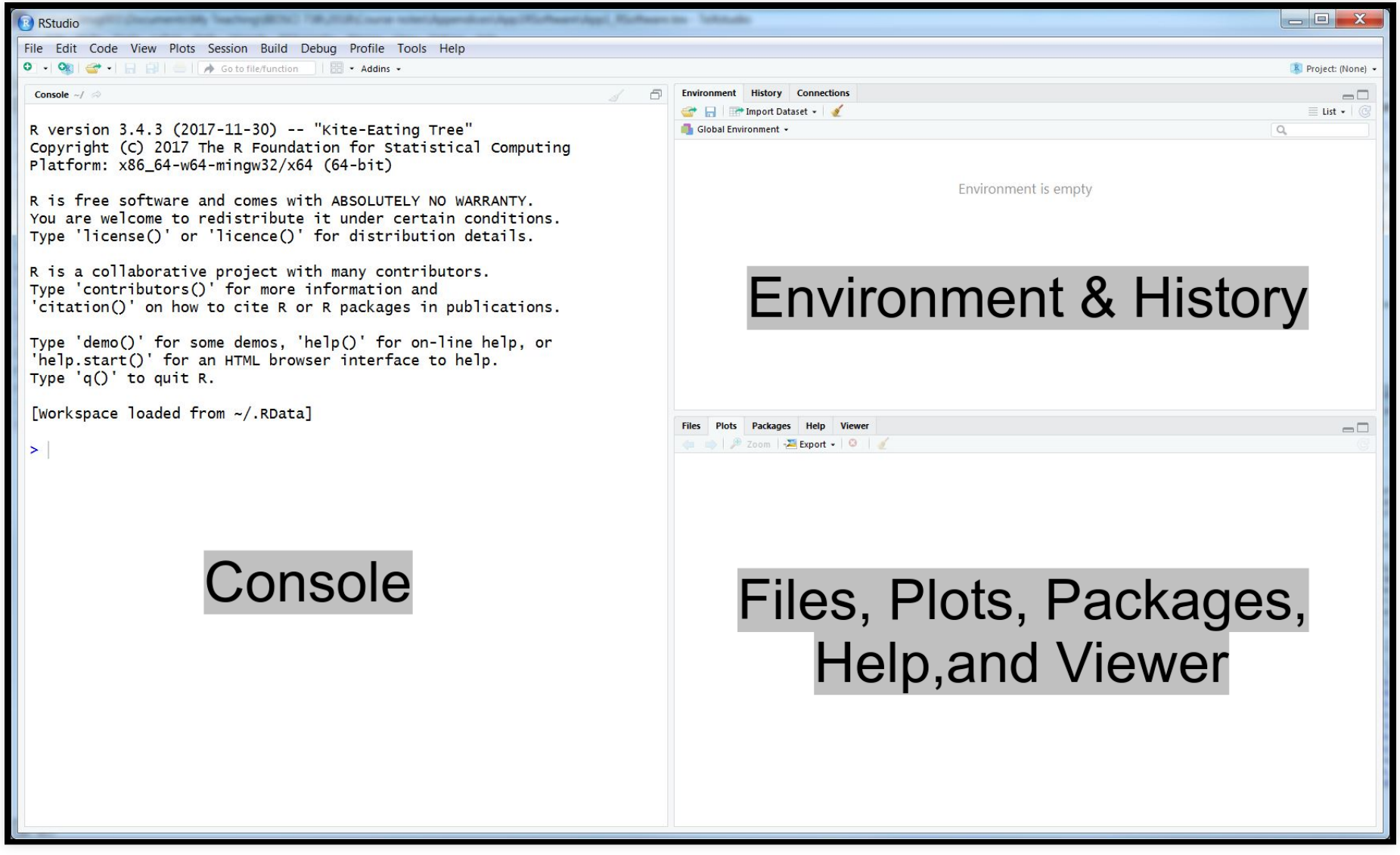
The RStudio interface



Introduction to **R**

Using R as a calculator

```
1 + 2
```

```
#R:  [1] 3
```

```
1 + 3^2
```

```
#R:  [1] 10
```

```
log(15) - sqrt(3.4)
```

```
#R:  [1] 0.8641413
```

```
pnorm(1.96)
```

```
#R:  [1] 0.9750021
```

```
# This is a comment and is not evaluated
```

