R: A Hitchhikers Guide to Reproducible Research

- We built this software on base R code

Brendan Palmer,

Clinical Research Facility - Cork &

School of Public Health

@B_A_Palmer

CRF-C
HRB Clinical Research Facility Cork

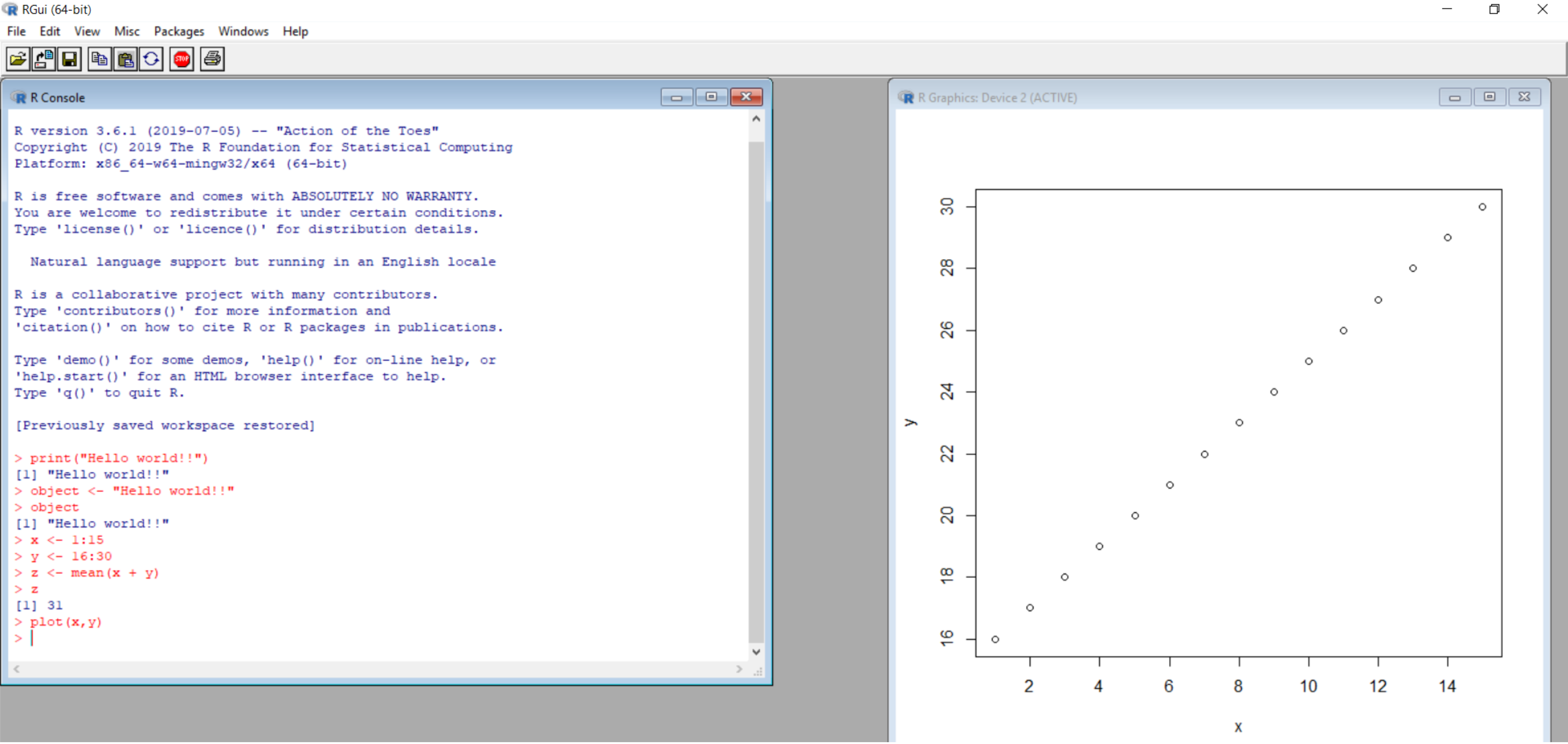
UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh
School of Public Health

