Workshop 2: Loading and Manipulating Data

QCBS R Workshop Series

Québec Centre for Biodiversity Science

About this workshop

REPO DEV | WIKI 02 | SLIDES 02 | SLIDES 02 | SCRIPT 02

Required packages

- dplyr
- tidyr
- magrittr

```
install.packages(c('dplyr', 'tidyr', 'magrittr'))
```

Learning Objectives

1. Creating an R project
2. Writing a script
3. Loading, exploring, and saving data
4. Learn to manipulate data frames with tidyr, dplyr, maggritr *(For more advanced users)*

RStudio projects

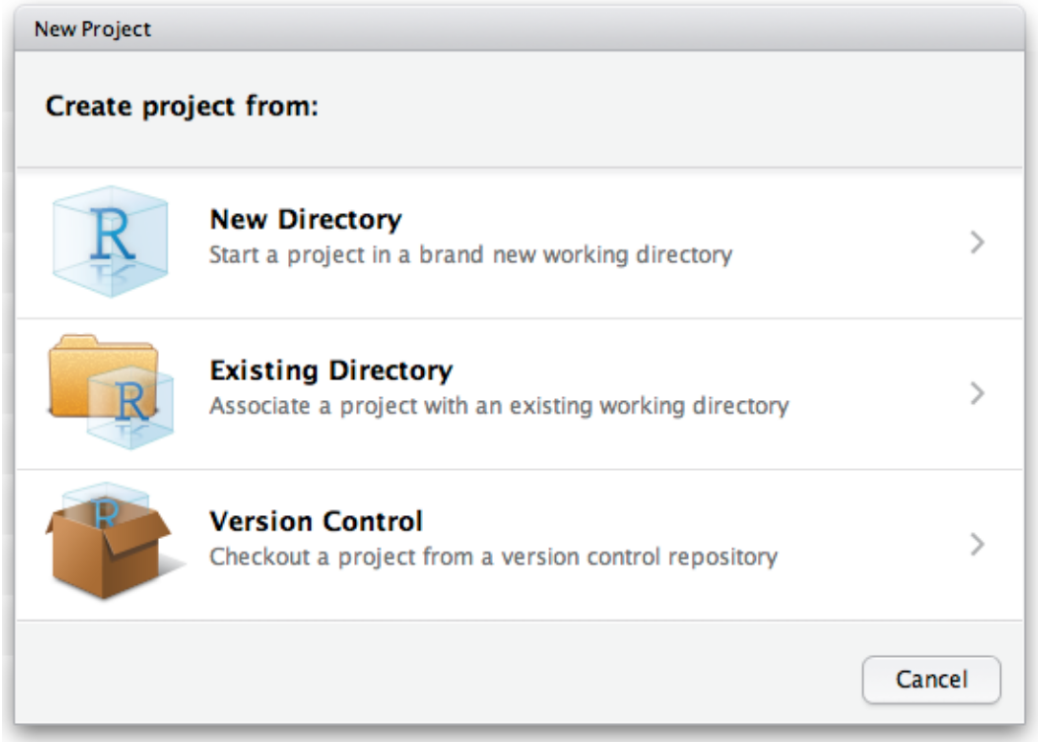
- Projects make easy to keep your work organized.
- All files, scripts, and documentation related to a specific project are bound together with a .Rproj file

Encourages reproducible code!

Creating a new project

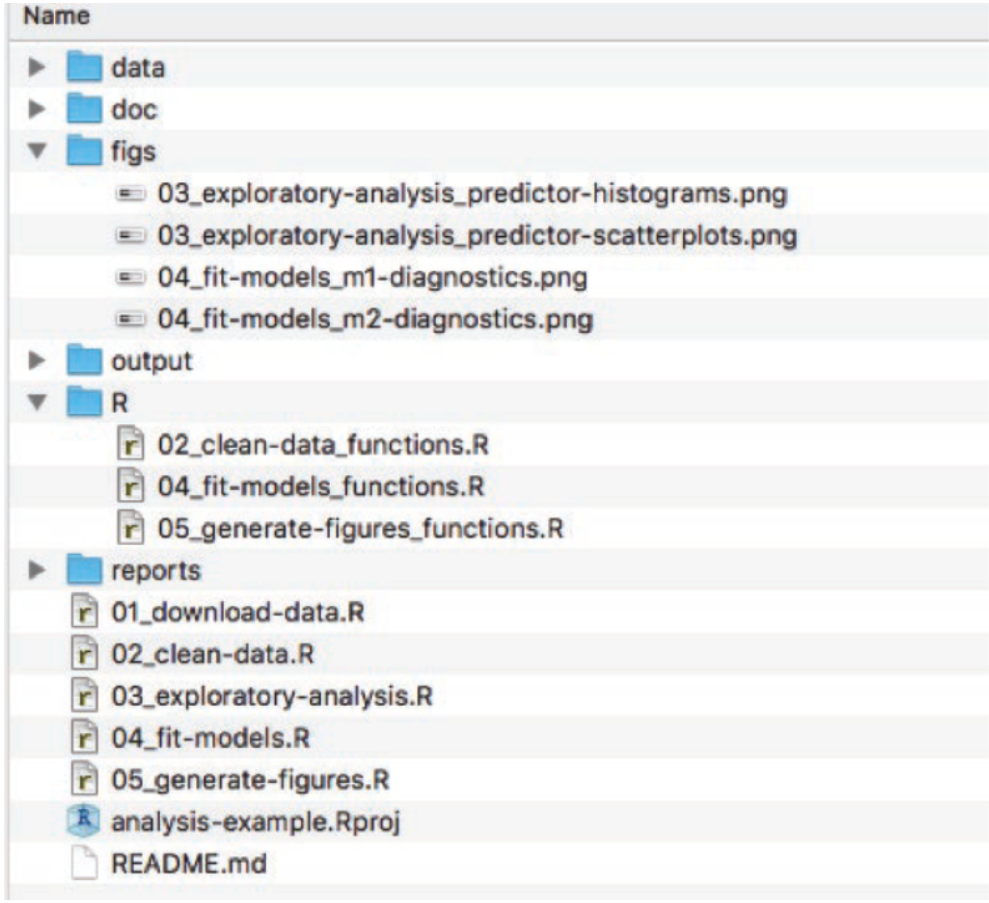
Use the **Create project** command

(available in the Projects menu and the global toolbar)



One project = one folder

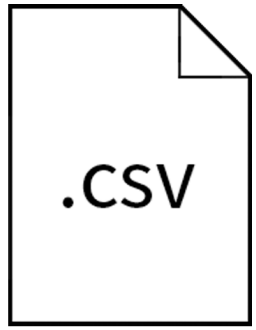
Keep your files organized!



Preparing data for R

Dataset should be stored as comma-separated value files (.csv) in the working directory

- Can be created from almost all applications (Excel, LibreOffice, Google Docs)
- File -> Save as **.csv**...



Naming files

Avoid using spaces, accents or special characters for your names.

Wise

- rawDatasetAgo2017.csv
- co2_concentrations_QB.csv
- 01_figIntro.R

not-wise

- final.csv
- safnnejs.csv
- 1-4.csv

Naming variables

Avoid using spaces, accents or special characters for your names Use separators (e.g. "_") to add details

Wise

- Measurements
- speciesNames
- Site_001_Rep_002

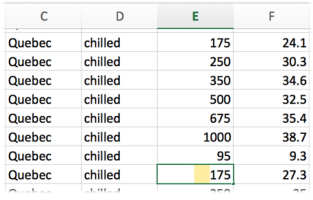
not-wise

- a
- 3
- complicatedverylongname

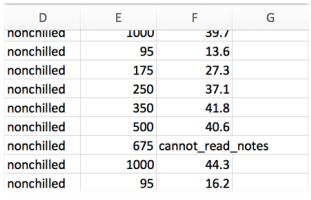
Common data preparation mistakes

- text in numeric columns
- typos
- numeric names for non-numeric data
- inconsistent formats for dates, numbers, metrics, etc.
- incorrect headings
- merged cells

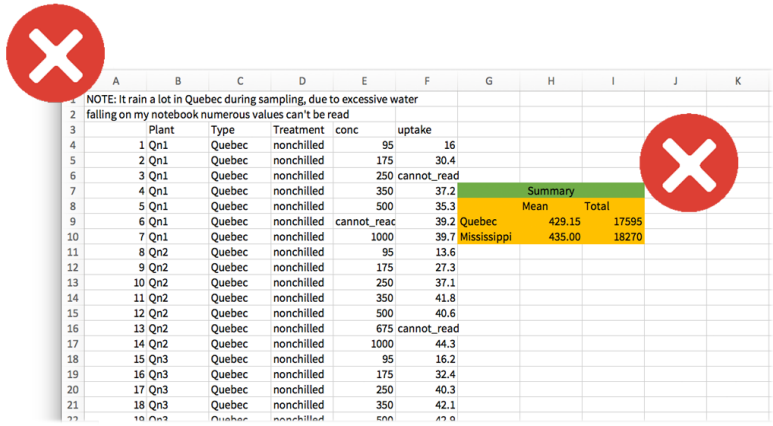
Bad data sheets:



C	D	E	F
Quebec	chilled	175	24.1
Quebec	chilled	250	30.3
Quebec	chilled	350	34.6
Quebec	chilled	500	32.5
Quebec	chilled	675	35.4
Quebec	chilled	1000	38.7
Quebec	chilled	95	9.3
Quebec	chilled	175	27.3