Using **R** as a calculator

- `=` is the assignment operator (you can also use `<-`).
- For example, `x = 2` means that we have assigned the value 2 to the object `x`.

```
x = 2
y = 3

x^2 - 3 * y + 5
```

```
#R:  [1] 0
```

Note that **R** is case-sensitive

```
X
```

```
#R:  Error in eval(expr, envir, enclos): object 'X' not found
```

Different types of data objects in **R**

R has 6 different data types:

- character (alphanumeric; `"hello world"`)
- numeric (real or decimal; `3.14159`)
- integer (whole numbers; `256`)
- logical (`TRUE` or `FALSE`)
- factor (numeric or alphanumeric, treated as categorical)
- complex (numbers with imaginary components; `3i`)

Vectors

- Use `c()` to combine multiple elements separated by comma's.
- A *vector* is a combination of multiple elements of the same data type in 1 dimension (a one-dimensional array).

```
# A character vector contains strings
c("hello", "world")
```

```
#R:  [1] "hello" "world"
```

```
# A numeric vector contains numbers
c(1, 2, 3, 4, 5, 6)
```

```
#R:  [1] 1 2 3 4 5 6
```

```
# We can easily produce sequences using ':'
1:6
```

```
#R:  [1] 1 2 3 4 5 6
```

Matrices

- Use `matrix()` to create a matrix in **R**.
- A *matrix* is a combination of multiple elements of the same data type in 2 dimensions (a 2-dimensional array).

```r
# Create a matrix with 2 rows
matrix(1:6, nrow = 2)
```

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

```r
# Create a matrix with 2 columns
matrix(1:6, ncol = 2)
```

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

Dataframes

- Use `data.frame` to create a dataframe in **R**.
- A *dataframe* is a collection of multiple vectors (as different columns) that can be different types.

```
my_characters = c("one", "two", "three")
my_numbers = 1:3
my_logicals = c(TRUE, FALSE, F)

data.frame(my_characters, my_numbers, my_logicals)
```

```
#R:     my_characters my_numbers my_logicals
#R:   1           one          1        TRUE
#R:   2           two          2       FALSE
#R:   3         three          3       FALSE
```

Getting help

- Google!
  - "How to calculate the average in R?"
  - The search results tell you that the `mean()` function is useful.
- Quick-R: https://www.statmethods.net/
- R-Bloggers: https://www.r-bloggers.com/
- Stack Overflow (SO): https://stackoverflow.com/questions/tagged/r

Getting help

- `?`

  For example, `?mean` brings up the help file for the `mean` function. It will tell you almost everything you need to know to use `mean()`.

- `??`

  For example, `??mean` searches for everything related to "mean" in all the **R** packages installed on your computer.

- `RSiteSearch("mean")`

  Searches everything on CRAN (an online repository of **R** packages). This requires interenet connection.

Basic principles for data organisation in spreadsheets

Be consistent

> *Whatever you do, do it consistently*

Use consistent:

- codes for categorical variables (not M, Male, and male).
- codes for missing values (can use NA, -, or leave blank).
  - Do not use a numeric value (999).
- variable names in all files (glucose_10wk, Gluc10wk)
- subject identifiers (mouse153, M153, 153)
- date formats (YYYY-MM-DD, YY/DD/MM)

Also, be careful about spaces within cells. A blank cell is different to a cell with a space in, and “Male” is different to “ Male”.

Choose good names for things

> *It is worth putting some time and thought into picking good names for things*

In general:

