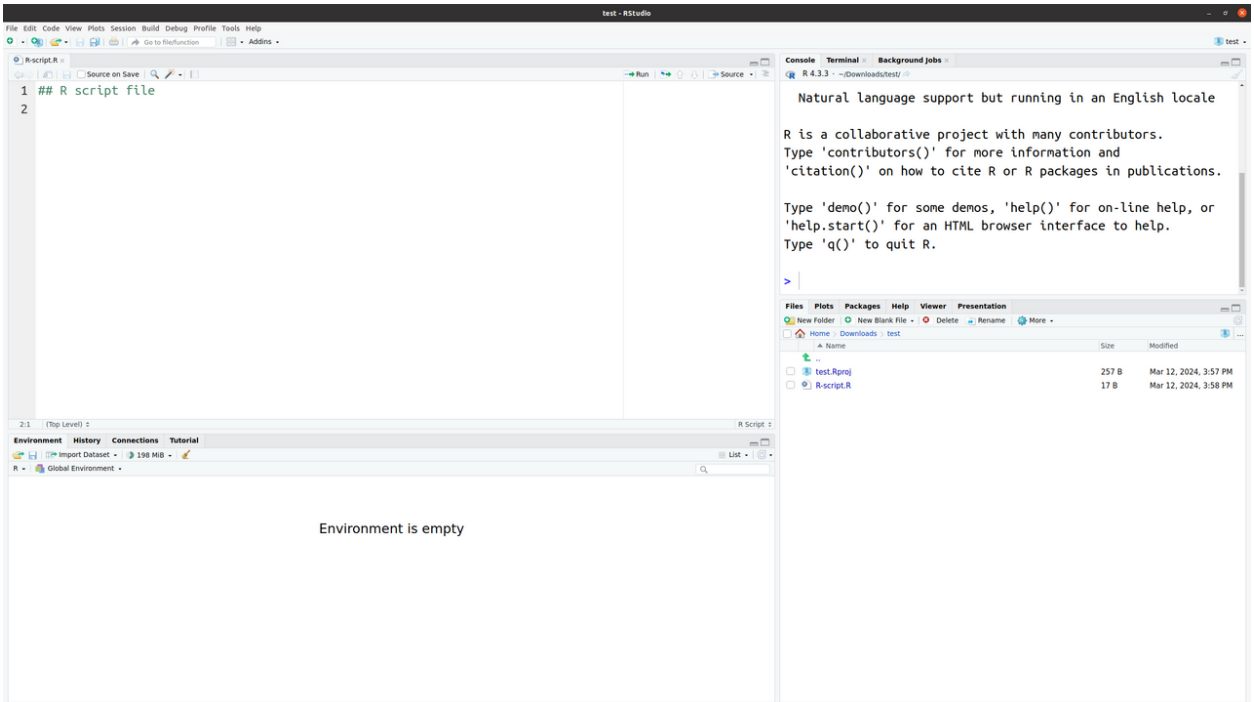
Intro to R and RStudio

Contents

1 Introduction to R

1.1 R is just a "cursor"

1.2 RStudio is the best IDE for R



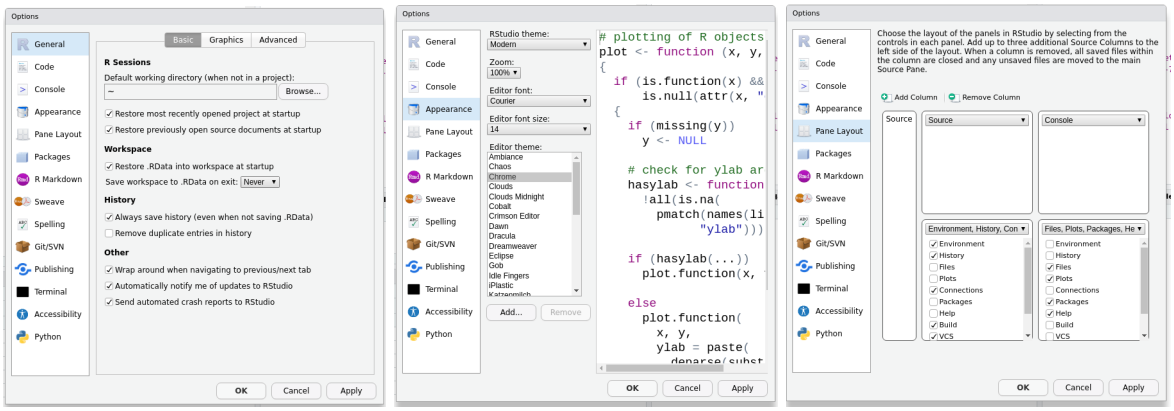
1.3 Hints for RStudio

RStudio has many features you will learn to appreciate and use in the future

Some useful hints at first encounter:

