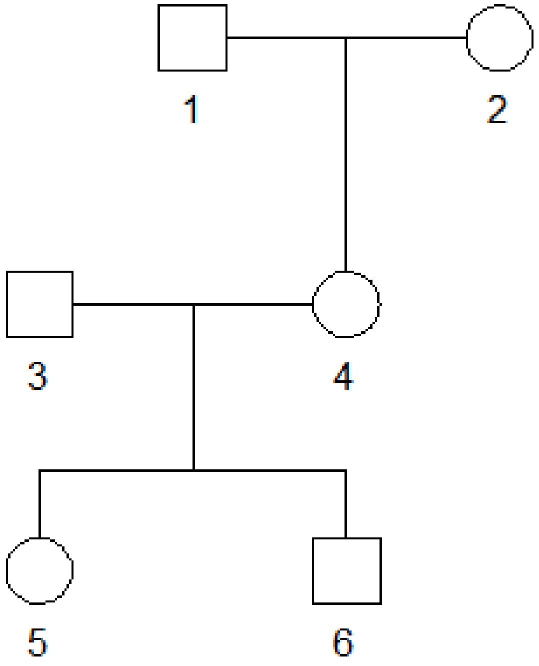
```
>   peds = list(linearPed(3),
                singleton("NN"),
                cousinPed(1))
```

```
>   plotPedList(peds,
                widths = c(2, 1, 3),
                hatched = leaves)
```



Alternative pedigree creation: ped file

A text file describing a pedigree structure.



0 if founder

famid	id	fid	mid	sex
1	1	0	0	1
1	2	0	0	2
1	3	0	0	1
1	4	1	2	2
1	5	3	4	2
1	6	3	4	1

Contents of *example.ped*

In pedtools:

```
>  x = readPed("example.ped")
>  plot(x)
```

Columns

famid	= family ID (optional)
id	= individual ID
fid	= ID of father
mid	= ID of mother
sex	= 1 (male), 2 (female) or 0 (unknown)

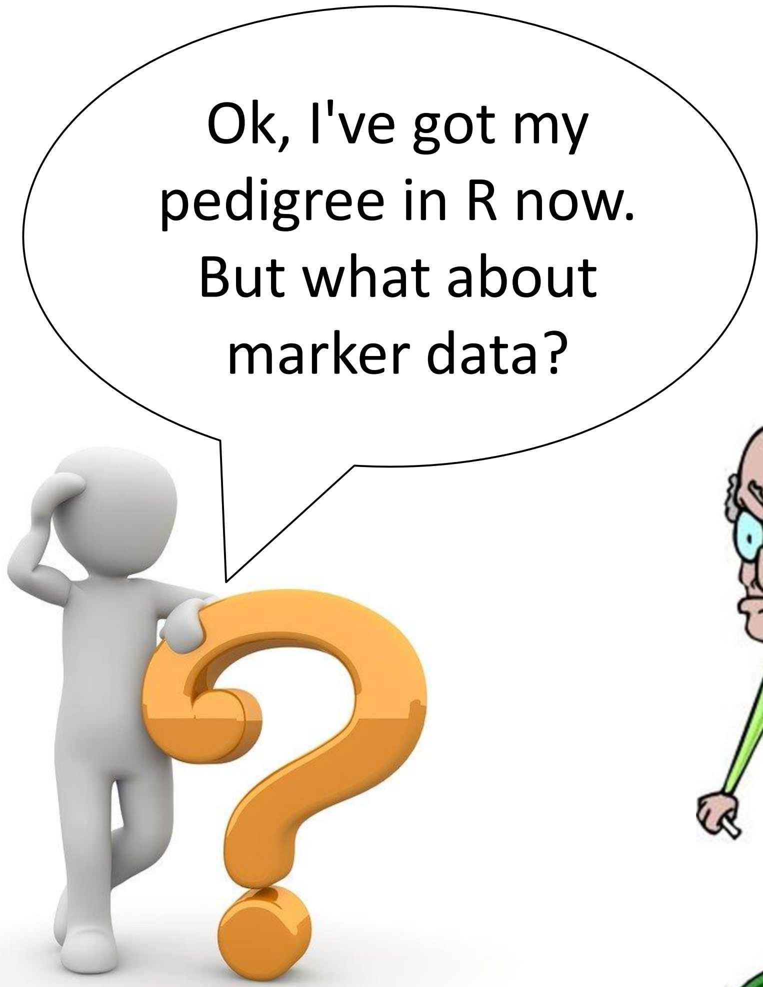
Oh my! Do I have to write these ped-files by hand?

