

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  
}
```

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	Output	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

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: $m \cdot x = n$

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]]`

Second element of l.

`l[1]`

New list with only the first element.

`l$x`

Element named x.

`l['y']`

New list with only element named y.

Also see the `dplyr` package.

Data Frames

`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

Matrix subsetting

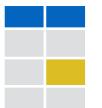
`df[, 2]`



`df[2,]`



`df[2, 2]`



List subsetting



Understanding a data frame

`View(df)`

See the full data frame.

`head(df)`

See the first 6 rows.

`nrow(df)`

Number of rows.

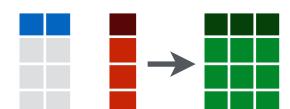
`ncol(df)`

Number of columns.

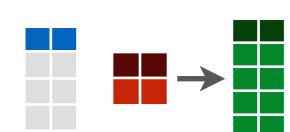
`dim(df)`

Number of columns and rows.

`cbind` - Bind columns.



`rbind` - Bind rows.



Strings

`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.

`aov`

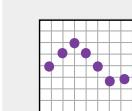
Analysis of variance.

Distributions

	Random Variates	Density Function	Cumulative Distribution	Quantile
Normal	<code>rnorm</code>	<code>dnorm</code>	<code>pnorm</code>	<code>qnorm</code>
Poisson	<code>rpois</code>	<code>dpois</code>	<code>ppois</code>	<code>qpois</code>
Binomial	<code>rbinom</code>	<code>dbinom</code>	<code>pbinom</code>	<code>qbinom</code>
Uniform	<code>runif</code>	<code>dunif</code>	<code>unif</code>	<code>qunif</code>

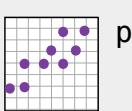
Plotting

Also see the `ggplot2` package.



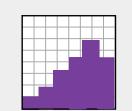
`plot(x)`

Values of x in order.



`plot(x, y)`

Values of x against y.



`hist(x)`

Histogram of x.

Dates

See the `lubridate` package.

Data Import :: CHEAT SHEET

R's **tidyverse** is built around **tidy data** stored in **tibbles**, which are enhanced data frames.

The front side of this sheet shows how to read text files into R with **readr**.

The reverse side shows how to create tibbles with **tibble** and to layout tidy data with **tidyr**.

OTHER TYPES OF DATA

Try one of the following packages to import other types of files

- **haven** - SPSS, Stata, and SAS files
- **readxl** - excel files (.xls and .xlsx)
- **DBI** - databases
- **jsonlite** - json
- **xml2** - XML
- **httr** - Web APIs
- **rvest** - HTML (Web Scraping)

Save Data

Save **x**, an R object, to **path**, a file path, as:

Comma delimited file

```
write_csv(x, path, na = "NA", append = FALSE,  
          col_names = !append)
```

File with arbitrary delimiter

```
write_delim(x, path, delim = " ", na = "NA",  
            append = FALSE, col_names = !append)
```

CSV for excel

```
write_excel_csv(x, path, na = "NA", append =  
                FALSE, col_names = !append)
```

String to file

```
write_file(x, path, append = FALSE)
```

String vector to file, one element per line

```
write_lines(x, path, na = "NA", append = FALSE)
```

Object to RDS file

```
write_rds(x, path, compress = c("none", "gz",  
                                "bz2", "xz"), ...)
```

Tab delimited files

```
write_tsv(x, path, na = "NA", append = FALSE,  
          col_names = !append)
```



Read Tabular Data

- These functions share the common arguments:

```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),  
       quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,  
       n_max), progress = interactive())
```

a,b,c	1,2,3	4,5,NA
1	2	3
4	5	NA

a;b;c	1;2;3	4;5;NA
1	2	3
4	5	NA

a b c	1 2 3	4 5 NA
1	2	3
4	5	NA

a b c	1 2 3	4 5 NA
1	2	3
4	5	NA

Comma Delimited Files

```
read_csv("file.csv")
```

To make file.csv run:

```
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")
```

Semi-colon Delimited Files

```
read_csv2("file2.csv")
```

```
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")
```

Files with Any Delimiter

```
read_delim("file.txt", delim = "|")
```

```
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")
```

Fixed Width Files

```
read_fwf("file.fwf", col_positions = c(1, 3, 5))
```

```
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")
```

Tab Delimited Files

```
read_tsv("file.tsv") Also read_table().
```

```
write_file(x = "a\tb\tc\n1\t2\t3\n4\t5\tNA", path = "file.tsv")
```

USEFUL ARGUMENTS

a,b,c	1,2,3	4,5,NA
1	2	3
4	5	NA

Example file

```
write_file("a,b,c\n1,2,3\n4,5,NA","file.csv")  
f <- "file.csv"
```

A	B	C
1	2	3
4	5	NA

Skip lines

```
read_csv(f, skip = 1)
```

a,b,c	1,2,3	4,5,NA
1	2	3
4	5	NA

No header

```
read_csv(f, col_names = FALSE)
```

A	B	C
1	2	3
4	5	NA

Read in a subset

```
read_csv(f, n_max = 1)
```

x	y	z
A	B	C
1	2	3
4	5	NA

Provide header

```
read_csv(f, col_names = c("x", "y", "z"))
```

A	B	C
NA	2	3
4	5	NA

Missing Values

```
read_csv(f, na = c("1", "?"))
```

Read Non-Tabular Data

Read a file into a single string

```
read_file(file, locale = default_locale())
```

Read each line into its own string

```
read_lines(file, skip = 0, n_max = -1L, na = character(),  
          locale = default_locale(), progress = interactive())
```

Read Apache style log files

```
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, progress = interactive())
```

Read a file into a raw vector

```
read_file_raw(file)
```

Read each line into a raw vector

```
read_lines_raw(file, skip = 0, n_max = -1L,  
               progress = interactive())
```



Data types

readr functions guess the types of each column and convert types when appropriate (but will NOT convert strings to factors automatically).

A message shows the type of each column in the result.

```
## Parsed with column specification:  
## cols(  
##   age = col_integer(),  
##   sex = col_character(),  
##   earn = col_double()  
## )
```

age is an integer

sex is a character

1. Use **problems()** to diagnose problems.

```
x <- read_csv("file.csv"); problems(x)
```

2. Use a **col_** function to guide parsing.

- **col_guess()** - the default
 - **col_character()**
 - **col_double()**, **col_euro_double()**
 - **col_datetime(format = "")** Also **col_date(format = "")**, **col_time(format = "")**
 - **col_factor(levels, ordered = FALSE)**
 - **col_integer()**
 - **col_logical()**
 - **col_number()**, **col_numeric()**
 - **col_skip()**
- x <- read_csv("file.csv", col_types = cols(
A = col_double(),
B = col_logical(),
C = col_factor()))**

3. Else, read in as character vectors then parse with a **parse_** function.

- **parse_guess()**
 - **parse_character()**
 - **parse_datetime()** Also **parse_date()** and **parse_time()**
 - **parse_double()**
 - **parse_factor()**
 - **parse_integer()**
 - **parse_logical()**
 - **parse_number()**
- x\$A <- parse_number(x\$A)**

Tibbles - an enhanced data frame

The **tibble** package provides a new S3 class for storing tabular data, the **tibble**. Tibbles inherit the data frame class, but improve three behaviors:

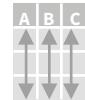
- **Subsetting** - [always returns a new tibble, [[and \$ always return a vector.
- **No partial matching** - You must use full column names when subsetting
- **Display** - When you print a tibble, R provides a concise view of the data that fits on one screen

# A tibble: 234 x 6	manufacturer	model	displ	year	cyl
1 audi	a4	1.8	1999	4	
2 audi	a4	1.8	1999	4	
3 audi	a4	2.0	1999	4	
4 audi	a4	2.0	1999	4	
5 audi	a4	2.0	1999	4	
6 audi	a4	2.0	1999	4	
7 audi	a4	2.0	1999	4	
8 audi	a4 quattro	1.8	1999	4	
9 audi	a4 quattro	1.8	1999	4	
10 audi	a4 quattro	1.8	1999	4	
11 audi	a4 quattro	1.8	1999	4	
12 audi	a4 quattro	1.8	1999	4	
13 audi	a4 quattro	1.8	1999	4	
14 audi	a4 quattro	1.8	1999	4	
15 audi	a4 quattro	1.8	1999	4	
16 audi	a4 quattro	1.8	1999	4	
17 audi	a4 quattro	1.8	1999	4	
18 audi	a4 quattro	1.8	1999	4	
19 audi	a4 quattro	1.8	1999	4	
20 audi	a4 quattro	1.8	1999	4	
21 audi	a4 quattro	1.8	1999	4	
22 audi	a4 quattro	1.8	1999	4	
23 audi	a4 quattro	1.8	1999	4	
24 audi	a4 quattro	1.8	1999	4	
25 audi	a4 quattro	1.8	1999	4	
26 audi	a4 quattro	1.8	1999	4	
27 audi	a4 quattro	1.8	1999	4	
28 audi	a4 quattro	1.8	1999	4	
29 audi	a4 quattro	1.8	1999	4	
30 audi	a4 quattro	1.8	1999	4	
31 audi	a4 quattro	1.8	1999	4	
32 audi	a4 quattro	1.8	1999	4	
33 audi	a4 quattro	1.8	1999	4	
34 audi	a4 quattro	1.8	1999	4	
35 audi	a4 quattro	1.8	1999	4	
36 audi	a4 quattro	1.8	1999	4	
37 audi	a4 quattro	1.8	1999	4	
38 audi	a4 quattro	1.8	1999	4	
39 audi	a4 quattro	1.8	1999	4	
40 audi	a4 quattro	1.8	1999	4	
41 audi	a4 quattro	1.8	1999	4	
42 audi	a4 quattro	1.8	1999	4	
43 audi	a4 quattro	1.8	1999	4	
44 audi	a4 quattro	1.8	1999	4	
45 audi	a4 quattro	1.8	1999	4	
46 audi	a4 quattro	1.8	1999	4	
47 audi	a4 quattro	1.8	1999	4	
48 audi	a4 quattro	1.8	1999	4	
49 audi	a4 quattro	1.8	1999	4	
50 audi	a4 quattro	1.8	1999	4	
51 audi	a4 quattro	1.8	1999	4	
52 audi	a4 quattro	1.8	1999	4	
53 audi	a4 quattro	1.8	1999	4	
54 audi	a4 quattro	1.8	1999	4	
55 audi	a4 quattro	1.8	1999	4	
56 audi	a4 quattro	1.8	1999	4	
57 audi	a4 quattro	1.8	1999	4	
58 audi	a4 quattro	1.8	1999	4	
59 audi	a4 quattro	1.8	1999	4	
60 audi	a4 quattro	1.8	1999	4	
61 audi	a4 quattro	1.8	1999	4	
62 audi	a4 quattro	1.8	1999	4	
63 audi	a4 quattro	1.8	1999	4	
64 audi	a4 quattro	1.8	1999	4	
65 audi	a4 quattro	1.8	1999	4	
66 audi	a4 quattro	1.8	1999	4	
67 audi	a4 quattro	1.8	1999	4	
68 audi	a4 quattro	1.8	1999	4	
69 audi	a4 quattro	1.8	1999	4	
70 audi	a4 quattro	1.8	1999	4	
71 audi	a4 quattro	1.8	1999	4	
72 audi	a4 quattro	1.8	1999	4	
73 audi	a4 quattro	1.8	1999	4	
74 audi	a4 quattro	1.8	1999	4	
75 audi	a4 quattro	1.8	1999	4	
76 audi	a4 quattro	1.8	1999	4	
77 audi	a4 quattro	1.8	1999	4	
78 audi	a4 quattro	1.8	1999	4	
79 audi	a4 quattro	1.8	1999	4	
80 audi	a4 quattro	1.8	1999	4	
81 audi	a4 quattro	1.8	1999	4	
82 audi	a4 quattro	1.8	1999	4	
83 audi	a4 quattro	1.8	1999	4	
84 audi	a4 quattro	1.8	1999	4	
85 audi	a4 quattro	1.8	1999	4	
86 audi	a4 quattro	1.8	1999	4	
87 audi	a4 quattro	1.8	1999	4	
88 audi	a4 quattro	1.8	1999	4	
89 audi	a4 quattro	1.8	1999	4	
90 audi	a4 quattro	1.8	1999	4	
91 audi	a4 quattro	1.8	1999	4	
92 audi	a4 quattro	1.8	1999	4	
93 audi	a4 quattro	1.8	1999	4	
94 audi	a4 quattro	1.8	1999	4	
95 audi	a4 quattro	1.8	1999	4	
96 audi	a4 quattro	1.8	1999	4	
97 audi	a4 quattro	1.8	1999	4	
98 audi	a4 quattro	1.8	1999	4	
99 audi	a4 quattro	1.8	1999	4	
100 audi	a4 quattro	1.8	1999	4	
101 audi	a4 quattro	1.8	1999	4	
102 audi	a4 quattro	1.8	1999	4	
103 audi	a4 quattro	1.8	1999	4	
104 audi	a4 quattro	1.8	1999	4	
105 audi	a4 quattro	1.8	1999	4	
106 audi	a4 quattro	1.8	1999	4	
107 audi	a4 quattro	1.8	1999	4	
108 audi	a4 quattro	1.8	1999	4	
109 audi	a4 quattro	1.8	1999	4	
110 audi	a4 quattro	1.8	1999	4	
111 audi	a4 quattro	1.8	1999	4	
112 audi	a4 quattro	1.8	1999	4	
113 audi	a4 quattro	1.8	1999	4	
114 audi	a4 quattro	1.8	1999	4	
115 audi	a4 quattro	1.8	1999	4	
116 audi	a4 quattro	1.8	1999	4	
117 audi	a4 quattro	1.8	1999	4	
118 audi	a4 quattro	1.8	1999	4	
119 audi	a4 quattro	1.8	1999	4	
120 audi	a4 quattro	1.8	1999	4	
121 audi	a4 quattro	1.8	1999	4	
122 audi	a4 quattro	1.8	1999	4	
123 audi	a4 quattro	1.8	1999	4	
124 audi	a4 quattro	1.8	1999	4	
125 audi	a4 quattro	1.8	1999	4	
126 audi	a4 quattro	1.8	1999	4	
127 audi	a4 quattro	1.8	1999	4	
128 audi	a4 quattro	1.8	1999	4	
129 audi	a4 quattro	1.8	1999	4	
130 audi	a4 quattro	1.8	1999	4	
131 audi	a4 quattro	1.8	1999	4	
132 audi	a4 quattro	1.8	1999	4	
133 audi	a4 quattro	1.8	1999	4	
134 audi	a4 quattro	1.8	1999	4	
135 audi	a4 quattro	1.8	1999	4	
136 audi	a4 quattro	1.8	1999	4	
137 audi	a4 quattro	1.8	1999	4	
138 audi	a4 quattro	1.8	1999	4	
139 audi	a4 quattro	1.8	1999	4	
140 audi	a4 quattro	1.8	1999	4	
141 audi	a4 quattro	1.8	1999	4	
142 audi	a4 quattro	1.8	1999	4	
143 audi	a4 quattro	1.8	1999	4	
144 audi	a4 quattro	1.8	1999	4	
145 audi	a4 quattro	1.8	1999	4	
146 audi	a4 quattro	1.8	1999	4	
147 audi	a4 quattro	1.8	1999	4	
148 audi	a4 quattro	1.8	1999	4	
149 audi	a4 quattro	1.8	1999	4	
150 audi	a4 quattro	1.8	1999	4	
151 audi	a4 quattro	1.8	1999	4	
152 audi	a4 quattro	1.8	1999	4	
153 audi	a4 quattro	1.8	1999	4	
154 audi	a4 quattro	1.8	1999	4	
155 audi	a4 quattro	1.8	1999	4	
156 audi	a4 quattro	1.8	1999	4	
157 audi	a4 quattro	1.8	1999	4	
158 audi	a4 quattro	1.8	1999	4	
159 audi	a4 quattro	1.8	1999	4	
160 audi	a4 quattro	1.8	1999	4	
161 audi	a4 quattro	1.8	1999	4	
162 audi	a4 quattro	1.8	1999	4	
163 audi	a4 quattro	1.8	1999	4	
164 audi	a4 quattro	1.8	1999	4	
165 audi	a4 quattro	1.8	1999	4	
166 audi	a4 quattro	1.8	1999	4	
167 audi	a4 quattro	1.8	1999	4	
168 audi	a4 quattro	1.8	1999	4	
169 audi	a4 quattro	1.8	1999	4	
170 audi	a4 quattro	1.8	1999	4	
171 audi	a4 quattro	1.8	1999	4	
172 audi	a4 quattro	1.8	1999</td		

Data Transformation with dplyr :: CHEAT SHEET



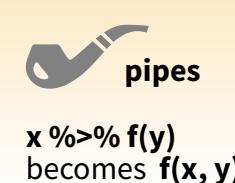
dplyr functions work with pipes and expect **tidy data**. In tidy data:



Each **variable** is in its own **column**



Each **observation**, or **case**, is in its own **row**



Summarise Cases

These apply **summary functions** to columns to create a new table. Summary functions take vectors as input and return one value (see back).

	summary function
	summarise(.data, ...) Compute table of summaries. Also summarise_() . <code>summarise(mtcars, avg = mean(mpg))</code>
	count(x, ..., wt = NULL, sort = FALSE) Count number of rows in each group defined by the variables in ... Also tally() . <code>count(iris, Species)</code>

VARIATIONS

- summarise_all()** - Apply funs to every column.
- summarise_at()** - Apply funs to specific columns.
- summarise_if()** - Apply funs to all cols of one type.

Group Cases

Use **group_by()** to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.

	<code>mtcars %>% group_by(cyl) %>% summarise(avg = mean(mpg))</code>
--	--

group_by(.data, ..., add = FALSE)
Returns copy of table grouped by ...
`g_iris <- group_by(iris, Species)`

ungroup(x, ...)
Returns ungrouped copy of table.
`ungroup(g_iris)`

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

	filter(.data, ...) Extract rows that meet logical criteria. Also filter_() . <code>filter(iris, Sepal.Length > 7)</code>
	distinct(.data, ..., .keep_all = FALSE) Remove rows with duplicate values. Also distinct_() . <code>distinct(iris, Species)</code>
	sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select fraction of rows. <code>sample_frac(iris, 0.5, replace = TRUE)</code>
	sample_n(tbl, size, replace = FALSE, weight = NULL, .env = parent.frame()) Randomly select size rows. <code>sample_n(iris, 10, replace = TRUE)</code>
	slice(.data, ...) Select rows by position. Also slice_() . <code>slice(iris, 10:15)</code>
	top_n(x, n, wt) Select and order top n entries (by group if grouped data). <code>top_n(iris, 5, Sepal.Width)</code>

Logical and boolean operators to use with filter()

<	<=	is.na()	%in%		xor()
>	>=	!is.na()	!	&	

See **?base:::logic** and **?Comparison** for help.

ARRANGE CASES

	arrange(.data, ...) Order rows by values of a column or columns (low to high), use with desc() to order from high to low. <code>arrange(mtcars, mpg)</code> <code>arrange(mtcars, desc(mpg))</code>
--	---

ADD CASES

	add_row(.data, ..., .before = NULL, .after = NULL) Add one or more rows to a table. <code>add_row(faithful, eruptions = 1, waiting = 1)</code>
--	---

Column functions return a set of columns as a new table. Use a variant that ends in _ for non-standard evaluation friendly code.

	select(.data, ...) Extract columns by name. Also select_if() . <code>select(iris, Sepal.Length, Species)</code>
--	---

Use these helpers with **select ()**,
e.g. `select(iris, starts_with("Sepal"))`

contains(match)	num_range(prefix, range)	:, e.g. <code>mpg:cyl</code>
ends_with(match)	one_of(...)	-, e.g. <code>-Species</code>
matches(match)	starts_with(match)	

MAKE NEW VARIABLES

These apply **vectorized functions** to columns. Vectorized funs take vectors as input and return vectors of the same length as output (see back).

vectorized function	
	mutate(.data, ...) Compute new column(s). <code>mutate(mtcars, gpm = 1/mpg)</code>
	transmute(.data, ...) Compute new column(s), drop others. <code>transmute(mtcars, gpm = 1/mpg)</code>
	mutate_all(.tbl, .funs, ...) Apply funs to every column. Use with funs() . <code>mutate_all(faithful, funs(log(.), log2(.)))</code>
	mutate_at(.tbl, .cols, .funs, ...) Apply funs to specific columns. Use with funs() , vars() and the helper functions for select() . <code>mutate_at(iris, vars(-Species), funs(log(.)))</code>
	mutate_if(.tbl, .predicate, .funs, ...) Apply funs to all columns of one type. Use with funs() . <code>mutate_if(iris, is.numeric, funs(log(.)))</code>
	add_column(.data, ..., .before = NULL, .after = NULL) Add new column(s). <code>add_column(mtcars, new = 1:32)</code>
	rename(.data, ...) Rename columns. <code>rename(iris, Length = Sepal.Length)</code>



Vector Functions

TO USE WITH MUTATE ()

mutate() and **transmute()** apply vectorized functions to columns to create new columns. Vectorized functions take vectors as input and return vectors of the same length as output.

vectorized function →

OFFSETS

dplyr::lag() - Offset elements by 1
dplyr::lead() - Offset elements by -1

CUMULATIVE AGGREGATES

dplyr::cumall() - Cumulative all()
dplyr::cumany() - Cumulative any()
 cummax() - Cumulative max()
dplyr::cummean() - Cumulative mean()
 cummin() - Cumulative min()
 cumprod() - Cumulative prod()
 cumsum() - Cumulative sum()

RANKINGS

dplyr::cume_dist() - Proportion of all values <=
dplyr::dense_rank() - rank with ties = min, no gaps
dplyr::min_rank() - rank with ties = min
dplyr::ntile() - bins into n bins
dplyr::percent_rank() - min_rank scaled to [0,1]
dplyr::row_number() - rank with ties = "first"

MATH

+, -, *, /, ^, %/%, %% - arithmetic ops
log(), **log2()**, **log10()** - logs
<, <=, >, >=, !=, == - logical comparisons

MISC

dplyr::between() - x >= left & x <= right
dplyr::case_when() - multi-case if_else()
dplyr::coalesce() - first non-NA values by element across a set of vectors
dplyr::if_else() - element-wise if() + else()
dplyr::na_if() - replace specific values with NA
 pmax() - element-wise max()
 pmin() - element-wise min()
dplyr::recode() - Vectorized switch()
dplyr::recode_factor() - Vectorized switch() for factors

Summary Functions

TO USE WITH SUMMARISE ()

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

summary function →

COUNTS

dplyr::n() - number of values/rows
dplyr::n_distinct() - # of uniques
 sum(!is.na()) - # of non-NA's

LOCATION

mean() - mean, also **mean(!is.na())**
median() - median

LOGICALS

mean() - Proportion of TRUE's
sum() - # of TRUE's

POSITION/ORDER

dplyr::first() - first value
dplyr::last() - last value
dplyr::nth() - value in nth location of vector

RANK

quantile() - nth quantile
min() - minimum value
max() - maximum value

SPREAD

IQR() - Inter-Quartile Range
mad() - median absolute deviation
sd() - standard deviation
var() - variance

Row Names

Tidy data does not use rownames, which store a variable outside of the columns. To work with the rownames, first move them into a column.

→
rownames_to_column()
Move row names into col.
a <- rownames_to_column(iris, var = "C")

→
column_to_rownames()
Move col in row names.
column_to_rownames(a, var = "C")

Also **has_rownames()**, **remove_rownames()**

Combine Tables

COMBINE VARIABLES

X	Y	=
A B C a t 1 b u 2 c v 3	A B D a t 3 b u 2 d w 1	A B C A B D a t 1 a t 3 b u 2 b u 2 c v 3 d w 1

Use **bind_cols()** to paste tables beside each other as they are.

bind_cols(...) Returns tables placed side by side as a single table.
BE SURE THAT ROWS ALIGN.

Use a "**Mutating Join**" to join one table to columns from another, matching values with the rows that they correspond to. Each join retains a different combination of values from the tables.

X	Y	Join
A B C D a t 1 3 b u 2 2 c v 3 NA	A B C D a t 1 3 b u 2 2 d w NA 1	left_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) Join matching values from y to x.

X	Y	Join
A B C D a t 1 3 b u 2 2 d w NA 1	A B C D a t 1 3 b u 2 2 d w NA 1	right_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) Join matching values from x to y.

X	Y	Join
A B C D a t 1 3 b u 2 2 c v 3 NA	A B C D a t 1 3 b u 2 2 c v 3 NA	inner_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) Join data. Retain only rows with matches.

X	Y	Join
A B C D a t 1 3 b u 2 2 c v 3 NA	A B C D a t 1 3 b u 2 2 d w NA 1	full_join(x, y, by = NULL, copy = FALSE, suffix = c(".x", ".y"), ...) Join data. Retain all values, all rows.

→
Use **by = c("col1", "col2")** to specify the column(s) to match on.
left_join(x, y, by = "A")

→
Use a named vector, **by = c("col1" = "col2")**, to match on columns with different names in each data set.
left_join(x, y, by = c("C" = "D"))

→
Use **suffix** to specify suffix to give to duplicate column names.
left_join(x, y, by = c("C" = "D"), suffix = c("1", "2"))

COMBINE CASES

X	Y	=
A B C a t 1 b u 2 c v 3	A B C C v 3 d w 4	

Use **bind_rows()** to paste tables below each other as they are.

bind_rows(..., .id = NULL)
Returns tables one on top of the other as a single table. Set .id to a column name to add a column of the original table names (as pictured)

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.

union(x, y, ...)
Rows that appear in x or y.
(Duplicates removed). **union_all()** retains duplicates.

Use **setequal()** to test whether two data sets contain the exact same rows (in any order).

EXTRACT ROWS

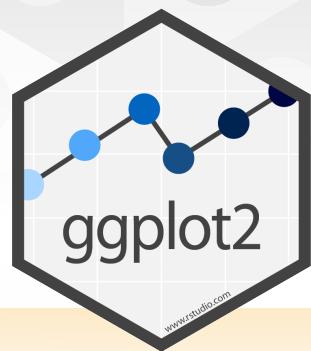
X	Y	=
A B C a t 1 b u 2 c v 3	A B D a t 3 b u 2 d w 1	

Use a "**Filtering Join**" to filter one table against the rows of another.

semi_join(x, y, by = NULL, ...)
Return rows of x that have a match in y.
USEFUL TO SEE WHAT WILL BE JOINED.

anti_join(x, y, by = NULL, ...)
Return rows of x that do not have a match in y. USEFUL TO SEE WHAT WILL NOT BE JOINED.

Data Visualization with ggplot2 :: CHEAT SHEET



Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same components: a **data** set, a **coordinate system**, and geoms—visual marks that represent data points.



To display values, map variables in the data to visual properties of the geom (**aesthetics**) like **size**, **color**, and **x** and **y** locations.



Complete the template below to build a graph.