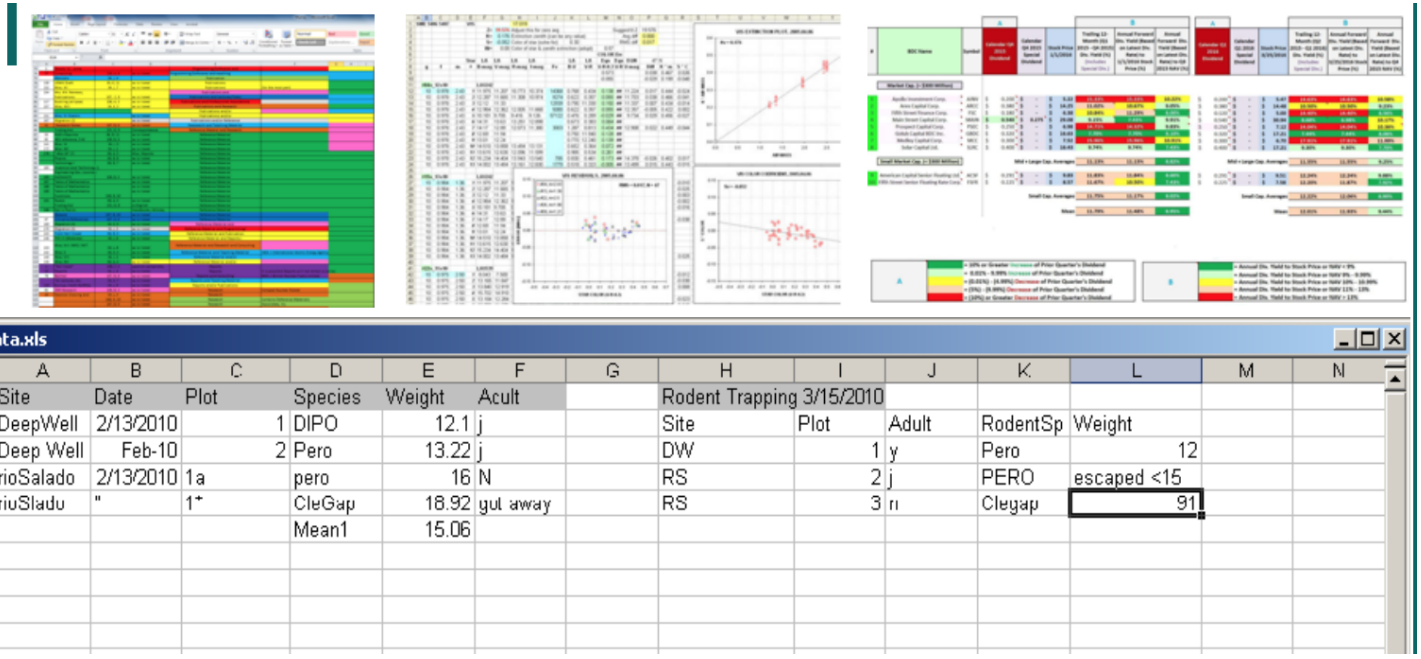
D	E	F	G
nonchilled	1000	39.7	
nonchilled	95	13.6	
nonchilled	175	27.3	
nonchilled	250	37.1	
nonchilled	350	41.8	
nonchilled	500	40.6	
nonchilled	675	cannot_read_notes	
nonchilled	1000	44.3	
nonchilled	95	16.2	



	A	B	C	D	E	F	G	H	I	J	K
1	NOTE: It rain a lot in Quebec during sampling, due to excessive water										
2	falling on my notebook numerous values can't be read										
3		Plant	Type	Treatment	conc	uptake					
4	1	Qn1	Quebec	nonchilled	95	16					
5	2	Qn1	Quebec	nonchilled	175	30.4					
6	3	Qn1	Quebec	nonchilled	250	cannot_read					
7	4	Qn1	Quebec	nonchilled	350	37.2		Summary			
8	5	Qn1	Quebec	nonchilled	500	35.3		Mean	Total		
9	6	Qn1	Quebec	nonchilled	cannot_reac	39.2	Quebec	429.15	17595		
10	7	Qn1	Quebec	nonchilled	1000	39.7	Mississippi	435.00	18270		
11	8	Qn2	Quebec	nonchilled	95	13.6					
12	9	Qn2	Quebec	nonchilled	175	27.3					
13	10	Qn2	Quebec	nonchilled	250	37.1					
14	11	Qn2	Quebec	nonchilled	350	41.8					
15	12	Qn2	Quebec	nonchilled	500	40.6					
16	13	Qn2	Quebec	nonchilled	675	cannot_read					
17	14	Qn2	Quebec	nonchilled	1000	44.3					
18	15	Qn3	Quebec	nonchilled	95	16.2					
19	16	Qn3	Quebec	nonchilled	175	32.4					
20	17	Qn3	Quebec	nonchilled	250	40.3					
21	18	Qn3	Quebec	nonchilled	350	42.1					

Horrible data sheets:

Not the best practices for data preservation and interoperability



data.xls

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Site	Date	Plot	Species	Weight	Acult		Rodent Trapping 3/15/2010						
2	DeepWell	2/13/2010	1	DIPO	12.1	j		Site	Plot	Adult	RodentSp	Weight		
3	Deep Well	Feb-10	2	Pero	13.22	j		DW	1	y	Pero	12		
4	rioSalado	2/13/2010	1a	pero	16	N		RS	2	j	PERO	escaped <15		
5	rioSlado	"	1*	CleGap	18.92	gut away		RS	3	n	Clegap	91		
6				Mean1	15.06									
7														
8														
9														
10														

Preparing data in R

It is possible to do all your data preparation work within R

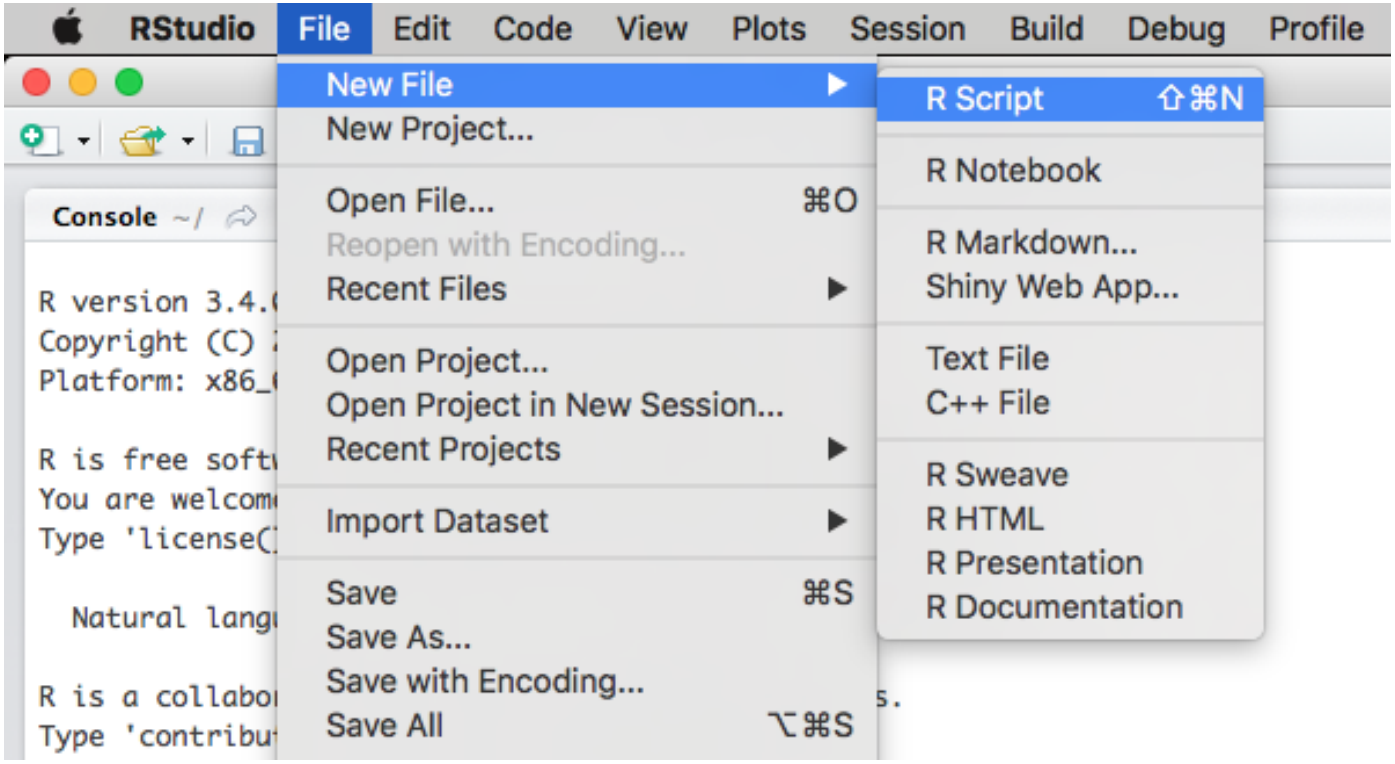
- Saves time for large datasets
- Keeps original data intact
- Keeps track of the manipulation and transformation you did
- Can switch between long- and wide-formats easily (more on this in future workshops)
- Useful tips on data preparation can be found here: https://www.zoology.ubc.ca/~schluter/R/data/

Writing a Script

R Scripts

- What is a script?
  - A text file storing the commands used for a particular analysis
- Once written and saved, your script file allows you to make changes and re-run analyses with minimal effort!
  - Just highlight text and click "Run" or press Command + Enter (Apple) or Ctrl + Enter (Windows and Linux)

Creating a R script



Creating a R script

Commands and comments

The `#` symbol in a script tells R to ignore anything remaining on this line of the script when running commands

```
# This is a comment not a command
```