No, that is tedious and error-prone!

Better: **QuickPed**

Quick **QuickPed** ped demo

https://magnusdv.shinyapps.io/quickped

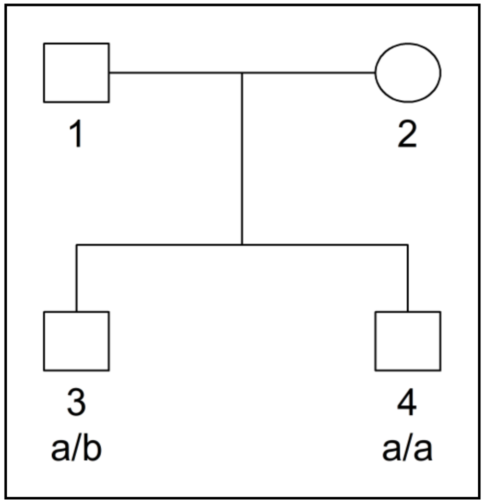

Ok, I've got my pedigree in R now. But what about marker data?

- ✓ Create with **pedtools**

  or
- ✓ Include in
  - ped file
  - fam file

Marker data 1

```
>   x = nuclearPed(2)
>   m = marker(x, geno = c(NA, NA, "a/b", "a/a"))
>   plot(x, marker = m)
```

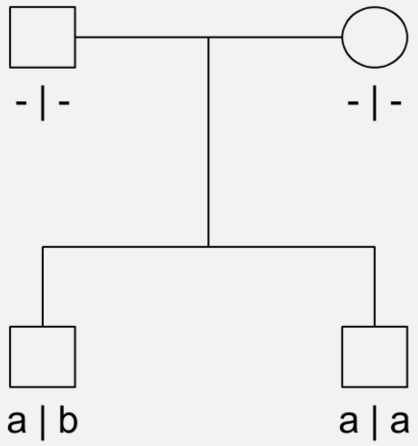


Print information about m:

```
>   m
 id  <NA>
  1   -/-
  2   -/-
  3   a/b
  4   a/a
* * * * *
Position: NA
Mutation: none
Frequencies:
   a   b
 0.5 0.5
```

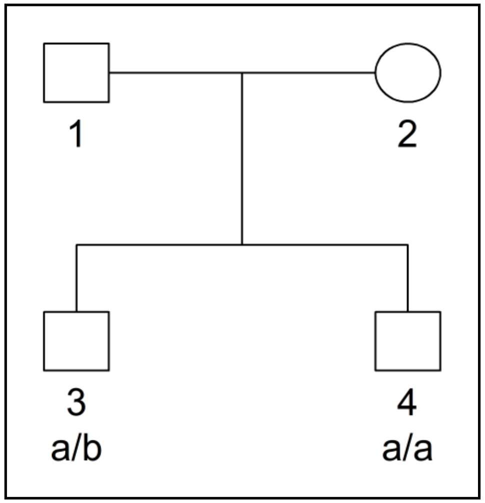
Plot options for markers:

```
>  plot(x, marker = m,
         labs = NULL,
         sep = " | ",
         showEmpty = TRUE)
```



Marker data 2: Attaching to pedigree

```
>   x = nuclearPed(2)
>   x = addMarker(x, geno = c(NA, NA, "a/b", "a/a"))
>   plot(x, marker = 1)
```



Or with pipe:

```
> x = nuclearPed(2) |>
    addMarker(geno = ...)
```

The genotypes are shown in a new column

```
>   x
id fid mid sex <1>
  1   *   *   1 -/-
  2   *   *   2 -/-
  3   1   2   1 a/b
  4   1   2   1 a/a
```

new column!

Changing allele frequencies

```
>   fr = c(a = 0.1, b = 0.9)
>   x = setAfreq(x, marker = 1,
                 afreq = fr)
```

Inspect the result

```
>   afreq(x, marker = 1)
  a   b
0.1 0.9
```

Functions for manipulating marker data

Get/set attributes

- afreq / setAfreq
- genotype / setGenotype
- chrom / setChrom
- posMb / setPosition
- name / setMarkername
- getMap / setMap

Attach/remove

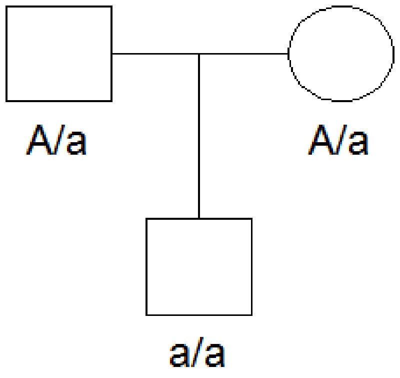
- setMarkers
- addMarkers
- selectMarkers
- removeMarkers