```
ggplot (data = <DATA>) +
<GEOM_FUNCTION>(mapping = aes(<MAPPINGS>),
stat = <STAT>, position = <POSITION>) +
<COORDINATE_FUNCTION> +
<FACET_FUNCTION> +
<SCALE_FUNCTION> +
<THEME_FUNCTION>
```

required
Not required, sensible defaults supplied

ggplot(data = mpg, **aes**(x = cty, y = hwy)) Begins a plot that you finish by adding layers to. Add one geom function per layer.

aesthetic mappings **data** **geom**

qplot(x = cty, y = hwy, data = mpg, geom = "point") Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

last_plot() Returns the last plot

gsave("plot.png", **width** = 5, **height** = 5) Saves last plot as 5' x 5' file named "plot.png" in working directory. Matches file type to file extension.

Geoms

Use a geom function to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

GRAPHICAL PRIMITIVES

- a + **geom_blank()**
(Useful for expanding limits)
- b + **geom_curve**(aes(yend = lat + 1, xend=long+1, curvature=z)) - x, yend, alpha, angle, color, curvature, linetype, size
- a + **geom_path**(lineend="butt", linejoin="round", linemitre=1) - x, y, alpha, color, group, linetype, size
- a + **geom_polygon**(aes(group = group)) - x, y, alpha, color, fill, group, linetype, size
- b + **geom_rect**(aes(xmin = long, ymin=lat, xmax=long + 1, ymax = lat + 1)) - xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size
- a + **geom_ribbon**(aes(ymin=unemploy - 900, ymax=unemploy + 900)) - x, ymax, ymin, alpha, color, fill, group, linetype, size

LINE SEGMENTS

- common aesthetics: x, y, alpha, color, linetype, size
- b + **geom_abline**(aes(intercept=0, slope=1))
 - b + **geom_hline**(aes(yintercept = lat))
 - b + **geom_vline**(aes(xintercept = long))

- b + **geom_segment**(aes(yend=lat+1, xend=long+1))
- b + **geom_spoke**(aes(angle = 1:1155, radius = 1))

ONE VARIABLE continuous

- c + **geom_area**(stat = "bin") - x, y, alpha, color, fill, linetype, size
- c + **geom_density**(kernel = "gaussian") - x, y, alpha, color, fill, group, linetype, size, weight
- c + **geom_dotplot**() - x, y, alpha, color, fill
- c + **geom_freqpoly**() - x, y, alpha, color, group, linetype, size
- c + **geom_histogram**(binwidth = 5) - x, y, alpha, color, fill, linetype, size, weight
- c2 + **geom_qq**(aes(sample = hwy)) - x, y, alpha, color, fill, linetype, size, weight

discrete

- d - **ggplot**(mpg, aes(f1))
- d + **geom_bar**() - x, alpha, color, fill, linetype, size, weight

TWO VARIABLES

continuous x , continuous y

- e - **ggplot**(mpg, aes(cty, hwy))
- e + **geom_label**(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

- e + **geom_jitter**(height = 2, width = 2) - x, y, alpha, color, fill, shape, size

- e + **geom_point**() - x, y, alpha, color, fill, shape, size, stroke

- e + **geom_quantile**() - x, y, alpha, color, group, linetype, size, weight

- e + **geom_rug**(sides = "bl") - x, y, alpha, color, linetype, size

- e + **geom_smooth**(method = lm) - x, y, alpha, color, fill, group, linetype, size, weight

- A B + **geom_text**(aes(label = cty), nudge_x = 1, nudge_y = 1, check_overlap = TRUE) - x, y, label, alpha, angle, color, family, fontface, hjust, lineheight, size, vjust

discrete x , continuous y

- f - **ggplot**(mpg, aes(class, hwy))

- f + **geom_col**() - x, y, alpha, color, fill, group, linetype, size

- f + **geom_boxplot**() - x, y, lower, middle, upper, ymax, ymin, alpha, color, fill, group, linetype, shape, size, weight

- f + **geom_dotplot**(binaxis = "y", stackdir = "center") - x, y, alpha, color, fill, group

- f + **geom_violin**(scale = "area") - x, y, alpha, color, fill, group, linetype, size, weight

discrete x , discrete y

- g - **ggplot**(diamonds, aes(cut, color))

- g + **geom_count**() - x, y, alpha, color, fill, shape, size, stroke

THREE VARIABLES

- seals\$z - with(seals, sqrt(delta_long^2 + delta_lat^2))
- l - **ggplot**(seals, aes(long, lat))

- l + **geom_contour**(aes(z = z)) - x, y, z, alpha, colour, group, linetype, size, weight

continuous bivariate distribution

- h - **ggplot**(diamonds, aes(carat, price))
- h + **geom_bin2d**(binwidth = c(0.25, 500)) - x, y, alpha, color, fill, linetype, size, weight

- h + **geom_density2d**() - x, y, alpha, colour, group, linetype, size
- h + **geom_hex**() - x, y, alpha, colour, fill, size

continuous function

- i - **ggplot**(economics, aes(date, unemploy))

- i + **geom_area**() - x, y, alpha, color, fill, linetype, size
- i + **geom_line**() - x, y, alpha, color, group, linetype, size
- i + **geom_step**(direction = "hv") - x, y, alpha, color, group, linetype, size

visualizing error

- df - **data.frame**(grp = c("A", "B"), fit = 4.5, se = 1.2)
- j - **ggplot**(df, aes(grp, fit, ymin = fit-se, ymax = fit+se))

- j + **geom_crossbar**(fatten = 2) - x, y, ymax, ymin, alpha, color, fill, group, linetype, size
- j + **geom_errorbar**() - x, y, ymax, ymin, alpha, color, group, linetype, size, width (also **geom_errorbarh**())
- j + **geom_linerange**() - x, ymin, ymax, alpha, color, group, linetype, size
- j + **geom_pointrange**() - x, y, ymin, ymax, alpha, color, fill, group, linetype, shape, size

maps

- data - **data.frame**(murder = USArrests\$Murder, state = tolower(rownames(USArrests)))
- map - **map_data**("state")
- k - **ggplot**(data, aes(fill = murder))

- k + **geom_map**(aes(map_id = state), map = map) + **expand_limits**(x = map\$long, y = map\$lat), map_id, alpha, color, fill, linetype, size

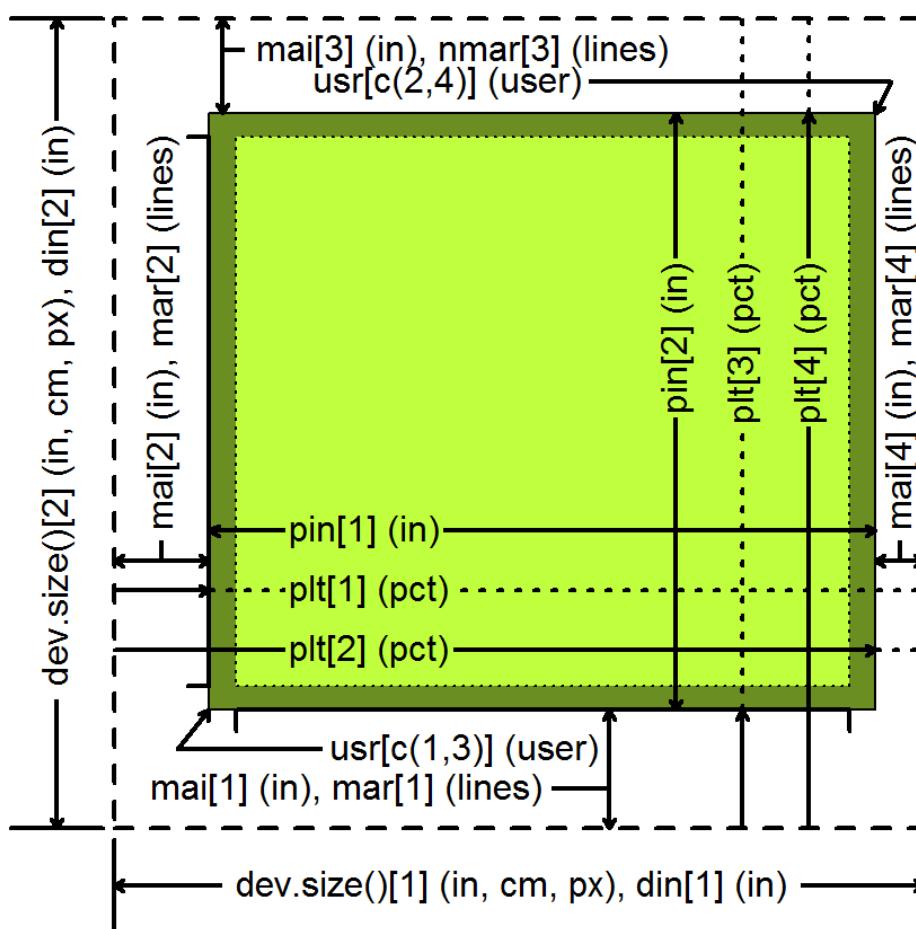
- l + **geom_raster**(aes(fill = z), hjust=0.5, vjust=0.5, interpolate=FALSE) - x, y, alpha, fill
- l + **geom_tile**(aes(fill = z)) - x, y, alpha, color, fill, linetype, size, width

How Big is Your Graph?

An R Cheat Sheet

Introduction

All functions that open a device for graphics will have **height** and **width** arguments to control the size of the graph and a **pointsize** argument to control the relative font size. In **knitr**, you control the size of the graph with the chunk options, **fig.width** and **fig.height**. This sheet will help you with calculating the size of the graph and various parts of the graph within R.



Your graphics device

dev.size() (width, height)
par("din") (r.o.) (width, height) in inches

Both the **dev.size** function and the **din** argument of **par** will tell you the size of the graphics device. The **dev.size** function will report the size in

1. inches (**units="in"**), the default
2. centimeters (**units="cm"**)
3. pixels (**units="px"**)

Like several other **par** arguments, **din** is read only (r.o.) meaning that you can ask its current value (**par("din")**) but you cannot change it (**par(din=c(5,7))** will fail).

Your plot margins

par("mai") (bottom, left, top, right) in inches
par("mar") (bottom, left, top, right) in lines

Margins provide you space for your axes, axis labels, and titles.

A "line" is the amount of vertical space needed for a line of text.

If your graph has no axes or titles, you can remove the margins (and maximize the plotting region) with

```
par(mar=rep(0,4))
```

Your plotting region

par("pin") (width, height) in inches
par("plt") (left, right, bottom, top) in pct

The **pin** argument **par** gives you the size of the plotting region (the size of the device minus the size of the margins) in inches.

The **plt** argument **par** gives you the percentage of the device from the left/bottom edge up to the left edge of the plotting region, the right edge, the bottom edge, and the top edge. The first and third values are equivalent to the percentage of space devoted to the left and bottom margins. Subtract the second and fourth values from 1 to get the percentage of space devoted to the right and top margins.

Your x-y coordinates

par("usr") (xmin, ymin, xmax, ymax)

Your x-y coordinates are the values you use when plotting your data. This normally is not the same as the values you specified with the **xlim** and **ylim** arguments in **plot**. By default, R adds an extra 4% to the plotting range (see the dark green region on the figure) so that points right up on the edges of your plot do not get partially clipped. You can override this by setting **xaxs="i"** and/or the **yaxs="i"** in **par**.

Run **par("usr")** to find the minimum X value, the maximum X value, the minimum Y value, and the maximum Y value. If you assign new values to **usr**, you will update the x-y coordinates to the new values.

Getting a square graph

par("pty")

You can produce a square graph manually by setting the width and height to the same value and setting the margins so that the sum of the top and bottom margins equal the sum of the left and right margins. But a much easier way is to specify **pty="s"**, which adjusts the margins so that the size of the plotting region is always square, even if you resize the graphics window.

Converting units

For many applications, you need to be able to translate user coordinates to pixels or inches. There are some cryptic shortcuts, but the simplest way is to get the range in user coordinates and measure the proportion of the graphics device devoted to the plotting region.

```
user.range <- par("usr")[c(2,4)] - par("usr")[c(1,3)]
```

```
region.pct <- par("plt")[c(2,4)] - par("plt")[c(1,3)]
```

```
region.px <- dev.size(units="px") * region.pct
```

```
px.per.xy <- region.px / user.range
```

To convert a horizontal or distance from the x-coordinate value to pixels, multiply by **px.per.xy[1]**. To convert a vertical distance, multiply by **region.px.per.xy[2]**. To convert a diagonal distance, you need to invoke Pythagoras.

```
a.px <- x.dist*px.per.xy[1]
b.px <- y.dist*px.per.xy[2]
c.px <- sqrt(a.px^2+b.px^2)
```

To rotate a string to match the slope of a line segment, you need to convert the distances to pixels, calculate the arctangent, and convert from radians to degrees.

```
segments(x0, y0, x1, y1)
delta.x <- (x1 - x0) * px.per.xy[1]
delta.y <- (y1 - y0) * px.per.xy[2]
angle.radians <- atan2(delta.y, delta.x)
angle.degrees <- angle.radians * 180 / pi
text(x1, y1, "TEXT", srt=angle.degrees)
```

Panels

`par("fig")` (width, height) in pct
`par("fin")` (width, height) in inches

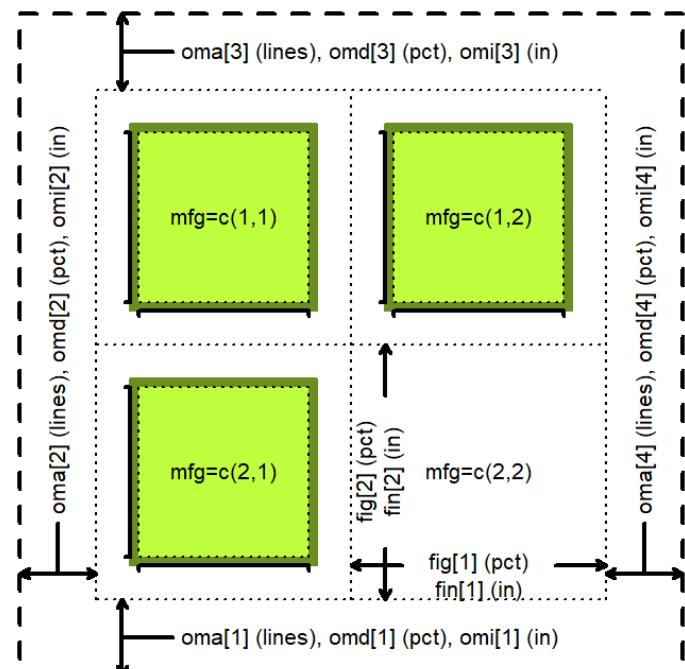
If you display multiple plots within a single graphics window (e.g., with the `mfrow` or `mfcol` arguments of `par` or with the `layout` function), then the `fig` and `fin` arguments will tell you the size of the current subplot window in percent or inches, respectively.

`par("oma")` (bottom, left, top, right) in lines
`par("omd")` (bottom, left, top, right) in pct
`par("omi")` (bottom, left, top, right) in inches

Each subplot will have margins specified by `mai` or `mar`, but no outer margin around the entire set of plots, unless you specify them using `oma`, `omd`, or `omi`. You can place text in the outer margins using the `mtext` function with the argument `outer=TRUE`.

`par("mfg")` (r, c) or (r, c, maxr, maxc)

The `mfg` argument of `par` will allow you to jump to a subplot in a particular row and column. If you query with `par("mfg")`, you will get the current row and column followed by the maximum row and column.



Character and string sizes

`strheight()`

The `strheight` functions will tell you the height of a specified string in inches (`units="inches"`), x-y user coordinates (`units="user"`) or as a percentage of the graphics device (`units="figure"`).

For a single line of text, `strheight` will give you the height of the letter "M". If you have a string with one or more linebreaks ("n"), the `strheight` function will measure the height of the letter "M" plus the height of one or more additional lines. The height of a line is dependent on the line spacing, set by the `lheight` argument of `par`. The default line height (`lheight=1`), corresponding to single spaced lines, produces a line height roughly 1.5 times the height of "M".

`strwidth()`

The `strwidth` function will produce different widths to individual characters, representing the proportional spacing used by most fonts ("W" using much more space than an "i"). For the width of a string, the `strwidth` function will sum up the lengths of the individual characters in the string.

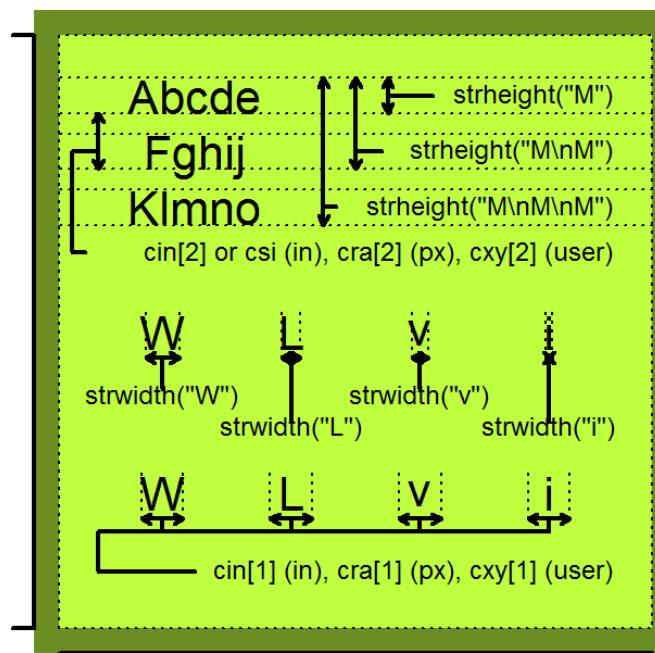
`par("cin")` (r.o.) (width, height) in inches
`par("csi")` (r.o.) height in inches
`par("cra")` (r.o.) (width, height) in pixels
`par("cxy")` (r.o.) (width, height) in xy coordinates

The single value returned by the `csi` argument of `par` gives you the height of a line of text in inches. The second of the two values returned by `cin`, `cra`, and `cxy` gives you the height of a line, in inches, pixels, or xy (user) coordinates.

The first of the two values returned by the `cin`, `cra`, and `cxy` arguments to `par` gives you the approximate width of a single character, in inches, pixels, or xy (user) coordinates. The width, very slightly smaller than the actual width of the letter "W", is a rough estimate at best and ignores the variable width of individual letters.

These values are useful, however, in providing fast ratios of the relative sizes of the differing units of measure

`px.per.in <- par("cra") / par("cin")`
`px.per.xy <- par("cra") / par("cxy")`
`xy.per.in <- par("cxy") / par("cin")`



If your fonts are too big or too small

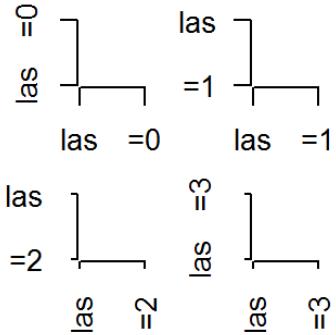
Fixing this takes a bit of trial and error.

1. Specify a larger/smaller value for the `pointsize` argument when you open your graphics device.
2. Try opening your graphics device with different values for `height` and `width`. Fonts that look too big might be better proportioned in a larger graphics window.
3. Use the `cex` argument to increase or decrease the relative size of your fonts.

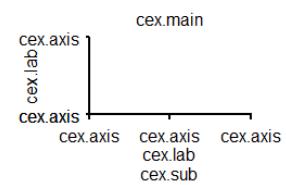
If your axes don't fit

There are several possible solutions.

1. You can assign wider margins using the `mar` or `mai` argument in `par`.
2. You can change the orientation of the axis labels with `las`. Choose among
 - a. `las=0` both axis labels parallel
 - b. `las=1` both axis labels horizontal
 - c. `las=2` both axis labels perpendicular
 - d. `las=3` both axis labels vertical.



3. change the relative size of the font
 - a. `cex.axis` for the tick mark labels.
 - b. `cex.lab` for `xlab` and `ylab`.
 - c. `cex.main` for the main title
 - d. `cex.sub` for the subtitle.



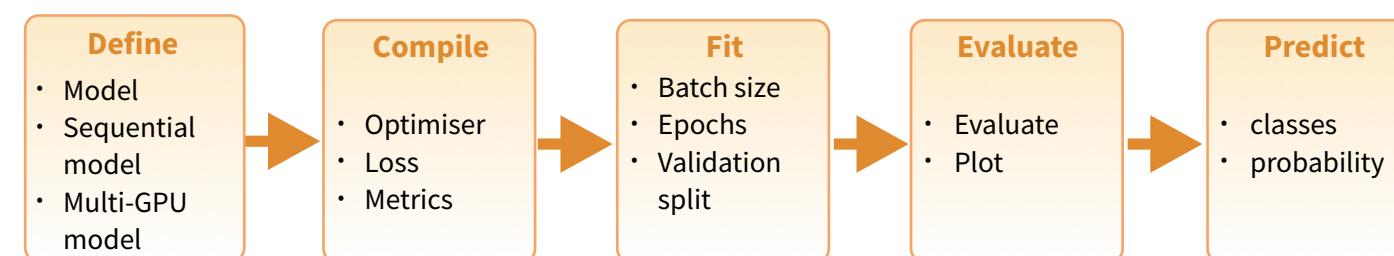
Deep Learning with Keras :: CHEAT SHEET



Intro

[Keras](#) is a high-level neural networks API developed with a focus on enabling fast experimentation. It supports multiple backends, including TensorFlow, CNTK and Theano.

TensorFlow is a lower level mathematical library for building deep neural network architectures. The [keras](#) R package makes it easy to use Keras and TensorFlow in R.



<https://keras.rstudio.com>

<https://www.manning.com/books/deep-learning-with-r>

The “Hello, World!”
of deep learning

Working with keras models

DEFINE A MODEL

`keras_model()` Keras Model

`keras_model_sequential()` Keras Model composed of a linear stack of layers

`multi_gpu_model()` Replicates a model on different GPUs

COMPILE A MODEL

`compile(object, optimizer, loss, metrics = NULL)`

Configure a Keras model for training

FIT A MODEL

`fit(object, x = NULL, y = NULL, batch_size = NULL, epochs = 10, verbose = 1, callbacks = NULL, ...)`
Train a Keras model for a fixed number of epochs (iterations)

`fit_generator()` Fits the model on data yielded batch-by-batch by a generator

`train_on_batch(); test_on_batch()` Single gradient update or model evaluation over one batch of samples

EVALUATE A MODEL

`evaluate(object, x = NULL, y = NULL, batch_size = NULL)` Evaluate a Keras model

`evaluate_generator()` Evaluates the model on a data generator

PREDICT

`predict()` Generate predictions from a Keras model

`predict_proba() and predict_classes()`

Generates probability or class probability predictions for the input samples

`predict_on_batch()` Returns predictions for a single batch of samples

`predict_generator()` Generates predictions for the input samples from a data generator

OTHER MODEL OPERATIONS

`summary()` Print a summary of a Keras model

`export_savedmodel()` Export a saved model

`get_layer()` Retrieves a layer based on either its name (unique) or index

`pop_layer()` Remove the last layer in a model

`save_model_hdf5(); load_model_hdf5()` Save/Load models using HDF5 files

`serialize_model(); unserialize_model()`

Serialize a model to an R object

`clone_model()` Clone a model instance

`freeze_weights(); unfreeze_weights()`

Freeze and unfreeze weights

CORE LAYERS



`layer_input()` Input layer



`layer_dense()` Add a densely-connected NN layer to an output



`layer_activation()` Apply an activation function to an output



`layer_dropout()` Applies Dropout to the input



`layer_reshape()` Reshapes an output to a certain shape



`layer_permute()` Permute the dimensions of an input according to a given pattern



`layer_repeat_vector()` Repeats the input n times



`layer_lambda(object, f)` Wraps arbitrary expression as a layer



`layer_activity_regularization()` Layer that applies an update to the cost function based on input activity



`layer_masking()` Masks a sequence by using a mask value to skip timesteps



`layer_flatten()` Flattens an input

INSTALLATION

The [keras](#) R package uses the Python [keras](#) library. You can install all the prerequisites directly from R.

https://keras.rstudio.com/reference/install_keras.html

```
library(keras)  
install_keras()
```

See `?keras_install`
for GPU instructions

This installs the required libraries in an Anaconda environment or virtual environment '`r-tensorflow`'.

TRAINING AN IMAGE RECOGNIZER ON MNIST DATA

input layer: use MNIST images

5041

```
mnist <- dataset_mnist()  
x_train <- mnist$train$x; y_train <- mnist$train$y  
x_test <- mnist$test$x; y_test <- mnist$test$y
```

reshape and rescale

```
x_train <- array_reshape(x_train, c(nrow(x_train), 784))  
x_test <- array_reshape(x_test, c(nrow(x_test), 784))  
x_train <- x_train / 255; x_test <- x_test / 255
```

```
y_train <- to_categorical(y_train, 10)  
y_test <- to_categorical(y_test, 10)
```

defining the model and layers

```
model <- keras_model_sequential()  
model %>%  
  layer_dense(units = 256, activation = 'relu',  
             input_shape = c(784)) %>%  
  layer_dropout(rate = 0.4) %>%  
  layer_dense(units = 128, activation = 'relu') %>%  
  layer_dense(units = 10, activation = 'softmax')
```

compile (define loss and optimizer)

```
model %>% compile(  
  loss = 'categorical_crossentropy',  
  optimizer = optimizer_rmsprop(),  
  metrics = c('accuracy'))
```

train (fit)