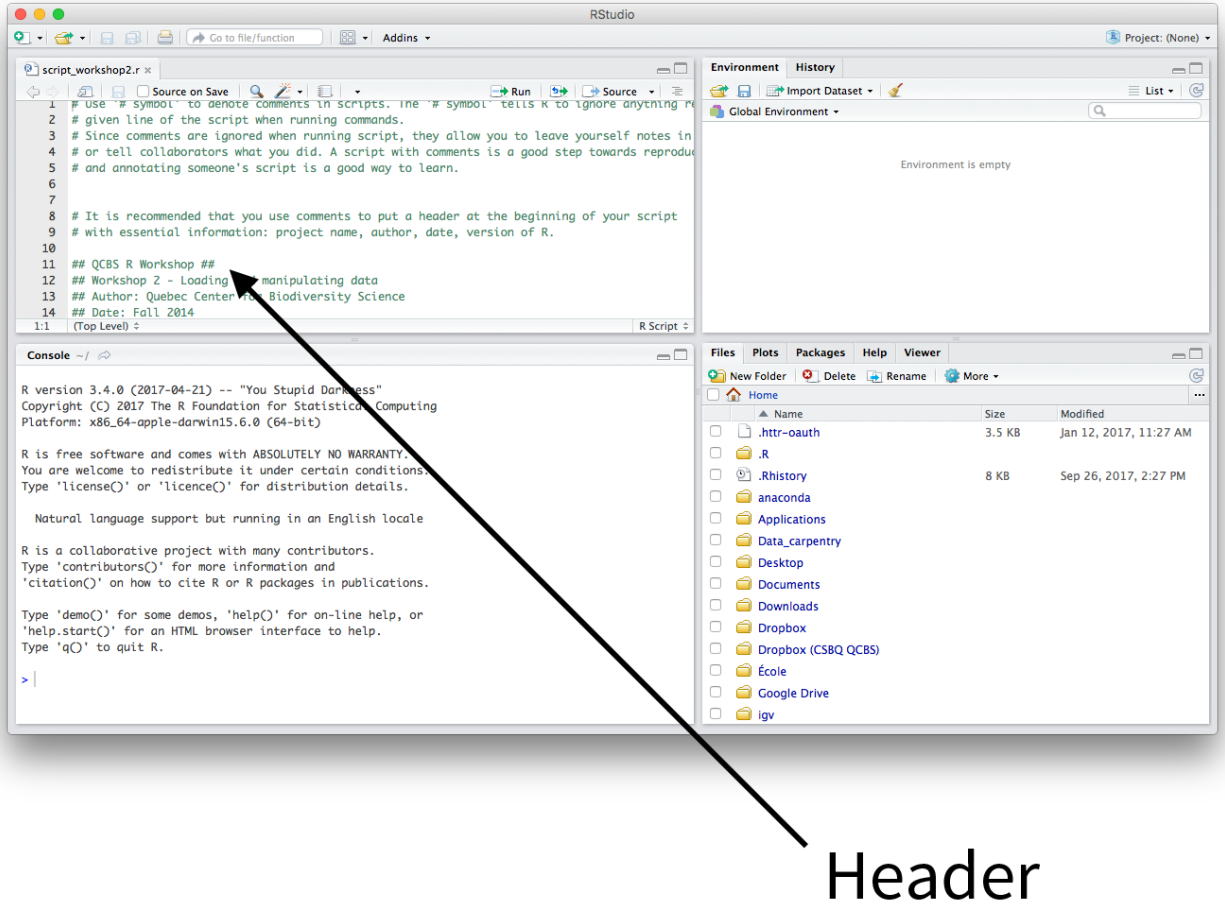
Why should I use #?

- Annotating someone's script is a good way to learn
- Tell collaborators what you did
- Good step towards reproducible science

Be as detailed as possible!

Header

It is recommended that you start your script with a header using comments:



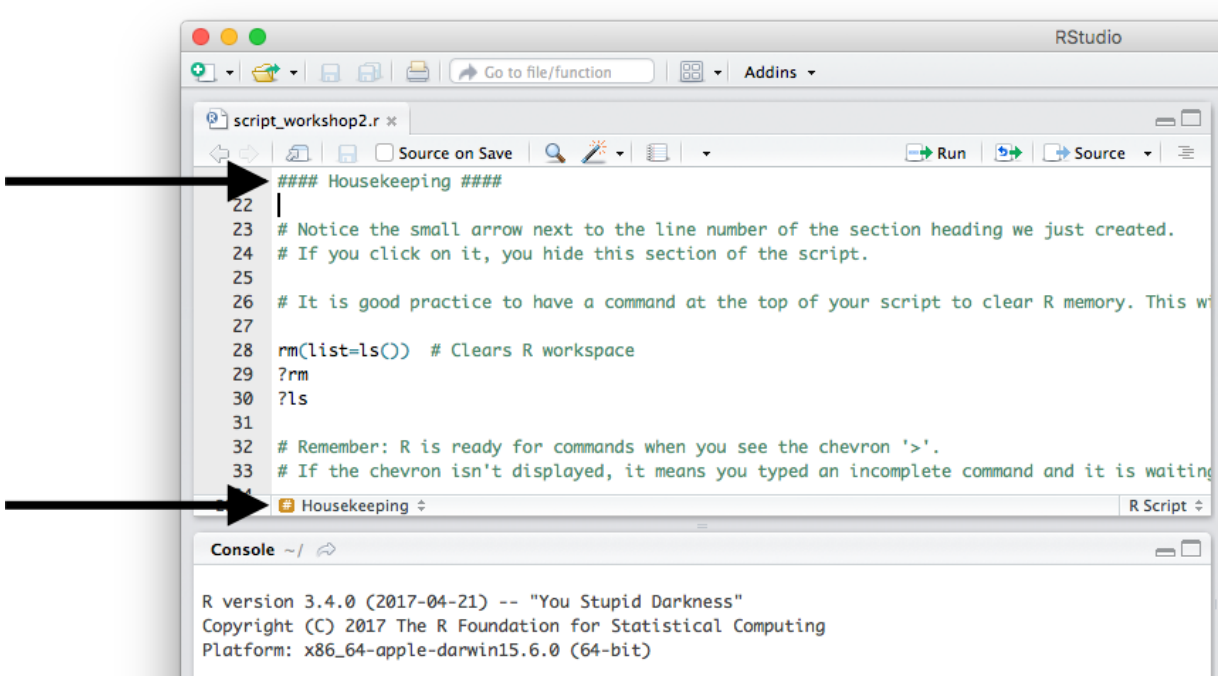
Section Headings

You can make a section heading in R Studio four `#` signs

```
# You can comment using this, but look below how to create section header

## Heading name ####
```

This allows you to move quickly between sections and hide sections



Housekeeping

The first command at the top of all scripts may be: `rm(list = ls())`. This command:

- Clears R memory
- Prevents errors related to the use of older data

Clearing the workspace

```
# Clear the R workspace
rm(list = ls())
?rm
?ls
```

Housekeeping

Demo – Try to add some test data to R and then see how `rm(list = ls())` removes it

```
A←"Test" # Put some data in workspace
A ← "Test" # Note that you can use spaces!
A = "Test" # ← or = can be used equally

#Note that it is best practice to use "←" for assigment instead of "="

# Check objects in the workspace
ls()
# [1] "A"

A
# [1] "Test"

# clean workspace
rm(list=ls())

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

Remember

- R is ready for commands when you see the chevron `>` in the console. If you don't see it, press ESC
- R is case sensitive!

Loading, exploring and saving data

Download today's data

You can download the data and the script for this workshop from the wiki:

http://qcbs.ca/wiki/r/workshop2

Save the files in the folder where your created your R project.

NOTE Many databases are already available on R

```
# Complete list of available data on base R
library(help = "datasets")
```

Working Directory

Tells R where your scripts and data are. You need to set the right working directory to load a data file. Type `getwd()` in the console to see your working directory:

```
getwd()
```

Display contents of the directory

You can display contents of the working directory using `dir()`:

```
dir()
# [1] "data"                "images"              "qcbsR-fonts.css"
# [4] "qcbsR-header.html"   "qcbsR-macros.js"     "qcbsR.css"
# [7] "workshop02-en.html" "workshop02-en.Rmd"
```

It helps to:

- Check that the file you plan to open is present in the folder that R is currently working in
- Check for correct spelling (e.g. "co2_good.csv" instead of "CO2_good.csv")

Importing Data

Import data into R using `read.csv`:

```
CO2 ← read.csv("data/co2_good.csv", header=TRUE)
```

Note that this will:

- Create a new object in R called `CO2`;
- The file name is written within quotation marks ('file' or "file");
- If you needed to fetch the file name from another directory that is not yours, you would have to write the full extension (e.g., `"C:/Users/Mario/Downloads/co2_good.csv"`)
- `header=TRUE` tells R that the first line of your dataset contains column names

Importing Data

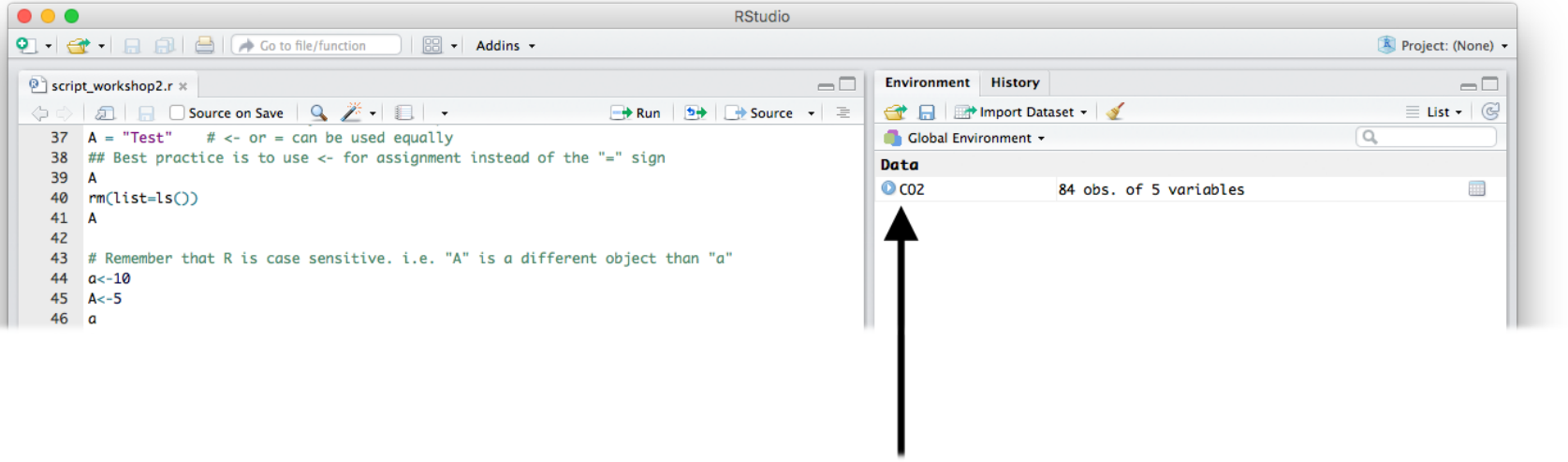
Recall to find out what arguments the function requires, use help "?" I

```
?read.csv
```

Note that if your operating system or .CSV editor (e.g. Excel) is in French, you may have to use `read.csv2`

```
?read.csv2
```

Importing Data



Notice that RStudio now provides information on the CO2 data in your **workspace**.

The **workspace** refers to all the objects that you create during an R session.

Inspecting the data

R Command	Action
`CO2`	look at the whole dataframe
`view(CO2)`	look data as a spreadsheet
`head(CO2)`	look at the first few rows
`tail(CO2)`	look at the last few rows
`names(CO2)`	names of the columns in the dataframe
`attributes(CO2)`	attributes of the dataframe
`dim(CO2)`	dimensions of the dataframe
`ncol(CO2)`	number of columns
`nrow(CO2)`	number of rows
`summary(CO2)`	basic statistics for each variables

Inspecting the data

```
str(CO2)
# 'data.frame':    84 obs. of  5 variables:
#  $ Plant    : Factor w/ 12 levels "Mc1","Mc2","Mc3",..: 10 10 10 10 10
#  $ Type     : Factor w/ 2 levels "Mississippi",..: 2 2 2 2 2 2 2 2 2 2
#  $ Treatment: Factor w/ 2 levels "chilled","nonchilled": 2 2 2 2 2 2 2
#  $ conc     : int  95 175 250 350 500 675 1000 95 175 250 ...
#  $ uptake   : num  16 30.4 34.8 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

This shows the structure of the dataframe. Very useful to check data type (mode) of all columns to make sure R loaded data properly.