To understand R, remember the following

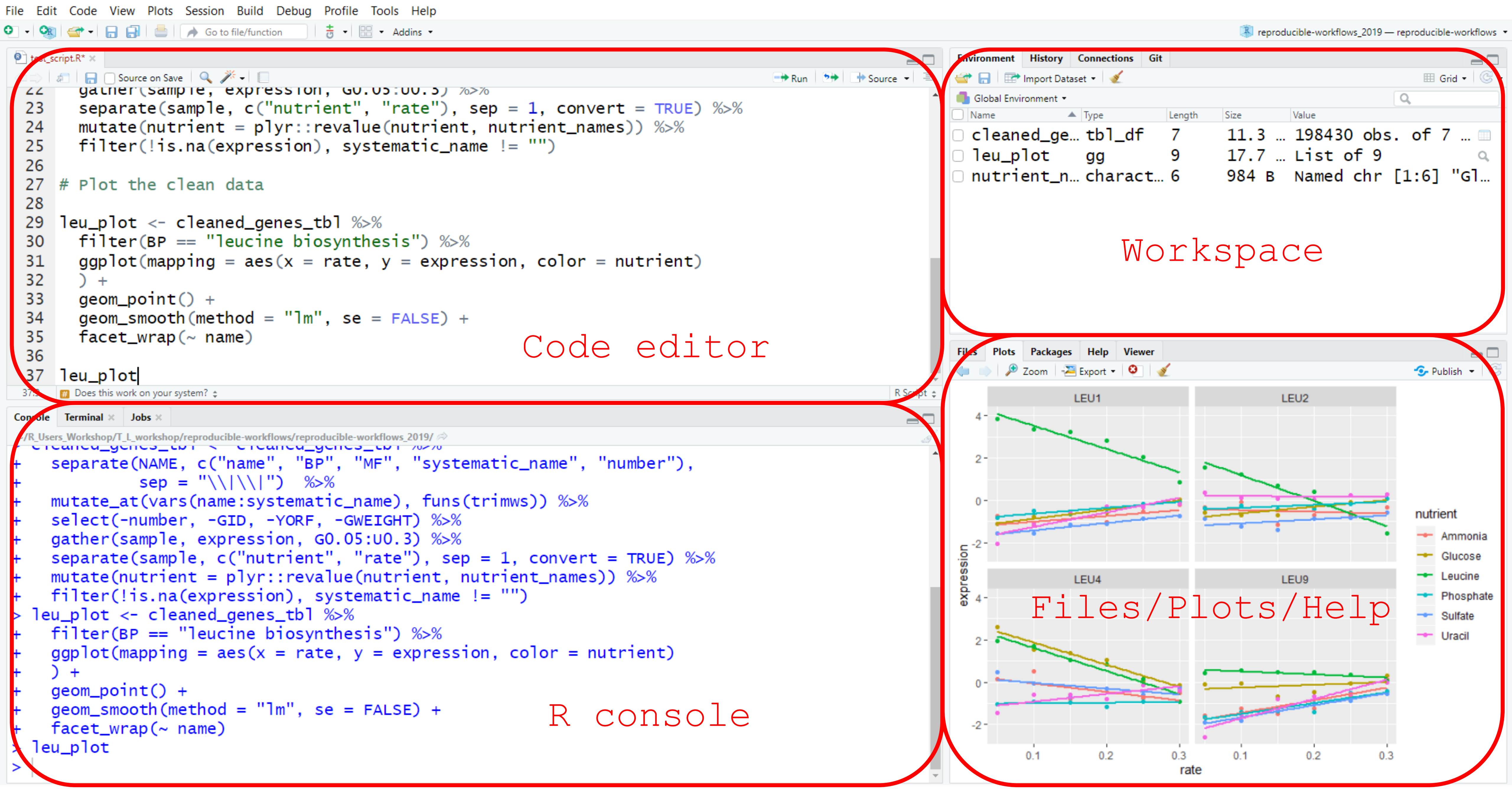
- **Everything that exists is an object**

- **Everything that happens is a function**

R user interface versus RStudio


RGui (64-bit)
File Edit View Misc Packages Windows Help
R Console
R Graphics: Device 2 (ACTIVE)
y
16
18
20
22
24
26
28
30
x
2
4
6
8
10
12
14