```
model %>% fit(  
  x_train, y_train,  
  epochs = 30, batch_size = 128,  
  validation_split = 0.2)  
model %>% evaluate(x_test, y_test)  
model %>% predict_classes(x_test)
```

More layers

CONVOLUTIONAL LAYERS

	<code>layer_conv_1d()</code> 1D, e.g. temporal convolution
	<code>layer_conv_2d_transpose()</code> Transposed 2D (deconvolution)
	<code>layer_conv_2d()</code> 2D, e.g. spatial convolution over images
	<code>layer_conv_3d_transpose()</code> Transposed 3D (deconvolution) <code>layer_conv_3d()</code> 3D, e.g. spatial convolution over volumes
	<code>layer_conv_lstm_2d()</code> Convolutional LSTM
	<code>layer_separable_conv_2d()</code> Depthwise separable 2D
	<code>layer_upsampling_1d()</code> <code>layer_upsampling_2d()</code> <code>layer_upsampling_3d()</code> Upsampling layer
	<code>layer_zero_padding_1d()</code> <code>layer_zero_padding_2d()</code> <code>layer_zero_padding_3d()</code> Zero-padding layer
	<code>layer_cropping_1d()</code> <code>layer_cropping_2d()</code> <code>layer_cropping_3d()</code> Cropping layer

POOLING LAYERS

	<code>layer_max_pooling_1d()</code> <code>layer_max_pooling_2d()</code> <code>layer_max_pooling_3d()</code> Maximum pooling for 1D to 3D
	<code>layer_average_pooling_1d()</code> <code>layer_average_pooling_2d()</code> <code>layer_average_pooling_3d()</code> Average pooling for 1D to 3D
	<code>layer_global_max_pooling_1d()</code> <code>layer_global_max_pooling_2d()</code> <code>layer_global_max_pooling_3d()</code> Global maximum pooling
	<code>layer_global_average_pooling_1d()</code> <code>layer_global_average_pooling_2d()</code> <code>layer_global_average_pooling_3d()</code> Global average pooling

ACTIVATION LAYERS

	<code>layer_activation()</code> object, activation Apply an activation function to an output
	<code>layer_activation_leaky_relu()</code> Leaky version of a rectified linear unit
	<code>layer_activation_parametric_relu()</code> Parametric rectified linear unit
	<code>layer_activation_thresholded_relu()</code> Thresholded rectified linear unit
	<code>layer_activation_elu()</code> Exponential linear unit

DROPOUT LAYERS

	<code>layer_dropout()</code> Applies dropout to the input
	<code>layer_spatial_dropout_1d()</code> <code>layer_spatial_dropout_2d()</code> <code>layer_spatial_dropout_3d()</code> Spatial 1D to 3D version of dropout

RECURRENT LAYERS

	<code>layer_simple_rnn()</code> Fully-connected RNN where the output is to be fed back to input
	<code>layer_gru()</code> Gated recurrent unit - Cho et al
	<code>layer_cudnn_gru()</code> Fast GRU implementation backed by CuDNN

	<code>layer_lstm()</code> Long-Short Term Memory unit - Hochreiter 1997
	<code>layer_cudnn_lstm()</code> Fast LSTM implementation backed by CuDNN

LOCALLY CONNECTED LAYERS

	<code>layer_locally_connected_1d()</code> <code>layer_locally_connected_2d()</code> Similar to convolution, but weights are not shared, i.e. different filters for each patch
---	--

Preprocessing

SEQUENCE PREPROCESSING

`pad_sequences()`
Pads each sequence to the same length (length of the longest sequence)

`skipgrams()`
Generates skipgram word pairs

`make_sampling_table()`
Generates word rank-based probabilistic sampling table

TEXT PREPROCESSING

`text_tokenizer()` Text tokenization utility

`fit_text_tokenizer()` Update tokenizer internal vocabulary

`save_text_tokenizer(); load_text_tokenizer()`
Save a text tokenizer to an external file

`texts_to_sequences(); texts_to_sequences_generator()`
Transforms each text in texts to sequence of integers

`texts_to_matrix(); sequences_to_matrix()`
Convert a list of sequences into a matrix

`text_one_hot()` One-hot encode text to word indices

`text_hashing_trick()`
Converts a text to a sequence of indexes in a fixed-size hashing space

`text_to_word_sequence()`
Convert text to a sequence of words (or tokens)

IMAGE PREPROCESSING

`image_load()` Loads an image into PIL format.

`flow_images_from_data(); flow_images_from_directory()`
Generates batches of augmented/normalized data from images and labels, or a directory

`image_data_generator()` Generate minibatches of image data with real-time data augmentation.

`fit_image_data_generator()` Fit image data generator internal statistics to some sample data

`generator_next()` Retrieve the next item

`image_to_array(); image_array_resize(); image_array_save()` 3D array representation

Keras applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.
`application_xception()`
`xception_preprocess_input()`
 Xception v1 model

`application_inception_v3()`
`inception_v3_preprocess_input()`
 Inception v3 model, with weights pre-trained on ImageNet

`application_inception_resnet_v2()`
`inception_resnet_v2_preprocess_input()`
 Inception-ResNet v2 model, with weights trained on ImageNet

`application_vgg16(); application_vgg19()`
 VGG16 and VGG19 models

`application_resnet50()` ResNet50 model
`application_mobilenet()`
`mobilenet_preprocess_input()`
`mobilenet_decode_predictions()`
`mobilenet_load_model_hdf5()`
 MobileNet model architecture

IMAGENET

[ImageNet](#) is a large database of images with labels, extensively used for deep learning

`imagenet_preprocess_input()`
`imagenet_decode_predictions()`
 Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions

Callbacks

A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

`callback_early_stopping()` Stop training when a monitored quantity has stopped improving
`callback_learning_rate_scheduler()` Learning rate scheduler
`callback_tensorboard()` TensorBoard basic visualizations

LATEX 2 ϵ Cheat Sheet

Document classes

`book` Default is two-sided.
`report` No `\part` divisions.
`article` No `\part` or `\chapter` divisions.
`letter` Letter (?).
`slides` Large sans-serif font.

Used at the very beginning of a document: `\documentclass{class}`. Use `\begin{document}` to start contents and `\end{document}` to end the document.

Common documentclass options

`10pt/11pt/12pt` Font size.
`letterpaper/a4paper` Paper size.
`twocolumn` Use two columns.
`twoside` Set margins for two-sided.
`landscape` Landscape orientation. Must use `dvips -t landscape`.
`draft` Double-space lines.
Usage: `\documentclass[opt,opt]{class}`.

Packages

`fullpage` Use 1 inch margins.
`anysize` Set margins: `\marginsize{l}{r}{t}{b}`.
`multicol` Use n columns: `\begin{multicols}{n}`.
`latexsym` Use LATEX symbol font.
`graphicx` Show image: `\includegraphics[width=x]{file}`.
`url` Insert URL: `\url{http://...}`.
Use before `\begin{document}`. Usage: `\usepackage{package}`

Title

`\author{text}` Author of document.
`\title{text}` Title of document.
`\date{text}` Date.
These commands go before `\begin{document}`. The declaration `\maketitle` goes at the top of the document.

Miscellaneous

`\pagestyle{empty}` Empty header, footer and no page numbers.
`\tableofcontents` Add a table of contents here.

Document structure

`\part{title}` `\subsubsection{title}`
`\chapter{title}` `\paragraph{title}`
`\section{title}` `\subparagraph{title}`
`\subsection{title}`
Use `\setcounter{secnumdepth}{x}` suppresses heading numbers of depth $> x$, where `chapter` has depth 0. Use a *, as in `\section*{title}`, to not number a particular item—these items will also not appear in the table of contents.

Text environments

`\begin{comment}` Comment (not printed). Requires `verbatim` package.
`\begin{quote}` Indented quotation block.
`\begin{quotation}` Like quote with indented paragraphs.
`\begin{verse}` Quotation block for verse.

Lists

`\begin{enumerate}` Numbered list.
`\begin{itemize}` Bulleted list.
`\begin{description}` Description list.
`\item text` Add an item.
`\item[x] text` Use x instead of normal bullet or number. Required for descriptions.

References

`\label{marker}` Set a marker for cross-reference, often of the form `\label{sec:item}`.
`\ref{marker}` Give section/body number of marker.
`\pageref{marker}` Give page number of marker.
`\footnote{text}` Print footnote at bottom of page.

Floating bodies

`\begin{table}[place]` Add numbered table.
`\begin{figure}[place]` Add numbered figure.
`\begin{equation}[place]` Add numbered equation.
`\caption{text}` Caption for the body.
The `place` is a list valid placements for the body. `t=top`, `h=here`, `b=bottom`, `p=separate page`, `!=place even if ugly`. Captions and label markers should be within the environment.

Text properties

Font face

Command	Declaration	Effect
<code>\textrm{text}</code>	<code>\rmfamily text</code>	Roman family
<code>\textsf{text}</code>	<code>\sffamily text</code>	Sans serif family
<code>\texttt{text}</code>	<code>\ttfamily text</code>	Typewriter family
<code>\textmd{text}</code>	<code>\mdseries text</code>	Medium series
<code>\textbf{text}</code>	<code>\bfseries text</code>	Bold series
<code>\textup{text}</code>	<code>\upshape text</code>	Upright shape
<code>\textit{text}</code>	<code>\itshape text</code>	<i>Italic shape</i>
<code>\textsl{text}</code>	<code>\slshape text</code>	Slanted shape
<code>\textsc{text}</code>	<code>\scshape text</code>	SMALL CAPS SHAPE
<code>\emph{text}</code>	<code>\em text</code>	<i>Emphasized</i>
<code>\textnormal{text}</code>	<code>\normalfont text</code>	Document font
<code>\underline{text}</code>		<u>Underline</u>

The command (tttt) form handles spacing better than the declaration (tttt) form.

Font size

<code>\tiny</code>	<code>tiny</code>	<code>\Large</code> Large
<code>\scriptsize</code>	<code>scriptsize</code>	<code>\LARGE</code> LARGE
<code>\footnotesize</code>	<code>footnotesize</code>	
<code>\small</code>	<code>small</code>	<code>\huge</code> huge
<code>\normalsize</code>	<code>normalsize</code>	<code>\Huge</code> Huge
<code>\large</code>	<code>large</code>	

These are declarations and should be used in the form `{\small ...}`, or without braces to affect the entire document.

Verbatim text

`\begin{verbatim}` Verbatim environment.
`\begin{verbatim*}` Spaces are shown as `_`.
`\verb!text!` Text between the delimiting characters (in this case '!') is verbatim.

Justification

Environment	Declaration
<code>\begin{center}</code>	<code>\centering</code>
<code>\begin{flushleft}</code>	<code>\raggedright</code>
<code>\begin{flushright}</code>	<code>\raggedleft</code>

Miscellaneous

`\linespread{x}` changes the line spacing by the multiplier x .

Text-mode symbols

Symbols

<code>\&</code>	<code>\&</code>	<code>-</code>	<code>_</code>	<code>\ldots</code>	<code>\ldots</code>	<code>\textbullet</code>
<code>\\$</code>	<code>\\$</code>	<code>^</code>	<code>\^{}{}</code>	<code> </code>	<code>\textbar</code>	<code>\backslash</code>
<code>\%</code>	<code>\%</code>	<code>~</code>	<code>\~{}{}</code>	<code>#</code>	<code>\#</code>	<code>\\$</code>

Accents

<code>\`o</code>	<code>\`o</code>	<code>\`o</code>	<code>\`o</code>	<code>\`o</code>	<code>\`o</code>	<code>\`o</code>
<code>\.o</code>	<code>\.o</code>	<code>\.o</code>	<code>\.o</code>	<code>\.o</code>	<code>\.o</code>	<code>\.o</code>
<code>\c{c}</code>	<code>\c{c}</code>	<code>\c{d} o</code>	<code>\c{b} o</code>	<code>\c{t} oo</code>	<code>\c{o}e</code>	<code>\c{oe}</code>
<code>\OE</code>	<code>\OE</code>	<code>\ae</code>	<code>\AE</code>	<code>\aa</code>	<code>\AA</code>	<code>\AA</code>
<code>\o{o}</code>	<code>\o{o}</code>	<code>\O O</code>	<code>\l l</code>	<code>\L L</code>	<code>\i i</code>	<code>\i i</code>
<code>\j{j}</code>	<code>\j{j}</code>	<code>\~{`}</code>	<code>\?{`}</code>			

Delimiters

<code>‘‘</code>	<code>‘‘</code>	<code>{</code>	<code>\{</code>	<code>[</code>	<code>[</code>	<code>(</code>	<code>(</code>	<code><</code>	<code>\textless</code>
<code>‘‘</code>	<code>‘‘</code>	<code>}</code>	<code>\}</code>	<code>]</code>	<code>]</code>	<code>)</code>	<code>)</code>	<code>></code>	<code>\textgreater</code>

Dashes

Name	Source	Example	Usage
hyphen	-	X-ray	In words.
en-dash	--	1–5	Between numbers.
em-dash	---	Yes—or no?	Punctuation.

Line and page breaks

`\`\\` Begin new line without new paragraph.
`\`*` Prohibit pagebreak after linebreak.
`\kill` Don't print current line.
`\pagebreak` Start new page.
`\noindent` Do not indent current line.

Miscellaneous

`\today` March 28, 2017.
`\sim` Prints \sim instead of `\~{}`, which makes `\~{}`.
`\`.` Space, disallow linebreak (W.J.\`Clinton).
`\@.` Indicate that the `.` ends a sentence when following an uppercase letter.
`\hspace{l}` Horizontal space of length l (Ex: $l = 20pt$).
`\vspace{l}` Vertical space of length l .
`\rule{w}{h}` Line of width w and height h .

Tabular environments

tabbing environment

`\`=` Set tab stop. `\`>` Go to tab stop.
Tab stops can be set on “invisible” lines with `\kill` at the end of the line. Normally `\`\\` is used to separate lines.

tabular environment

```
\begin{array}{[pos]}{cols}
\begin{tabular}{[pos]}{cols}
\begin{tabular*}{width}{[pos]}{cols}
```

tabular column specification

l Left-justified column.
c Centered column.
r Right-justified column.
 $p\{width\}$ Same as `\parbox[t]\{width\}`.
 $\@{decl}$ Insert *decl* instead of inter-column space.
| Inserts a vertical line between columns.

tabular elements

`\hline` Horizontal line between rows.
`\cline{x-y}` Horizontal line across columns *x* through *y*.
`\multicolumn{n}{cols}{text}` A cell that spans *n* columns, with *cols* column specification.

Math mode

For inline math, use `\(...\)` or `$...$`. For displayed math, use `\[...\]` or `\begin{equation}`.

Superscript x $^{\{x\}}$ Subscript x $_{\{x\}}$
 $\frac{x}{y}$ $\frac{\{x\}}{\{y\}}$ $\sum_{k=1}^n$ $\sum_{\{k=1\}}^n$
 $\sqrt[n]{x}$ $\sqrt[n]{\{x\}}$ $\prod_{k=1}^n$ $\prod_{\{k=1\}}^n$

Math-mode symbols

$\leq \backslash leq$ $\geq \backslash geq$ $\neq \backslash neq$ $\approx \backslash approx$
 $\times \backslash times$ $\div \backslash div$ $\pm \backslash pm$ $\cdot \backslash cdot$
 $\circ \backslash circ$ $\circ \backslash circ$ $/ \backslash prime$ $\dots \backslash cdots$
 $\infty \backslash infty$ $\neg \backslash neg$ $\wedge \backslash wedge$ $\vee \backslash vee$
 $\supset \backslash supset$ $\forall \backslash forall$ $\in \backslash in$ $\rightarrow \backslash rightarrow$
 $\subset \backslash subset$ $\exists \backslash exists$ $\notin \backslash notin$ $\Rightarrow \backslash Rightarrow$
 $\cup \backslash cup$ $\cap \backslash cap$ $\mid \backslash mid$ $\Leftrightarrow \backslash Leftrightarrow$
 $\dot{a} \backslash dot a$ $\hat{a} \backslash hat a$ $\bar{a} \backslash bar a$ $\tilde{a} \backslash tilde a$
 $\alpha \backslash alpha$ $\beta \backslash beta$ $\gamma \backslash gamma$ $\delta \backslash delta$
 $\epsilon \backslash epsilon$ $\zeta \backslash zeta$ $\eta \backslash eta$ $\varepsilon \backslash varepsilon$
 $\theta \backslash theta$ $\iota \backslash iota$ $\kappa \backslash kappa$ $\vartheta \backslash vartheta$
 $\lambda \backslash lambda$ $\mu \backslash mu$ $\nu \backslash nu$ $\xi \backslash xi$
 $\pi \backslash pi$ $\rho \backslash rho$ $\sigma \backslash sigma$ $\tau \backslash tau$
 $\upsilon \backslash upsilon$ $\phi \backslash phi$ $\chi \backslash chi$ $\psi \backslash psi$
 $\omega \backslash omega$ $\Gamma \backslash Gamma$ $\Delta \backslash Delta$ $\Theta \backslash Theta$
 $\Lambda \backslash Lambda$ $\Xi \backslash Xi$ $\Pi \backslash Pi$ $\Sigma \backslash Sigma$
 $\Upsilon \backslash Upsilon$ $\Phi \backslash Phi$ $\Psi \backslash Psi$ $\Omega \backslash Omega$

Bibliography and citations

When using $\text{BIB}\text{\TeX}$, you need to run `latex`, `bibtex`, and `latex` twice more to resolve dependencies.

Citation types

`\cite{key}` Full author list and year. (Watson and Crick 1953)
`\citeA{key}` Full author list. (Watson and Crick)
`\citeN{key}` Full author list and year. Watson and Crick (1953)
`\shortcite{key}` Abbreviated author list and year. ?
`\shortciteA{key}` Abbreviated author list. ?
`\shortciteN{key}` Abbreviated author list and year. ?
`\citeyear{key}` Cite year only. (1953)
All the above have an NP variant without parentheses; Ex. `\citeNP`.

BIB \TeX entry types

`@article` Journal or magazine article.
`@book` Book with publisher.
`@booklet` Book without publisher.
`@conference` Article in conference proceedings.
`@inbook` A part of a book and/or range of pages.
`@incollection` A part of book with its own title.
`@misc` If nothing else fits.
`@phdthesis` PhD. thesis.
`@proceedings` Proceedings of a conference.
`@techreport` Tech report, usually numbered in series.
`@unpublished` Unpublished.

BIB \TeX fields

`address` Address of publisher. Not necessary for major publishers.
`author` Names of authors, of format
`booktitle` Title of book when part of it is cited.
`chapter` Chapter or section number.
`edition` Edition of a book.
`editor` Names of editors.
`institution` Sponsoring institution of tech. report.
`journal` Journal name.
`key` Used for cross ref. when no author.
`month` Month published. Use 3-letter abbreviation.
`note` Any additional information.
`number` Number of journal or magazine.
`organization` Organization that sponsors a conference.
`pages` Page range (2,6,9–12).
`publisher` Publisher's name.
`school` Name of school (for thesis).
`series` Name of series of books.
`title` Title of work.
`type` Type of tech. report, ex. "Research Note".
`volume` Volume of a journal or book.
`year` Year of publication.

Not all fields need to be filled. See example below.

Common BIB \TeX style files

abbrv	Standard	abstract	alpha with abstract
alpha	Standard	apa	APA
plain	Standard	unsrt	Unsorted

The \TeX document should have the following two lines just before `\end{document}`, where `bibfile.bib` is the name of the BIB \TeX file.

```
\bibliographystyle{plain}
\bibliography{bibfile}
```

BIB \TeX example

The BIB \TeX database goes in a file called `file.bib`, which is processed with `bibtex` file.

```
@String{N = {Na\text{\textendash}ture}}
@Article{WC:1953,
  author = {James Watson and Francis Crick},
  title = {A structure for Deoxyribose Nucleic Acid},
  journal = N,
  volume = {171},
  pages = {737},
  year = 1953
}
```

Sample \TeX document

```
\documentclass[11pt]{article}
\usepackage{fullpage}
\title{Template}
\author{Name}
\begin{document}
\maketitle

\section{section}
\subsection*[subsection without number]
text \textbf{bold text} text. Some math: $2+2=5$ 
\subsection{subsection}
text \emph{emphasized text} text. \cite{WC:1953} discovered the structure of DNA.
```

A table:

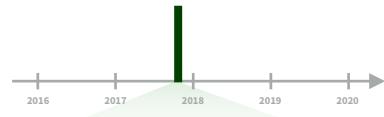
```
\begin{table}[!th]
\begin{tabular}{|l|c|r|}
\hline
first & row & data \\
second & row & data \\
\hline
\end{tabular}
\caption{This is the caption}
\label{ex:table}
\end{table}
```

The table is numbered `\ref{ex:table}`.
`\end{document}`

Dates and times with lubridate :: CHEAT SHEET



Date-times



2017-11-28 12:00:00

2017-11-28 12:00:00

A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

```
dt <- as_datetime(1511870400)
## "2017-11-28 12:00:00 UTC"
```

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a wide variety of input formats.

2017-11-28T14:02:00

ymd_hms(), ymd_hm(), ymd_h().
ymd_hms("2017-11-28T14:02:00")

2017-22-12 10:00:00

ydm_hms(), ydm_hm(), ydm_h().
ydm_hms("2017-22-12 10:00:00")

11/28/2017 1:02:03

mdy_hms(), mdy_hm(), mdy_h().
mdy_hms("11/28/2017 1:02:03")

1 Jan 2017 23:59:59

dmy_hms(), dmy_hm(), dmy_h().
dmy_hms("1 Jan 2017 23:59:59")

20170131

ymd(), ydm(). ymd(20170131)

July 4th, 2000

mdy(), myd(). mdy("July 4th, 2000")

4th of July '99

dmy(), dym(). dmy("4th of July '99")

2001: Q3

yq() Q for quarter. yq("2001: Q3")

2:01

hms::hms() Also lubridate::hms(), hm() and ms(), which return periods.* hms::hms(sec = 0, min = 1, hours = 2)

2017.5

date_decimal(decimal, tz = "UTC")
date_decimal(2017.5)



now(tzone = "") Current time in tz (defaults to system tz). now()

today(tzone = "") Current date in a tz (defaults to system tz). today()

fast.strptime() Faster strftime.
fast.strptime('9/1/01', '%y/%m/%d')

parse_date_time() Easier strftime.
parse_date_time("9/1/01", "ymd")

2017-11-28

A **date** is a day stored as the number of days since 1970-01-01

```
d <- as_date(17498)
## "2017-11-28"
```

12:00:00

An hms is a **time** stored as the number of seconds since 00:00:00

```
t <- hms::as.hms(85)
## 00:01:25
```

GET AND SET COMPONENTS

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

```
d ## "2017-11-28"
day(d) ## 28
day(d) <- 1
d ## "2017-11-01"
```

2018-01-31 11:59:59

date(x) Date component. date(dt)

2018-01-31 11:59:59

year(x) Year. year(dt)
isoyear(x) The ISO 8601 year.
epiyear(x) Epidemiological year.

2018-01-31 11:59:59

month(x, label, abbr) Month.
month(dt)

2018-01-31 11:59:59

day(x) Day of month. day(dt)
wday(x, label, abbr) Day of week.
qday(x) Day of quarter.

2018-01-31 11:59:59

hour(x) Hour. hour(dt)

2018-01-31 11:59:59

minute(x) Minutes. minute(dt)

2018-01-31 11:59:59

second(x) Seconds. second(dt)

2018-01-31 11:59:59

week(x) Week of the year. week(dt)
isoweek() ISO 8601 week.
epiweek() Epidemiological week.

2018-01-31 11:59:59

quarter(x, with_year = FALSE)
Quarter. quarter(dt)

2018-01-31 11:59:59

semester(x, with_year = FALSE)
Semester. semester(dt)

2018-01-31 11:59:59

am(x) Is it in the am? am(dt)
pm(x) Is it in the pm? pm(dt)

2018-01-31 11:59:59

dst(x) Is it daylight savings? dst(dt)

2018-01-31 11:59:59

leap_year(x) Is it a leap year?
leap_year(dt)

2018-01-31 11:59:59

update(object, ..., simple = FALSE)
update(dt, mday = 2, hour = 1)

Round Date-times



floor_date(x, unit = "second")
Round down to nearest unit.
floor_date(dt, unit = "month")



round_date(x, unit = "second")
Round to nearest unit.
round_date(dt, unit = "month")



ceiling_date(x, unit = "second", change_on_boundary = NULL)
Round up to nearest unit.
ceiling_date(dt, unit = "month")

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE)
Roll back to last day of previous month. **rollback**(dt)

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date()** and **stamp_time()**.

1. Derive a template, create a function
`sf <- stamp("Created Sunday, Jan 17, 1999 3:34")`

Tip: use a date with day > 12

2. Apply the template to dates
`sf(ymd("2010-04-05"))`
`## [1] "Created Monday, Apr 05, 2010 00:00"`

Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns one time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. **OlsonNames()**

5:00 Mountain 6:00 Central
4:00 Pacific 7:00 Eastern

PT MT CT ET
7:00 Pacific 7:00 Mountain 7:00 Central

7:00 Pacific 7:00 Mountain 7:00 Central

with_tz(time, tzone = "") Get the same date-time in a new time zone (a new clock time). **with_tz**(dt, "US/Pacific")

force_tz(time, tzone = "") Get the same clock time in a new time zone (a new date-time). **force_tz**(dt, "US/Pacific")



Math with Date-times

Lubridate provides three classes of timespans to facilitate math with dates and date-times

Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

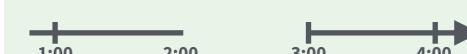
A normal day

```
nor <- ymd_hms("2018-01-01 01:30:00", tz = "US/Eastern")
```



The start of daylight savings (spring forward)

```
gap <- ymd_hms("2018-03-11 01:30:00", tz = "US/Eastern")
```



The end of daylight savings (fall back)

```
lap <- ymd_hms("2018-11-04 00:30:00", tz = "US/Eastern")
```



Leap years and leap seconds

```
leap <- ymd("2019-03-01")
```



Periods track changes in clock times, which ignore time line irregularities.

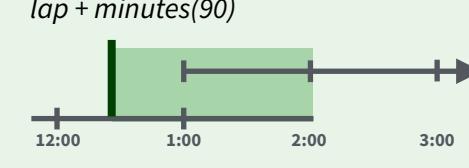
`nor + minutes(90)`



`gap + minutes(90)`



`lap + minutes(90)`

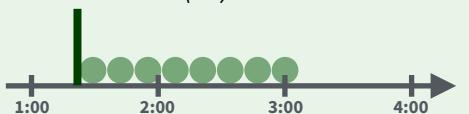


`leap + years(1)`

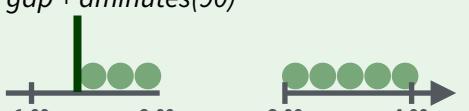


Durations track the passage of physical time, which deviates from clock time when irregularities occur.

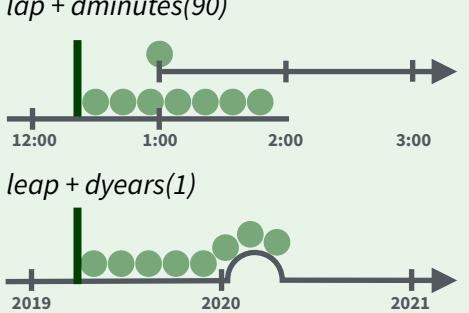
`nor + dminutes(90)`



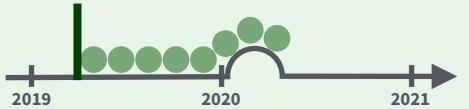
`gap + dminutes(90)`



`lap + dminutes(90)`



`leap + dyears(1)`



Intervals represent specific intervals of the timeline, bounded by start and end date-times.

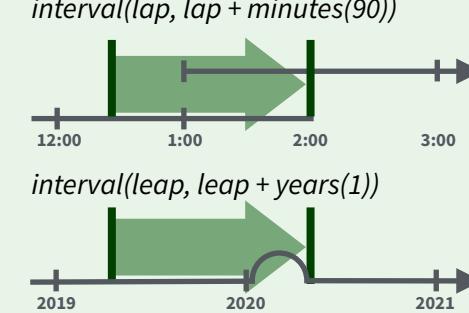
`interval(nor, nor + minutes(90))`



`interval(gap, gap + minutes(90))`



`interval(lap, lap + minutes(90))`



`interval(leap, leap + years(1))`



Not all years are 365 days due to **leap days**.

Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding **months**, e.g. February 31st

```
jan31 <- ymd(20180131)
jan31 + months(1)
## NA
```

`%m+%` and `%m-%` will roll imaginary dates to the last day of the previous month.

```
jan31 %m+% months(1)
## "2018-02-28"
```

`add_with_rollback(e1, e2, roll_to_first = TRUE)` will roll imaginary dates to the first day of the new month.

```
add_with_rollback(jan31, months(1),
roll_to_first = TRUE)
## "2018-03-01"
```

PERIODS

Add or subtract periods to model events that happen at specific clock times, like the NYSE opening bell.

Make a period with the name of a time unit **pluralized**, e.g.

```
p <- months(3) + days(12)
```

"3m 12d 0H 0M 0S"

Number of months Number of days etc.

`years(x = 1) x years.`

`months(x) x months.`

`weeks(x = 1) x weeks.`

`days(x = 1) x days.`

`hours(x = 1) x hours.`

`minutes(x = 1) x minutes.`

`seconds(x = 1) x seconds.`

`milliseconds(x = 1) x milliseconds.`

`microseconds(x = 1) x microseconds`

`nanoseconds(x = 1) x nanoseconds.`

`picoseconds(x = 1) x picoseconds.`

`period(num = NULL, units = "second", ...)`

An automation friendly period constructor.

`period(5, unit = "years")`

`as.period(x, unit)` Coerce a timespan to a period, optionally in the specified units.

Also `is.period()`. `as.period(i)`

`period_to_seconds(x)` Convert a period to the "standard" number of seconds implied by the period. Also `seconds_to_period()`.
`period_to_seconds(p)`

DURATIONS

Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length.

Difftimes are a class of durations found in base R.

Make a duration with the name of a period prefixed with a **d**, e.g.

```
dd <- ddays(14)
```

"1209600s (~2 weeks)"

Exact length in seconds Equivalent in common units

`dyears(x = 1) 31536000x seconds.`

`dweeks(x = 1) 604800x seconds.`

`ddays(x = 1) 86400x seconds.`

`dhours(x = 1) 3600x seconds.`

`dminutes(x = 1) 60x seconds.`

`dseconds(x = 1) x seconds.`

`dmilliseconds(x = 1) x × 10-3 seconds.`

`dmicroseconds(x = 1) x × 10-6 seconds.`

`dnanoseconds(x = 1) x × 10-9 seconds.`

`dpicoseconds(x = 1) x × 10-12 seconds.`

`duration(num = NULL, units = "second", ...)`

An automation friendly duration constructor. `duration(5, unit = "years")`

`as.duration(x, ...)` Coerce a timespan to a duration. Also `is.duration()`, `is.difftime()`. `as.duration(i)`

`make_difftime(x)` Make difftime with the specified number of units.
`make_difftime(99999)`

INTERVALS

Divide an interval by a duration to determine its physical length, divide and interval by a period to determine its implied length in clock time.

Make an interval with **interval()** or `%--%`, e.g.