Note: the CO2 dataset includes repeated measurements of CO2 uptake from 6 plants from Quebec and 6 plants from Mississippi at several levels of CO2 concentration. Half the plants of each type were chilled overnight before the experiment was conducted.

Inspecting the data

Common problems:

- Factors loaded as text (character) or viceversa;
- Factors includes too many (and unexpected) levels because of typos
- Data (integer or numeric) is loaded as character because of typos (*e.g.* a space or a "," instead of a "." to separate decimal numbers)

Working with datasets in R

Exercise:

Load the data with:

```
CO2 ← read.csv("data/co2_good.csv", header = FALSE)
```

Check data types with `str()` again.

What is wrong here?

Do not forget to re-load data with `header = T` afterwards

Reminder from Workshop 1: Accessing data

Imagine a data frame called `mydata`:

Variable1	Variable2	Variable3	Variable4

```
mydata[1,] # Extracts the first row
mydata[2,3] # Extracts the content of row 2 / column 3
mydata[,1] # Extracts the first column
mydata[,1][2] # [...] can be also be used recursively
mydata$Variable1 # Also extracts the first column
```

Renaming variables

Variables names can be changed within R.

```
# First lets make a copy of the dataset to play with
CO2copy ← CO2
# names() gives you the names of the variables present in the data frame
names(CO2copy)
# [1] "Plant"     "Type"      "Treatment" "conc"      "uptake"

# Changing from English to French names (make sure you have the same leve
names(CO2copy) ← c("Plante","Categorie", "Traitement", "conc", "absortio
names(CO2copy)
# [1] "Plante"    "Categorie" "Traitement" "conc"      "absortion"
```

Creating new variables

Variables and strings can be concatenated together. The function `paste()` is very useful for concatenating. See `?paste` and `?paste0`.

```
# Let's create an unique id for our samples:
# Don't forget to use "" for strings
CO2copy$uniqueID ← paste0(CO2copy$Plante,
                          "_",CO2copy$Categorie,
                          "_", CO2copy$Traitement)

# observe the results
head(CO2copy$uniqueID)
# [1] "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchil
# [4] "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchilled" "Qn1_Quebec_nonchil
```

Creating new variables

Creating new variables works for numbers and mathematical operations as well!

```
# Let's standardize our variable "absortion" to relative values
CO2copy$absortionRel ← CO2copy$absortion/max(CO2copy$absortion)

# Observe the results
head(CO2copy$absortionRel)
```

Subsetting data

There are many ways to subset a data frame

```
# Let's keep working with our CO2copy data frame

# Select only "Plante" and "absortionRel" columns. (Don't forget the ","!
CO2copy[,c("Plante", "absortionRel")]

# Subset data frame from rows from 1 to 50
CO2copy[1:50,]
```

Subsetting data

```r
# Select observations matching only the nonchilled Traitement.
CO2copy[CO2copy$Traitement == "nonchilled",]

# Select observations with absortion higher or equal to 20
CO2copy[CO2copy$absortion ⩾ 20, ]

# Select observations with absortion higher or equal to 20
CO2copy[CO2copy$Traitement  == "nonchilled" & CO2copy$absortion ⩾ 20, ]

# We are done playing with the Dataset copy, lets erase it.
CO2copy ← NULL
```

Go here to check all the logical operators you can use

Data exploration

A good way to start your data exploration is to look at some basic statistics of your dataset.

Use the `summary()` function to do that!

```
summary(CO2)
```

This is also useful to spot some errors you might have missed!

Data exploration

You can also use other functions to calculate basic statistics on parts of your data frame. Let's try the `mean()`, `sd()` and `hist()` functions:

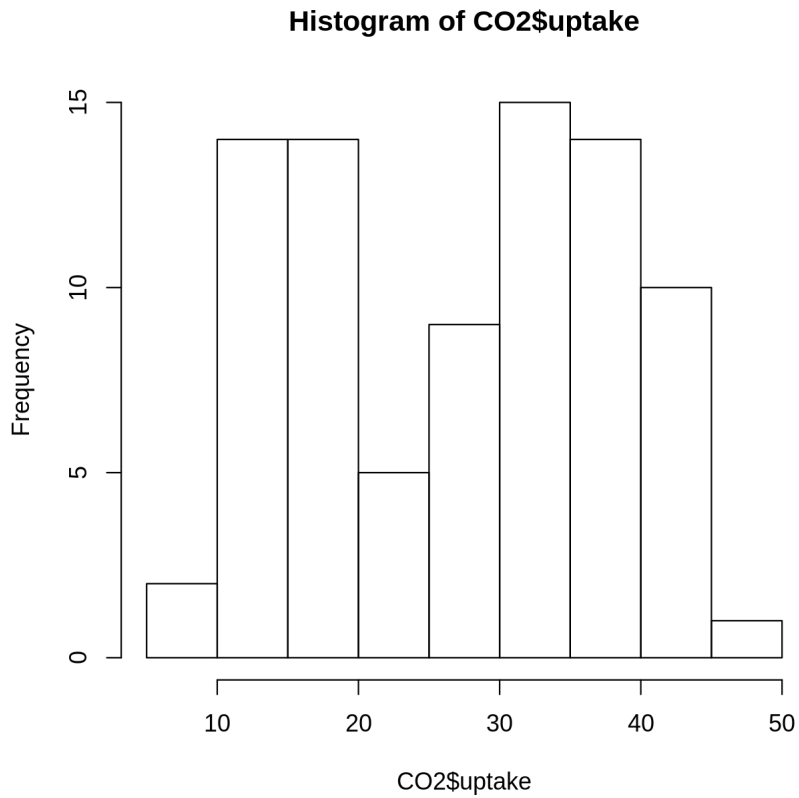
```
# Calculate the mean and the standard deviation of the CO2 concentration:
# Assign them to new variables
meanConc ← mean(CO2$conc)
sdConc ← sd(CO2$conc)
```

```
# print() prints any given value to the R console
print(paste("the mean of concentration is:", meanConc))
# [1] "the mean of concentration is: 435"

print(paste("the standard deviation of concentration is:", sdConc))
# [1] "the standard deviation of concentration is: 295.924119222056"
```

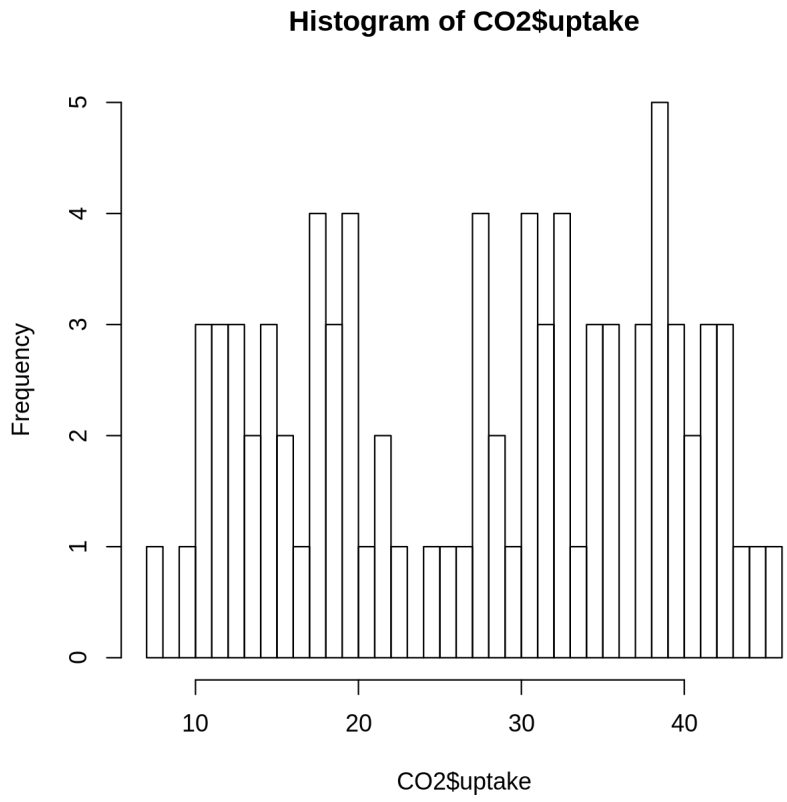
Data exploration

```
# Let's plot a histogram to explore the distribution of "uptake"
hist(CO2$uptake)
```



Data exploration

```
# Increasing the number of bins to observe better the pattern
hist(CO2$uptake, breaks = 40)
```



Saving your Workspace

```r
# Saving an R workspace file that stores all your objects
save.image(file="data/co2_project_Data.RData")


# Clear your memory
rm(list = ls())


# Reload your data
load("data/co2_project_Data.RData")
head(CO2) # Looking good!
```

```
#   Plant   Type  Treatment conc uptake
# 1   Qn1 Quebec nonchilled   95   16.0
# 2   Qn1 Quebec nonchilled  175   30.4
# 3   Qn1 Quebec nonchilled  250   34.8
# 4   Qn1 Quebec nonchilled  350   37.2
# 5   Qn1 Quebec nonchilled  500   35.3
# 6   Qn1 Quebec nonchilled  675   39.2
```

Exporting data

R disposes of `write` functions that allow you to write objects directly to files in your computer. Let us use the `write.csv` function to save our CO2 data into a .CSV file:

```
write.csv(CO2, file = "data/co2_new.csv")
```

Note that our arguments are both:

`CO2` ➔ Object (name)

`"co2_new.csv"` ➔ File to write (name)

Challenge: Use your data

- Try to load, explore, plot, and save your own data in R;
- If it does not load properly, try to make the appropriate changes;
- When you are finished, try opening your exported data in Excel, Numbers, Notepad or TextEdit.

If you don't have your own data, work with your neighbor

Remember to clean your workspace

Database repair

Database repair

Getting your data working properly can be tougher than you think!