- Create Projects: *File > New Project...* or in the upper right corner.
  - Helps you organise your files for different projects
  - Helps to set the correct working directory
- Working directory
  - Use More (cogwheel) in the files pane to figure out and change the working directory.
  - Use commands `getwd()`/`sedwd()` to do it in the R console.
- Keyboard shortcuts. See list: *Tools > Keyboard Shortcuts Help* Alt+Shift+K
  - Insert `<-` use Alt+-
  - Jump to *source pane*: Ctrl+1
  - Jump to *console pane*:Ctrl+2
  - Execute a single line (from editor to console): Ctrl+Enter
- My preference: Don't save the workspace data so change the options (better to save on purpose or recompute)

Many settings can be changed using: *Tools > Global Options...*



1.4 R

Calculator

```
> 1 + 1
[1] 2
```

Getting help in R (see the Example section at the bottom)

```
?lm
```

Everything in R is functions. We use functions to construct objects, fit models and visualise data.

We can name our objects the way we like. There are *almost* no restrictions.

- can't start by numbers
- can't use *hypen*/dash, `-`, *nor* colon, `:`, in the naming (*nor* *, +, % and other crazy things)
- OK to use underscore, `_`, and dots, `.`

1.5 Data types

There are various data types in R

- numerics (doubles/floats)
- integers
- boolians (`TRUE` and `FALSE` — alternatively `T` and `F`, but please **avoid**)
- characters / strings
- factors — corresponds to categorical variables
- lists
- data.frames (tibbles)
- and many more…

1.6 Built-in datasets (`data.frame/tibble`)

Example data are available directly in R:

```r
head(chickwts)
```

```
  weight      feed
1    179 horsebean
```

```
2    160 horsebean
3    136 horsebean
4    227 horsebean
5    217 horsebean
6    168 horsebean
```

```r
str(chickwts)
```

```
'data.frame':   71 obs. of  2 variables:
 $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
 $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

```r
summary(chickwts)
```

```
     weight             feed
 Min.   :108.0   casein   :12
 1st Qu.:204.5   horsebean:10
 Median :258.0   linseed  :12
 Mean   :261.3   meatmeal :11
 3rd Qu.:323.5   soybean  :14
 Max.   :423.0   sunflower:12
```

Many packages also include data:

```r
library(ggplot2)
head(mpg)
```

```
# A tibble: 6 x 11
  manufacturer model displ  year   cyl trans      drv     cty   hwy fl    class
  <chr>        <chr> <dbl> <int> <int> <chr>      <chr> <int> <int> <chr> <chr>
1 audi         a4      1.8  1999     4 auto(l5)   f        18    29 p     compa~
2 audi         a4      1.8  1999     4 manual(m5) f        21    29 p     compa~
3 audi         a4      2    2008     4 manual(m6) f        20    31 p     compa~
4 audi         a4      2    2008     4 auto(av)   f        21    30 p     compa~
5 audi         a4      2.8  1999     6 auto(l5)   f        16    26 p     compa~
6 audi         a4      2.8  1999     6 manual(m5) f        18    26 p     compa~
```

```r
str(mpg)
```

```
tibble [234 x 11] (S3: tbl_df/tbl/data.frame)
 $ manufacturer: chr [1:234] "audi" "audi" "audi" "audi" ...
 $ model       : chr [1:234] "a4" "a4" "a4" "a4" ...
 $ displ       : num [1:234] 1.8 1.8 2 2 2.8 2.8 3.1 1.8 1.8 2 ...
 $ year        : int [1:234] 1999 1999 2008 2008 1999 1999 2008 1999 1999 2008 ...
 $ cyl         : int [1:234] 4 4 4 4 6 6 6 4 4 4 ...
 $ trans       : chr [1:234] "auto(l5)" "manual(m5)" "manual(m6)" "auto(av)" ...
 $ drv         : chr [1:234] "f" "f" "f" "f" ...
 $ cty         : int [1:234] 18 21 20 21 16 18 18 18 16 20 ...
 $ hwy         : int [1:234] 29 29 31 30 26 26 27 26 25 28 ...
 $ fl          : chr [1:234] "p" "p" "p" "p" ...
 $ class       : chr [1:234] "compact" "compact" "compact" "compact" ...
```