```
i <- interval(ymd("2017-01-01"), d)
```

```
## 2017-01-01 UTC--2017-11-28 UTC
```

```
j <- d %--% ymd("2017-12-31")
```

```
## 2017-11-28 UTC--2017-12-31 UTC
```

Start Date End Date
a %within% b Does interval or date-time a fall within interval b? `now() %within% i`

`int_start(int)` Access/set the start date-time of an interval. Also `int_end()`. `int_start(i) < now(); int_start(i)`

`int_aligns(int1, int2)` Do two intervals share a boundary? Also `int_overlaps()`. `int_aligns(i, j)`

`int_diff(times)` Make the intervals that occur between the date-times in a vector.
`v <- c(dt, dt + 100, dt + 1000); int_diff(v)`

`int_flip(int)` Reverse the direction of an interval. Also `int_standardize()`. `int_flip(i)`

`int_length(int)` Length in seconds. `int_length(i)`

`int_shift(int, by)` Shifts an interval up or down the timeline by a timespan. `int_shift(i, days(-1))`

`as.interval(x, start, ...)` Coerce a timespans to an interval with the start date-time. Also `is.interval()`. `as.interval(days(1), start = now())`

Machine Learning with R



Introduction

mlr offers a unified interface for the basic building blocks of machine learning: tasks, learners, hyperparameters, etc.

Tasks contain a description of a task (classification, regression, clustering, etc.) and a data set.

Learners specify a machine learning algorithm (GLM, SVM, xgboost, etc.) and its parameters.

Hyperparameters are learner settings that can be specified directly or tuned. A **parameter set** lists the possible hyperparameters for a given learner.

Wrapped Models are learners that have been trained on a task and can be used to make predictions.

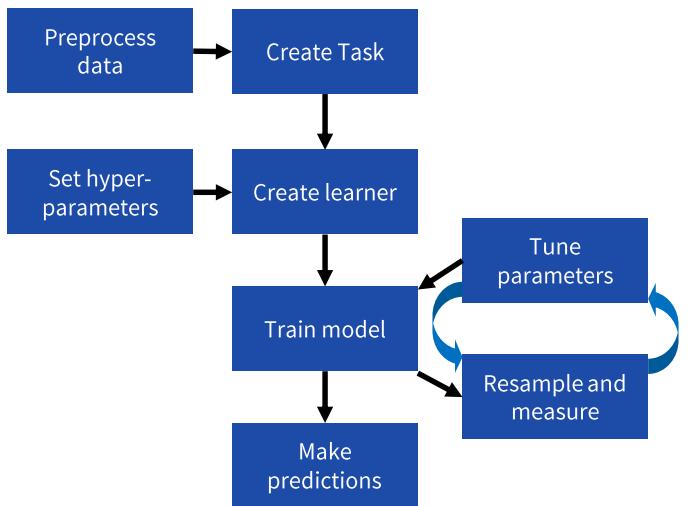
Predictions are the results of applying a model to either new data or the original training data.

Measures control how learner performance is evaluated, e.g. RMSE, LogLoss, AUC, etc.

Resampling estimates generalization performance by separating training data from test data. Common strategies include holdout and cross-validation.

Links: [Tutorial](#) | [CRAN](#) | [Github](#)

mlr workflow



Setup

Preprocessing data

`createDummyFeatures(obj=, target=, method=, cols=)`
Creates (0,1) flags for each non-numeric variable excluding `target`. Can be applied to entire dataset or only specific `cols`

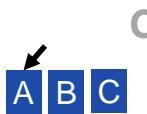
`normalizeFeatures(obj=, target=, method=, cols=, range=, on.constant=)`
Normalizes numerical features according to specified `method`:

- "center" (subtract mean)
- "scale" (divide by std. deviation)
- "standardize" (center and scale)
- "range" (linear scale to given range, default `range=c(0,1)`)

`mergeSmallFactorLevels(task=, cols=, min.perc=)`
Combine infrequent factor levels into a single merged level

`summarizeColumns(obj=)` where `obj` is a data.frame or task.
Provides type, NA, and distributional data about each column

See also `capLargeValues` `dropFeatures` `removeConstantFeatures` `summarizeLevels`

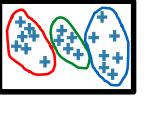


Creating a task

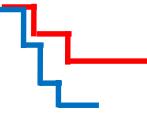
`makeClassifTask(data=, target=)`
Classification of a target variable, with optional positive class `positive`

`makeRegrTask(data=, target=)`
Regression on a target variable

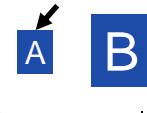
`makeMultilabelTask(data=, target=)`
Classification where the target can belong to more than one class per observation



`makeClusterTask(data=)`
Unsupervised clustering on a data set



`makeSurvTask(data=, target= c("time", "event"))`
Survival analysis with a survival time column and an event column



`makeCostSensTask(data=, costs=)`
Cost-sensitive classification where each observation-cost pair has a specified cost

Other arguments that can be passed to a `task`:

- `weights`= Weighting vector to apply to observations
- `blocking`= Factor vector where each level indicates a block of observations that will not be split up in resampling

Making a learner

`makeLearner(cl=, predict.type=, ..., par.vals=)`
Choose an algorithm class to perform the task and determine what that algorithm will predict

- `cl`=name of algorithm, e.g. `"classif.xgboost"` `"regr.randomForest"` `"cluster.kmeans"`
- `predict.type="response"` returns a prediction type that matches the source data; `"prob"` returns a predicted probability for classification problems only; `"se"` returns the standard error of the prediction for regression problems only. Only certain learners can return `"prob"` and `"se"`
- `par.vals`= takes a list of hyperparameters and passes them to the learner; parameters can also be passed directly (...)

You can make multiple learners at once with `makeLearners()`

mlr has integrated over 170 different learning algorithms

- Full list: `View(listLearners())` shows all learners
- Available learners for a task: `View(listLearners(task))`
- Filtered list: `View(listLearners("classif", properties=c("prob", "factors")))` shows all classification learners `"classif"` which can predict probabilities `"prob"` and handle factor inputs `"factors"`
- See also `getLearnerProperties()`

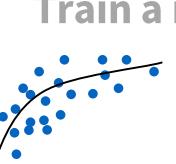
Training & Testing

Setting hyperparameters

`setHyperPars(learner=, ...)`
Set the hyperparameters (settings) for each learner, if you don't want to use the defaults. You can also specify hyperparameters in the `makeLearner()` call

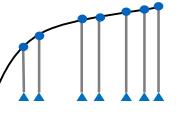


`getParamSet(learner=)`
Show the possible universe of parameters for your learner; can take a learner directly, or a text string such as `"classif.qda"`



Train a model and predict

`train(learner=, task=)`
Train a model (`WrappedModel`) by applying a learner to a task. By default, the model will train on all observations. The underlying model can be extracted with `getLearnerModel()`



`predict(object=, task=, newdata=)`
Use a trained model to make predictions on a task or dataset. The resulting `pred` object can be viewed with `View(pred)` or accessed by `as.data.frame(pred)`



Measuring performance

`performance(pred=, measures=)`
Calculate performance of predictions according to one or more of several measures (use `listMeasures()` for full list):

- `classif` `acc` `auc` `bac` `ber` `brier[,scaled]` `f1` `fdr` `fnr` `fpr` `gmean` `multiclass[,au1]` `aunp` `aunu` `brier` `npv` `ppv` `qsr` `ssr` `tn` `tnr` `tp` `tpr` `wkappa`
- `regr` `rsq` `expvar` `kendalltau` `mae` `mape` `medae` `medse` `mse` `msle` `rae` `rmse` `rmsle` `rrse` `rsq` `sae` `spearmanrho` `sse`
- `cluster` `db` `dunn` `G1` `G2` `silhouette`
- `multilabel` `multilabel[,f1]` `subset01` `.tpr` `.ppv` `.acc` `.hamloss`
- `costsens` `mcp` `meancosts`
- `surv` `cindex`
- `other` `featperc` `timeboth` `timelpredict` `timetrain`

For detailed performance data on classification tasks, use:

- `calculateConfusionMatrix(pred=)`
- `calculateROCMeasures(pred=)`

Resampling a learner

`makeResampleDesc(method=, ..., stratify=)`

`method` must be one of the following:

- "CV" (cross-validation, for number of folds use `iters=`)
 - "LOO" (leave-one-out cross-validation, for folds use `iters=`)
 - "RepCV" (repeated cross-validation, for number of repetitions use `reps=`, for folds use `folds=`)
 - "Subsample" (aka Monte-Carlo cross-validation, for iterations use `iters=`, for train % use `split=`)
 - "Bootstrap" (out-of-bag bootstrap, uses `iters=`)
 - "Holdout" (for train % use `split=`)
- `stratify` keeps target proportions consistent across samples.

`makeResampleInstance(desc=, task=)` can reduce noise by ensuring the resampling is done identically every time.

`resample(learner=, task=, resampling=, measures=)`
Train and test model according to specified resampling strategy.

mlr includes several pre-specified resample descriptions: `cv2` (2-fold cross-validation), `cv3`, `cv5`, `cv10`, `hout` (holdout with split 2/3 for training, 1/3 for testing). Convenience functions also exist to `resample()` with a specific strategy: `crossval()`, `repCV()`, `holdout()`, `subsample()`, `bootstrap00B()`, `bootstrapB632()`, `bootstrapB632plus()`

Refining Performance

Tuning hyperparameters

Set search space using `makeParamSet(make<type>Param())`

- `makeNumericParam(id=, lower=, upper=, trafo=)`
 - `makeIntegerParam(id=, lower=, upper=, trafo=)`
 - `makeIntegerVectorParam(id=, len=, lower=, upper=, trafo=)`
 - `makeDiscreteParam(id=, values=c(...))` (can also be used to test discrete values of numeric or integer parameters)
- `trafo` transforms the parameter output using a specified function, e.g. `lower=-2, upper=2, trafo=function(x) 10^x` would test values between 0.01 and 100, scaled exponentially
- Other acceptable parameter types include `Logical` `LogicalVector` `CharacterVector` `DiscreteVector`

Set a search algorithm with `makeTuneControl<type>()`

- `Grid(resolution=10)` Grid of all possible points
- `Random(maxit=100)` Randomly sample search space
- `MBO(budget=)` Use Bayesian model-based optimization
- `Irace(n.instances=)` Iterated racing process
- Other types: `CMAES`, `Design`, `GenSA`

Tune using `tuneParams(learner=, task=, resampling=, measures=, par.set=, control=)`

Quickstart

Prepare data for training and testing

```

library(mlbench)
data(Soybean)
soy = createDummyFeatures(Soybean, target="Class")
tsk = makeClassifTask(data=soy, target="Class")
ho = makeResampleInstance("Holdout", tsk)
tsk.train = subsetTask(tsk, ho$train.ind[1])
tsk.test = subsetTask(tsk, ho$test.ind[1])

```

Convert the factor inputs in the Soybean dataset into (0,1) dummy features which can be used by the XGboost algorithm. Create a task to predict the "Class" column. Create a train set with 2/3 of data and a test set with the remaining 1/3 (default).

Create learner and evaluate performance

```

lrn = makeLearner("classif.xgboost", nrounds=10)
cv = makeResampleDesc("CV", iters=5)
res = resample(lrn, tsk.train, cv, acc)

```

Create an XGboost learner which will build 10 trees. Then test performance using 5-fold cross-validation. Accuracy should be between 0.90-0.92.

Tune hyperparameters and retrain model

```

ps = makeParamSet(makeNumericParam("eta", 0, 1),
  makeNumericParam("lambda", 0, 200),
  makeIntegerParam("max_depth", 1, 20))
tc = makeTuneControlMBO(budget=100)
tr = tuneParams(lrn, tsk.train, cv5, acc, ps, tc)
lrn = setHyperPars(lrn, par.vals=tr$x)

```

Tune hyperparameters `eta`, `lambda`, and `max_depth` by defining a search space and using Model Based Optimization (MBO) to control the search. Then perform 100 rounds of 5-fold cross-validation, improving accuracy to ~0.93. Update the XGboost learner with the tuned hyperparameters.

```

mdl = train(lrn, tsk.train)
prd = predict(mdl, tsk.test)
calculateConfusionMatrix(prd)
mdl = train(lrn, tsk)

```

Train the model on the train set and make predictions on the test set. Show performance as a confusion matrix. Finally, re-train model on the full set to use on new data. You are now ready to go out into the real world and make 93% accurate predictions!

Legend for functions (not all parameters shown):

`function(required_parameters, optional_parameters=)`

Configuration

mlr's default settings can be changed using `configureMlr()`:

- `show.info` Whether to show verbose output by default when training, tuning, resampling, etc. (`TRUE`)
- `on.learner.error` How to handle a learner error. `"stop"` halts execution, `"warn"` returns NAs and displays a warning, `"quiet"` returns NAs with no warning (`"stop"`)
- `on.learner.warning` How to handle a learner warning. `"warn"` displays a warning, `"quiet"` suppresses it (`"warn"`)
- `on.par.without.desc` How to handle a parameter with no description. `"stop"`, `"warn"`, `"quiet"` (`"stop"`)
- `on.par.out.of.bounds` How to handle a parameter with an out-of-bounds value. `"stop"`, `"warn"`, `"quiet"` (`"stop"`)
- `on.measure.not.applicable` How to handle a measure not applicable to a learner. `"stop"`, `"warn"`, `"quiet"` (`"stop"`)
- `show.learner.output` Whether to show learner output to the console during training (`TRUE`)
- `on.error.dump` Whether to create an error dump for crashed learners if `on.learner.error` is not set to `"stop"` (`TRUE`)

Use `getMlrOptions()` to see current settings

Parallelization

mlr works with the `parallelMap` package to take advantage of multicore and cluster computing for faster operations. mlr automatically detects which operations are able to run in parallel.

To begin parallel operation use:

- ```
parallelStart(mode=, cpus=, level=)
```
- `mode` determines how the parallelization is performed:
    - `"local"` no parallelization applied, simply uses `mapply`
    - `"multicore"` multicore execution on a single machine, uses `parallel::mclapply`. Not available in Windows.
    - `"socket"` multicore execution in socket mode
    - `"mpi"` Snow MPI cluster on one or multiple machines using `parallel::makeCluster` and `parallel::clusterMap`
    - `"BatchJobs"` Batch queuing HPC clusters using `BatchJobs::batchMap`
  - `cpus` determines how many logical cores will be used
  - `level` controls parallelization: `"mlr.benchmark"`, `"mlr.resample"`, `"mlr.selectFeatures"`, `"mlr.tuneParams"`, `"mlr.ensemble"`

To end parallelization, use `parallelStop()`

# Imputation

`impute(obj=, target=, cols=, dummy.cols=, dummy.type=)`  
Applies specified logic to data frame or task containing NAs and returns an imputation description which can be used on new data

- `obj`=data frame or task on which to perform imputation
- `target`=specify target variable which will not be imputed
- `cols`=column names and logic for imputation\*
- `dummy.cols`=column names to create a NA (T/F) column\*
- `dummy.type`=set to `"numeric"` to use (0,1) instead of (T/F)

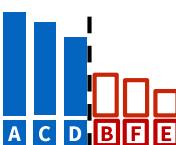
\*Can also use `classes` and `dummy.classes` in place of `cols`

Imputation logic is passed to `cols` or `classes` via a list, e.g.: `cols=list(V1=imputeMean())` where `V1` is the column to which to apply the imputation, and `imputeMean()` is the imputation method. Available imputation methods include:  
`imputeConst(const=)` `imputeMedian()` `imputeMode()` `imputeMin(multiplier=)` `imputeMax(multiplier=)` `imputeNormal(mean=, sd=)` `imputeHist(breaks=, use.mids=)` `imputeLearner(learner=, features=)` `impute` returns a list containing the imputed dataset or task as well as an imputation description that can be used to reapply the same imputation to new data using `reimpute`

`reimpute(obj=, desc=)` Imputes missing values on a task or dataset (`obj`) using a description (`desc`) created by `impute`

# Feature Extraction

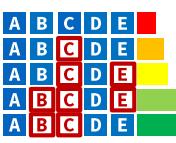
## Feature filtering



`filterFeatures(task=, method=, perc=, abs=, threshold=)`  
Uses a learner-agnostic feature evaluation method to rank feature importance, then includes only features in the top n percent (`perc=`), top n (`abs=`), or which meet a set performance threshold (`threshold=`).

Outputs a task with features that failed the test omitted. `method` defaults to `"randomForestSRC.rfsrc"`, but can be set to:  
`"anova.test"` `"carscore"` `"cforest.importance"`  
`"chi.squared"` `"gain.ratio"` `"information.gain"`  
`"kruskal.test"` `"linear.correlation"` `"mrmr"` `"oneR"`  
`"permutation.importance"` `"randomForest.importance"`  
`"randomForestSRC.rfsrc"` `"randomForestSRC.var.select"`  
`"rank.correlation"` `"relief"`  
`"symmetrical.uncertainty"` `"univariate.model.score"`  
`"variance"`

## Feature selection



`selectFeatures(learner=, task=, resampling=, measures=, control=)`  
Uses a feature selection algorithm (`control`) to resample and build a model repeatedly using different feature sets each time in order to find the best set.

Available controls include:

- `makeFeatSelControlExhaustive(max.features=)` Try every combination of features up to optional `max.features`
- `makeFeatSelControlRandom(maxit=, prob=, max.features=)` Randomly sample features with probability `prob` (default 0.5) until `maxit` (default 100) iterations; return the best one found
- `makeFeatSelControlSequential(method=, maxit=, max.features=, alpha=, beta=)` Perform an iterative search using a `method` from the following: `"sfs"` forward search, `"sbs"` backward search, `"sffs"` floating forward search, `"sfbs"` floating backward search. `alpha` indicates minimum improvement required to add a feature; `beta` indicates minimum required to remove a feature
- `makeFeatSelControlGA(maxit=, max.features=, mu=, lambda=, crossover.rate=, mutation.rate=)` Genetic algorithm trains on random feature vectors, then uses crossover on the best performers to produce 'offspring', repeated over generations. `mu` is size of parent population, `lambda` is size of children population, `crossover.rate` is probability of choosing a bit from first parent, `mutation.rate` is probability of flipping a bit (on or off)

`selectFeatures` returns a `FeatSelResult` object which contains optimal features and an optimization path. To apply feature selection result (`fsr`) to your task (`tsk`), use:  
`tsk = subsetTask(tsk, features=fsr$x)`

# Benchmarking

`benchmark(learners=, tasks=, resamplings=, measures=)`  
Allows easy comparison of multiple learners on a single task, a single learner on multiple tasks, or multiple learners on multiple tasks. Returns a benchmark result object.

Benchmark results can be accessed with a variety of functions beginning with `getBMR<object>.AggrPerformance`  
`FeatSelResults` `FilteredFeatures` `LearnerIds`  
`LeanerShortNames` `Learners` `MeasureIds` `Measures`  
`Models` `Performances` `Predictions` `TaskDescs` `TaskIds`  
`TuneResults`

mlr contains several toy tasks which are useful for benchmarking:  
`agri.task` `bc.task` `bh.task` `costiris.task` `iris.task`  
`lung.task` `mtcars.task` `pid.task` `sonar.task`  
`wpbc.task` `yeast.task`

# Visualization

## Performance

`generateThreshVsPerfData(obj=, measures=)` Measure performance at different probability cutoffs to determine optimal decision threshold for binary classification problems

- `plotThreshVsPerf(obj=)` Plot visual representation of threshold curve(s) from `ThreshVsPerfData`
- `plotROCCurves(obj=)` Plot receiver operating characteristic (ROC) curve from `ThreshVsPerfData`. Must set `measures=list(fpr, tpr)`

## Residuals

- `plotResiduals(obj=)` Plots residuals for `Prediction` or `BenchmarkResult`

## Learning curve

`generateLearningCurveData(learners=, task=, resampling=, percs=, measures=)` Measure performance of learner(s) trained on different percentages of task data

- `plotLearningCurve(obj=)` Plot curve showing learner performance vs. proportion of data used, uses `LearningCurveData`

## Feature importance

`generateFilterValuesData(task=, method=)` Get feature importance rankings using specified filter method

- `plotFilterValues(obj=)` Plot bar chart of feature importance based on filter method using `FilterValuesData`

## Hyperparameter tuning

`generateHyperParsEffectData(tune.result=)` Get the impact of different hyperparameter settings on model performance

- `plotHyperParsEffect(hyperpars.effec.t.data=, x=, y=, z=)` Create a plot showing hyperparameter impact on performance using `HyperParsEffectData`

See also:

- `plotOptPath(op=)` Display details of optimization process. Takes `<obj>$opt.path`, where `<obj>` is an object of class `tuneResult` or `featSelResult`
- `plotTuneMultiCritResult(res=)` Show pareto front for results of tuning to multiple performance measures

## Partial dependence

`generatePartialDependenceData(obj=, input=)` Get partial dependence of model (`obj`) prediction over each feature of data (`input`)

- `plotPartialDependence(obj=)` Plots partial dependence of model using `PartialDependenceData`

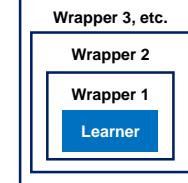
## Benchmarking

- `plotBMRBoxplots(bmr=)` Distribution of performances
- `plotBMRSummary(bmr=)` Scatterplot of avg. performances
- `plotBMRanksAsBarChart(bmr=)` Rank learners in bar plot

## Other

- `generateCritDifferencesData(bmr=, measure=, p.value=, test=)` Perform critical-differences test using either the Bonferroni-Dunn ("bd") or "Nemenyi" test
- `plotCritDifferences(obj=)`
- `generateCalibrationData(obj=)` Evaluate calibration of probability predictions vs. true incidence
- `plotCalibration(obj=)`

# Wrappers



**Wrappers** fuse a learner with additional functionality. mlr treats a learner with wrappers as a single learner, and hyperparameters of wrappers can be tuned jointly with underlying model parameters. Models trained with wrappers will apply them to new data.

## Preprocessing and imputation

`makeDummyFeaturesWrapper(learner=)`  
`makeImputeWrapper(learner=, classes=, cols=)`  
`makePreprocWrapper(learner=, train=, predict=)`  
`makePreprocWrapperCaret(learner=, ...)`  
`makeRemoveConstantFeaturesWrapper(learner=)`

## Class imbalance

`makeOverBaggingWrapper(learner=)`  
`makeSMOTEWrapper(learner=)`  
`makeUndersampleWrapper(learner=)`  
`makeWeightedClassesWrapper(learner=)`

## Cost-sensitive learning

`makeCostSensClassifWrapper(learner=)`  
`makeCostSensRegrWrapper(learner=)`  
`makeCostSensWeightedPairsWrapper(learner=)`

## Multilabel classification

`makeMultilabelBinaryRelevanceWrapper(learner=)`  
`makeMultilabelClassifierChainsWrapper(learner=)`  
`makeMultilabelDBRWrapper(learner=)`  
`makeMultilabelNestedStackingWrapper(learner=)`  
`makeMultilabelStackingWrapper(learner=)`

## Other

`makeBaggingWrapper(learner=)`  
`makeConstantClassWrapper(learner=)`  
`makeDownsampleWrapper(learner=, dw.perc=)`  
`makeFeatSelWrapper(learner=, resampling=, control=)`  
`makeFilterWrapper(learner=, fw.perc=, fw.abs=, fw.threshold=)`  
`makeMultiClassWrapper(learner=)`  
`makeTuneWrapper(learner=, resampling=, par.set=, control=)`

## Nested Resampling

mlr supports **nested resampling** for complex operations such as tuning and feature selection through wrappers. In order to get a good estimate of generalization performance and avoid data leakage, both an outer (for tuning/feature selection) and an inner (for the base model) resampling process are advised.

- Outer resampling can be specified in `resample` or `benchmark`
- Inner resampling can be specified in `makeTuneWrapper`, `makeFeatSelWrapper`, etc.

## Ensembles

`makeStackedLearner(base.learners=, super.learner=, method=)` Combines multiple learners to create an ensemble

- `base.learners`=learners to use for initial predictions
- `super.learner`=learner to use for final prediction
- `method`=how to combine base learner predictions:
  - `"average"` simple average of all base learners
  - `"stack.nocv", "stack.cv"` train super learner on results of base learners, with or without cross-validation
  - `"hill.climb"` search for optimal weighted average
  - `"compress"` with a neural network for faster performance

# Intro stats with mosaic

(lattice version)

## Essential R syntax

Names in R are *case sensitive*

Function and arguments

`rflip(10)`

Optional arguments

`rflip(10, prob = 0.8)`

Assignment

`x <- rflip(10, prob = 0.8)`

Getting help on any function

`help(mean)`

## Loading packages

`library(mosaic)`

## Arithmetic operations

`+ - * /` basic operations

`^` exponentiation

`( )` grouping

`sqrt(x)` square root

`abs(x)` absolute value

`log10(x)` logarithm, base 10

`log(x)` natural logarithm, base  $e$

`exp(x)` exponential function  $e^x$

`factorial(k)`  $k! = k(k-1) \dots 1$

## Logical operators

`==` is equal to (note double equal sign)

`!=` is not equal to

`<` is less than

`<=` is less than or equal to

`>` is greater than

`>=` is greater than or equal to

`&` **A & B** is TRUE if both **A** and **B** are TRUE

`|` **A | B** is TRUE if one or both of **A** and **B** are TRUE

`%in%` includes; for example

`"C" %in% c("A", "B")` is FALSE

## Formula interface

Use for graphics, statistics, inference, and modeling operations.

`goal(y ~ x, data = mydata)`

Read as “Calculate **goal** for **y** using **mydata** “broken down by” **x**, or “modeled by” **x**.

`mean(age ~ sex, data = HELPrc)`

For graphics:

`goal(y ~ x | z, groups = w, data = mydata)`

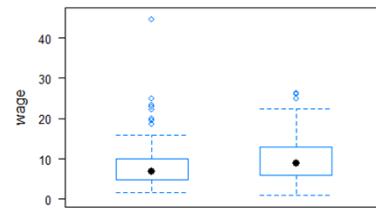
**y** : y-axis variable (*optional*)

**x** : x-axis variable (*required*)

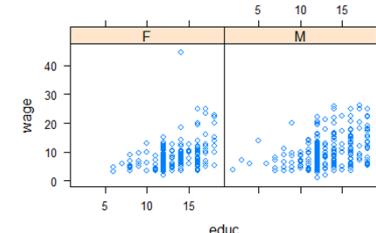
**z** : panel-by variable (*optional*)

**w** : color-by variable (*optional*)

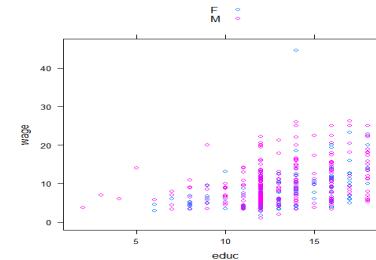
`bwplot(wage ~ sex, data = CPS85)`



`xyplot(wage ~ educ | sex, data = CPS85)`



`xyplot(wage ~ educ, groups = sex, data = CPS85, auto.key = TRUE)`



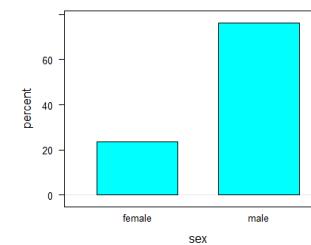
## One categorical variable

Counts by category

`tally(~ sex, data = HELPrc)`

Percentages by category

`tally(~ sex, format = "percent", data = HELPrc)`  
`bargraph(~ sex, type = "percent", data = HELPrc)`



Tests and confidence intervals

Exact test

`result1 <- binom.test(~ (homeless == "homeless"), data = HELPrc)`

Approximate test (large samples)

`result2 <- prop.test(~ (homeless == "homeless"), data = HELPrc)`

Extract confidence intervals and p-values

`confint(result1)`  
`pval(result2)`

## Examining data

Print short summary of all variables

`inspect(HELPrc)`

Number of rows and columns

`dim(HELPrc)`

`nrow(HELPrc)`

`ncol(HELPrc)`

Print first rows or last rows

`head(KidsFeet)`  
`tail(KidsFeet, 10)`

Names of variables

`names(HELPrc)`

## One quantitative variable

Make output more readable

`options(digits = 3)`

Compute summary statistics

`mean(~ cesd, data = HELPrc)`

Other summary statistics work similarly

`median()` `iqr()` `max()` `min()`  
`fivenum()` `sd()` `var()` `sum()`

Table of summary statistics

`favstats(~ cesd, data = HELPrc)`

Summary statistics by group

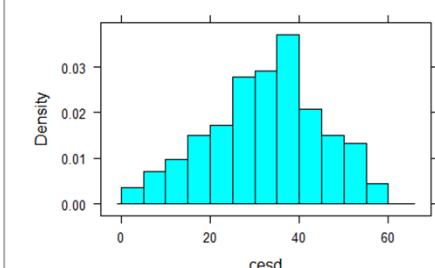
`favstats(cesd ~ sex, data = HELPrc)`

Quantiles

`quantile(~ cesd, data = HELPrc, prob = c(0.25, 0.5, 0.8))`

Histogram

`histogram(~ cesd, width = 5, center = 2.5, data = HELPrc)`



Normal probability plot

`qqmath(~ cesd, dist = "qnorm", data = HELPrc)`

Density plot

`densityplot(~ cesd, data = HELPrc)`

Dot plot

`dotPlot(~ cesd, data = HELPrc)`

One-sample t-test

`result <- t.test(~ cesd, mu = 34, data = HELPrc)`

Extract confidence intervals and p-values

`confint(result)`  
`pval(result)`

## Two categorical variables

Contingency table with margins

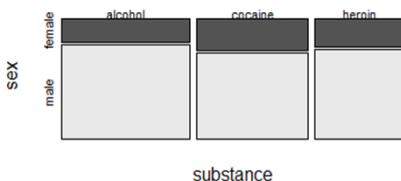
```
tally(~ substance + sex,
 margins = TRUE,
 data = HELPrc)
```

Percentages by column