For example, sharing data from an Apple computer to Windows, or between computers set up in different continents can lead to incompatible files (e.g. different decimal separators).

Let's practice how to solve these common errors.

Database repair

Exercise:

Read the file `co2_broken.csv`

```
CO2 ← read.csv("data/co2_broken.csv")
head(CO2)
#                NOTE..It.rain.a.lot.in.Quebec.during.sampling due.to.exce
# 1 falling on my notebook numerous values can't be read rain
# 2                       Plant\tType\tTreatment\tconc\tuptake
# 3                           Qn1\tQuebec\tnonchilled\t95\t16
# 4                         Qn1\tQuebec\tnonchilled\t175\t30.4
# 5             Qn1\tQuebec\tnonchilled\t250\tcannot_read_notes
# 6                         Qn1\tQuebec\tnonchilled\t350\t37.2
#   X.1 X.2 X.3
# 1  NA  NA  NA
# 2  NA  NA  NA
# 3  NA  NA  NA
# 4  NA  NA  NA
# 5  NA  NA  NA
# 6  NA  NA  NA
```

Database repair

Some useful functions:

- `head()` - first few rows
- `str()` - structure of data
- `class()` - class of the object
- `unique()` - unique observations
- `levels()` - levels of a factor
- `which()` - ask a question to your data frame
- `droplevels()` - get rid of undesired levels after sub setting factors

add "?" (e.g. `?read.csv` to the function name to access the function help page)

HINT There are 4 problems!

Database repair

ERROR 1 The data appears to be lumped into one column

```
head(CO2)
#                NOTE..It.rain.a.lot.in.Quebec.during.sampling due.to.exce
# 1 falling on my notebook numerous values can't be read rain
# 2                       Plant\tType\tTreatment\tconc\tuptake
# 3                           Qn1\tQuebec\tnonchilled\t95\t16
# 4                         Qn1\tQuebec\tnonchilled\t175\t30.4
# 5             Qn1\tQuebec\tnonchilled\t250\tcannot_read_notes
# 6                         Qn1\tQuebec\tnonchilled\t350\t37.2
#   X.1 X.2 X.3
# 1  NA  NA  NA
# 2  NA  NA  NA
# 3  NA  NA  NA
# 4  NA  NA  NA
# 5  NA  NA  NA
# 6  NA  NA  NA
```

Database repair

ERROR 1 - Solution

- Re-import the data, but specify the separation among entries
- The sep argument tells R what character separates the values on each line of the file
- Here, "TAB" was used instead of ","

```
CO2 ← read.csv("data/co2_broken.csv",sep = "")
```

Database repair

ERROR 2 The data does not start until the third line of the file, so you end up with notes on the file as the headings.

```
head(CO2)
#     NOTE.      It        rain         a                lot    in. Quebec du
# 1 falling      on          my notebook          numerous values  can't
# 2   Plant    Type  Treatment     conc            uptake
# 3     Qn1 Quebec nonchilled       95              16
# 4     Qn1 Quebec nonchilled      175            30.4
# 5     Qn1 Quebec nonchilled      250 cannot_read_notes
# 6     Qn1 Quebec nonchilled      350            37.2
#   sampling.       due to excessive X....
# 1      read rain,,,, NA        NA    NA
# 2                    NA        NA    NA
# 3                    NA        NA    NA
# 4                    NA        NA    NA
# 5                    NA        NA    NA
# 6                    NA        NA    NA
```

Database repair

ERROR 2 - Solution

Skip two lines when loading the file using the "skip" argument:

```
CO2 ← read.csv("data/co2_broken.csv", sep = "", skip = 2)
head(CO2)
#   Plant   Type  Treatment                conc            uptake
# 1   Qn1 Quebec nonchilled                  95              16
# 2   Qn1 Quebec nonchilled                 175            30.4
# 3   Qn1 Quebec nonchilled                 250 cannot_read_notes
# 4   Qn1 Quebec nonchilled                 350            37.2
# 5   Qn1 Quebec nonchilled                 500            35.3
# 6   Qn1 Quebec nonchilled cannot_read_notes            39.2
```

Database repair

ERROR 3 `conc` and `uptake` variables are considered factors instead of numbers, because there are comments in the numeric columns

```
str(CO2)
# 'data.frame':    84 obs. of  5 variables:
#  $ Plant    : Factor w/ 12 levels "Mc1","Mc2","Mc3",..: 10 10 10 10 10
#  $ Type     : Factor w/ 2 levels "Mississippi",..: 2 2 2 2 2 2 2 2 2 2
#  $ Treatment: Factor w/ 4 levels "chiled","chilled",..: 4 4 4 4 4 4 4 4
#  $ conc     : Factor w/ 8 levels "1000","175","250",..: 7 2 3 4 5 8 1 7
#  $ uptake   : Factor w/ 77 levels "10.5","10.6",..: 15 39 76 54 50 61 6
```

```
unique(CO2$conc)
# [1] 95                  175                 250                 350
# [5] 500                 cannot_read_notes 1000                675
# Levels: 1000 175 250 350 500 675 95 cannot_read_notes
```

- Due to missing values entered as "cannot_read_notes" and "na"
- Recall that R only recognizes "NA" (capital)

?read.csv

read.table {utils} R Documentation

Data Input

Description

Reads a file in table format and creates a data frame from it, with cases corresponding to lines and variables to fields in the file.

Usage

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

`na.strings` a character vector of strings which are to be interpreted as NA values. Blank fields are also considered to be missing values in logical, integer, numeric and complex fields. Note that the test happens *after* white space is stripped from the input, so `na.strings` values may need their own white space stripped in advance.

Database repair

ERROR 3 - Solution

Tell R that all of NA, "na", and "cannot_read_notes" should be considered NA. Then because all other values in those columns are numbers, `conc` and `uptake` will be loaded as numeric/integer.

```
CO2 ← read.csv("data/co2_broken.csv", sep = "", skip = 2,
               na.strings = c("NA","na","cannot_read_notes"))
str(CO2)
# 'data.frame':    84 obs. of  5 variables:
#  $ Plant    : Factor w/ 12 levels "Mc1","Mc2","Mc3",..: 10 10 10 10 10
#  $ Type     : Factor w/ 2 levels "Mississippi",..: 2 2 2 2 2 2 2 2 2 2
#  $ Treatment: Factor w/ 4 levels "chiled","chilled",..: 4 4 4 4 4 4 4 4
#  $ conc     : int  95 175 250 350 500 NA 1000 95 175 250 ...
#  $ uptake   : num  16 30.4 NA 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

Database repair

ERROR 4

There are only 2 treatments (chilled and non-chilled) but there are spelling errors causing it to look like 4 different treatments.

```
str(CO2)
```

```
levels(CO2$Treatment)
# [1] "chiled"     "chilled"    "nnchilled"  "nonchilled"
unique(CO2$Treatment)
# [1] nonchilled nnchilled  chilled    chiled
# Levels: chiled chilled nnchilled nonchilled
```

Database repair

ERROR 4 - Solution

```
# Identify all rows that contain "nnchilled" and replace with "nonchilled
CO2$Treatment[CO2$Treatment=="nnchilled"] ← "nonchilled"

# Identify all rows that contain "chiled" and replace with "chilled"
CO2$Treatment[CO2$Treatment=="chiled"] ← "chilled"
```

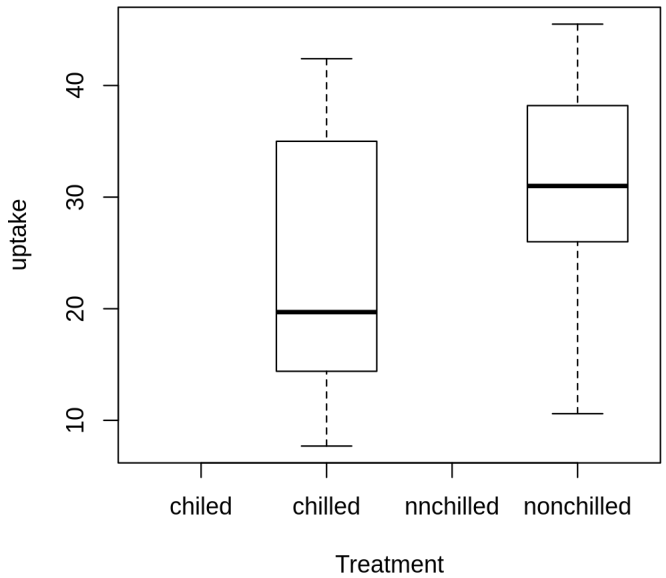
Database repair

ERROR 4 - Solution

After repairing the factors, we need to remove the unused factor levels

If not :

```
boxplot(uptake ~ Treatment, data = CO2)
```

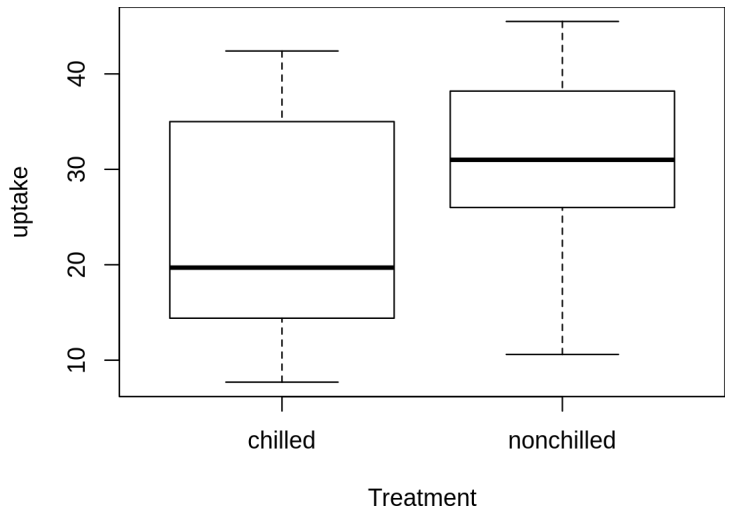


Database repair

ERROR 4 - Solution

```
CO2 ← droplevels(CO2)
str(CO2)
# 'data.frame':    84 obs. of  5 variables:
#  $ Plant    : Factor w/ 12 levels "Mc1","Mc2","Mc3",..: 10 10 10 10 10
#  $ Type     : Factor w/ 2 levels "Mississippi",..: 2 2 2 2 2 2 2 2 2 2
#  $ Treatment: Factor w/ 2 levels "chilled","nonchilled": 2 2 2 2 2 2 2
#  $ conc     : int  95 175 250 350 500 NA 1000 95 175 250 ...
#  $ uptake   : num  16 30.4 NA 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
```