```r
summary(mpg)
```

```
 manufacturer          model               displ            year
 Length:234         Length:234         Min.   :1.600   Min.   :1999
 Class :character   Class :character   1st Qu.:2.400   1st Qu.:1999
 Mode  :character   Mode  :character   Median :3.300   Median :2004
```

```
                                      Mean   :3.472   Mean   :2004
                                      3rd Qu.:4.600   3rd Qu.:2008
                                      Max.   :7.000   Max.   :2008
     cyl           trans               drv                 cty
 Min.   :4.000   Length:234         Length:234         Min.   : 9.00
 1st Qu.:4.000   Class :character   Class :character   1st Qu.:14.00
 Median :6.000   Mode  :character   Mode  :character   Median :17.00
 Mean   :5.889                                         Mean   :16.86
 3rd Qu.:8.000                                         3rd Qu.:19.00
 Max.   :8.000                                         Max.   :35.00
     hwy             fl               class
 Min.   :12.00   Length:234         Length:234
 1st Qu.:18.00   Class :character   Class :character
 Median :24.00   Mode  :character   Mode  :character
 Mean   :23.44
 3rd Qu.:27.00
 Max.   :44.00
```

You can of course read in your own data in R almost no matter which data format it is in (more on that later).

1.7 Assignment operator =, <- or ->

In R we can create/assign values to objects in three ways

- `<-` The standard assignment in R (right-hand-side assigned to left-hand-side), e.g. `x <- 3`
- `->` Less standard assignment (LHS assigned RHS), e.g. `4 -> y`
- `=` Implicitly assigns RHS to LHS, so `x = 3` works but `4 = y` doesn't.

1.8 Vectors and subsetting

We will focus on using `dplyr` and other packages to handle data and wrangling of these, but some knowledge of base R is convenient (and necessary).

We construct simple vectors using the `c` function

```
x <- c(3, 5, 10, 3, 2, 8 , 7)
```

Select the first three

```
x[1:3]
```

```
[1]  3  5 10
```

Deselect the first three

```
x[-(1:3)]
```

```
[1] 3 2 8 7
```

Select elements in `x` less than 4.

```
x[x<4]
```

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

And which ones

```r
x<4
```

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

```r
which(x<4)
```

```
[1] 1 4 5
```

Utilise the *sloppiness* of R to select every other variable (the indexer is recycled — without any warnings)

```r
x[c(TRUE,FALSE)]
```

```
[1]  3 10  2  7
```

```r
x[c(FALSE,TRUE)]
```

```
[1] 5 3 8
```

1.9 Data type `factor`

In particular the `factor` type is important to understand. Eventhough it appears to just be a character string it is **really not**:

```r
f <- sample(chickwts$feed, 10)
f
```

```
 [1] meatmeal  casein    soybean   linseed   casein    sunflower soybean
 [8] horsebean soybean   casein
Levels: casein horsebean linseed meatmeal soybean sunflower
```

A `factor` is just a vector of integers with extra info about what levels (groups) the integers represent.

Dangerous if you don't know what you are doing (not even a warning is given here):

```r
c(f, "sunflower")
```

```
 [1] "4"         "1"         "5"         "3"         "1"         "6"
 [7] "5"         "2"         "5"         "1"         "sunflower"
```

We see that R converts the factors to their underlying *encodings* and the result is converted to a string!

If we manually want to convert from `factor` to `string` one can use `paste` or `as.character`

```r
paste(f)
```

```
 [1] "meatmeal"  "casein"    "soybean"   "linseed"   "casein"    "sunflower"
 [7] "soybean"   "horsebean" "soybean"   "casein"
```

1.10 Missing values, `NA`'s

```r
x <- c(1, 2, NA)
mean(x)
```

```
[1] NA
```

```r
mean(x, na.rm = TRUE)
```

```
[1] 1.5
```

```r
x == NA
```

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

```r
is.na(x)
```

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

1.11 User-defined functions

Function to convert "miles per gallon" to "km per litre":