```
tally(~ sex | substance,
 format = "percent",
 data = HELPrc)
```

Mosaic plot

```
mosaicplot(~ substance + sex,
 color = TRUE, data = HELPrc)
```



Chi-square test

```
xchisq.test(~ substance + sex,
 data = HELPrc,
 correct = FALSE)
```

## Distributions

Normal distribution function

```
pnorm(13, mean = 10, sd = 2)
```

Normal distribution function with graph

```
xpnorm(1.645, mean = 0, sd = 1)
```

Normal distribution quantiles

```
qnorm(0.95) # mean = 0, sd = 1
```

Normal distribution quantiles with graph

```
xqnorm(0.85, mean = 10, sd = 2)
```

Binomial density function ("size" means  $n$ )

```
dbinom(5, size = 8, prob = 0.65)
```

Binomial distribution function

```
pbinom(5, size = 8, prob = 0.65)
```

Central portion of distribution

```
cdist("norm", 0.95)
```

```
cdist("t", c(0.90, 0.99), df = 5)
```

Plotting distributions

```
plotDist("binom", size = 8,
 prob = 0.65, xlim = c(-1, 9))
```

```
plotDist("norm", mean = 10,
 sd = 2)
```

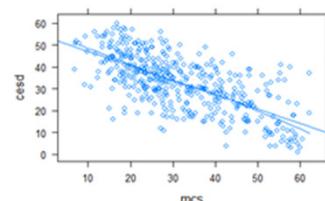
## Two quantitative variables

Correlation coefficient

```
cor(cesd ~ mcs, data = HELPrc)
```

Scatterplot with regression line and smooth

```
xyplot(cesd ~ mcs,
 type = c("p", "r", "smooth"),
 data = HELPrc)
```



Simple linear regression

```
cesdmodel <- lm(cesd ~ mcs,
 data = HELPrc)
```

```
msummary(cesdmodel)
```

Prediction

```
lmfunction <- makeFun(cesdmodel)
lmfunction(mcs = 35)
```

Extract useful quantities

```
anova(cesdmodel)
```

```
coef(cesdmodel)
```

```
confint(cesdmodel)
```

```
rsquared(cesdmodel)
```

Diagnostics; plot residuals

```
histogram(~resid(cesdmodel),
 density = TRUE)
```

```
qqmath(~resid(cesdmodel))
```

Diagnostics; plot residuals vs. fitted

```
xyplot(resid(cesdmodel) ~
 fitted(cesdmodel),
 type = c("p", "smooth", "r"))
```

## Categorical response, quantitative predictor

Logistic regression

```
logit_mod <-
 glm(homeless ~ age + female,
 family = binomial, data = HELPrc)
```

```
msummary(logitmod)
```

Odds ratios and confidence intervals

```
exp(coef(logit_mod))
```

```
exp(confint(logit_mod))
```

## Data management

From dplyr package

Drop or reorder variables

```
select()
```

Create new variables from existing ones

```
mutate()
```

Retain specific rows from data

```
filter()
```

Sort data rows

```
arrange()
```

Compute summary statistics by group

```
group_by()
```

```
summarize()
```

Merge data tables

```
left_join()
```

```
inner_join()
```

## Importing data

Import file from computer or URL

```
MustangPrice <-
 read.file("C:/MustangPrice.csv")
NOTE: R uses forward slashes!
Dome <-
 read.file("http://www.mosaic-
web.org/go/datasets/Dome.csv")
```

## Randomization and simulation

Fix random number sequence

```
set.seed(42)
```

Tossing coins

```
rflip(10) # default prob is 0.5
```

Do something repeatedly

```
do(5) * rflip(10, prob = 0.75)
```

Draw a simple random sample

```
sample(LETTERS, 10)
```

```
deal(Cards, 5) # poker hand
```

Resample with replacement

```
Small <- sample(KidsFeet, 10)
```

```
resample(�Small)
```

Random permutation (shuffling)

```
shuffle(Cards)
```

Random values from distributions

```
rbinom(5, size = 10, prob = 0.7)
```

```
rnorm(5, mean = 10, sd = 2)
```

## Quantitative response, categorical predictor

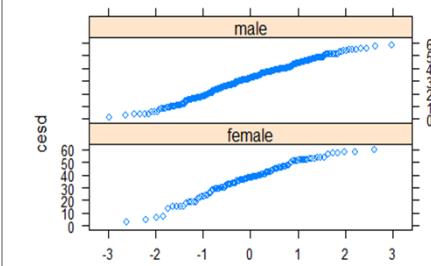
Two-level predictor: two-sample  $t$  test

Numeric summaries

```
favstats(~cesd | sex,
 data = HELPrc)
```

Comparative normal probability plot

```
qqmath(~cesd | sex, data = HELPrc,
 layout = c(1, 2)) # also bwplot
```



Dotplot for smaller samples

```
xyplot(sex ~ length, alpha = 0.6,
 cex = 1.4, data = KidsFeet)
```

Two-sample  $t$ -test and confidence interval

```
result <- t.test(cesd ~ sex,
 var.equal = FALSE, data = HELPrc)
```

```
confint(result)
```

More than two levels: Analysis of variance

Numeric summaries

```
favstats(cesd ~ substance,
 data = HELPrc)
```

Graphic summaries

```
bwplot(cesd ~ substance, pch = "|",
 data = HELPrc)
```

Fitt and summarize model

```
modsubstance <- lm(cesd ~ substance,
 data = HELPrc)
```

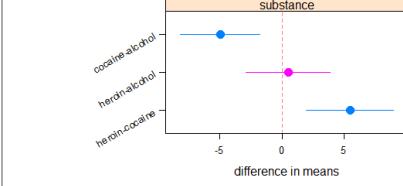
```
anova(modsubstance)
```

Which differences are significant?

```
pairwise <- TukeyHSD(modsubstance)
```

```
mplot(pairwise)
```

95% family-wise confidence level



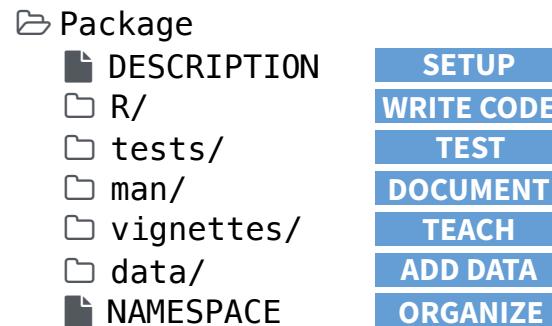
# Package Development: : CHEAT SHEET



## Package Structure

A package is a convention for organizing files into directories.

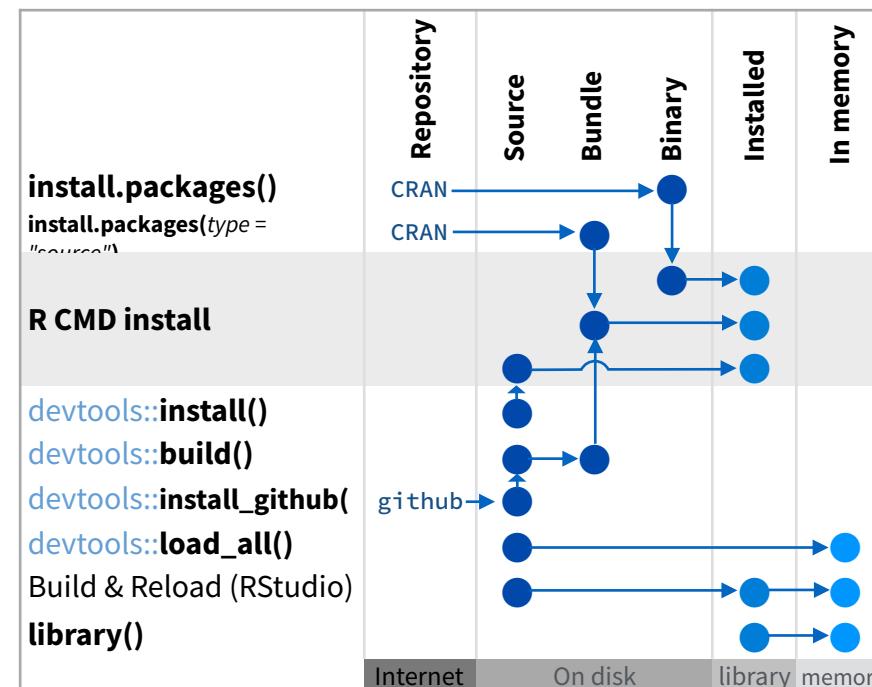
This sheet shows how to work with the 7 most common parts of an R package:



The contents of a package can be stored on disk as a:

- **source** - a directory with sub-directories (as above)
- **bundle** - a single compressed file (*.tar.gz*)
- **binary** - a single compressed file optimized for a specific OS

Or installed into an R library (loaded into memory during an R session) or archived online in a repository. Use the functions below to move between these states.



`devtools::use_build_ignore("file")`

Adds file to `.Rbuildignore`, a list of files that will not be included when package is built.

## Setup (DESCRIPTION)

The `DESCRIPTION` file describes your work, sets up how your package will work with other packages, and applies a copyright.

- You must have a `DESCRIPTION` file
- Add the packages that yours relies on with  
`devtools::use_package()`  
Adds a package to the Imports or Suggests field

### CC0

No strings attached.

### MIT

MIT license applies to your code if re-shared.

### GPL-2

GPL-2 license applies to your code, *and all code anyone bundles with it*, if re-shared.

Package: mypackage

Title: Title of Package

Version: 0.1.0

Authors@R: person("Hadley", "Wickham", email = "hadley@me.com", role = c("aut", "cre"))

Description: What the package does (one paragraph)

Depends: R (>= 3.1.0)

License: GPL-2

LazyData: true

Imports:

dplyr (>= 0.4.0),

ggvis (>= 0.2)

Suggests:

knitr (>= 0.1.0)

**Import** packages that your package *must* have to work. R will install them when it installs your package.

**Suggest** packages that are not very essential to yours. Users can install them manually, or not, as they like.

## Write Code (R/)

All of the R code in your package goes in `R/`. A package with just an `R/` directory is still a very useful package.

- Create a new package project with  
`devtools::create("path/to/name")`  
Create a template to develop into a package.
- Save your code in `R/` as scripts (extension `.R`)

### WORKFLOW

1. Edit your code.
2. Load your code with one of  
`devtools::load_all()`  
Re-loads all saved files in `R/` into memory.  
  
`Ctrl/Cmd + Shift + L` ([keyboard shortcut](#))  
Saves all open files then calls `load_all()`.
3. Experiment in the console.
4. Repeat.
  - Use consistent style with [r-pkgs.had.co.nz/r.html#style](http://r-pkgs.had.co.nz/r.html#style)
  - Click on a function and press **F2** to open its definition
  - Search for a function with **Ctrl +**.



Visit [r-pkgs.had.co.nz](http://r-pkgs.had.co.nz) to learn much more about writing and publishing packages for R

## Test (tests/)

Use `tests/` to store tests that will alert you if your code breaks.

- Add a `tests/` directory
- Import `testthat` with `devtools::use_testthat()`, which sets up package to use automated tests with `testthat`
- Write tests with `context()`, `test()`, and `expect` statements
- Save your tests as `.R` files in `tests/testthat/`

### WORKFLOW

1. Modify your code or tests.
2. Test your code with one of  
`devtools::test()`  
Runs all tests in `tests/`  
  
`Ctrl/Cmd + Shift + T` ([keyboard shortcut](#))
3. Repeat until all tests pass

### Example Test

```
context("Arithmetic")
test_that("Math works", {
 expect_equal(1 + 1, 2)
 expect_equal(1 + 2, 3)
 expect_equal(1 + 3, 4)
})
```

### Expect statement

|                                 |                                            |
|---------------------------------|--------------------------------------------|
| <code>expect_equal()</code>     | is equal within small numerical tolerance? |
| <code>expect_identical()</code> | is exactly equal?                          |
| <code>expect_match()</code>     | matches specified string or regular        |
| <code>expect_output()</code>    | prints specified output?                   |
| <code>expect_message()</code>   | displays specified message?                |
| <code>expect_warning()</code>   | displays specified warning?                |
| <code>expect_error()</code>     | throws specified error?                    |
| <code>expect_is()</code>        | output inherits from certain class?        |
| <code>expect_false()</code>     | returns FALSE?                             |
| <code>expect_true()</code>      | returns TRUE?                              |



## Document (📄 man/)

📄 man/ contains the documentation for your functions, the help pages in your package.

- Use roxygen comments to document each function beside its definition
- Document the name of each exported data set
- Include helpful examples for each function

### WORKFLOW

1. Add roxygen comments in your .R files
2. Convert roxygen comments into documentation with one of:

`devtools::document()`

Converts roxygen comments to .Rd files and places them in 📄 man/. Builds NAMESPACE.

**Ctrl/Cmd + Shift + D** (Keyboard Shortcut)

3. Open help pages with ? to preview documentation
4. Repeat

### .Rd FORMATTING TAGS

|                       |                                                                                    |
|-----------------------|------------------------------------------------------------------------------------|
| \emph{italic text}    | \email{name@@foo.com}                                                              |
| \strong{bold text}    | \href{url}{display}                                                                |
| \code{function(args)} | \url{url}                                                                          |
| \pkg{package}         |                                                                                    |
| \dontrun{code}        | \link[=dest]{display}                                                              |
| \dontshow{code}       | \linkS4class{class}                                                                |
| \donttest{code}       | \code{\link{function}}                                                             |
| \deqn{a + b (block)}  | \code{\link[package]{function}}}                                                   |
| \eqn{a + b (inline)}  | \tabular{lcr}{<br>left \tab centered \tab right \cr cell \tab cell \tab cell \cr } |

### ROXYGEN2

The **roxygen2** package lets you write documentation inline in your .R files with a shorthand syntax. devtools implements roxygen2 to make documentation.



- Add roxygen documentation as comment lines that begin with #’.
- Place comment lines directly above the code that defines the object documented.
- Place a roxygen @ tag (right) after #’ to supply a specific section of documentation.
- Untagged lines will be used to generate a title, description, and details section (in that order)

```
#' Add together two numbers.
#'
#' @param x A number.
#' @param y A number.
#' @return The sum of \code{x} and \code{y}.
#' @examples
#' add(1, 1)
#' @export
add <- function(x, y) {
 x + y
}
```

### COMMON ROXYGEN TAGS

|                  |                |          |      |
|------------------|----------------|----------|------|
| @aliases         | @inheritParams | @seealso |      |
| @concepts        | @keywords      | @format  |      |
| @describeln      | <b>@param</b>  | @source  | data |
| <b>@examples</b> | @rdname        | @include |      |
| <b>@export</b>   | <b>@return</b> | @slot    | S4   |
| @family          | @section       | @field   | RC   |

## Teach (📄 vignettes/)

📄 vignettes/ holds documents that teach your users how to solve real problems with your tools.

- Create a 📄 vignettes/ directory and a template vignette with `devtools::use_vignette()`  
Adds template vignette as vignettes/my-vignette.Rmd.
- Append YAML headers to your vignettes (like right)
- Write the body of your vignettes in R Markdown ([rmarkdown.rstudio.com](http://rmarkdown.rstudio.com))

```

```

```
title: "Vignette Title"
author: "Vignette Author"
date: "`r Sys.Date()`"
output: rmarkdown::html_vignette
vignette: >
 \%VignetteIndexEntry{Vignette Title}
 \%VignetteEngine{knitr::rmarkdown}
 \usepackage[utf8]{inputenc}
```

```

```

## Add Data (📄 data/)

The 📄 data/ directory allows you to include data with your package.

- Save data as .Rdata files (suggested)
- Store data in one of **data/**, **R/Sysdata.rda**, **inst/extdata**
- Always use **LazyData: true** in your DESCRIPTION file.

`devtools::use_data()`

Adds a data object to data/ (R/Sysdata.rda if **internal = TRUE**)

`devtools::use_data_raw()`

Adds an R Script used to clean a data set to data-raw/. Includes data-raw/ on .Rbuildignore.

### Store data in

- **data/** to make data available to package users
- **R/sysdata.rda** to keep data internal for use by your functions.
- **inst/extdata** to make raw data available for loading and parsing examples. Access this data with **system.file()**

## Organize (📄 NAMESPACE)

The 📄 NAMESPACE file helps you make your package self-contained: it won’t interfere with other packages, and other packages won’t interfere with it.

- Export functions for users by placing **@export** in their roxygen comments
- Import objects from other packages with **package::object** (recommended) or **@import**, **@importFrom**, **@importClassesFrom**, **@importMethodsFrom** (not always recommended)

### WORKFLOW

1. Modify your code or tests.
2. Document your package (`devtools::document()`)
3. Check NAMESPACE
4. Repeat until NAMESPACE is correct

### SUBMIT YOUR PACKAGE

[r-pkgs.had.co.nz/release.html](http://r-pkgs.had.co.nz/release.html)



# quanteda Cheat Sheet

## General syntax

- **corpus\_\*** manage text collections/metadata
- **tokens\_\*** create/modify tokenized texts
- **dfm\_\*** create/modify doc-feature matrices
- **fcm\_\*** work with co-occurrence matrices
- **textstat\_\*** calculate text-based statistics
- **textmodel\_\*** fit (un-)supervised models
- **textplot\_\*** create text-based visualizations

### Consistent grammar:

- **object()** constructor for the object type
- **object\_verb()** inputs & returns object type

## Extensions

**quanteda** works well with these companion packages:

- **readtext**: An easy way to read text data
- **spacyr**: NLP using the spaCy library
- **quantedaData**: additional textual data
- **LIWClike**: R implementation of the Linguistic Inquiry and Word Count approach

## Create a corpus from texts (corpus\_\*)

### Read texts (txt, pdf, csv, doc, docx, json, xml)

```
my_texts <- readtext::readtext("~/link/to/path/*")
```

### Construct a corpus from a character vector

```
x <- corpus(data_char_ukimmig2010, text_field = "text")
```

### Explore a corpus

```
summary(data_corpus_ inaugural, n = 2)
Corpus consisting of 58 documents, showing 2 documents:
Text Types Tokens Sentences Year President FirstName
1789-Washington 625 1538 23 1789 Washington George
1793-Washington 96 147 4 1793 Washington George
#
Source: Gerhard Peters and John T. Woolley. The American Presidency Project.
Created: Tue Jun 13 14:51:47 2017
Notes: http://www.presidency.ucsb.edu/inaugurals.php
```

### Extract or add document-level variables

```
party <- docvars(data_corpus_ inaugural, "Party")
docvars(x, "serial_number") <- 1:ndoc(x)
```

### Bind or subset corpora

```
corpus(x[1:5]) + corpus(x[7:9])
corpus_subset(x, Year > 1990)
```

### Change units of a corpus

```
corpus_reshape(x, to = c("sentences", use_docvars = TRUE)
```

### Segment texts on a pattern match

```
corpus_segment(x, pattern, valuetype, extract_pattern = TRUE)
```

### Take a random sample of corpus texts

```
corpus_sample(x, size = 10, replace = FALSE)
```

## Extract features (dfm\_\*; fcm\_\*)

### Create a document-feature matrix (dfm) from a corpus

```
x <- dfm(data_corpus_ inaugural,
 tolower = TRUE, stem = FALSE, remove_punct = TRUE,
 remove = stopwords("english"))
```

```
head(x, n = 2, nfeature = 4)
Document-feature matrix of: 2 documents, 4 features (41.7% sparse).
features
docs fellow-citizens senate house representatives
1789-Washington 1 1 2 2
1793-Washington 0 0 0 0
```

### Create a dictionary

```
dictionary(list(negative = c("bad", "awful", "sad"),
 positive = c("good", "wonderful", "happy")))
```

### Apply a dictionary

```
dfm_lookup(x, dictionary = data_dictionary_LSD2015)
```

### Select features

```
dfm_select(x, dictionary = data_dictionary_LSD2015)
```

### Compress a dfm by combining identical elements

```
dfm_compress(x, margin = c("both", "documents", "features"))
```

### Randomly sample documents or features

```
dfm_sample(x, what = c("documents", "features"))
```

### Weight or smooth the feature frequencies

```
dfm_weight(x, type = "relfreq") | dfm_smooth(x, smoothing = 0.5)
```

### Sort or group a dfm

```
dfm_sort(x, margin = c("features", "documents", "both"))
dfm_group(x, groups = "President")
```

### Combine identical dimension elements of a dfm

```
dfm_compress(x, margin = c("both", "documents", "features"))
```

### Create a feature co-occurrence matrix (fcm)

```
x <- fcm(data_corpus_ inaugural, context = "window", size = 5)
fcm_compress/remove/select/toupper/tolower are also available
```

## Useful additional functions

### Locate keywords-in-context

```
kwic(data_corpus_ inaugural, "america")
```

### Utility functions

|                             |                          |
|-----------------------------|--------------------------|
| texts(corpus)               | Show texts of a corpus   |
| ndoc(corpus/dfm/tokens)     | Count documents/features |
| nfeature(corpus/dfm/tokens) | Count features           |
| summary(corpus/dfm)         | Print summary            |
| head(corpus/dfm)            | Return first part        |
| tail(corpus/dfm)            | Return last part         |

## Tokenize a set of texts (tokens\_\*)

### Tokenize texts from a character vector or corpus

```
x <- tokens("Powerful tool for text analysis.",
 remove_punct = TRUE, stem = TRUE)
```

### Convert sequences into compound tokens

```
myseqs <- phrase(c("powerful", "tool", "text analysis"))
tokens_compound(x, myseqs)
```

### Select tokens

```
tokens_select(x, c("powerful", "text"), selection = "keep")
```

### Create ngrams and skipgrams from tokens

```
tokens_ngrams(x, n = 1:3)
tokens_skipgrams(toks, n = 2, skip = 0:1)
```

### Convert case of tokens

```
tokens_tolower(x) | tokens_topupper(x)
```

### Stem the terms in an object

```
tokens_wordstem(x)
```

## Calculate text statistics (textstat\_\*)

### Tabulate feature frequencies from a dfm

```
textstat_frequency(x) | topfeatures(x)
```

### Identify and score collocations from a tokenized text

```
toks <- tokens(c("quanteda is a pkg for quant text analysis",
 "quant text analysis is a growing field"))
textstat_collocations(toks, size = 3, min_count = 2)
```

### Calculate readability of a corpus

```
textstat_readability(data_corpus_inaugural, measure = "Flesch")
```

### Calculate lexical diversity of a dfm

```
textstat_lexdiv(x, measure = "TTR")
```

### Measure distance or similarity from a dfm

```
textstat_simil(x, "2017-Trump", method = "cosine")
textstat_dist(x, "2017-Trump", margin = "features")
```

### Calculate keyness statistics

```
textstat_keyness(x, target = "2017-Trump")
```

## Fit text models based on a dfm (textmodel\_\*)

### Correspondence Analysis (CA)

```
textmodel_ca(x, threads = 2, sparse = TRUE, residual_floor = 0.1)
```

### Naïve Bayes classifier for texts

```
textmodel_NB(x, y = training_labels, distribution = "multinomial")
```

### Wordscores text model

```
refscores <- c(seq(-1.5, 1.5, .75), NA)
textmodel_wordscores(data_dfm_lbgexample, refscores)
```

### Wordfish Poisson scaling model

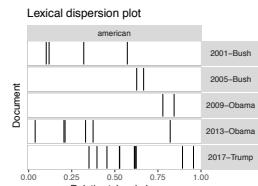
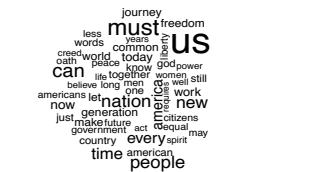
```
textmodel_wordfish(dfm(data_corpus_irishbudget2010), dir = c(6,5))
```

**Textmodel methods:** predict(), coef(), summary(), print()

## Plot features or models (textplot\_\*)

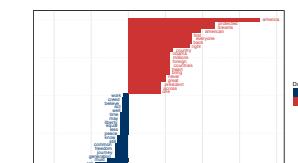
### Plot features as a wordcloud

```
data_corpus_inaugural %>%
 corpus_subset(President == "Obama") %>%
 dfm(remove = stopwords("english")) %>%
 textplot_wordcloud()
```



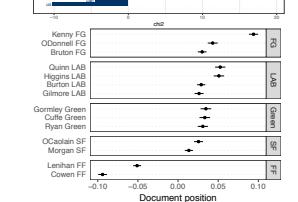
### Plot the dispersion of key word(s)

```
data_corpus_inaugural %>%
 corpus_subset(Year > 1945) %>%
 kwic("american") %>%
 textplot_xray()
```



### Plot word keyness

```
data_corpus_inaugural %>%
 corpus_subset(President %in%
 c("Obama", "Trump")) %>%
 dfm(groups = "President",
 remove = stopwords("english")) %%
 textstat_keyness(target = "Trump") %>%
 textplot_keyness()
```



### Plot Wordfish, Wordscores or CA models

```
textplot_scale1d(scaling_model,
 groups = party,
 margin = c("documents"))
```

## Convert dfm to a non-quanteda format