```
boxplot(uptake ~ Treatment, data
```



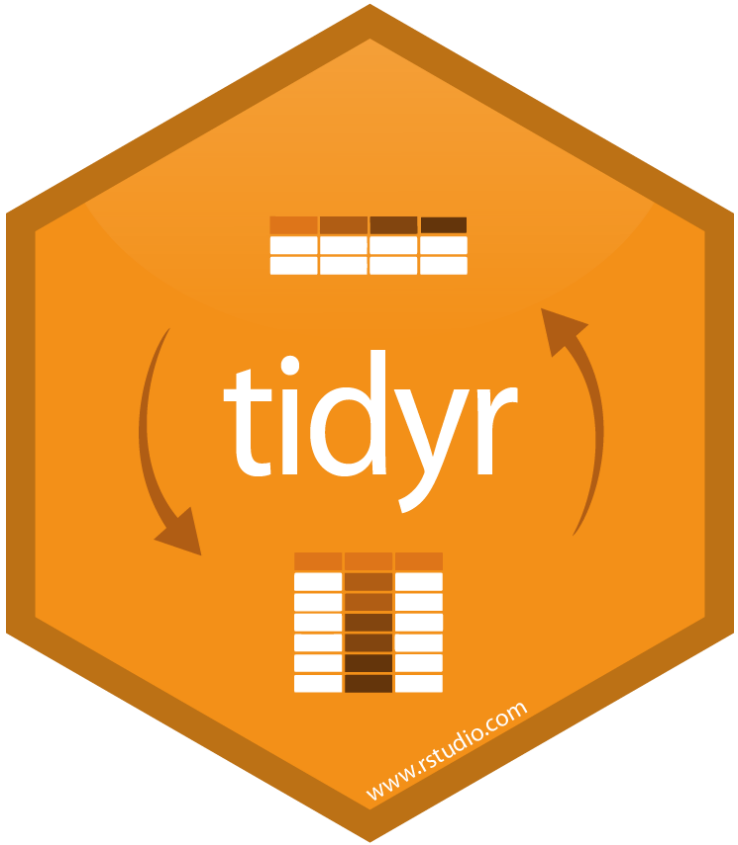
Advanced users section

Learn to manipulate data with

tidyr, dyplr, maggritr

Using `tidyr` to reshape data frames

```
library(tidyr)
```



Data formats

Wide format

```
#   Species DBH Height
# 1     Oak  12     56
# 2     Elm  20     85
# 3     Ash  13     55
```

Long format

```
#   Species Measurement Value
# 1     Oak         DBH    12
# 2     Elm         DBH    20
# 3     Ash         DBH    13
# 4     Oak      Height    56
# 5     Elm      Height    85
# 6     Ash      Height    55
```

long vs wide format

Wide data format has a separate column for each variable or each factor in your study

Long data format has a column for possible variables and a column for the values of those variables

---

Wide data frame can be used for some basic plotting in `ggplot2`, but more complex plots require long format (example to come)

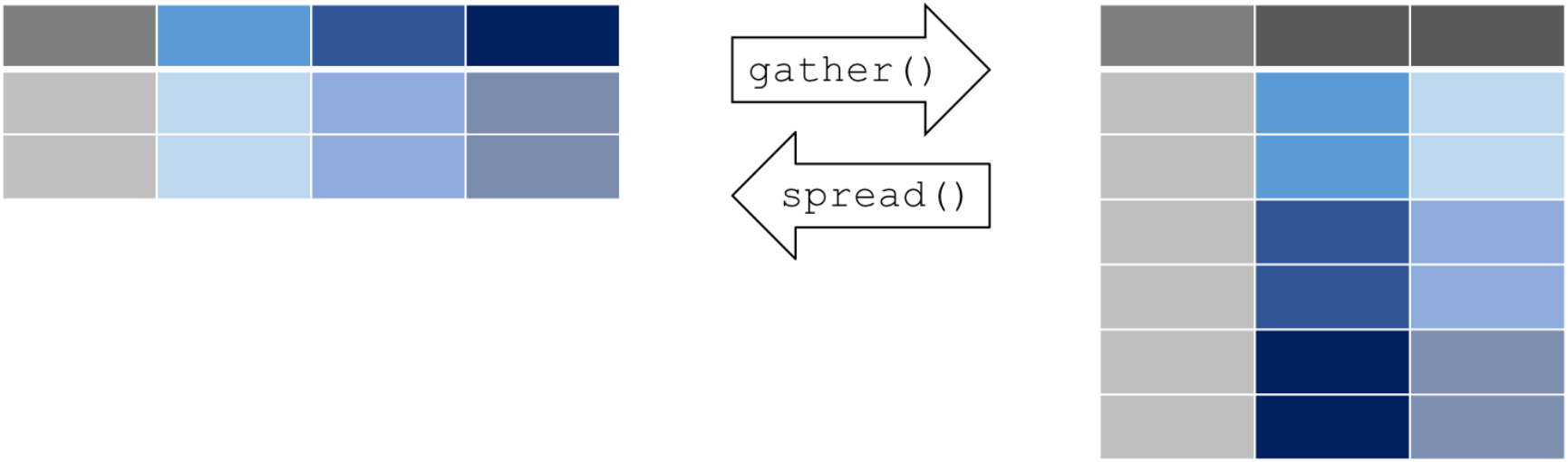
`dplyr`, `lm()`, `glm()`, `gam()` all require long data format

Tidying your data

Tidying allows you to manipulate the structure of your data while preserving all original information

`gather()` - convert from wide to long format

`spread()` - convert from long to wide format



`tidyr` installation

```
install.packages("tidyr")
library(tidyr)
```

`gather` columns into rows

```
gather(data, key, value, ...)
```

- `data` A data frame (e.g. `wide`)
- `key` name of the new column containing variable names (e.g. `Measurement`)
- `value` name of the new column containing variable values (e.g. `Value`)
- `...` name or numeric index of the columns we wish to gather (e.g. `DBH`, `Height`)

`gather` columns into rows

```
wide ← data.frame(Species = c("Oak", "Elm", "Ash"),
                  DBH = c(12, 20, 13), Height = c(56, 85, 55))
wide
#   Species DBH Height
# 1     Oak  12     56
# 2     Elm  20     85
# 3     Ash  13     55
```

```
long = tidyr::gather(wide, Measurement, Value, DBH, Height)
long
#   Species Measurement Value
# 1     Oak         DBH    12
# 2     Elm         DBH    20
# 3     Ash         DBH    13
# 4     Oak      Height    56
# 5     Elm      Height    85
# 6     Ash      Height    55
```

`spread` rows into columns

```
spread(data, key, value)
```

- `data` A data frame (e.g. `long`)
- `key` Name of the column containing variable names (e.g. `Measurement`)
- `value` Name of the column containing variable values (e.g. `Value`)

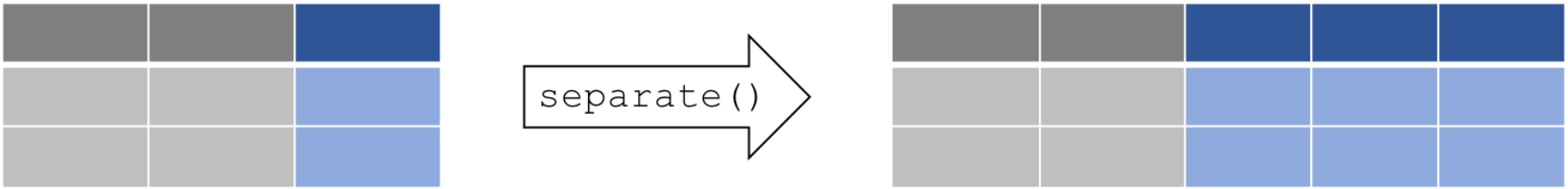
`spread` rows into columns

```
long
#   Species Measurement Value
# 1     Oak         DBH    12
# 2     Elm         DBH    20
# 3     Ash         DBH    13
# 4     Oak      Height    56
# 5     Elm      Height    85
# 6     Ash      Height    55
```

```
wide2 = tidyr::spread(long, Measurement, Value)
wide2
#   Species DBH Height
# 1     Ash  13     55
# 2     Elm  20     85
# 3     Oak  12     56
```

`separate` columns

`separate()` splits a columns by a character string separator



`separate(data, col, into, sep)`

- `data` A data frame (e.g. `long`)
- `col` Name of the column you wish to separate
- `into` Names of new variables to create
- `sep` Character which indicates where to separate

Using `separate()` example

Create a fictional dataset about fish and plankton

```
set.seed(8)
messy ← data.frame(id = 1:4,
                   trt = sample(rep(c('control', 'farm'), each = 2)),
                   zooplankton.T1 = runif(4),
                   fish.T1 = runif(4),
                   zooplankton.T2 = runif(4),
                   fish.T2 = runif(4))
messy
#   id     trt zooplankton.T1    fish.T1 zooplankton.T2   fish.T2
# 1  1    farm      0.7189275 0.64449114    0.544962116 0.2644589
# 2  2    farm      0.2908734 0.45704489    0.138224346 0.2765322
# 3  3 control      0.9322698 0.08930101    0.927812252 0.5211070
# 4  4 control      0.7691470 0.43239137    0.001301721 0.2236889
```

Using `separate()` example

First convert the messy data frame from wide to long format

```
messy.long ← tidyr::gather(messy, taxa, count, -id, -trt)
head(messy.long)
#   id     trt           taxa     count
# 1  1    farm zooplankton.T1 0.7189275
# 2  2    farm zooplankton.T1 0.2908734
# 3  3 control zooplankton.T1 0.9322698
# 4  4 control zooplankton.T1 0.7691470
# 5  1    farm        fish.T1 0.6444911
# 6  2    farm        fish.T1 0.4570449
```

Using `separate()` example

Then we want to split the 2 sampling time (T1 and T2).

```
messy.long.sep ← tidyr::separate(messy.long, taxa,
                                 into = c("species", "time"), sep = "\\.
head(messy.long.sep)
#   id     trt     species time     count
# 1  1    farm zooplankton   T1 0.7189275
# 2  2    farm zooplankton   T1 0.2908734
# 3  3 control zooplankton   T1 0.9322698
# 4  4 control zooplankton   T1 0.7691470
# 5  1    farm        fish   T1 0.6444911
# 6  2    farm        fish   T1 0.4570449
```

The argument `sep = "\\."` tells R to splits the character string around the period (.). We cannot type directly `"."` because it is a regular expression that matches any single character.

Recap of `tidyr`

A package that reshapes the layout of data sets.

Converting from wide to long format using `gather()`

Converting from long format to wide format using `spread()`

Split and merge columns with `unite()` and `separate()`

Data Wrangling with dplyr and tidyr Cheat Sheet

Challenge #1

Using the airquality dataset, `gather` all the columns (except Month and Day) into rows.