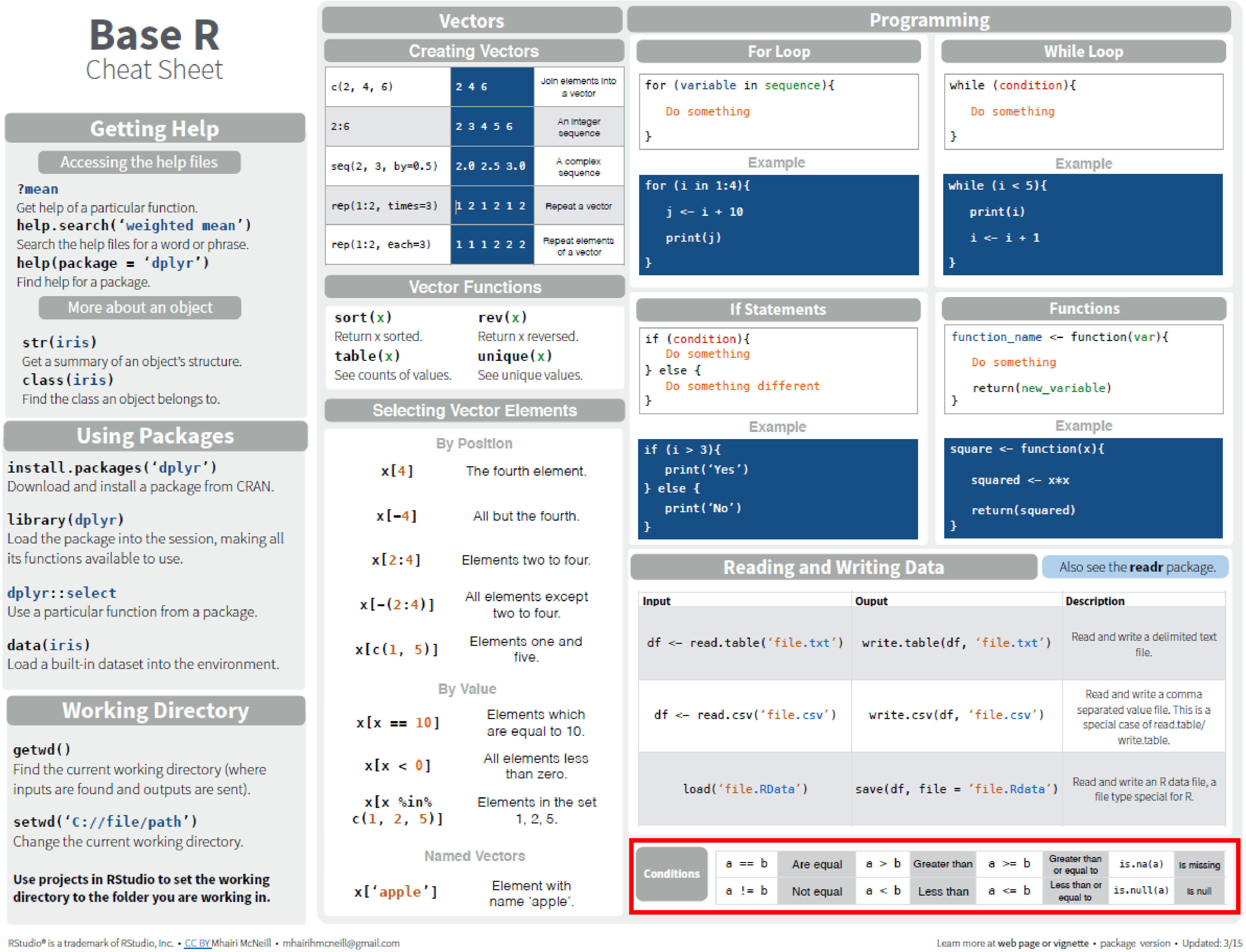
```
R version 3.6.1 (2019-07-05) -- "Action of the Toes"
Copyright (C) 2019 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[Previously saved workspace restored]

> print("Hello world!!")
[1] "Hello world!!"
> object <- "Hello world!!"
> object
[1] "Hello world!!"
> x <- 1:15
> y <- 16:30
> z <- mean(x + y)
> z
[1] 31
> plot(x,y)
> |
```



```
  gather(sample, expression, G0.05:U0.3) %>%
  separate(sample, c("nutrient", "rate"), sep = 1, convert = TRUE) %>%
  mutate(nutrient = plyr::revalue(nutrient, nutrient_names)) %>%
  filter(!is.na(expression), systematic_name != "")

# Plot the clean data

leu_plot <- cleaned_genes_tbl %>%
  filter(BP == "leucine biosynthesis") %>%
  ggplot(mapping = aes(x = rate, y = expression, color = nutrient)
  ) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE) +
  facet_wrap(~ name)

leu_plot
```

Name	Type	Length	Size	Value
cleaned_ge…	tbl_df	7	11.3 …	198430 obs. of 7 …
leu_plot	gg	9	17.7 …	List of 9
nutrient_n…	charact…	6	984 B	Named chr [1:6] "Gl…

```
> cleaned_genes_tbl <- cleaned_genes_tbl %>%
+   separate(NAME, c("name", "BP", "MF", "systematic_name", "number"),
+            sep = "\\|\\|")  %>%
+   mutate_at(vars(name:systematic_name), funs(trimws)) %>%
+   select(-number, -GID, -YORF, -GWEIGHT) %>%
+   gather(sample, expression, G0.05:U0.3) %>%
+   separate(sample, c("nutrient", "rate"), sep = 1, convert = TRUE) %>%
+   mutate(nutrient = plyr::revalue(nutrient, nutrient_names)) %>%
+   filter(!is.na(expression), systematic_name != "")
> leu_plot <- cleaned_genes_tbl %>%
+   filter(BP == "leucine biosynthesis") %>%
+   ggplot(mapping = aes(x = rate, y = expression, color = nutrient)
+   ) +
+   geom_point() +
+   geom_smooth(method = "lm", se = FALSE) +
+   facet_wrap(~ name)
> leu_plot
>
```



Base R
Cheat Sheet

Getting Help

Accessing the help files

`?mean`
Get help of a particular function.

`help.search(‘weighted mean’)`
Search the help files for a word or phrase.

`help(package = ‘dplyr’)`
Find help for a package.

More about an object

`str(iris)`
Get a summary of an object’s structure.

`class(iris)`
Find the class an object belongs to.

Using Packages

`install.packages(‘dplyr’)`
Download and install a package from CRAN.