```
convert(x, to = c("lda", "tm", "stm", "austin", "topicmodels",
 "lsa", "matrix", "data.frame"))
```



# Work with strings with stringr :: CHEAT SHEET

The `stringr` package provides a set of internally consistent tools for working with character strings, i.e. sequences of characters surrounded by quotation marks.

## Detect Matches

|  |                                                                                                                                                                         |
|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>str_detect(string, pattern)</code> Detect the presence of a pattern match in a string.<br><code>str_detect(fruit, "a")</code>                                     |
|  | <code>str_which(string, pattern)</code> Find the indexes of strings that contain a pattern match.<br><code>str_which(fruit, "a")</code>                                 |
|  | <code>str_count(string, pattern)</code> Count the number of matches in a string.<br><code>str_count(fruit, "a")</code>                                                  |
|  | <code>str_locate(string, pattern)</code> Locate the positions of pattern matches in a string. Also <code>str_locate_all</code> .<br><code>str_locate(fruit, "a")</code> |

## Subset Strings

|  |                                                                                                                                                                                                                                                |
|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>str_sub(string, start = 1L, end = -1L)</code> Extract substrings from a character vector.<br><code>str_sub(fruit, 1, 3); str_sub(fruit, -2)</code>                                                                                       |
|  | <code>str_subset(string, pattern)</code> Return only the strings that contain a pattern match.<br><code>str_subset(fruit, "b")</code>                                                                                                          |
|  | <code>str_extract(string, pattern)</code> Return the first pattern match found in each string, as a vector. Also <code>str_extract_all</code> to return every pattern match.<br><code>str_extract(fruit, "[aeiou]")</code>                     |
|  | <code>str_match(string, pattern)</code> Return the first pattern match found in each string, as a matrix with a column for each () group in pattern. Also <code>str_match_all</code> .<br><code>str_match(sentences, "(a the) ([^ ]+)")</code> |

## Manage Lengths

|  |                                                                                                                                                                                                  |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>str_length(string)</code> The width of strings (i.e. number of code points, which generally equals the number of characters).<br><code>str_length(fruit)</code>                            |
|  | <code>str_pad(string, width, side = c("left", "right", "both"), pad = " ")</code> Pad strings to constant width.<br><code>str_pad(fruit, 17)</code>                                              |
|  | <code>str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")</code> Truncate the width of strings, replacing content with ellipsis.<br><code>str_trunc(fruit, 3)</code> |
|  | <code>str_trim(string, side = c("both", "left", "right"))</code> Trim whitespace from the start and/or end of a string.<br><code>str_trim(fruit)</code>                                          |

## Mutate Strings

|  |                                                                                                                                                                                                |
|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>str_sub()</code> <- value. Replace substrings by identifying the substrings with <code>str_sub()</code> and assigning into the results.<br><code>str_sub(fruit, 1, 3) &lt;- "str"</code> |
|  | <code>str_replace(string, pattern, replacement)</code> Replace the first matched pattern in each string.<br><code>str_replace(fruit, "a", "-")</code>                                          |
|  | <code>str_replace_all(string, pattern, replacement)</code> Replace all matched patterns in each string.<br><code>str_replace_all(fruit, "a", "-")</code>                                       |
|  | <code>str_to_lower(string, locale = "en")<sup>1</sup></code> Convert strings to lower case.<br><code>str_to_lower(sentences)</code>                                                            |
|  | <code>str_to_upper(string, locale = "en")<sup>1</sup></code> Convert strings to upper case.<br><code>str_to_upper(sentences)</code>                                                            |
|  | <code>str_to_title(string, locale = "en")<sup>1</sup></code> Convert strings to title case.<br><code>str_to_title(sentences)</code>                                                            |

## Join and Split

|  |                                                                                                                                                                                                                                                                                      |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>str_c(..., sep = "", collapse = NULL)</code> Join multiple strings into a single string.<br><code>str_c(letters, LETTERS)</code>                                                                                                                                               |
|  | <code>str_c(..., sep = "", collapse = NULL)</code> Collapse a vector of strings into a single string.<br><code>str_c(letters, collapse = "")</code>                                                                                                                                  |
|  | <code>str_dup(string, times)</code> Repeat strings times times.<br><code>str_dup(fruit, times = 2)</code>                                                                                                                                                                            |
|  | <code>str_split_fixed(string, pattern, n)</code> Split a vector of strings into a matrix of substrings (splitting at occurrences of a pattern match). Also <code>str_split</code> to return a list of substrings.<br><code>str_split_fixed(fruit, " ", n=2)</code>                   |
|  | <code>glue::glue(..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}")</code> Create a string from strings and {expressions} to evaluate.<br><code>glue::glue("Pi is {pi}")</code>                                                                                     |
|  | <code>glue::glue_data(.x, ..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}")</code> Use a data frame, list, or environment to create a string from strings and {expressions} to evaluate.<br><code>glue::glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")</code> |

## Order Strings

|  |                                                                                                                                                                                                            |
|--|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)<sup>1</sup></code> Return the vector of indexes that sorts a character vector.<br><code>x[str_order(x)]</code> |
|  | <code>str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)<sup>1</sup></code> Sort a character vector.<br><code>str_sort(x)</code>                                         |

## Helpers

|  |                                                                                                                                                          |
|--|----------------------------------------------------------------------------------------------------------------------------------------------------------|
|  | <code>str_conv(string, encoding)</code> Override the encoding of a string.<br><code>str_conv(fruit, "ISO-8859-1")</code>                                 |
|  | <code>str_view(string, pattern, match = NA)</code> View HTML rendering of first regex match in each string.<br><code>str_view(fruit, "[aeiou]")</code>   |
|  | <code>str_view_all(string, pattern, match = NA)</code> View HTML rendering of all regex matches.<br><code>str_view_all(fruit, "[aeiou]")</code>          |
|  | <code>str_wrap(string, width = 80, indent = 0, exdent = 0)</code> Wrap strings into nicely formatted paragraphs.<br><code>str_wrap(sentences, 20)</code> |

<sup>1</sup> See [bit.ly/ISO639-1](http://bit.ly/ISO639-1) for a complete list of locales.



# Data Science in Spark with sparklyr :: CHEAT SHEET

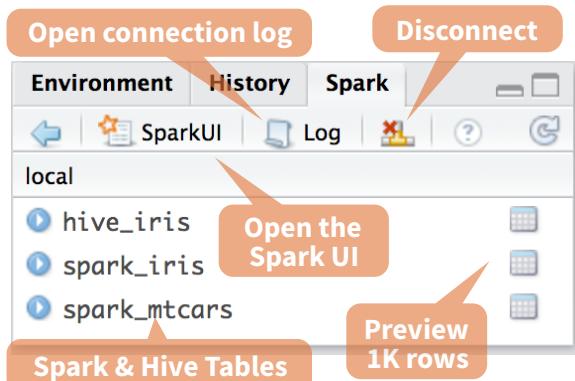


## Intro

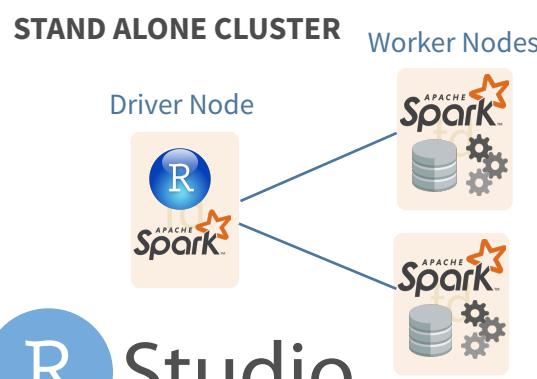
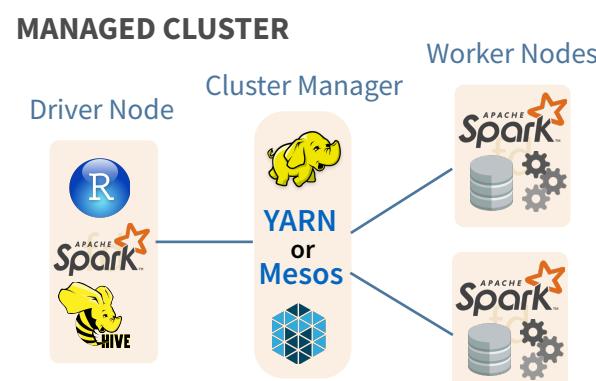
**sparklyr** is an R interface for Apache Spark™, it provides a complete **dplyr** backend and the option to query directly using **Spark SQL** statement. With sparklyr, you can orchestrate distributed machine learning using either **Spark's MLLib** or **H2O** Sparkling Water.

Starting with **version 1.044, RStudio Desktop, Server and Pro include integrated support for the sparklyr package**. You can create and manage connections to Spark clusters and local Spark instances from inside the IDE.

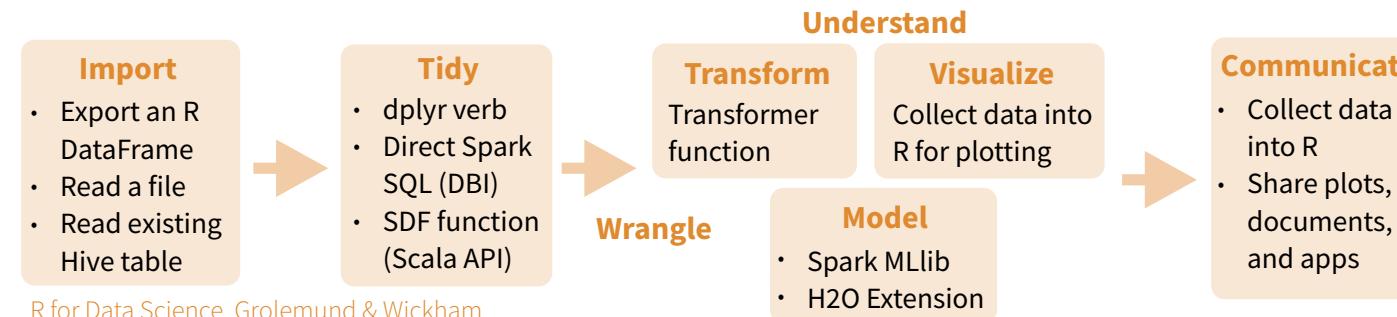
### RStudio Integrates with sparklyr



## Cluster Deployment



## Data Science Toolchain with Spark + sparklyr



## Getting Started

### LOCAL MODE (No cluster required)

1. Install a local version of Spark:  
`spark_install ("2.0.1")`
2. Open a connection  
`sc <- spark_connect (master = "local")`

### ON A MESOS MANAGED CLUSTER

1. Install RStudio Server or Pro on one of the existing nodes
2. Locate path to the cluster's Spark Home Directory, it normally is "/usr/lib/spark"
3. Open a connection  
`spark_connect(master="mesos URL", version = "1.6.2", spark_home = [Cluster's Spark path])`

### ON A YARN MANAGED CLUSTER

1. Install RStudio Server or RStudio Pro on one of the existing nodes, preferably an edge node
2. Locate path to the cluster's Spark Home Directory, it normally is "/usr/lib/spark"
3. Open a connection  
`spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Cluster's Spark path])`

### ON A SPARK STANDALONE CLUSTER

1. Install RStudio Server or RStudio Pro on one of the existing nodes or a server in the same LAN
2. Install a local version of Spark:  
`spark_install (version = "2.0.1")`
3. Open a connection  
`spark_connect(master="spark://host:port", version = "2.0.1", spark_home = spark_home_dir())`

### USING LIVY (Experimental)

1. The Livy REST application should be running on the cluster
2. Connect to the cluster  
`sc <- spark_connect(method = "livy", master = "http://host:port")`

## Tuning Spark

### EXAMPLE CONFIGURATION

```
config <- spark_config()
config$spark.executor.cores <- 2
config$spark.executor.memory <- "4G"
sc <- spark_connect (master="yarn-client",
 config = config, version = "2.0.1")
```

### IMPORTANT TUNING PARAMETERS with defaults

- spark.yarn.am.cores
- spark.yarn.am.memory **512m**
- spark.network.timeout **120s**
- spark.executor.memory **1g**
- spark.executor.cores **1**
- spark.executor.instances
- spark.executor.extraJavaOptions
- spark.executor.heartbeatInterval **10s**
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

## Using sparklyr

A brief example of a data analysis using Apache Spark, R and sparklyr in local mode

```
library(sparklyr); library(dplyr); library(ggplot2);
library(tidyr);
set.seed(100)
```

**Install Spark locally**

```
spark_install("2.0.1")
```

**Connect to local version**

```
sc <- spark_connect(master = "local")
```

```
import_iris <- copy_to(sc, iris, "spark_iris",
 overwrite = TRUE)
```

**Copy data to Spark memory**

```
partition_iris <- sdf_partition(
 import_iris, training=0.5, testing=0.5)
```

**Partition data**

```
sdf_register(partition_iris,
 c("spark_iris_training","spark_iris_test"))
```

**Create a hive metadata for each partition**

```
tidy_iris <-tbl(sc,"spark_iris_training") %>%
 select(Species, Petal_Length, Petal_Width)
```

**Spark ML Decision Tree Model**

```
model_iris <- tidy_iris %>%
 ml_decision_tree(response="Species",
 features=c("Petal_Length","Petal_Width"))
```

```
test_iris <-tbl(sc,"spark_iris_test")
```

**Create reference to Spark table**

```
pred_iris <- sdf_predict(
 model_iris, test_iris) %>%
 collect
```

```
pred_iris %>%
 inner_join(data.frame(prediction=0:2,
 lab=model_iris$model.parameters$labels)) %>%
 ggplot(aes(Petal_Length, Petal_Width, col=lab)) +
 geom_point()
```

**Bring data back into R memory for plotting**

```
spark_disconnect(sc)
```

**Disconnect**



# Reactivity

## COPY A DATA FRAME INTO SPARK

`sdf_copy_to(sc, iris, "spark_iris")`

`sdf_copy_to(sc, x, name, memory, repartition, overwrite)`

## IMPORT INTO SPARK FROM A FILE

Arguments that apply to all functions:

`sc, name, path, options = list(), repartition = 0, memory = TRUE, overwrite = TRUE`

**CSV** `spark_read_csv(header = TRUE, columns = NULL, infer_schema = TRUE, delimiter = "", quote = "", escape = "\\", charset = "UTF-8", null_value = NULL)`

**JSON** `spark_read_json()`

**PARQUET** `spark_read_parquet()`

## SPARK SQL COMMANDS

`DBI::dbWriteTable(sc, "spark_iris", iris)`

`DBI::dbWriteTable(conn, name, value)`

## FROM A TABLE IN HIVE

`my_var <- tbl_cache(sc, name = "hive_iris")`

`tbl_cache(sc, name, force = TRUE)`  
Loads the table into memory

`my_var <- dplyr::tbl(sc, name = "hive_iris")`

`dplyr::tbl(sc, ...)`  
Creates a reference to the table without loading it into memory

# Wrangle

## SPARK SQL VIA DPLYR VERBS

Translates into Spark SQL statements

`my_table <- my_var %>%  
filter(Species == "setosa") %>%  
sample_n(10)`

## DIRECT SPARK SQL COMMANDS

`my_table <- DBI::dbGetQuery(sc, "SELECT *  
FROM iris LIMIT 10")`

`DBI::dbGetQuery(conn, statement)`

## SCALA API VIA SDF FUNCTIONS

`sdf_mutate(.data)`

Works like dplyr mutate function

`sdf_partition(x, ..., weights = NULL, seed = sample (.Machine$integer.max, 1))`

`sdf_partition(x, training = 0.5, test = 0.5)`

`sdf_register(x, name = NULL)`

Gives a Spark DataFrame a table name

`sdf_sample(x, fraction = 1, replacement = TRUE, seed = NULL)`

`sdf_sort(x, columns)`  
Sorts by >=1 columns in ascending order

`sdf_with_unique_id(x, id = "id")`

`sdf_predict(object, newdata)`  
Spark DataFrame with predicted values

## ML TRANSFORMERS

`ft_binarizer(my_table, input.col = "Petal_Length", output.col = "petal_large", threshold = 1.2)`

Arguments that apply to all functions:  
`x, input.col = NULL, output.col = NULL`

`ft_binarizer(threshold = 0.5)`  
Assigned values based on threshold

`ft_bucketizer(splits)`  
Numeric column to discretized column

`ft_discrete_cosine_transform(inverse = FALSE)`  
Time domain to frequency domain

`ft_elementwise_product(scaling.col)`  
Element-wise product between 2 cols

`ft_index_to_string()`  
Index labels back to label as strings

`ft_one_hot_encoder()`  
Continuous to binary vectors

`ft_quantile_discretizer(n.buckets = 5L)`  
Continuous to binned categorical values

`ft_sql_transformer(sql)`  
Column of labels into a column of label indices.

`ft_string_indexer(params = NULL)`  
Combine vectors into single row-vector

# Visualize & Communicate

## DOWNLOAD DATA TO R MEMORY

`r_table <- collect(my_table)`  
`plot(Petal_Width ~ Petal_Length, data = r_table)`

`dplyr::collect(x)`  
Download a Spark DataFrame to an R DataFrame

`sdf_read_column(x, column)`

Returns contents of a single column to R

## SAVE FROM SPARK TO FILE SYSTEM

Arguments that apply to all functions: `x, path`

**CSV** `spark_read_csv(header = TRUE, delimiter = "", quote = "", escape = "\\", charset = "UTF-8", null_value = NULL)`

**JSON** `spark_read_json(mode = NULL)`

**PARQUET** `spark_read_parquet(mode = NULL)`

# Model (MLlib)

`ml_decision_tree(my_table, response = "Species", features = c("Petal_Length", "Petal_Width"))`

`ml_als_factorization(x, user.column = "user", rating.column = "rating", item.column = "item", rank = 10L, regularization.parameter = 0.1, iter.max = 10L, ml.options = ml_options())`

`ml_decision_tree(x, response, features, max.bins = 32L, max.depth = 5L, type = c("auto", "regression", "classification"), ml.options = ml_options())` Same options for: `ml_gradient_boosted_trees`

`ml_generalized_linear_regression(x, response, features, intercept = TRUE, family = gaussian(link = "identity"), iter.max = 100L, ml.options = ml_options())`

`ml_kmeans(x, centers, iter.max = 100, features = dplyr::tbl_vars(x), compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options())`

`ml_lda(x, features = dplyr::tbl_vars(x), k = length(features), alpha = (50/k) + 1, beta = 0.1 + 1, ml.options = ml_options())`

`ml_linear_regression(x, response, features, intercept = TRUE, alpha = 0, lambda = 0, iter.max = 100L, ml.options = ml_options())`  
Same options for: `ml_logistic_regression`

`ml_multilayer_perceptron(x, response, features, layers, iter.max = 100, seed = sample(.Machine$integer.max, 1), ml.options = ml_options())`

`ml_naive_bayes(x, response, features, lambda = 0, ml.options = ml_options())`

`ml_one_vs_rest(x, classifier, response, features, ml.options = ml_options())`

`ml_pca(x, features = dplyr::tbl_vars(x), ml.options = ml_options())`

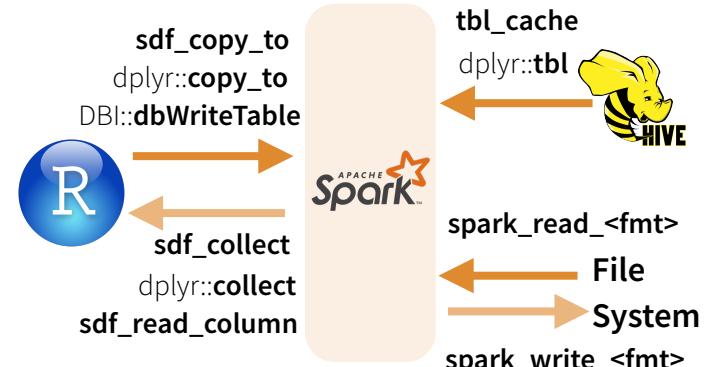
`ml_random_forest(x, response, features, max.bins = 32L, max.depth = 5L, num.trees = 20L, type = c("auto", "regression", "classification"), ml.options = ml_options())`

`ml_survival_regression(x, response, features, intercept = TRUE, censor = "censor", iter.max = 100L, ml.options = ml_options())`

`ml_binary_classification_eval(predicted_tbl_spark, label, score, metric = "areaUnderROC")`

`ml_classification_eval(predicted_tbl_spark, label, predicted_lbl, metric = "f1")`

`ml_tree_feature_importance(sc, model)`



# Extensions

Create an R package that calls the full Spark API & provide interfaces to Spark packages.

## CORE TYPES

`spark_connection()` Connection between R and the Spark shell process

`spark_jobj()` Instance of a remote Spark object

`spark_dataframe()` Instance of a remote Spark DataFrame object

## CALL SPARK FROM R

`invoke()` Call a method on a Java object

`invoke_new()` Create a new object by invoking a constructor

`invoke_static()` Call a static method on an object

## MACHINE LEARNING EXTENSIONS

`ml_create_dummy_variables()` `ml_options()`

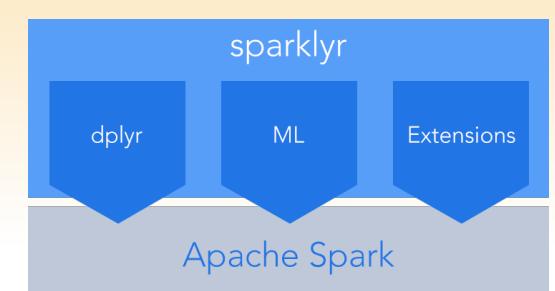
`ml_prepare_dataframe()` `ml_model()`

`ml_prepare_response_features_intercept()`

**sparklyr**

is an R  
interface  
for

APACHE  
SPARK™



# Spark数据科学之

## sparklyr 速查表



### 简介

sparklyr是Apache Spark™与R接的口。它提供了一个完整的dplyr后端并且直接支持对Spark SQL语句的调用。它还能够基于Spark MLlib或者H2O Sparkling Water实现分布式机器学习算法。



从版本1.0.44开始，R Studio软件在Desktop, Server和Pro版本中集成了对sparklyr软件包的支持。从IDE内我们可以直接创建或管理与Spark远程集群或本地Spark的连接。

### RStudio平台中的sparklyr集成

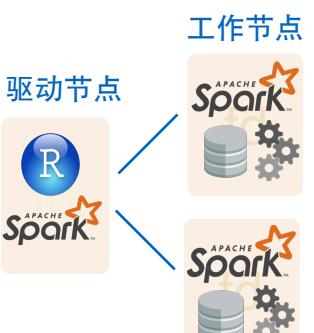


### 分布式部署

#### 分布式部署选项



#### 单例模式



## Spark与sparklyr下的数据科学工具链



### 如何开始

#### 本地模式

轻松设置; 无需群集

- 安装本地模式Spark:  
`spark_install ("2.0.1")`
- 建立连接:  
`sc <- spark_connect (master = "local")`

#### 使用Mesos集群管理模式

- 在现有节点上安装RStudio Server或Pro
- 确定Spark Home的集群路径
- 建立集群连接:  
`spark_connect(master=[Mesos-URL], version = "1.6.2", spark_home = [Spark Home集群路径])`

#### 使用Livy (测试阶段)

- Livy REST应用需已在集群上运行
- 建立集群连接:  
`sc <- spark_connect(master = "http://host:port", method = "livy")`

#### 使用YARN集群管理模式

- 在现有节点上安装RStudio Server或Pro
- 确定Spark Home集群路径 (通常位于 "/usr/lib/spark")
- 建立集群连接:  
`spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Spark Home集群路径])`

#### 使用Spark单例模式

- 在现有节点上或在同个局域网的服务器上安装RStudio Server或Pro
- 安装本地模式Spark:  
`spark_install (version = "2.0.1")`
- 建立连接:  
`spark_connect(master="spark://host:port", version = "2.0.1", spark_home = spark_home_dir())`

### Spark调试

#### 配置示例

```
config <- spark_config()
config$spark.executor.cores <- 2
config$spark.executor.memory <- "4G"
sc <- spark_connect (master = "yarn-client", config = config, version = "2.0.1")
```

#### 调试参数 附默认值

- spark.yarn.am.cores
- spark.yarn.am.memory 512m

#### 调试参数 附默认值 (续)

- spark.executor.heartbeatInterval 10s
- spark.network.timeout 120s
- spark.executor.memory 1g
- spark.executor.cores 1
- spark.executor.extraJavaOptions
- spark.executor.instances
- sparklyr.shell.executor-memory
- sparklyr.shell.driver-memory

## 使用sparklyr

### 简单示例：在本地模式下使用Apache Spark, R和sparklyr进行数据分析

```
library(sparklyr); library(dplyr); library(ggplot2);
library(tidyverse);
set.seed(100)
```

安装本地模式Spark

```
spark_install("2.0.1")
```

连接本地Spark

```
sc <- spark_connect(master = "local")
```

```
import_iris <- copy_to(sc, iris, "spark_iris",
overwrite = TRUE)
```

复制数据集到Spark

```
partition_iris <- sdf_partition(
import_iris, training=0.5,
testing=0.5)
```

分割Spark  
数据集

```
sdf_register(partition_iris,
c("spark_iris_training", "spark_iris_test"))
```

为每个分区注册Hive表格

```
tidy_iris <- tbl(sc,"spark_iris_training") %>%
select(Species, Petal_Length, Petal_Width)
```

创建Spark ML  
决策树模型

```
model_iris <- tidy_iris %>%
ml_decision_tree(response="Species",
features=c("Petal_Length", "Petal_Width"))
```

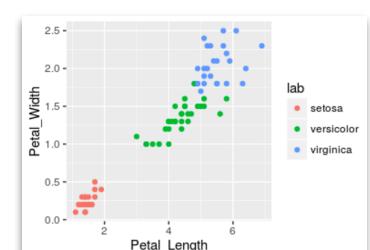
```
test_iris <- tbl(sc,"spark_iris_test")
```

建立Spark  
表格指针

```
pred_iris <- sdf_predict(
model_iris, test_iris) %>%
collect
```

读取预测数据至R中  
用于之后演示之用

```
pred_iris %>%
inner_join(data.frame(prediction=0:2,
lab=model_iris$model.parameters$labels)) %>%
ggplot(aes(Petal_Length, Petal_Width, col=lab)) +
geom_point()
```



```
spark_disconnect(sc)
```

断开连接

## 导入

### 复制DataFrame到Spark

```
sdf_copy_to(sc, iris, "spark_iris")
```

```
sdf_copy_to(sc, x, name, memory, repartition,
overwrite)
```

### 在Spark中读取文件

适用所有函数的参数:  
`sc, name, path, options = list(), repartition = 0,  
memory = TRUE, overwrite = TRUE`

**CSV**

```
spark_read_csv(header = TRUE,
columns = NULL, infer_schema = TRUE,
delimiter = ",", quote = "\"", escape = "\\\",
charset = "UTF-8", null_value = NULL)
```

**JSON**

```
spark_read_json()
```

**PARQUET**

```
spark_read_parquet()
```

### Spark SQL指令

```
DBI::dbWriteTable(
 sc, "spark_iris", iris)
```

```
DBI::dbWriteTable(conn, name,
value)
```

### 读取Hive表格

```
my_var <- tbl_cache(sc,
name = "hive_iris")
```

```
tbl_cache(sc, name, force = TRUE)
将整个表导入内存
```

```
my_var <- dplyr::tbl(sc,
name = "hive_iris")
```

```
dplyr::tbl(scr, ...)
创建指向表的延迟加载指针
```

## 可视化与交流

### 将数据载入R内存

```
r_table <- collect(my_table)
plot(Petal_Width~Petal_Length, data=r_table)
```

**dplyr::collect(x)**

将Spark DataFrame转为R DataFrame

**sdf\_read\_column(x, column)**

返回R单个列下的内容

### 从Spark储存数据到文件系统

适用所有函数的参数: `x, path`

**CSV**

```
spark_read_csv(header = TRUE,
delimiter = ", ", quote = "\"", escape = "<\"\\\"",
charset = "UTF-8", null_value =
NULL)
```

**JSON**

```
spark_read_json(mode = NULL)
```

**PARQUET**

```
spark_read_parquet(mode = NULL)
```

## 数据整理

### 通过dplyr语句使用Spark SQL

转换为Spark SQL语句

```
my_table <- my_var %>%
filter(Species == "setosa") %>%
sample_n(10)
```

### Direct Spark SQL指令

```
my_table <- DBI::dbGetQuery(sc, "SELECT
* FROM iris LIMIT 10")
```

```
DBI::dbGetQuery(conn, statement)
```

### 通过SDF函数使用Scala API

**sdf\_mutate(.data)**

工作原理于dplyr mutate函数类似

```
sdf_partition(x, ..., weights = NULL, seed =
sample(.Machine$integer.max, 1))
```