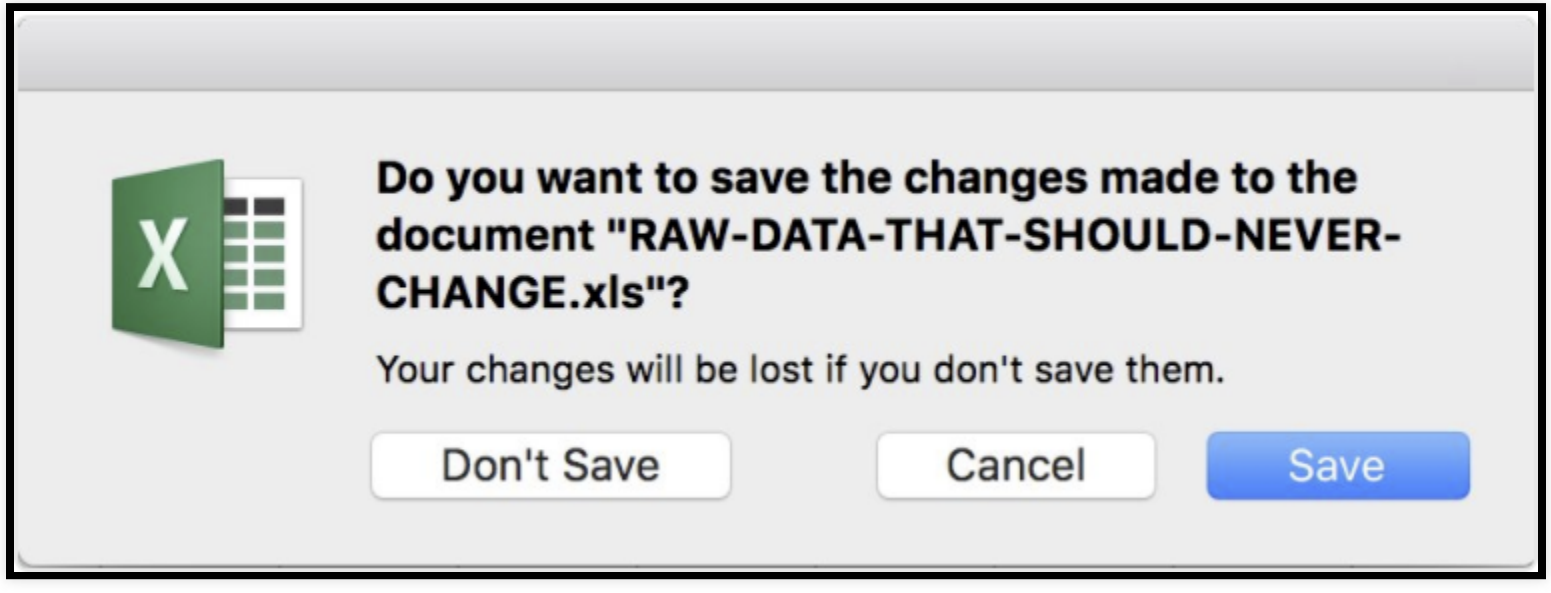
- Do not use spaces in variable (column) names or file names.
  - Use underscores or hyphens instead (but not both).
- Avoid special characters ($, @, %, #, &, (,), !, /, etc.).
- Never use “final” in the file name.
- Use short but *meaningful* names.

Other important guidelines

- Put just one thing in a cell (i.e. separate lat, lon columns).
- Make it a rectangle:
  - Rows corresponding to subjects (or observations).
  - Columns corresponding to variables.
  - Do not scatter tables around a worksheet.
- Create a data dictionary.
- No calculations in the raw data files.
- Do not use font colour or highlighting as data.
- Save the data in plain text files (i.e. a CSV).
- Make backups (or use a version control system).

Other important guidelines

> *Do not overwrite original data files!*



Reading data into **R**

Read and check

- Always set a working directory using `setwd()`. This can be a directory where you store the data and/or output the results.
- Use `read.csv()` to read a CSV file into **R**.
- `dim()` returns the number of observations (rows) and variables (columns).
- `head()` and `tail()` return the first and last few rows of the data set, respectively.
- `names()` returns the names of the variables in the data set.
- `str()` returns the structure of the dataset, e.g. dimension, column names, type of data object, first few values of each variable.

CSV file containing patient information

The patient CSV file has 7 variables:

- `Patient.ID`: Unique ID number.
- `Age`: Age in years.
- `Gender`: 0 = Female, 1 = Male.
- `Ethnicity`: 1 = Caucasian, 2 = African, 3 = Other.
- `Weight`: Weight in pounds.
- `Height`: Height in inches.
- `Smoke`: 1 = Yes, 2 = No.

Reading the data file into **R**

```
setwd("Data/")
patient.df = read.csv("Patient.csv")
head(patient.df)
```

```
#R:     Patient.ID Age Gender Ethnicity Weight Height Smoke Cholesterol
#R:  1           3  21   Male         1  179.5   70.4    NA         268
#R:  2           4  32 Female         1     NA   63.9    NA         160
#R:  3           9  48 Female         1  149.7   61.8     2         236
#R:  4          10  35   Male         1  203.5   69.8    NA         225
#R:  5          11  48   Male         1  155.3     NA     2         260
#R:  6          19  44   Male         2  189.6   70.2     1         187
```

Check the variable names

```r
# Names of the variables
names(patient.df)
```

```
#R:  [1] "Patient.ID"  "Age"          "Gender"       "Ethnicity"   "Weight
#R:  [6] "Height"       "Smoke"        "Cholesterol"
```

- Anything following the `#` symbol is treated as a comment, which is ignored by **R**.
- Writing comments is a very good habit to develop!