- transferMarkers

Frequency database

- getFreqDatabase
- setFreqDatabase
- readFreqDatabase
- writeFreqDatase

pedprobr: Pedigree probabilities in R



What it does

Compute the probability

$$\mathcal{P}(genotypes \mid pedigree;\ params)$$

Features

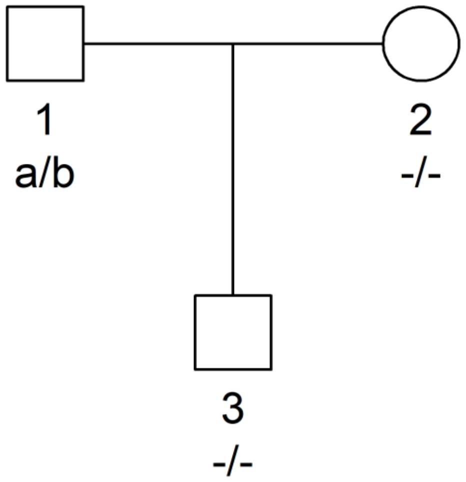
- arbitrary inbreeding
- autosomal & X-linked
- linked markers
- mutation models
- Elston-Stewart algorithm

A simple likelihood

```
>   library(pedprobr) # or library(pedsuite)
```

Create a pedigree with a SNP marker

```
>   x = nuclearPed(1)
>   x = addMarker(x, geno = c("a/b", NA, NA))
>   plot(x, marker = 1)
```



Compute the pedigree likelihood

```
>     likelihood(x, marker = 1)
[1] 0.5
```

Control

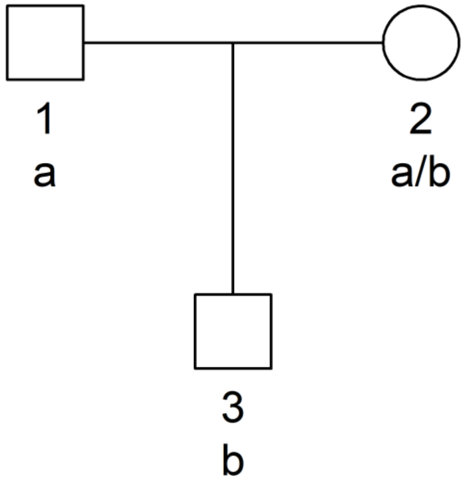
- By default, $P(a) = P(b) = 0.5$
- Thus HWE implies

$$P(a/b) = 2pq = 2 \cdot 0.5 \cdot 0.5 = 0.5$$

An example on X

Add X-marker to the pedigree

```
>   mx = marker(x, geno = c("a", "a/b", "b"),
                chrom = "X")
>   x = setMarkers(x, mx)
>   plot(x, marker = 1)
```



Compute the pedigree likelihood

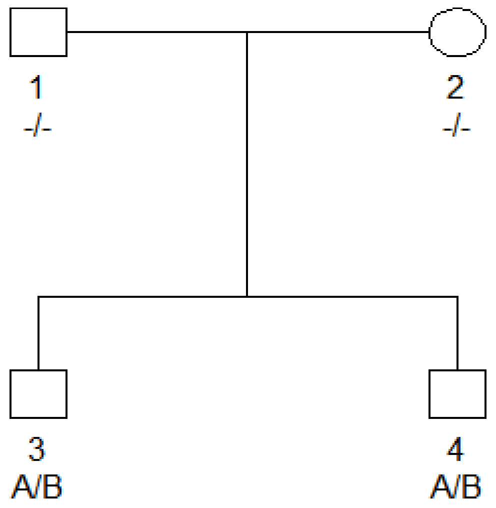
```
>   likelihood(x)
[1] 0.125
```

By default, `likelihood(x)` returns the likelihood for each attached markers.

oneMarkerDistribution

Computes the genotype distribution for one or several pedigree members, conditional on the observed genotypes

```
>   x = nuclearPed(2)
>   x = addMarker(x, geno = c(NA, NA, "A/B", "A/B"))
>   plot(x, 1)
```



```
>   oneMarkerDistribution(x, partial = 1, ids = 1:2)
Joint genotype probability distribution for individuals 1 and 2:
    A/A B/B A/B
A/A 0.0 0.2 0.1
B/B 0.2 0.0 0.1
A/B 0.1 0.1 0.2
```

Now: Exercises!