Week01: Overview

Preamble

Firing up python for Rmd

```r
library(reticulate)
```

```
## Warning: package 'reticulate' was built under R version 4.0.2
```

```r
use_python("C:/Users/PICHAU/anaconda3/python.exe")
```

Additional packages and notes

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

History of R

- First written in Fortran, then in C
- Environment for basic analysis, and then programming

Features i didn't know of

- Identical to S
- Run on anything (even Playstation 3)
- Constant releases
- User turns into a programmer

Free software?

1. Run the program
2. Study the program and source code
3. Freedom to redistribute
4. Freedom to improve the program

Drawbacks

1. 40-year old technology
2. 3D/dynamic graphics sucks
3. Functionality is based on demand
4. Objects are (usually) stored in physical memory
5. Slow if you don't know C++can

Asking questions

Pretty basic, i'm a master forumer already.

Code: Input and Evaluation

```
## Assignment operators
x = 1
y <- 2
print(x)
```

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

```
print(y)
```

```
## [1] 2
```

```
## Incomplete expression
# ex:
# x <-
```

Evaluation: whenever some object is called, it is auto-printed:

```
print(x)
```

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

Let's try a sequence:

```
seq = 1:20
print(seq)
```

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

R Objects and Attributes

- Everything is an object
- Atomic types:
    - Char
    - Num
    - Int
    - Complex
    - Logicals
- The most basic object is a vector
- Lists are represented as vector, but may contain anything inside

Numbers

- Num objects
- An integer must be specified with L, such as: `1L`
- Other numbers:
  - `Inf` is infinity
  - `NaN` is "not a number", or missing value

Attributes

R objects can have attributes:

- names, dimnames
- dimensions
- class
- length
- anything user-defined
- check with `attributes()`

Data types: vectors and functions

c() - concatenates values Creates vectors with the desired content

```
x = c(0.5, 0.6)
x = c(2L, 4L)
x = 9:29
x = c(1+0i, 2+4i)
```

```


x = vector("numeric", length = 15)
print(x)
```

vector() - creates vectors

```
##  [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Mixing objects and coercion when mixing objects in a vector, the least enforced type is coerced, from boolean, to integer, to character

Explicit coercion: as.type() We can force a class to be whatever type we want explicitly:

```
x = 0:5
as.numeric(x)
```

```
## [1] 0 1 2 3 4 5
```

```r
as.logical(x)
```

```
## [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
```

```r
as.character(x)
```

```
## [1] "0" "1" "2" "3" "4" "5"
```

```r
as.complex(x)
```

```
## [1] 0+0i 1+0i 2+0i 3+0i 4+0i 5+0i
```

Nonsensical coercion results in NAs:

```r
x = c("a","b","c")
as.numeric(x)
```

```
## Warning: NAs introduzidos por coerção
```

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

```r
as.logical(x)
```

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

```r
as.complex(x)
```

```
## Warning: NAs introduzidos por coerção
```

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

Lists

Super vectors that can carry anything as an indexed element

```r
x = list(1, "a", T, 1+4i)
print(x)
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] "a"
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] 1+4i
```

Matrices

Vectors with a dimension attribute, an integer

```r
m = matrix(nrow = 2, ncol = 3)
m
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```r
dim(m)
```

```
## [1] 2 3
```

```r
attributes(m)
```

```
## $dim
## [1] 2 3
```

Constructing a matrix column-wise:

```r
m2 = matrix(1:6,
            nrow = 2,
            ncol = 3)

print(m2)
```

```
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
```

Can be creating by dimensions after a vector:

```r
m = 1:10
dim(m) = c(2,5)
print(m)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    5    7    9
## [2,]    2    4    6    8   10
```

Matrices can also be created by binding stuff:

```r
x = 1:3
y = 10:12
cbind(x, y)
```

```
##      x  y
## [1,] 1 10
## [2,] 2 11
## [3,] 3 12
```

```
rbind(x,y)
```

```
##   [,1] [,2] [,3]
## x    1    2    3
## y   10   11   12
```

Data types - Factors

Special type of vector to represent categorical data, examples:

- Male and female
- Treatment and control

They are used by special functions such as `lm()`, `DESeq()`, and `glm()`. Their use is encouraged since factors are self-describing.

factor() - creating factors Factors are created as such:

```
x = factor(c("yes", "yes", "no", "yes", "no"))
print(x)
```

```
## [1] yes yes no  yes no
## Levels: no yes
```

Can be shown as tables:

```
table(x)
```

```
## x
##  no yes
##   2   3
```

Classes can be stripped from vectors:

```
unclass(x)
```

```
## [1] 2 2 1 2 1
## attr(,"levels")
## [1] "no"  "yes"
```

levels() - ordering factors Factors can be ordered, which is important for linear modelling, especially determining the baseline. Let's set up yes for the baseline:

```
x = factor(c("yes", "yes", "no", "yes", "no"),
           levels = c("yes", "no"))
print(x)
```

```
## [1] yes yes no  yes no
## Levels: yes no
```

Missing values