```
?airquality

data(airquality)
```

Solution #1

Using the airquality dataset, `gather` all the columns (except Month and Day) into rows.

```
air.long ← tidyr::gather(airquality, variable, value, -Month, -Day)
head(air.long)
#   Month Day variable value
# 1     5   1    Ozone    41
# 2     5   2    Ozone    36
# 3     5   3    Ozone    12
# 4     5   4    Ozone    18
# 5     5   5    Ozone    NA
# 6     5   6    Ozone    28
```

Note that the syntax used here indicates that we wish to gather ALL the columns except Month and Day. It is equivalent to: `gather(airquality, value, Ozone, Solar.R, Temp, Wind)`

Challenge #2

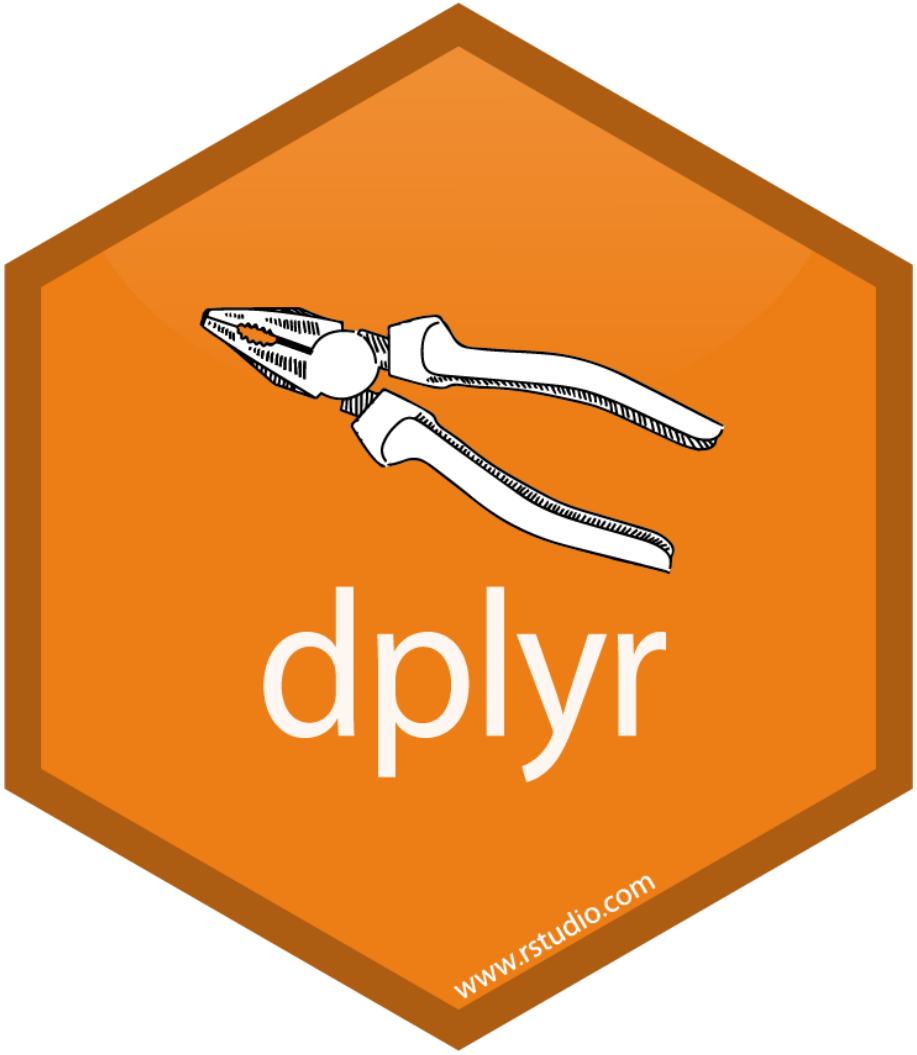
`spread` the resulting data frame to return to the original data format.

Solution #2

`spread` the resulting data frame to return to the original data format.

```
air.wide ← tidyr::spread(air.long, variable, value)
head(air.wide)
#   Month Day Ozone Solar.R Temp Wind
# 1     5   1    41     190   67  7.4
# 2     5   2    36     118   72  8.0
# 3     5   3    12     149   74 12.6
# 4     5   4    18     313   62 11.5
# 5     5   5    NA      NA   56 14.3
# 6     5   6    28      NA   66 14.9
```

Data manipulation with `dplyr`



Intro to `dplyr`

- Package that contains a set of functions (or “verbs”) for data manipulation such as filtering rows, selecting specific columns, re-ordering rows, adding new columns and summarizing data;
- Easy and intuitive functions
- Fast and efficient
- Can interface with external databases and translate your R code into SQL queries

Some corresponding R base functions: `split()`, `subset()`, `apply()`, `sapply()`, `lapply()`, `tapply()` and `aggregate()`

Intro to `dplyr`

```
library(dplyr)
```

Basic functions in `dplyr`

These 4 core functions tackle the most common manipulations when working with data frames

- `select()`: select columns from a data frame
- `filter()`: filter rows according to defined criteria
- `arrange()`: re-order data based on criteria (e.g. ascending, descending)
- `mutate()`: create or transform values in a column

`select` columns

`select(data, ...)`

- `...` Can be column names or positions or complex expressions separated by commas

Examples:

```
select(data, column1, column2) # select columns 1 and 2
select(data, c(2:4,6)) # select columns 2 to 4 and 6
select(data, -column1) # select all columns except column 1
select(data, start_with(x.)) # select all columns that start with "x."
```

`select` columns

Helper functions for select - ?select

`select(iris, contains("."))`

Select columns whose name contains a character string.

`select(iris, ends_with("Length"))`

Select columns whose name ends with a character string.

`select(iris, everything())`

Select every column.

`select(iris, matches(".t."))`

Select columns whose name matches a regular expression.

`select(iris, num_range("x", 1:5))`

Select columns named x1, x2, x3, x4, x5.

`select(iris, one_of(c("Species", "Genus")))`

Select columns whose names are in a group of names.

`select(iris, starts_with("Sepal"))`

Select columns whose name starts with a character string.

`select(iris, Sepal.Length:Petal.Width)`

Select all columns between Sepal.Length and Petal.Width (inclusive).

`select(iris, -Species)`

Select all columns except Species.

`select` columns

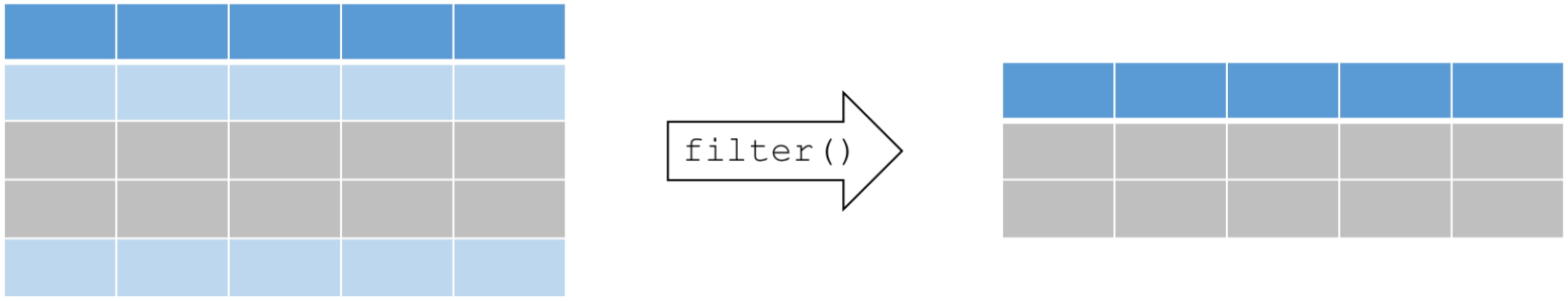
Example: suppose we are only interested in the variation of Ozone over time within the airquality dataset

```
ozone ← dplyr::select(airquality, Ozone, Month, Day)
head(ozone)
#   Ozone Month Day
# 1    41     5   1
# 2    36     5   2
# 3    12     5   3
# 4    18     5   4
# 5    NA     5   5
# 6    28     5   6
```

`filter` rows

Extract a subset of rows that meet one or more specific conditions

```
filter(dataframe, logical statement 1, logical statement 2, ... )
```



Logic in R - ?Comparison, ?base::Logic			
<	Less than	!=	Not equal to
>	Greater than	%in%	Group membership
==	Equal to	is.na	Is NA
<=	Less than or equal to	!is.na	Is not NA
>=	Greater than or equal to	&, |, !, xor, any, all	Boolean operators

`filter` rows

Example: we are interested in analyses that focus on the month of August during high temperature events

```
august ← dplyr::filter(airquality, Month == 8, Temp ⩾ 90)
# same as: filter(airquality, Month == 8 & Temp ⩾ 90)
head(august)
#   Ozone Solar.R Wind Temp Month Day
# 1    89     229 10.3   90     8   8
# 2   110     207  8.0   90     8   9
# 3    NA     222  8.6   92     8  10
# 4    76     203  9.7   97     8  28
# 5   118     225  2.3   94     8  29
# 6    84     237  6.3   96     8  30
```

Sort rows with `arrange`

Re-order rows by a particular column, by default in ascending order

Use `desc()` for descending order.