`library(dplyr)`
Load the package into the session, making all its functions available to use.

`dplyr::select`
Use a particular function from a package.

`data(iris)`
Load a built-in dataset into the environment.

Working Directory

`getwd()`
Find the current working directory (where inputs are found and outputs are sent).

`setwd(‘C://file/path’)`
Change the current working directory.

Use projects in RStudio to set the working directory to the folder you are working in.

Vectors

Creating Vectors

`c(2, 4, 6)`	`2 4 6`	Join elements into a vector
`2:6`	`2 3 4 5 6`	An integer sequence
`seq(2, 3, by=0.5)`	`2.0 2.5 3.0`	A complex sequence
`rep(1:2, times=3)`	`1 2 1 2 1 2`	Repeat a vector
`rep(1:2, each=3)`	`1 1 1 2 2 2`	Repeat elements of a vector

Vector Functions

`sort(x)`
Return x sorted.

`rev(x)`
Return x reversed.

`table(x)`
See counts of values.

`unique(x)`
See unique values.

Selecting Vector Elements

By Position

`x[4]`	The fourth element.
`x[-4]`	All but the fourth.
`x[2:4]`	Elements two to four.
`x[-(2:4)]`	All elements except two to four.
`x[c(1, 5)]`	Elements one and five.

By Value

`x[x == 10]`	Elements which are equal to 10.
`x[x < 0]`	All elements less than zero.
`x[x %in% c(1, 2, 5)]`	Elements in the set 1, 2, 5.

Named Vectors

`x[‘apple’]`	Element with name ‘apple’.

Programming

For Loop

```
for (variable in sequence){
    Do something
}
```

Example

```
for (i in 1:4){
    j <- i + 10
    print(j)
}
```

While Loop

```
while (condition){
    Do something
}
```

Example

```
while (i < 5){
    print(i)
    i <- i + 1
}
```

If Statements

```
if (condition){
    Do something
} else {
    Do something different
}
```

Example

```
if (i > 3){
    print(‘Yes’)
} else {
    print(‘No’)
}
```

Functions

```
function_name <- function(var){
    Do something
    return(new_variable)
}
```

Example

```
square <- function(x){
    squared <- x*x
    return(squared)
}
```

Reading and Writing Data

Also see the **readr** package.

Input	Ouput	Description
`df <- read.table(‘file.txt’)`	`write.table(df, ‘file.txt’)`	Read and write a delimited text file.
`df <- read.csv(‘file.csv’)`	`write.csv(df, ‘file.csv’)`	Read and write a comma separated value file. This is a special case of read.table/ write.table.
`load(‘file.RData’)`	`save(df, file = ’file.Rdata’)`	Read and write an R data file, a file type special for R.

Conditions								
	`a == b`	Are equal	`a > b`	Greater than	`a >= b`	Greater than or equal to	`is.na(a)`	Is missing
	`a != b`	Not equal	`a < b`	Less than	`a <= b`	Less than or equal to	`is.null(a)`	Is null

RStudio® is a trademark of RStudio, Inc. • CC BY Mhairi McNeill • mhairihmcneill@gmail.com

Learn more at web page or vignette • package version • Updated: 3/15

Types

Converting between common data types in R. Can always go from a higher value in the table to a lower value.

`as.logical`	`TRUE, FALSE, TRUE`	Boolean values (TRUE or FALSE).
`as.numeric`	`1, 0, 1`	Integers or floating point numbers.
`as.character`	`'1', '0', '1'`	Character strings. Generally preferred to factors.

`as.factor`	`'1', '0', '1',`
	`levels: '1', '0'`
		Character strings with preset levels. Needed for some statistical models.

Maths Functions

`log(x)`	Natural log.	`sum(x)`	Sum.
`exp(x)`	Exponential.	`mean(x)`	Mean.
`max(x)`	Largest element.	`median(x)`	Median.
`min(x)`	Smallest element.	`quantile(x)`	Percentage quantiles.
`round(x, n)`	Round to n decimal places.	`rank(x)`	Rank of elements.
`signif(x, n)`	Round to n significant figures.	`var(x)`	The variance.
`cor(x, y)`	Correlation.	`sd(x)`	The standard deviation.

Variable Assignment

```
> a <- 'apple'
> a
[1] 'apple'
```

The Environment

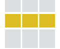
`ls()`	List all variables in the environment.
`rm(x)`	Remove x from the environment.
`rm(list = ls())`	Remove all variables from the environment.

You can use the environment panel in RStudio to browse variables in your environment.

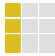
Matrices

`m <- matrix(x, nrow = 3, ncol = 3)`

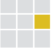
Create a matrix from x.



`m[2, ]` - Select a row



`m[ , 1]` - Select a column



`m[2, 3]` - Select an element

`t(m)`

Transpose

`m %*% n`

Matrix Multiplication

`solve(m, n)`

Find x in: m * x = n

Lists

`l <- list(x = 1:5, y = c('a', 'b'))`

A list is a collection of elements which can be of different types.