```
sdf_partition(x, training = 0.5, test = 0.5)
```

**sdf\_register(x, name = NULL)**

给Spark DataFrame命名

```
sdf_sample(x, fraction = 1, replacement =
TRUE, seed = NULL)
```

**sdf\_sort(x, columns)**  
按升序排列1个或多个数据列**sdf\_with\_unique\_id(x, id = "id")**  
添加唯一ID列**sdf\_predict(object, newdata)**  
创建含有预测值的Spark DataFrame

### ML转换器

```
ft_binarizer(my_table, input.col = "Petal_
Length", output.col = "petal_large",
threshold = 1.2)
```

适用于所有函数的参数:  
`x, input.col = NULL, output.col =  
NULL`

```
ft_binarizer(threshold = 0.5)
基于阈值把数值列二元化
```

**ft\_bucketizer(splits)**

把数字列转为离散列

```
ft_discrete_cosine_transform(inver
se = FALSE)
把时域转为频域
```

```
ft_elementwise_product(scaling.col)
计算两个数列的元素乘积
```

**ft\_index\_to\_string()**

把索引列转为字符串列

**ft\_one\_hot\_encoder()**

把连续值转为二进制向量

```
ft_quantile_discretizer(n.buckets
= 5L)
把连续值转为分箱分类值
```

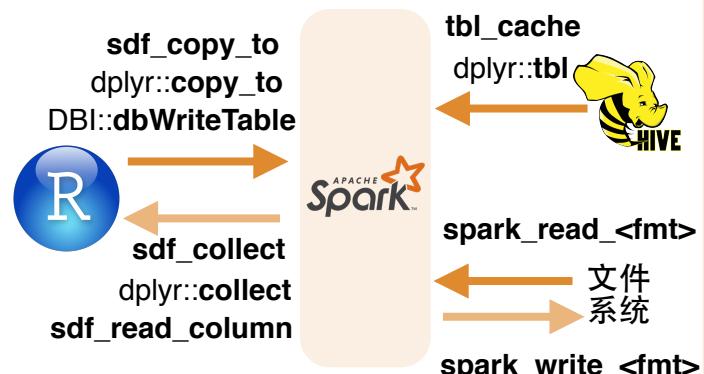
**ft\_sql\_transformer(sql)**

```
ft_string_indexer(params = NULL)
把字符串列转为索引列
```

**ft\_vectorAssembler()**

合并向量为单行向量

## 从Apache Spark读入或写出



### 扩展包

创建一个R软件包，用于调用完整的Spark API  
并为Spark软件包提供接口

#### 核心类型

**spark\_connection()** R与Spark Shell进程之间的连接对象**spark\_jobj()** 远程Spark对象实例**spark\_dataframe()** 远程Spark DataFrame对象实例

#### 在R中调用Spark

**invoke()** 调用Java对象函数**invoke\_new()** 通过构造函数创建一个新对象**invoke\_static()** 调用静态函数

#### 机器学习扩展包

**ml\_create\_dummy\_variables()** ml\_options()**ml\_prepare\_dataframe()** ml\_model()**ml\_prepare\_response\_features\_intercept()**

## 建模 (MLlib)

```
ml_decision_tree(my_table, response = "Species", features =
c("Petal_Length", "Petal_Width"))
```

```
ml_als_factorization(x, rating.column = "rating", user.column =
"user", item.column = "item", rank = 10L, regularization.parameter
= 0.1, iter.max = 10L, ml.options = ml_options())
```

```
ml_decision_tree(x, response, features, max.bins = 32L, max.depth
= 5L, type = c("auto", "regression", "classification"), ml.options =
ml_options())
```

相同选项也适用于: `ml_gradient_boosted_trees`

```
ml_generalized_linear_regression(x, response, features,
intercept = TRUE, family = gaussian(link = "identity"), iter.max =
100L, ml.options = ml_options())
```

```
ml_kmeans(x, centers, iter.max = 100, features = dplyr::tbl_vars(x),
compute.cost = TRUE, tolerance = 1e-04, ml.options =
ml_options())
```

```
ml_lda(x, features = dplyr::tbl_vars(x), k = length(features), alpha =
(50/k) + 1, beta = 0.1 + 1, ml.options = ml_options())
```

```
ml_linear_regression(x, response, features, intercept = TRUE,
alpha = 0, lambda = 0, iter.max = 100L, ml.options = ml_options())
```

相同选项也适用于: `ml_logistic_regression`

```
ml_multilayer_perceptron(x, response, features, layers, iter.max
= 100, seed = sample(.Machine$integer.max, 1), ml.options =
ml_options())
```

```
ml_naive_bayes(x, response, features, lambda = 0, ml.options =
ml_options())
```

```
ml_one_vs_rest(x, classifier, response, features, ml.options =
ml_options())
```

```
ml_pca(x, features = dplyr::tbl_vars(x), ml.options = ml_options())
```

```
ml_random_forest(x, response, features, max.bins = 32L,
max.depth = 5L, num.trees = 20L, type = c("auto", "regression",
"classification"), ml.options = ml_options())
```

```
ml_survival_regression(x, response, features, intercept =
TRUE, censor = "censor", iter.max = 100L, ml.options =
ml_options())
```

```
ml_binary_classification_eval(predicted_tbl_spark, label,
score, metric = "areaUnderROC")
```

```
ml_classification_eval(predicted_tbl_spark, label, predicted_lbl,
metric = "f1")
```

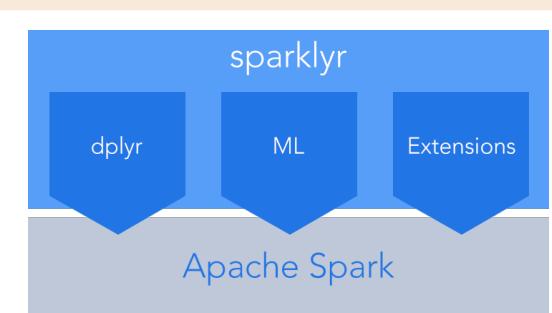
```
ml_tree_feature_importance(sc, model)
```

### sparklyr

是实现



与R的接口

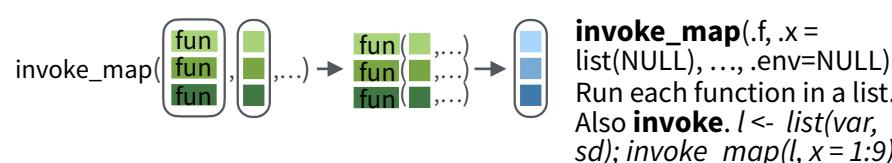
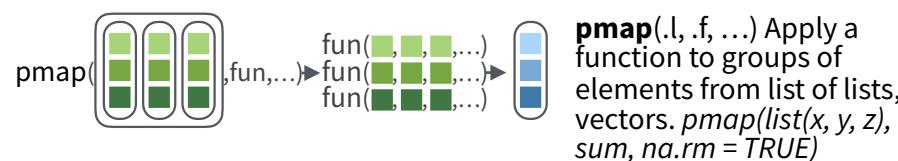
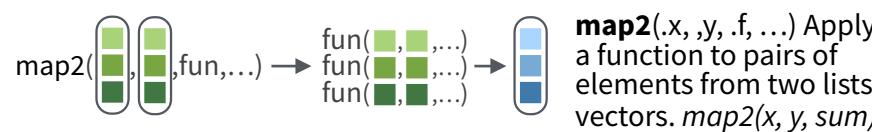
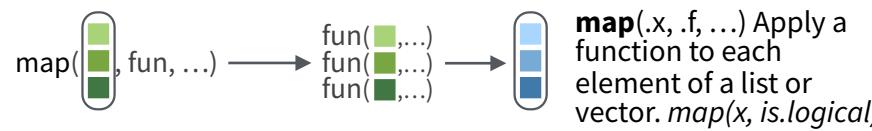


# Apply functions with purrr :: CHEAT SHEET



## Apply Functions

Map functions apply a function iteratively to each element of a list or vector.



**lmap(x, f, ...)** Apply function to each list-element of a list or vector.  
**imap(x, f, ...)** Apply .f to each element of a list or vector and its index.

### OUTPUT

**map()**, **map2()**, **pmap()**, **imap** and **invoke\_map** each return a list. Use a suffixed version to return the results as a specific type of flat vector, e.g. **map2\_chr**, **pmap\_lgl**, etc.

Use **walk**, **walk2**, and **pwalk** to trigger side effects. Each return its input invisibly.

### SHORTCUTS - within a purrr function:

"**name**" becomes **function(x)x[["name"]]**, e.g. **map(l, "a")** extracts *a* from each element of *l*

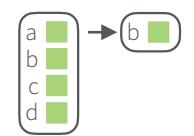
**~.x** becomes **function(x)x**, e.g. **map(l, ~.x + y)** becomes **map2(l, p, function(l, p) l + p)**

**~..1..2** etc becomes **function(..1, ..2, etc)** **..1..2** etc, e.g. **pmap(list(a, b, c), ~..3 + ..1 ..2)** becomes **pmap(list(a, b, c), function(a, b, c) c + a - b)**

| function       | returns                                            |
|----------------|----------------------------------------------------|
| <b>map</b>     | list                                               |
| <b>map_chr</b> | character vector                                   |
| <b>map_dbl</b> | double (numeric) vector                            |
| <b>map_dfc</b> | data frame (column bind)                           |
| <b>map_dfr</b> | data frame (row bind)                              |
| <b>map_int</b> | integer vector                                     |
| <b>map_lgl</b> | logical vector                                     |
| <b>walk</b>    | triggers side effects, returns the input invisibly |

## Work with Lists

### FILTER LISTS



**pluck(x, ..., .default=NULL)** Select an element by name or index, **pluck(x, "b")**, or its attribute with **attr\_getter**. **pluck(x, "b", attr\_getter("n"))**



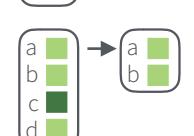
**keep(x, .p, ...)** Select elements that pass a logical test. **keep(x, is.character)**



**discard(x, .p, ...)** Select elements that do not pass a logical test. **discard(x, is.na)**

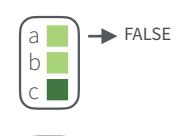


**compact(x, .p = identity)** Drop empty elements. **compact(x)**

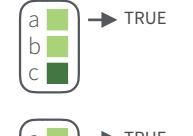


**head\_while(x, .p, ...)** Return head elements until one does not pass. Also **tail\_while**. **head\_while(x, is.character)**

### SUMMARISE LISTS



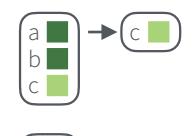
**every(x, .p, ...)** Do all element pass a test? **every(x, is.character)**



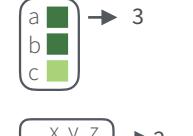
**some(x, .p, ...)** Do some elements pass a test? **some(x, is.character)**



**has\_element(x, "foo")** Does a list contain an element? **has\_element(x, "foo")**



**detect(x, .f, ..., .right=FALSE, .p)** Find first element to pass. **detect(x, is.character)**

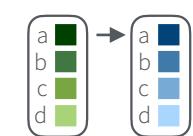


**detect\_index(x, .f, ..., .right=FALSE, .p)** Find index of first element to pass. **detect\_index(x, is.character)**

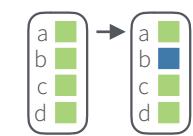


**vec\_depth(x)** Return depth (number of levels of indexes). **vec\_depth(x)**

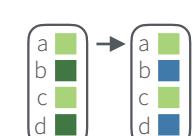
### TRANSFORM LISTS



**modify(x, .f, ...)** Apply function to each element. Also **map**, **map\_chr**, **map\_dbl**, **map\_dfc**, **map\_dfr**, **map\_int**, **map\_lgl**. **modify(x, ~.x + 2)**



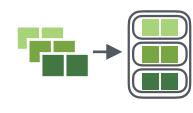
**modify\_at(x, .at, .f, ...)** Apply function to elements by name or index. Also **map\_at**. **modify\_at(x, "b", ~.x + 2)**



**modify\_if(x, .p, .f, ...)** Apply function to elements that pass a test. Also **map\_if**. **modify\_if(x, is.numeric, ~.x + 2)**

**modify\_depth(x, .depth, .f, ...)** Apply function to each element at a given level of a list. **modify\_depth(x, 1, ~.x + 2)**

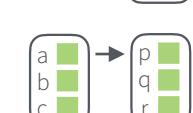
### WORK WITH LISTS



**array\_tree(array, margin = NULL)** Turn array into list. Also **array\_branch**. **array\_tree(x, margin = 3)**



**cross2(x, y, .filter = NULL)** All combinations of *x* and *y*. Also **cross**, **cross3**, **cross\_df**. **cross2(1:3, 4:6)**

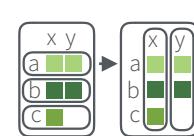


**set\_names(x, nm = x)** Set the names of a vector/list directly or with a function. **set\_names(x, c("p", "q", "r"))** **set\_names(x, tolower)**

### RESHAPE LISTS

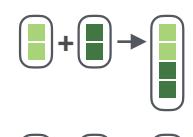


**flatten(x)** Remove a level of indexes from a list. Also **flatten\_chr**, **flatten\_dbl**, **flatten\_dfc**, **flatten\_dfr**, **flatten\_int**, **flatten\_lgl**. **flatten(x)**

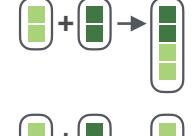


**transpose(x, .names = NULL)** Transposes the index order in a multi-level list. **transpose(x)**

### JOIN (TO) LISTS



**append(x, values, after = length(x))** Add to end of list. **append(x, list(d = 1))**

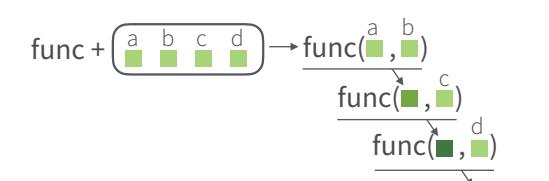


**prepend(x, values, before = 1)** Add to start of list. **prepend(x, list(d = 1))**

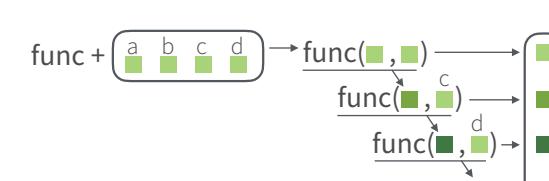


**splice(...)** Combine objects into a list, storing S3 objects as sub-lists. **splice(x, y, "foo")**

## Reduce Lists



**reduce(x, .f, ..., .init)** Apply function recursively to each element of a list or vector. Also **reduce\_right**, **reduce2**, **reduce2\_right**. **reduce(x, sum)**



**accumulate(x, .f, ..., .init)** Reduce, but also return intermediate results. Also **accumulate\_right**. **accumulate(x, sum)**

**compose()** Compose multiple functions.

**lift()** Change the type of input a function takes. Also **lift\_dl**, **lift\_lv**, **lift\_vl**.

**rerun()** Rerun expression n times.

**safely()** Modify func to return list of results and errors.

**negate()** Negate a predicate function (a pipe friendly !)

**partial()** Create a version of a function that has some args preset to values.

**quietly()** Modify function to return list of results, output, messages, warnings.

**possibly()** Modify function to return default value whenever an error occurs (instead of error).



# Nested Data

A **nested data frame** stores individual tables within the cells of a larger, organizing table.

| "cell" contents |         |         |         |
|-----------------|---------|---------|---------|
| Sepal.L         | Sepal.W | Petal.L | Petal.W |
| 5.1             | 3.5     | 1.4     | 0.2     |
| 4.9             | 3.0     | 1.4     | 0.2     |
| 4.7             | 3.2     | 1.3     | 0.2     |
| 4.6             | 3.1     | 1.5     | 0.2     |
| 5.0             | 3.6     | 1.4     | 0.2     |

`n_iris$data[[1]]`

| nested data frame |                   |  |  |
|-------------------|-------------------|--|--|
| Species           | data              |  |  |
| setosa            | <tibble [50 x 4]> |  |  |
| versicolor        | <tibble [50 x 4]> |  |  |
| virginica         | <tibble [50 x 4]> |  |  |

`n_iris`

| Sepal.L | Sepal.W | Petal.L | Petal.W |
|---------|---------|---------|---------|
| 7.0     | 3.2     | 4.7     | 1.4     |
| 6.4     | 3.2     | 4.5     | 1.5     |
| 6.9     | 3.1     | 4.9     | 1.5     |
| 5.5     | 2.3     | 4.0     | 1.3     |
| 6.5     | 2.8     | 4.6     | 1.5     |

`n_iris$data[[2]]`

| Sepal.L | Sepal.W | Petal.L | Petal.W |
|---------|---------|---------|---------|
| 6.3     | 3.3     | 6.0     | 2.5     |
| 5.8     | 2.7     | 5.1     | 1.9     |
| 7.1     | 3.0     | 5.9     | 2.1     |
| 6.3     | 2.9     | 5.6     | 1.8     |
| 6.5     | 3.0     | 5.8     | 2.2     |

`n_iris$data[[3]]`

Use a nested data frame to:

- preserve relationships between observations and subsets of data
- manipulate many sub-tables at once with the **purrr** functions **map()**, **map2()**, or **pmap()**.

Use a two step process to create a nested data frame:

1. Group the data frame into groups with **dplyr::group\_by()**
2. Use **nest()** to create a nested data frame with one row per group

| Species | S.L | S.W | P.L | P.W | Species | S.L | S.W | P.L | P.W |
|---------|-----|-----|-----|-----|---------|-----|-----|-----|-----|
| setosa  | 5.1 | 3.5 | 1.4 | 0.2 | setosa  | 5.1 | 3.5 | 1.4 | 0.2 |
| setosa  | 4.9 | 3.0 | 1.4 | 0.2 | setosa  | 4.9 | 3.0 | 1.4 | 0.2 |
| setosa  | 4.7 | 3.2 | 1.3 | 0.2 | setosa  | 4.7 | 3.2 | 1.3 | 0.2 |
| setosa  | 4.6 | 3.1 | 1.5 | 0.2 | setosa  | 4.6 | 3.1 | 1.5 | 0.2 |
| setosa  | 5.0 | 3.6 | 1.4 | 0.2 | setosa  | 5.0 | 3.6 | 1.4 | 0.2 |
| versi   | 7.0 | 3.2 | 4.7 | 1.4 | versi   | 7.0 | 3.2 | 4.7 | 1.4 |
| versi   | 6.4 | 3.2 | 4.5 | 1.5 | versi   | 6.4 | 3.2 | 4.5 | 1.5 |
| versi   | 6.9 | 3.1 | 4.9 | 1.5 | versi   | 6.9 | 3.1 | 4.9 | 1.5 |
| versi   | 5.5 | 2.3 | 4.0 | 1.3 | versi   | 5.5 | 2.3 | 4.0 | 1.3 |
| virgini | 6.3 | 2.8 | 4.6 | 1.5 | virgini | 6.3 | 2.8 | 4.6 | 1.5 |
| virgini | 6.3 | 3.3 | 6.0 | 2.5 | virgini | 6.3 | 3.3 | 6.0 | 2.5 |
| virgini | 5.8 | 2.7 | 5.1 | 1.9 | virgini | 5.8 | 2.7 | 5.1 | 1.9 |
| virgini | 7.1 | 3.0 | 5.9 | 2.1 | virgini | 7.1 | 3.0 | 5.9 | 2.1 |
| virgini | 6.3 | 2.9 | 5.6 | 1.8 | virgini | 6.3 | 2.9 | 5.6 | 1.8 |
| virgini | 6.5 | 3.0 | 5.8 | 2.2 | virgini | 6.5 | 3.0 | 5.8 | 2.2 |

`n_iris <- iris %>% group_by(Species) %>% nest()`

**tidy::nest**(data, ..., .key = data)

For grouped data, moves groups into cells as data frames.

Unnest a nested data frame with **unnest()**:

| Species | data            |         |     |     |     |     |
|---------|-----------------|---------|-----|-----|-----|-----|
| Species | S.L             | S.W     | P.L | P.W |     |     |
| setos   | <tibble [50x4]> | setos   | 5.1 | 3.5 | 1.4 | 0.2 |
| versi   | <tibble [50x4]> | setos   | 4.9 | 3.0 | 1.4 | 0.2 |
| virgini | <tibble [50x4]> | setos   | 4.7 | 3.2 | 1.3 | 0.2 |
|         |                 | setosa  | 4.6 | 3.1 | 1.5 | 0.2 |
|         |                 | setos   | 5.0 | 3.6 | 1.4 | 0.2 |
|         |                 | versi   | 7.0 | 3.2 | 4.7 | 1.4 |
|         |                 | versi   | 6.4 | 3.2 | 4.5 | 1.5 |
|         |                 | versi   | 6.9 | 3.1 | 4.9 | 1.5 |
|         |                 | versi   | 5.5 | 2.3 | 4.0 | 1.3 |
|         |                 | virgini | 6.3 | 2.8 | 4.6 | 1.5 |
|         |                 | virgini | 6.3 | 3.3 | 6.0 | 2.5 |
|         |                 | virgini | 5.8 | 2.7 | 5.1 | 1.9 |
|         |                 | virgini | 7.1 | 3.0 | 5.9 | 2.1 |
|         |                 | virgini | 6.3 | 2.9 | 5.6 | 1.8 |
|         |                 | virgini | 6.5 | 3.0 | 5.8 | 2.2 |

`n_iris %>% unnest()`

**tidy::unnest**(data, ..., .drop = NA, .id=NULL, .sep=NULL)  
Unnests a nested data frame.

# List Column Workflow

Nested data frames use a **list column**, a list that is stored as a column vector of a data frame. A typical **workflow** for list columns:

## 1 Make a list column

| Species | S.L | S.W | P.L | P.W |
|---------|-----|-----|-----|-----|
| setosa  | 5.1 | 3.5 | 1.4 | 0.2 |
| setosa  | 4.9 | 3.0 | 1.4 | 0.2 |
| setosa  | 4.7 | 3.2 | 1.3 | 0.2 |
| setosa  | 4.6 | 3.1 | 1.5 | 0.2 |
| versi   | 7.0 | 3.2 | 4.7 | 1.4 |
| versi   | 6.4 | 3.2 | 4.5 | 1.5 |
| versi   | 6.9 | 3.1 | 4.9 | 1.5 |
| versi   | 5.5 | 2.3 | 4.0 | 1.3 |
| virgini | 6.3 | 2.8 | 4.6 | 1.5 |
| virgini | 6.3 | 3.3 | 6.0 | 2.5 |
| virgini | 5.8 | 2.7 | 5.1 | 1.9 |
| virgini | 7.1 | 3.0 | 5.9 | 2.1 |
| virgini | 6.3 | 2.9 | 5.6 | 1.8 |
| virgini | 6.5 | 3.0 | 5.8 | 2.2 |

```
n_iris <- iris %>%
 group_by(Species) %>%
 nest()
```

## 2 Work with list columns

| Species | data            | model    |
|---------|-----------------|----------|
| setosa  | <tibble [50x4]> | <S3: lm> |
| versi   | <tibble [50x4]> | <S3: lm> |
| virgini | <tibble [50x4]> | <S3: lm> |

```
mod_fun <- function(df)
 lm(Sepal.Length ~ ., data = df)
```

```
m_iris <- n_iris %>%
 mutate(model = map(data, mod_fun))
```

## 3 Simplify the list column

| Species | beta |
|---------|------|
| setos   | 2.35 |
| versi   | 1.89 |
| virgini | 0.69 |