```
arrange(data, variable1, desc(variable2), ... )
```

Sort rows with `arrange`

Example:

1. Let's use the following code to create a scrambled version of the airquality dataset

```
air_mess ← dplyr::sample_frac(airquality, 1)
head(air_mess)
#   Ozone Solar.R Wind Temp Month Day
# 1    23     115  7.4   76     8  18
# 2    28     273 11.5   82     8  13
# 3     8      19 20.1   61     5   9
# 4   135     269  4.1   84     7   1
# 5    23     299  8.6   65     5   7
# 6    30     322 11.5   68     5  19
```

Sort rows with `arrange`

Example:

1. Now let's arrange the data frame back into chronological order, sorting by Month then Day

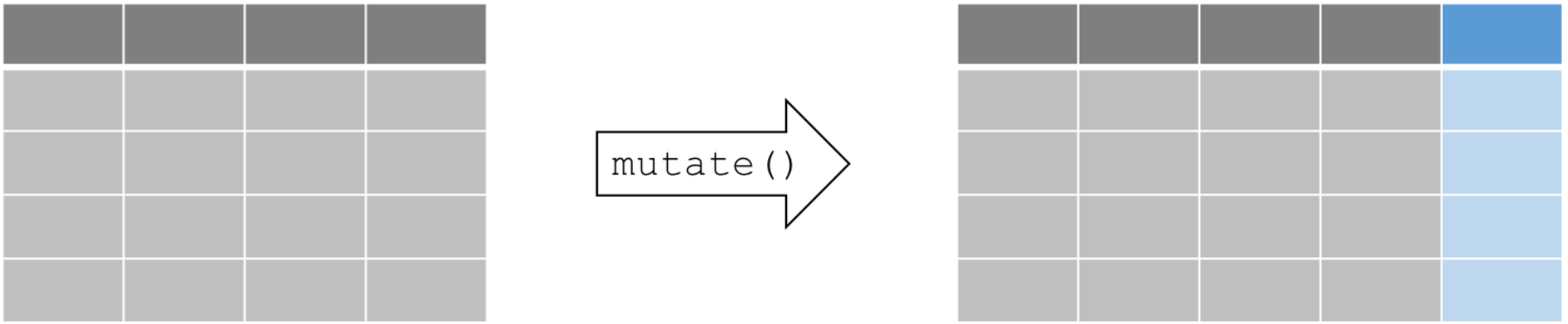
```
air_chron ← dplyr::arrange(air_mess, Month, Day)
head(air_chron)
#   Ozone Solar.R Wind Temp Month Day
# 1    41     190  7.4   67     5   1
# 2    36     118  8.0   72     5   2
# 3    12     149 12.6   74     5   3
# 4    18     313 11.5   62     5   4
# 5    NA      NA 14.3   56     5   5
# 6    28      NA 14.9   66     5   6
```

Try : `arrange(air_mess, Day, Month)` *and see the difference.*

Create new columns using `mutate`

Compute and add new columns

```
mutate(data, newVar1 = expression1, newVar2 = expression2, ...)
```



Create new columns using `mutate`

Example: we want to convert the temperature variable form degrees Fahrenheit to degrees Celsius

```
airquality_C ← dplyr::mutate(airquality, Temp_C = (Temp-32)*(5/9))
head(airquality_C)
#   Ozone Solar.R Wind Temp Month Day   Temp_C
# 1    41     190  7.4   67     5   1 19.44444
# 2    36     118  8.0   72     5   2 22.22222
# 3    12     149 12.6   74     5   3 23.33333
# 4    18     313 11.5   62     5   4 16.66667
# 5    NA      NA 14.3   56     5   5 13.33333
# 6    28      NA 14.9   66     5   6 18.88889
```

Usually data manipulation require multiple steps, the magrittr package offers a pipe operator `%>%` which allows us to link multiple operations

magrittr

```
library(magrittr)
```

Suppose we want to analyse only the month of June, then convert the temperature variable to degrees Celsius. We can create the required data frame by combining 2 dplyr verbs we learned

```
june_C ← mutate(filter(airquality, Month == 6),
                Temp_C = (Temp-32)*(5/9))
```

As we add more operations, wrapping functions one inside the other becomes increasingly illegible. But, step by step would be redundant and write a lot of objects to the workspace.

magrittr

Alternatively, we can use maggritr's pipe operator to link these successive operations

```
june_C ← airquality %>%
         dplyr::filter(Month == 6) %>%
         dplyr::mutate(Temp_C = (Temp-32)*(5/9))
```

Advantages :

- less redundant code
- easy to read and write because functions are executed in order

`dplyr::group_by` and `summarise`

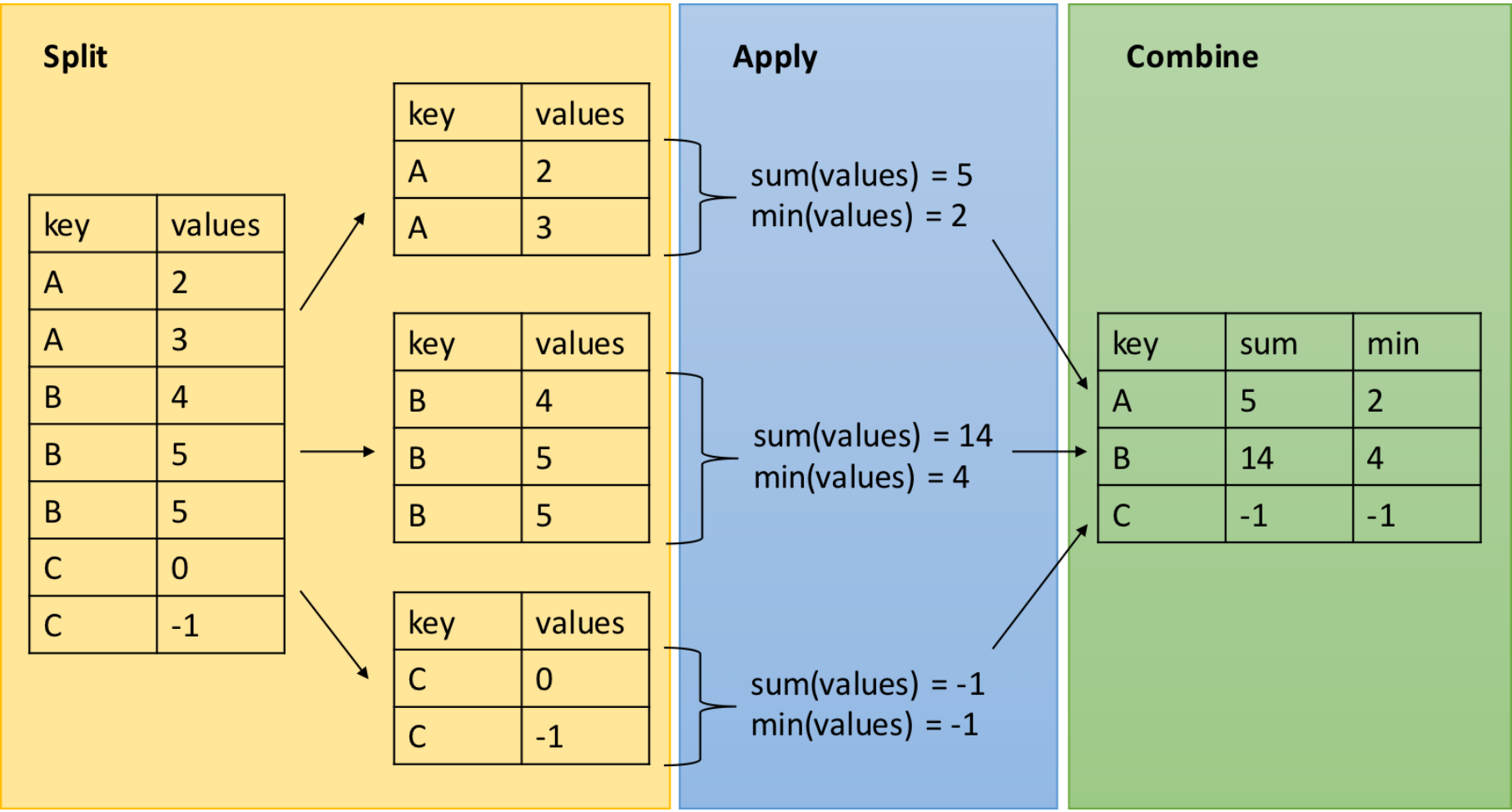
The `dplyr` verbs become especially powerful when they are are combined using the pipe operator `%>%`. The following `dplyr` functions allow us to split our data frame into groups on which we can perform operations individually

`group_by()` : group data frame by a factor for downstream operations (usually summarise)

`summarise()` : summarise values in a data frame or in groups within the data frame with aggregation functions (e.g. `min()`, `max()`, `mean()`, etc...)

`dplyr` - Split-Apply-Combine

The `group_by` function is key to the Split-Apply-Combine strategy



`dplyr` - Split-Apply-Combine

dplyr - Split-Apply-Combine

Example: we are interested in the mean temperature and standard deviation within each month if the airquality dataset

```
month_sum ← airquality %>%
      group_by(Month) %>%
      summarise(mean_temp = mean(Temp),
                sd_temp = sd(Temp))
month_sum
# # A tibble: 5 x 3
#   Month mean_temp sd_temp
#   <int>     <dbl>   <dbl>
# 1     5      65.5    6.85
# 2     6      79.1    6.60
# 3     7      83.9    4.32
# 4     8      84.0    6.59
# 5     9      76.9    8.36
```

Challenge - `dplyr` and `magrittr`

Using the `ChickWeight` dataset, create a summary table which displays the difference in weight between the maximum and minimum weight of each chick in the study.

Employ `dplyr` verbs and the `%>%` operator.

```
## ?ChickWeight
data(ChickWeight)
```

Solution

1. Use `group_by()` to divide the dataset by `Chick`
2. Use `summarise()` to calculate the weight gain within each group

```
weight_diff ← ChickWeight %>%
              dplyr::group_by(Chick) %>%
              dplyr::summarise(weight_diff = max(weight) - min(weight))
```

```
head(weight_diff)
# # A tibble: 6 x 2
#   Chick weight_diff
#   <ord>       <dbl>
# 1 18              4
# 2 16             16
# 3 15             27
# 4 13             55
# 5 9              58
# 6 20             76
```

Ninja challenge

- Using the `ChickWeight` dataset, create a summary table which displays, for each diet, the average individual difference in weight between the end and the beginning of the study.
- Employ `dplyr` verbs and the `%>%` operator.

Hint: `first()` and `last()` may be useful here

Ninja solution

```
diet_summ ← ChickWeight %>%
              dplyr::group_by(Diet, Chick) %>%
              dplyr::summarise(weight_gain = last(weight) - first(weight))
              dplyr::group_by(Diet) %>%
              dplyr::summarise(mean_gain = mean(weight_gain))
```

```
diet_summ
# # A tibble: 4 x 2
#   Diet  mean_gain
#   <fct>     <dbl>
# 1 1          115.
# 2 2          174
# 3 3          230.
# 4 4          188.
```

More on data manipulation

Learn more on dplyr

dplyr and tidyr cheatsheet

Thank you for attending!