`l[[2]]`	`l[1]`	`l$x`	`l['y']`
Second element of l.	New list with only the first element.	Element named x.	New list with only element named y.

Data Frames

Also see the **dplyr** package.

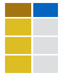
`df <- data.frame(x = 1:3, y = c('a', 'b', 'c'))`

A special case of a list where all elements are the same length.

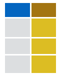
x	y
1	a
2	b
3	c

List subsetting

`df$x`



`df[[2]]`

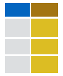


Understanding a data frame

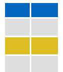
`View(df)`	See the full data frame.
`head(df)`	See the first 6 rows.

Matrix subsetting

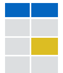
`df[ , 2]`



`df[2, ]`



`df[2, 2]`

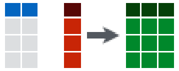


`nrow(df)`
Number of rows.

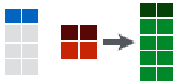
`ncol(df)`
Number of columns.

`dim(df)`
Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

Also see the **stringr** package.

`paste(x, y, sep = ' ')`	Join multiple vectors together.
`paste(x, collapse = ' ')`	Join elements of a vector together.
`grep(pattern, x)`	Find regular expression matches in x.
`gsub(pattern, replace, x)`	Replace matches in x with a string.
`toupper(x)`	Convert to uppercase.
`tolower(x)`	Convert to lowercase.
`nchar(x)`	Number of characters in a string.

Factors

`factor(x)`
Turn a vector into a factor. Can set the levels of the factor and the order.

`cut(x, breaks = 4)`
Turn a numeric vector into a factor by 'cutting' into sections.

Statistics

`lm(y ~ x, data=df)`
Linear model.

`glm(y ~ x, data=df)`
Generalised linear model.

`summary`
Get more detailed information out a model.

`t.test(x, y)`
Perform a t-test for difference between means.

`pairwise.t.test`
Perform a t-test for paired data.

`prop.test`
Test for a difference between proportions.

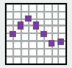
`aov`
Analysis of variance.

Distributions

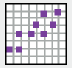
	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	`rnorm`	`dnorm`	`pnorm`	`qnorm`
Poisson	`rpois`	`dpois`	`ppois`	`qpois`
Binomial	`rbinom`	`dbinom`	`pbinom`	`qbinom`
Uniform	`runif`	`dunif`	`punif`	`qunif`

Plotting

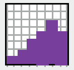
Also see the **ggplot2** package.



`plot(x)`
Values of x in order.



`plot(x, y)`
Values of x against y.



`hist(x)`
Histogram of x.

Dates

See the **lubridate** package.

Basics of R code

Symbol	What it does	Example 1	Example 2
<-	Assign operator Creates new objects	> x <- 5 > x [1] 5	> y <- "This" > y [1] "This"
c()	Helps create objects with more than one element	> v <- c(5,6,7,8) > v [1] 5 6 7 8	> w <- c("This", "is", "easy! ") > w [1] "This" "is" "easy!"
#	Computer ignores what is written. Used for adding notes to code	> # print(“hello”) >	> print(“hello”) [1] "hello"
%>%	Literally translates as “then do this”	> data %>% do_something_to(data)	> data %>% do_something_to(data) %>% do_something_else_to(data)
%in%	returns a logical vector indicating if there is a match	> "x" %in% c("x", "y", "z") [1] TRUE	> c("x", "y", "z") %in% "x" [1] TRUE FALSE FALSE
?	Access help	> ?mean()	> ?geom_point()

FYI: R is case sensitive!! Name.of.data ≠ name.of.data

Creating objects

For most of us, R is simply the creation of and manipulation of objects:

```
new_object <- c(1, 2, 3)
```

- the objects are then fed into functions to create amazing new objects

```
amazing_new_object <- function(new_object)
```

Broadly speaking the following is true in R:

- information

```
> data_frame  <- function(information)
> plot        <- function(data frame)
> model       <- function(data frame)
```

Naming objects

There are a few simple rules to follow initially:

- Object names must start with a letter and can only contain letters, numbers, '_' and '.'

- Certain characters should not be used, e.g:
    - c is the concatenate function 'c()'
    - T is used as shorthand for TRUE
    - F is used as shorthand for FALSE

- In this course I'll always use x, y and z as object names when demonstrating quick examples

Types of data structure

The main data types are;

```
# double (for double precision floating point numbers)
typeof(1.23)

# character
typeof("string")

# logical
typeof(FALSE)

# missing values are represented by NA
example <- c(1, 2, NA, 4)
```

Other examples include integers and complex numbers

Types of data structure

- Vectors come in two forms

A: Atomic vectors contain exactly one type of data

```
all_numbers           <- c(1, 2, 0.5, -0.5, 3.4)
all_characters        <- c("One", "too", "3")
all_logical           <- c(TRUE, FALSE) # NOTE: Type it out
```