Check the structure of the data set

```
str(patient.df)
```

```
#R:  'data.frame':  17030 obs. of  8 variables:
#R:   $ Patient.ID : int  3 4 9 10 11 19 34 44 45 48 ...
#R:   $ Age        : int  21 32 48 35 48 44 42 24 67 56 ...
#R:   $ Gender     : Factor w/ 2 levels "Female","Male": 2 1 1 2 2 2 1 1
#R:   $ Ethnicity  : int  1 1 1 1 1 2 2 1 2 1 ...
#R:   $ Weight     : num  180 NA 150 204 155 ...
#R:   $ Height     : num  70.4 63.9 61.8 69.8 NA 70.2 62.6 64.4 64.3 67.
#R:   $ Smoke      : int  NA NA 2 NA 2 1 1 1 NA 2 ...
#R:   $ Cholesterol: int  268 160 236 225 260 187 216 137 NA 156 ...
```

- Note that the *character* vector `Gender` is automatically converted to a *factor* vector.

Check the structure of the data set

We can set the `stringsAsFactors` argument to `FALSE`, so character strings are not converted to factors.

```
patient.df = read.csv("Patient.csv", stringsAsFactors = FALSE)
str(patient.df)
```

```
#R:  'data.frame':  17030 obs. of  8 variables:
#R:   $ Patient.ID : int  3 4 9 10 11 19 34 44 45 48 ...
#R:   $ Age        : int  21 32 48 35 48 44 42 24 67 56 ...
#R:   $ Gender     : chr  "Male" "Female" "Female" "Male" ...
#R:   $ Ethnicity  : int  1 1 1 1 1 2 2 1 2 1 ...
#R:   $ Weight     : num  180 NA 150 204 155 ...
#R:   $ Height     : num  70.4 63.9 61.8 69.8 NA 70.2 62.6 64.4 64.3 67.
#R:   $ Smoke      : int  NA NA 2 NA 2 1 1 1 NA 2 ...
#R:   $ Cholesterol: int  268 160 236 225 260 187 216 137 NA 156 ...
```

Descriptive statistics

Calculating averages

Calculate the average height of patients:

```
mean(Height)
```

```
#R:  Error in mean(Height): object 'Height' not found
```

You must tell **R** that `Height` is a variable (column) *within* the `patient.df` data frame:

```
mean(patient.df$Height)
```

```
#R:  [1] NA
```

There are missing values is the `Height` variable that **R** does not know what to do with.

Calculating averages with missing values

We can tell **R** to remove the missing values before calculating the average height:

```
mean(patient.df$Height, na.rm = TRUE)
```

```
#R:  [1] 65.43787
```

Table of counts

```r
# One-way table of counts
table(patient.df$Gender)
```

```
#R:
#R:  Female   Male
#R:    9077   7953
```

```r
# Same table of counts using 'with'
with(patient.df, table(Gender))
```

```
#R:  Gender
#R:  Female   Male
#R:    9077   7953
```

Table of proportions

```r
# Table of proportions for the gender variable
prop.table(table(patient.df$Gender))
```

```
#R:
#R:      Female       Male
#R:   0.5330006 0.4669994
```

```r
# Convert to % and round to 1dp
round(prop.table(table(patient.df$Gender)) * 100, 1)
```

```
#R:
#R:   Female    Male
#R:     53.3    46.7
```

Two-way frequency tables

```
gender_eth_table = with(patient.df, table(Gender, Ethnicity))
gender_eth_table
```

```
#R:           Ethnicity
#R:  Gender       1    2    3
#R:    Female 6114 2687  274
#R:    Male   5498 2173  279
```

Two-way frequency tables (proportions)

We can calculate proportions for each row (`margin = 1`) or for each column (`margin = 2`)

```
# Calculate proportions across gender for each ethnicity
prop.table(gender_eth_table, margin = 2)
```

```
#R:           Ethnicity
#R:  Gender            1         2         3
#R:    Female 0.5265243 0.5528807 0.4954792
#R:    Male   0.4734757 0.4471193 0.5045208
```

Summary

- Quick introduction to **R** and RStudio
- Spreadsheet guidelines
- Getting data into **R**
- Calculate averages
- Frequency tables