```r
mpg_to_kmpl <- function(x){
  x_kmpg <- x*1.609
  x_kmpl <- x_kmpg/3.785
  return(x_kmpl)
}
```

```r
mpg_to_kmpl(c(20,30))
```

```
[1]  8.501982 12.752972
```

And the other way:

```r
kmpl_to_mpg <- function(x){
  x_mpl <- x/1.609
  x_mpg <- x_mpl*3.785
  return(x_mpg)
}
```

```r
kmpl_to_mpg(c(10,20))
```

```
[1] 23.52393 47.04786
```

1.12 Function with multiple arguments

Two way conversion:

```r
fuel_conv <- function(x, to = "kmpl"){
  if(to == "kmpl"){
    return(mpg_to_kmpl(x))
  } else{ # Any other vaule than "kmpl" ends here
    return(kmpl_to_mpg(x))
  }
}
```

```r
fuel_conv(30)
```

```
[1] 12.75297
```

```r
fuel_conv(30, to = "mpg")
```

```
[1] 70.57178
```

Notice: This is **not** a great way to do this. Any other value of `to` than `"kmpl"` will convert to miles per gallon:

```r
fuel_conv(30, "kml")
```

```
[1] 70.57178
```

```r
fuel_conv(30, 1)
```

```
[1] 70.57178
```

1.13 The pipe operator: |>

- Originally designed by Danish Stefan Milton Bache, the pipe operator `%>%` from the `magrittr` package (also in `tidyverse`) enables easier to read (and write) programming in R.
- Later a native pipe operator was introduced in base R: `|>`
- **Keyboard shortcut:** Ctrl+Shift+M (Windows and Linux) Cmd+Shift+M (Mac)
- You choose which pipe to use in the menu *Tools > Global Options > Code* (tick "Use native pipe...")

1.14 Basic piping

- `x |> f()` is equivalent to `f(x)`
- `x |> f(y)` is equivalent to `f(x, y)`
- `x |> f() |> g() |> h()` is equivalent to `h(g(f(x)))`

Here, "equivalent"" is not technically exact: evaluation is non-standard, and the left-hand side is evaluated before it is passed on to the right-hand side expression.

However, in most cases this has no practical implication.

1.15 Read left to right

- `|>` is pronounced as "and then".
- For example to take the highway milage `hwy`, convert to km per litre and calculate the variance we would write:

```r
mpg$hwy |>
  fuel_conv(to = "kmpl") |>
  var()
```

```
[1] 6.407548
```

- Traditional function calling would be:

```r
var(fuel_conv(mpg$hwy, to = "kmpl"))
```

```
[1] 6.407548
```

- It does not make a big difference in this example but later we will see how the pipe helps us break code into chunks that are easier to read, write, debug, extend, modify and combine.

```
DATA |>
  SOME_CHANGE_TO_THE_DATA |>
  A_SUBSEQUENT_OPERATION  |>
  ...  |>
  FINAL_OPERATION_OR_PLOT
```

1.16 The argument placeholder

You may find code that contains the placeholder `_` such as:

- `x |> f(y, z = _)` which is equivalent to `f(y, z = x)`

(The magrittr pipe `%>%` uses `.` as placeholder.)

1.17 Three different workflows

- Typing commands in R console
  - Great for testing things quickly and exploring.
  - Bad for things you want to keep/save.
- Working with an R script
  - Good for saving a single analysis that doesn't need a lot of text explanation.
  - Very little overhead compared to typing directly in the console.
  - Good for checking reproducibility when you rerun everything with the "Compile report"-button (Ctrl+Shift+K)
- Working with a qmd (Quarto) or Rmd (Rmarkdown) file
  - Great for reports where text and code/graphics is used together.
  - A bit of overhead in changing between code and text.
  - Good for checking reproducibility when you rerun everything with the "Knit"-button (Ctrl+Shift+K)
- **Note for both R script and Quarto/Rmarkdown:**
  - Ctrl+Shift+K (Knit / Compile report) starts an **empty** R session and executes the commands in the file sequentially.
  - So you have to include **all** necessary commands including `library()` etc.
  - You cannot rely on commands you have typed previously in the R session.