B: Lists allow combinations of different types of data

```
this_is_a_list        <- list(1, TRUE, "Three", "4")
typeof(this_is_a_list)
[1] "list"

this_is_also_a_list <- list(all_numbers, all_characters)
```

Types of data structure

```
# Matrices/Arrays:

- You can have a matrix of two or more dimensions
  a_matrix <- matrix(1:9, 3, 3)

- Vectors and matrices can only contain one type of data

- VERY VERY VERY NB: If you try to create a vector with more
  than one data type, then it will undergo coercion to the
  least common denominator

- The coercion rule goes:
      logical -> integer -> numeric -> complex -> character

- You can perform coercions yourself on vectors
```

Walkthrough

- 01_baseR_introduction.R
    - Basic Code entry

Types of data structures

# Dataframes:

- These are a special type of list
- Observations are in rows
- Variables are in columns
- Labels or other metadata may also be present

```
> a_data_frame <- data.frame(number = 1:10,
                           char = sample(letters, 10),
                           this_really_a_col_name  = rep(c(TRUE, FALSE), 5))
```

- In the tidyverse dataframes are called ‘tibbles’

- Some older functions don’t work with tibbles

- We’ll go through this in more detail later

Indexing

- Indexing can occur in one or two dimensions
- One dimension:

```
new_object <- c(1, 2, 3)
new_object[1]
[1] 1
```

- Two dimensions

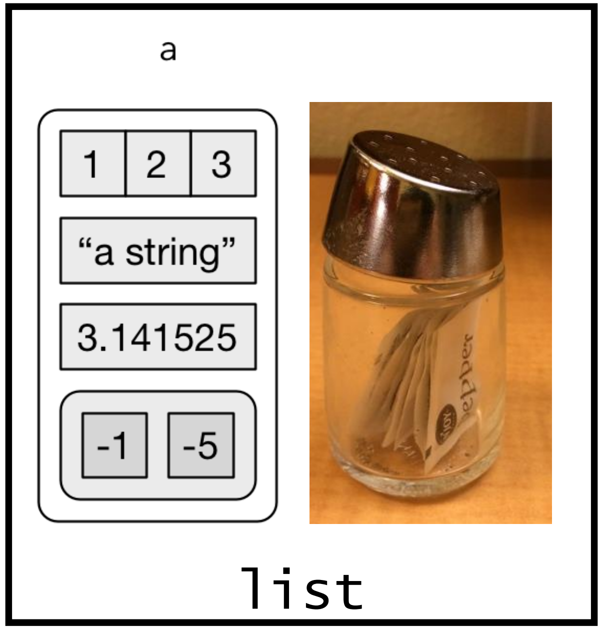
```
a_data_frame[1, 1]
a_data_frame$number[1]
```

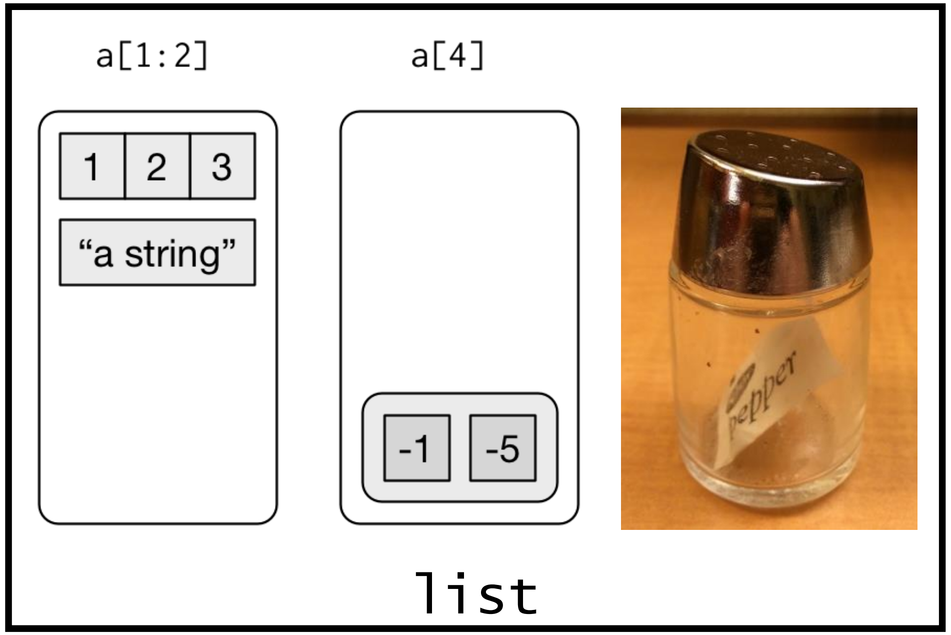
- In the tidyverse we don’t use ‘[’ much as dplyr::filter() and dplyr::select() allow you to solve the same problems

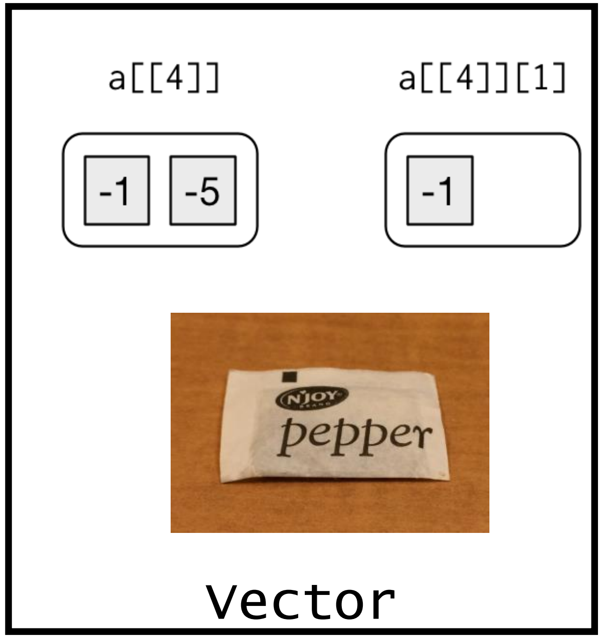
- However, given so much of the R has been written using these, it’s worth recognising and understanding them

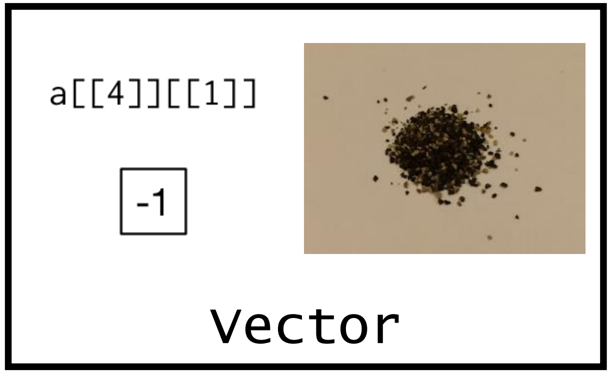
Indexing

```
# Recall
- this_is_also_a_list <- list(all_numbers, all_characters, all_logical)
```









```
# Important
-   [ extracts a sublist, results will be a list
-   [[ extracts a single component
```

Image: R for Data Science, Wickham and Grolemund

Walkthrough

- 01_baseR_introduction.R
    - Indexing dataframes and lists

Types of data structures

# Factors:

- In R, factors are used to work with categorical variables

- Historically they were easier to work with than characters, hence many baseR functions automatically convert characters to factors

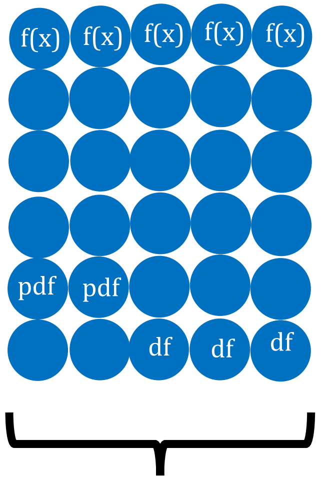
- This does not happen in the tidyverse

- One of the most important uses of factors is in statistical modeling;
    - since categorical variables are entered into statistical models differently than continuous variables, storing data as factors insures that the modeling functions will treat such data correctly

Walkthrough

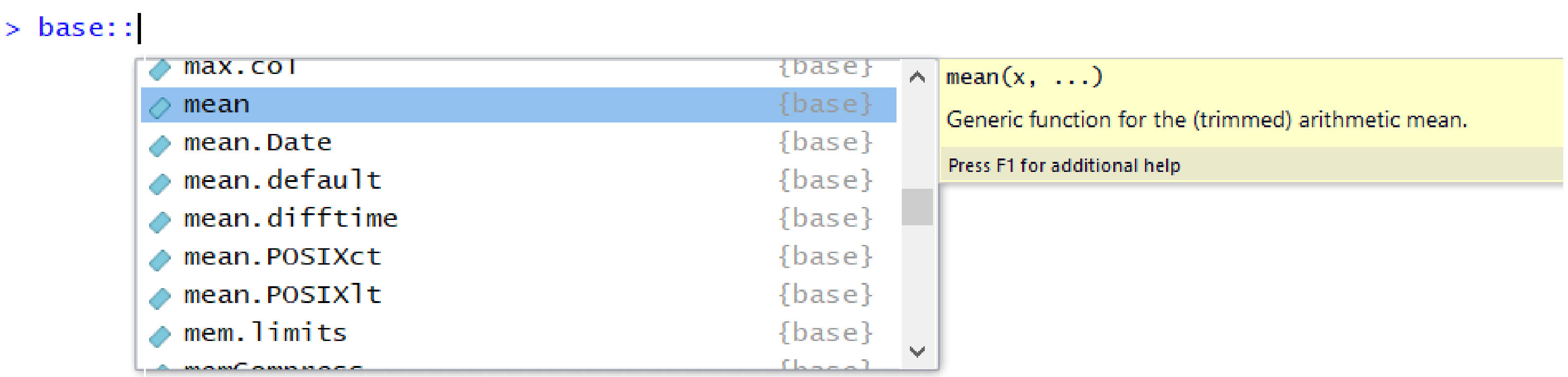
- 01_baseR_introduction.R
  - Factors example

Package contents



Base R: Comes pre-loaded

Functions



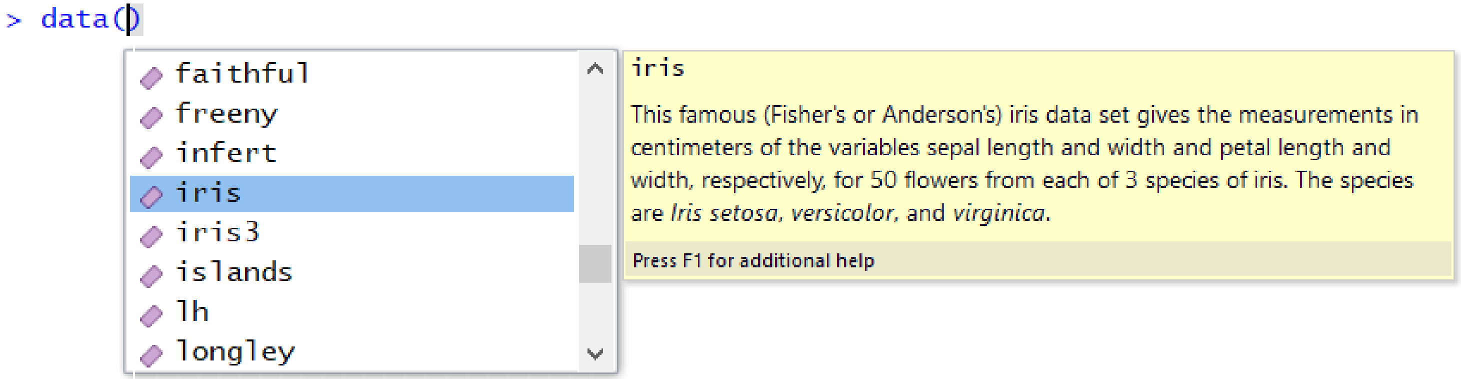
```
> base::
max.col                                   {base}
mean                                      {base}
mean.Date                                 {base}
mean.default                              {base}
mean.difftime                             {base}
mean.POSIXct                              {base}
mean.POSIXlt                              {base}
mem.limits                                {base}
memCompress                               {base}
```

mean(x, ...)

Generic function for the (trimmed) arithmetic mean.

Press F1 for additional help

Data sets



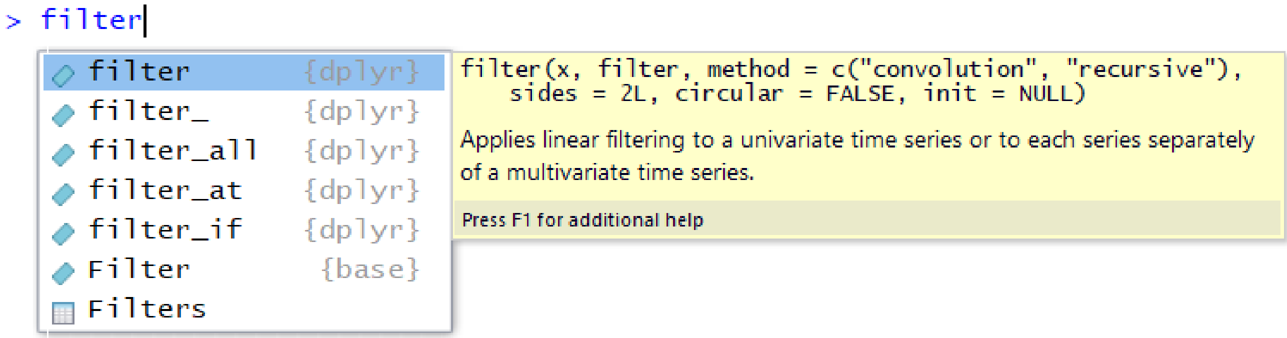
```
> data()
faithful
freeny
infert
iris
iris3
islands
lh
longley
```

iris

This famous (Fisher's or Anderson's) iris data set gives the measurements in centimeters of the variables sepal length and width and petal length and width, respectively, for 50 flowers from each of 3 species of iris. The species are *Iris setosa*, *versicolor*, and *virginica*.

Press F1 for additional help

Conflicts



```
> filter
filter         {dplyr}
filter_        {dplyr}
filter_all     {dplyr}
filter_at      {dplyr}
filter_if      {dplyr}
Filter          {base}
Filters
```

```
filter(x, filter, method = c("convolution", "recursive"),
    sides = 2L, circular = FALSE, init = NULL)
```

Applies linear filtering to a univariate time series or to each series separately of a multivariate time series.

Press F1 for additional help

Walkthrough

- 02_navigating_R_packages.R

Worksheet

- 03_practice_worksheet.R