```
b_fun <- function(mod)
 coefficients(mod)[[1]]
```

```
m_iris %>% transmute(Species,
 beta = map_dbl(model, b_fun))
```

# RStudio IDE :: CHEAT SHEET

## Documents and Apps

   Open Shiny, R Markdown, knitr, Sweave, LaTeX, .Rd files and more in Source Pane

Check spelling  Render output  Choose output format  Choose output location  Insert code chunk 

Jump to previous chunk  Jump to next chunk  Run selected lines  Publish to server  Show file outline 

Access markdown guide at **Help > Markdown Quick Reference**

Jump to chunk  Set knitr chunk options  Run this and all previous code chunks  Run this code chunk 

RStudio recognizes that files named **app.R**, **server.R**, **ui.R**, and **global.R** belong to a shiny app

Run app  Choose location to view app  Publish to shinyapps.io or server  Manage publish accounts 

## Debug Mode

Open with **debug()**, **browser()**, or a breakpoint. RStudio will open the debugger mode when it encounters a breakpoint while executing code.

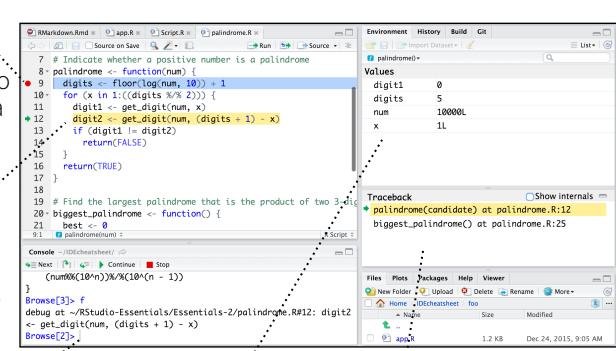
Click next to line number to add/remove a breakpoint.

Highlighted line shows where execution has paused

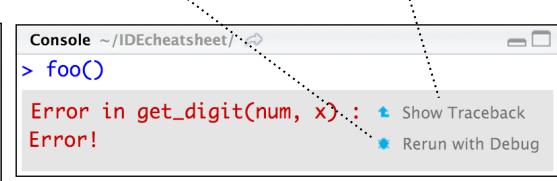
Run commands in environment where execution has paused

Examine variables in executing environment

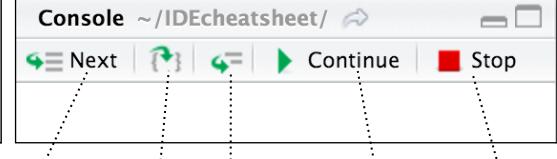
Select function in traceback to debug



Launch debugger mode from origin of error



Open traceback to examine the functions that R called before the error occurred



Step through code one line at a time

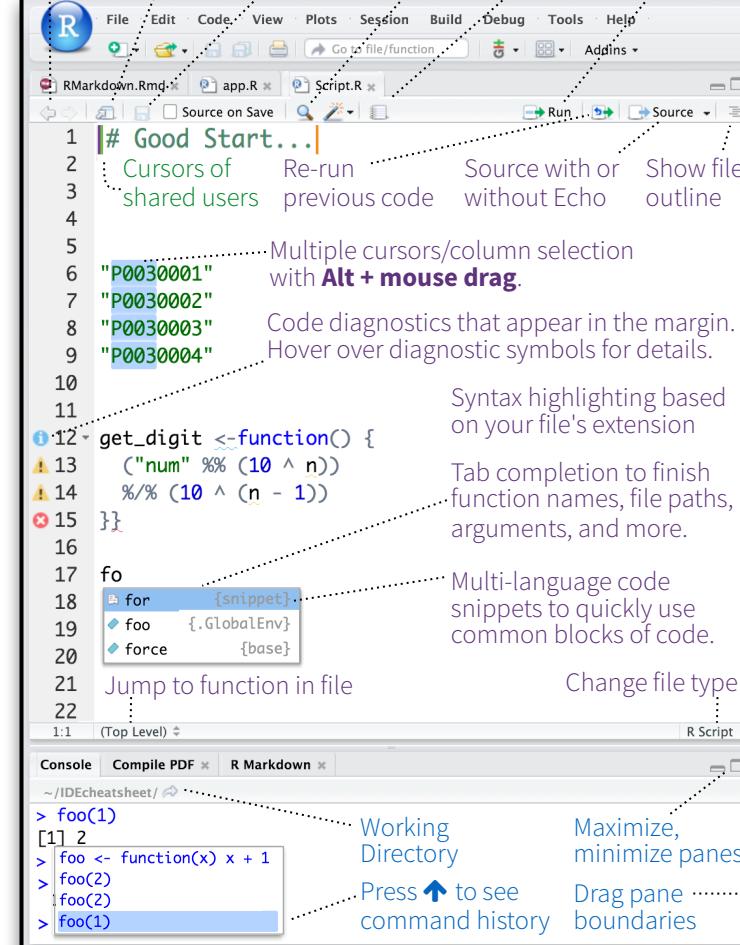
Step into and out of functions to run

Resume

Quit debug execution mode

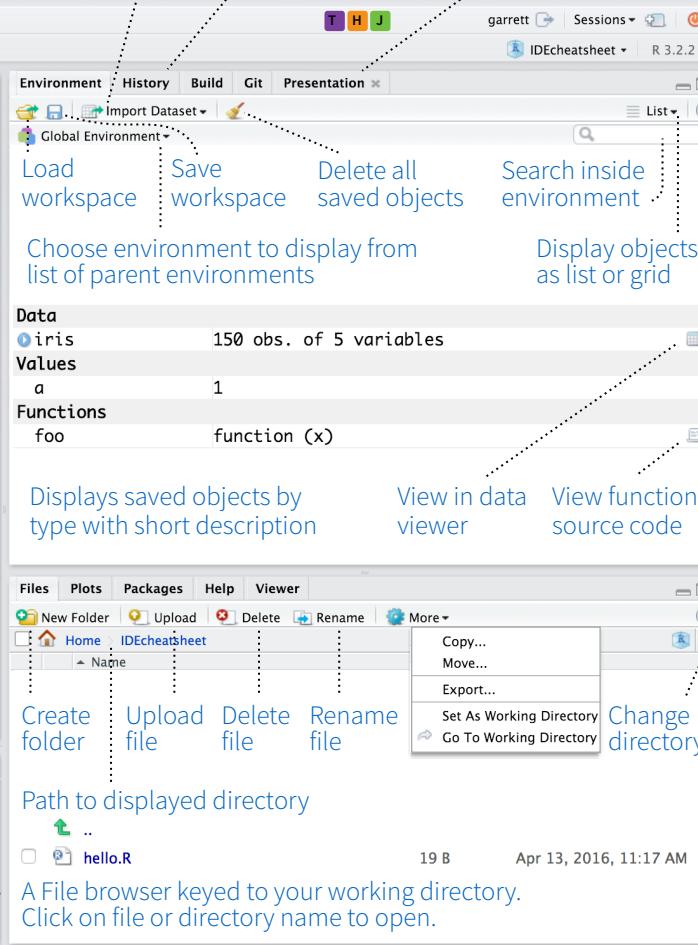
## Write Code

Navigate tabs  Open in new window  Save  Find and replace  Compile as notebook  Run selected code 



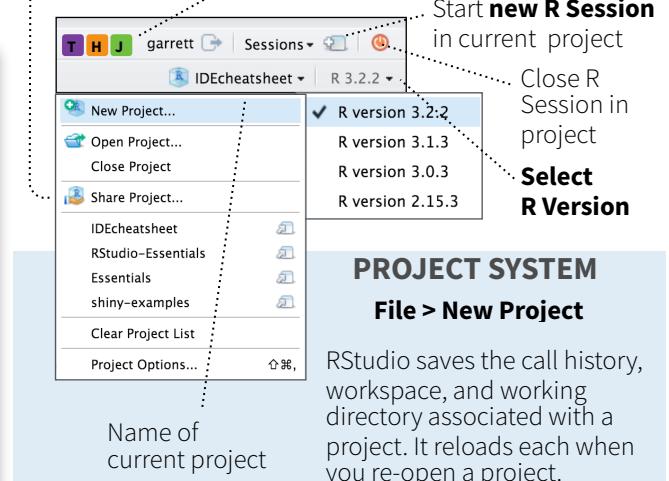
## R Support

**Import data** with wizard  History of past commands to run/copy  Display .RPres slideshows  **File > New File > R Presentation**



## Pro Features

**Share Project** Active shared with Collaborators... 



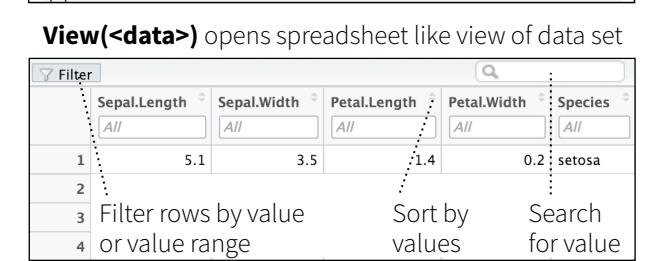
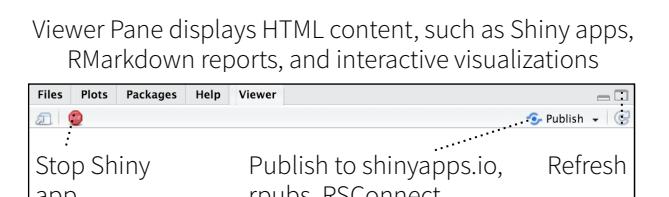
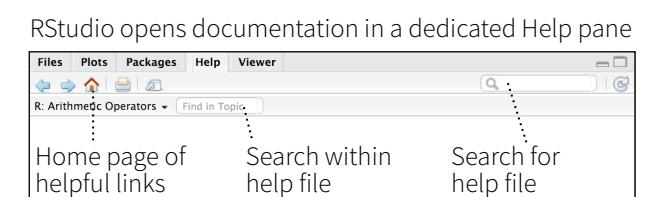
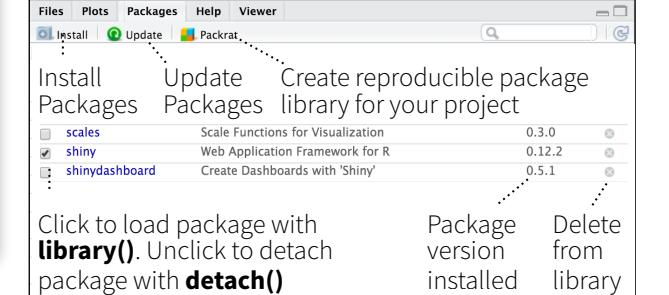
### PROJECT SYSTEM

**File > New Project**

RStudio saves the call history, workspace, and working directory associated with a project. It reloads each when you re-open a project.



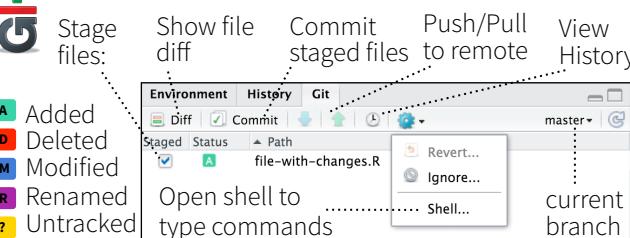
GUI Package manager lists every installed package



## Version Control

with Git or SVN

Turn on at **Tools > Project Options > Git/SVN**

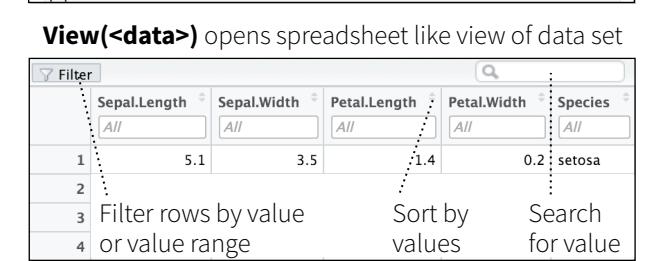
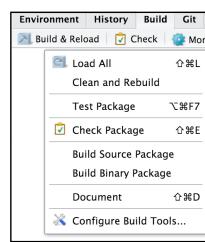


## Package Writing

### File > New Project > New Directory > R Package

Turn project into package, Enable roxygen documentation with **Tools > Project Options > Build Tools**

Roxygen guide at **Help > Roxygen Quick Reference**



## 1 LAYOUT

Move focus to Source Editor  
Move focus to Console  
Move focus to Help  
Show History  
Show Files  
Show Plots  
Show Packages  
Show Environment  
Show Git/SVN  
Show Build

**Windows/Linux Mac**  
Ctrl+1 Ctrl+1  
Ctrl+2 Ctrl+2  
Ctrl+3 Ctrl+3  
Ctrl+4 Ctrl+4  
Ctrl+5 Ctrl+5  
Ctrl+6 Ctrl+6  
Ctrl+7 Ctrl+7  
Ctrl+8 Ctrl+8  
Ctrl+9 Ctrl+9  
Ctrl+0 Ctrl+0

## 2 RUN CODE

### Search command history

Navigate command history  
Move cursor to start of line  
Move cursor to end of line  
Change working directory

### Interrupt current command

### Clear console

Quit Session (desktop only)

### Restart R Session

Run current (retain cursor)  
Run from current to end  
Run the current function  
Source a file

### Source the current file

Source with echo

**Windows/Linux Mac**

**Ctrl+↑** Cmd+↑  
**↑/↓** ↑/↓  
Home Cmd+←  
End Cmd+→  
Ctrl+Shift+H Ctrl+Shift+H

**Esc** Esc  
**Ctrl+L** Ctrl+L  
Ctrl+Q Cmd+Q

**Ctrl+Shift+F10** Cmd+Shift+F10  
**Ctrl+Enter** Cmd+Enter

Alt+Enter Option+Enter  
Ctrl+Alt+E Cmd+Option+E  
Ctrl+Alt+F Cmd+Option+F  
Ctrl+Alt+G Cmd+Option+G

**Ctrl+Shift+S** Cmd+Shift+S

Ctrl+Shift+Enter Cmd+Shift+Enter

## 4 WRITE CODE

### Attempt completion

Navigate candidates  
Accept candidate  
Dismiss candidates  
Undo  
Redo  
Cut  
Copy  
Paste  
Select All  
Delete Line

Select  
Select Word

Select to Line Start  
Select to Line End  
Select Page Up/Down  
Select to Start/End  
Delete Word Left  
Delete Word Right  
Delete to Line End  
Delete to Line Start

Indent

Outdent  
Yank line up to cursor  
Yank line after cursor  
Insert yanked text

**Insert <->**

**Insert %>%**

Show help for function

Show source code

New document

New document (Chrome)

Open document

Save document

Close document

Close document (Chrome)

Close all documents

Extract function

Extract variable

Reindent lines

**(Un)Comment lines**

Reflow Comment

Reformat Selection

Select within braces

Show Diagnostics

Transpose Letters

Move Lines Up/Down

Copy Lines Up/Down

Add New Cursor Above

Add New Cursor Below

Move Active Cursor Up

Move Active Cursor Down

Find and Replace

Use Selection for Find

Replace and Find

## Windows /Linux

### Tab or Ctrl+Space

↑/↓ Enter, Tab, or →  
Esc Ctrl+Z  
Ctrl+Shift+Z  
Ctrl+X  
Cmd+C  
Cmd+V  
Ctrl+A  
Ctrl+D  
Shift+[Arrow]  
Ctrl+Shift+←/→

Alt+Shift+←  
Alt+Shift+→

Shift+PageUp/Down

Shift+Alt+↑/↓

Ctrl+Opt+Backspace  
Option+Delete  
Ctrl+K

Option+Backspace

Tab (at start of line)

Shift+Tab

Ctrl+U

Ctrl+K

Ctrl+Y

**Alt+-**

**Ctrl+Shift+M**

F1

F2

Cmd+Shift+N

Ctrl+Alt+Shift+N

Ctrl+O

Ctrl+S

Cmd+W

Cmd+Option+W

Ctrl+Shift+W

Ctrl+Alt+X

Cmd+Option+X

Ctrl+Alt+V

Ctrl+I

**Ctrl+Shift+C**

Ctrl+Shift+/

Cmd+Shift+A

Ctrl+Shift+E

Ctrl+Shift+E

Cmd+Shift+Opt+P

Ctrl+T

Alt+↑/↓

Shift+Alt+↑/↓

Cmd+Option+↑/↓

Ctrl+Option+Up

Ctrl+Option+Down

Ctrl+Option+Shift+Up

Ctrl+Opt+Shift+Down

Ctrl+F

Ctrl+F3

Ctrl+Shift+J

## Mac

### Tab or Cmd+Space

↑/↓

Enter, Tab, or →

Esc

Cmd+Z

Cmd+Shift+Z

Cmd+X

Cmd+C

Cmd+V

Cmd+A

Cmd+D

Shift+[Arrow]

Option+Shift+←/→

Cmd+Shift+←

Cmd+Shift+→

Shift+PageUp/Down

Cmd+Shift+↑/↓

Ctrl+Opt+Backspace

Option+Delete

Ctrl+K

Option+Backspace

Tab (at start of line)

Shift+Tab

Ctrl+U

Ctrl+K

Ctrl+Y

**Alt+-**

**Ctrl+Shift+M**

F1

F2

Cmd+Shift+N

Cmd+Shift+Opt+N

Cmd+O

Ctrl+S

Cmd+W

Cmd+Option+W

Cmd+Shift+W

Ctrl+Alt+X

Cmd+Option+X

Cmd+Option+V

Cmd+I

**Cmd+Shift+C**

Cmd+Shift+/

Cmd+Shift+A

Ctrl+Shift+E

Ctrl+Shift+E

Cmd+Shift+Opt+P

Ctrl+T

Alt+↑/↓

Shift+Alt+↑/↓

Cmd+Option+↑/↓

Ctrl+Option+Up

Ctrl+Option+Down

Ctrl+Option+Shift+Up

Ctrl+Opt+Shift+Down

Ctrl+F

Ctrl+F3

Cmd+E

Cmd+Shift+J

## WHY RSTUDIO SERVER PRO?

RSP extends the open source server with a commercial license, support, and more:

- open and run multiple R sessions at once
  - tune your resources to improve performance
  - edit the same project at the same time as others
  - see what you and others are doing on your server
  - switch easily from one version of R to a different version
  - integrate with your authentication, authorization, and audit practices
- Download a free 45 day evaluation at [www.rstudio.com/products/rstudio-server-pro/](http://www.rstudio.com/products/rstudio-server-pro/)



## 5 DEBUG CODE

Toggle Breakpoint

Execute Next Line

Step Into Function

Finish Function/Loop

Continue

Stop Debugging

## Windows/Linux Mac

Shift+F9 Shift+F9

F10 F10

Shift+F4 Shift+F4

Shift+F6 Shift+F6

Shift+F5 Shift+F5

Shift+F8 Shift+F8

## 6 VERSION CONTROL

Show diff

Commit changes

Scroll diff view

Stage/Unstage (Git)

Stage/Unstage and move to next

## Windows/Linux Mac

Ctrl+Alt+D Ctrl+Option+D

# R Syntax Comparison :: CHEAT SHEET

## Dollar sign syntax

```
goal(data$x, data$y)
```

### SUMMARY STATISTICS:

one continuous variable:

```
mean(mtcars$mpg)
```

one categorical variable:

```
table(mtcars$cyl)
```

two categorical variables:

```
table(mtcars$cyl, mtcars$am)
```

one continuous, one categorical:

```
mean(mtcars$mpg [mtcars$cyl==4])
```

```
mean(mtcars$mpg [mtcars$cyl==6])
```

```
mean(mtcars$mpg [mtcars$cyl==8])
```

### PLOTTING:

one continuous variable:

```
hist(mtcars$disp)
```

```
boxplot(mtcars$disp)
```

one categorical variable:

```
barplot(table(mtcars$cyl))
```

two continuous variables:

```
plot(mtcars$disp, mtcars$mpg)
```

two categorical variables:

```
mosaicplot(table(mtcars$am, mtcars$cyl))
```

one continuous, one categorical:

```
histogram(mtcars$disp[mtcars$cyl==4])
```

```
histogram(mtcars$disp[mtcars$cyl==6])
```

```
histogram(mtcars$disp[mtcars$cyl==8])
```

```
boxplot(mtcars$disp[mtcars$cyl==4])
boxplot(mtcars$disp[mtcars$cyl==6])
boxplot(mtcars$disp[mtcars$cyl==8])
```

### WRANGLING:

subsetting:

```
mtcars[mtcars$mpg>30,]
```

making a new variable:

```
mtcars$efficient[mtcars$mpg>30] <- TRUE
```

```
mtcars$efficient[mtcars$mpg<30] <- FALSE
```

## Formula syntax

```
goal(y~x|z, data=data, group=w)
```

### SUMMARY STATISTICS:

one continuous variable:

```
mosaic::mean(~mpg, data=mtcars)
```

one categorical variable:

```
mosaic::tally(~cyl, data=mtcars)
```

two categorical variables:

```
mosaic::tally(cyl~am, data=mtcars)
```

one continuous, one categorical:

```
mosaic::mean(mpg~cyl, data=mtcars)
```

tilde

### PLOTTING:

one continuous variable:

```
lattice::histogram(~disp, data=mtcars)
```

```
lattice::bwplot(~disp, data=mtcars)
```

one categorical variable:

```
mosaic::bargraph(~cyl, data=mtcars)
```

two continuous variables:

```
lattice::xyplot(mpg~disp, data=mtcars)
```

two categorical variables:

```
mosaic::bargraph(~am, data=mtcars, group=cyl)
```

one continuous, one categorical:

```
lattice::histogram(~disp|cyl, data=mtcars)
```

```
lattice::bwplot(cyl~disp, data=mtcars)
```

The variety of R syntaxes give you many ways to “say” the same thing

read across the cheatsheet to see how different syntaxes approach the same problem

## Tidyverse syntax

```
data %>% goal(x)
```

### SUMMARY STATISTICS:

one continuous variable:

```
mtcars %>% dplyr::summarize(mean(mpg))
```

one categorical variable:

```
mtcars %>% dplyr::group_by(cyl) %>%
dplyr::summarize(n())
```

two categorical variables:

```
mtcars %>% dplyr::group_by(cyl, am) %>%
dplyr::summarize(n())
```

one continuous, one categorical:

```
mtcars %>% dplyr::group_by(cyl) %>%
dplyr::summarize(mean(mpg))
```

the pipe

### PLOTTING:

one continuous variable:

```
ggplot2::qplot(x=mpg, data=mtcars, geom = "histogram")
```

```
ggplot2::qplot(y=disp, x=1, data=mtcars, geom="boxplot")
```

one categorical variable:

```
ggplot2::qplot(x=cyl, data=mtcars, geom="bar")
```

two continuous variables:

```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

two categorical variables:

```
ggplot2::qplot(x=factor(cyl), data=mtcars, geom="bar") +
facet_grid(.~am)
```

one continuous, one categorical:

```
ggplot2::qplot(x=disp, data=mtcars, geom = "histogram") +
facet_grid(.~cyl)
```

```
ggplot2::qplot(y=disp, x=factor(cyl), data=mtcars,
geom="boxplot")
```

### WRANGLING:

subsetting:

```
mtcars %>% dplyr::filter(mpg>30)
```

making a new variable:

```
mtcars <- mtcars %>%
dplyr::mutate(efficient = if_else(mpg>30, TRUE, FALSE))
```

# R Syntax Comparison :: CHEAT SHEET

**Syntax** is the set of rules that govern what code works and doesn't work in a programming language. Most programming languages offer one standardized syntax, but R allows package developers to specify their own syntax. As a result, there is a large variety of (equally valid) R syntaxes.

The three most prevalent R syntaxes are:

1. The **dollar sign syntax**, sometimes called **base R syntax**, expected by most base R functions. It is characterized by the use of `dataset$variablename`, and is also associated with square bracket subsetting, as in `dataset[1, 2]`. Almost all R functions will accept things passed to them in dollar sign syntax.
2. The **formula syntax**, used by modeling functions like `lm()`, lattice graphics, and `mosaic` summary statistics. It uses the tilde (~) to connect a response variable and one (or many) predictors. Many base R functions will accept formula syntax.
3. The **tidyverse syntax** used by `dplyr`, `tidyR`, and more. These functions expect data to be the first argument, which allows them to work with the "pipe" (%>%) from the `magrittr` package. Typically, `ggplot2` is thought of as part of the tidyverse, although it has its own flavor of the syntax using plus signs (+) to string pieces together. `ggplot2` author Hadley Wickham has said the package would have had different syntax if he had written it after learning about the pipe.

Educators often try to teach within one unified syntax, but most R programmers use some combination of all the syntaxes.

## Internet research tip:

If you are searching on google, StackOverflow, or another favorite online source and see code in a syntax you don't recognize:

- Check to see if the code is using one of the three common syntaxes listed on this cheatsheet
- Try your search again, using a keyword from the syntax name ("tidyverse") or a relevant package ("mosaic")



Sometimes particular syntaxes work, but are considered dangerous to use, because they are so easy to get wrong. For example, passing variable names without assigning them to a named argument.

## Even more ways to say the same thing

Even within one syntax, there are often variations that are equally valid. As a case study, let's look at the `ggplot2` syntax. `ggplot2` is the plotting package that lives within the tidyverse. If you read down this column, all the code here produces the same graphic.

### quickplot

`qplot()` stands for quickplot, and allows you to make quick plots. It doesn't have the full power of `ggplot2`, and it uses a slightly different syntax than the rest of the package.

```
ggplot2::qplot(x=disp, y=mpg, data=mtcars, geom="point")
```

```
ggplot2::qplot(x=disp, y=mpg, data=mtcars)
```

```
ggplot2::qplot(disp, mpg, data=mtcars)
```

read down this column for many pieces of code in one syntax that look different but produce the same graphic

### ggplot

To unlock the power of `ggplot2`, you need to use the `ggplot()` function (which sets up a plotting region) and add geoms to the plot.

```
ggplot2::ggplot(mtcars) +
 geom_point(aes(x=disp, y=mpg))
```

```
ggplot2::ggplot(data=mtcars) +
 geom_point(mapping=aes(x=disp, y=mpg))
```

plus adds layers

```
ggplot2::ggplot(mtcars, aes(x=disp, y=mpg)) +
 geom_point()
```

```
ggplot2::ggplot(mtcars, aes(x=disp)) +
 geom_point(aes(y=mpg))
```

### ggformula

The "third and a half way" to use the formula syntax, but get `ggplot2`-style graphics

```
ggformula::gf_point(mpg~disp, data= mtcars)
```

### formulas in base plots

Base R plots will also take the formula syntax, although it's not as commonly used

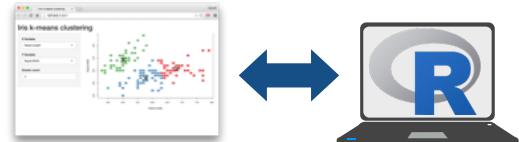
```
plot(mpg~disp, data=mtcars)
```



# Shiny :: CHEAT SHEET

## Basics

A **Shiny** app is a web page (**UI**) connected to a computer running a live R session (**Server**)



Users can manipulate the UI, which will cause the server to update the UI's displays (by running R code).

## APP TEMPLATE

Begin writing a new app with this template. Preview the app by running the code at the R command line.

```
library(shiny)
ui <- fluidPage(
 numericInput(inputId = "n",
 "Sample size", value = 25),
 plotOutput(outputId = "hist"))
server <- function(input, output) {
 output$hist <- renderPlot({
 hist(rnorm(input$n))
 })
}
shinyApp(ui = ui, server = server)
```

- **ui** - nested R functions that assemble an HTML user interface for your app
- **server** - a function with instructions on how to build and rebuild the R objects displayed in the UI
- **shinyApp** - combines **ui** and **server** into an app. Wrap with **runApp()** if calling from a sourced script or inside a function.

## SHARE YOUR APP

 [shinyapps.io](http://shinyapps.io) The easiest way to share your app is to host it on shinyapps.io, a cloud based service from RStudio

1. Create a free or professional account at <http://shinyapps.io>
2. Click the **Publish** icon in the RStudio IDE or run:  
`rsconnect::deployApp("<path to directory>")`

**Build or purchase your own Shiny Server**  
at [www.rstudio.com/products/shiny-server/](http://www.rstudio.com/products/shiny-server/)



## Building an App

Complete the template by adding arguments to `fluidPage()` and a body to the server function.

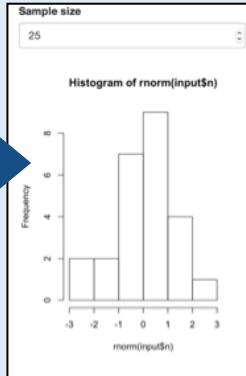
Add inputs to the UI with `*Input()` functions

Add outputs with `*Output()` functions

Tell server how to render outputs with R in the server function. To do this:

1. Refer to outputs with `output$<id>`
2. Refer to inputs with `input$<id>`
3. Wrap code in a `render*>()` function before saving to output

```
library(shiny)
ui <- fluidPage(
 numericInput(inputId = "n",
 "Sample size", value = 25),
 plotOutput(outputId = "hist"))
server <- function(input, output) {
 output$hist <- renderPlot({
 hist(rnorm(input$n))
 })
}
shinyApp(ui = ui, server = server)
```



Save your template as **app.R**. Alternatively, split your template into two files named **ui.R** and **server.R**.

```
library(shiny)
ui <- fluidPage(
 numericInput(inputId = "n",
 "Sample size", value = 25),
 plotOutput(outputId = "hist"))
server <- function(input, output) {
 output$hist <- renderPlot({
 hist(rnorm(input$n))
 })
}
shinyApp(ui = ui, server = server)
```

```
ui.R
fluidPage(
 numericInput(inputId = "n",
 "Sample size", value = 25),
 plotOutput(outputId = "hist"))

server.R
function(input, output) {
 output$hist <- renderPlot({
 hist(rnorm(input$n))
 })
}
```

**ui.R** contains everything you would save to ui.

**server.R** ends with the function you would save to server.

No need to call **shinyApp()**.

Save each app as a directory that holds an **app.R** file (or a **server.R** file and a **ui.R** file) plus optional extra files.

● ● ● **app-name**  
 .r  
 app.R  
 global.R  
 DESCRIPTION  
 README  
 <other files>  
 www

- The directory name is the name of the app
- (optional) defines objects available to both ui.R and server.R
- (optional) used in showcase mode
- (optional) data, scripts, etc.
- (optional) directory of files to share with web browsers (images, CSS, .js, etc.) Must be named "www"

Launch apps with  
`runApp(<path to directory>)`

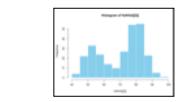
## Outputs - `render*`() and `*Output()` functions work together to add R output to the UI



`DT::renderDataTable(expr, options, callback, escape, env, quoted)`



`renderImage(expr, env, quoted, deleteFile)`



`renderPlot(expr, width, height, res, ..., env, quoted, func)`



`renderPrint(expr, env, quoted, func, width)`

`renderTable(expr, ..., env, quoted, func)`

`renderText(expr, env, quoted, func)`

`renderUI(expr, env, quoted, func)`

works with

`dataTableOutput(outputId, icon, ...)`

`imageOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)`

`plotOutput(outputId, width, height, click, dblclick, hover, hoverDelay, inline, hoverDelayType, brush, clickId, hoverId)`

`verbatimTextOutput(outputId)`

`tableOutput(outputId)`

`textOutput(outputId, container, inline)`

`uiOutput(outputId, inline, container, ...)`

`htmlOutput(outputId, inline, container, ...)`

## Inputs

collect values from the user

Access the current value of an input object with `input$<inputId>`. Input values are **reactive**.

Action

`actionButton(inputId, label, icon, ...)`

Link

`actionLink(inputId, label, icon, ...)`

- Choice 1
- Choice 2
- Choice 3
- Check me

`checkboxInput(inputId, label, value)`

`dateInput(inputId, label, value, min, max, format, startview, weekstart, language)`

`dateRangeInput(inputId, label, start, end, min, max, format, startview, weekstart, language, separator)`

Choose File

`fileInput(inputId, label, multiple, accept)`

1

`numericInput(inputId, label, value, min, max, step)`

.....

`passwordInput(inputId, label, value)`

- Choice A
- Choice B
- Choice C

Choice 1

Choice 1

Choice 2

0

5

10

Apply Changes

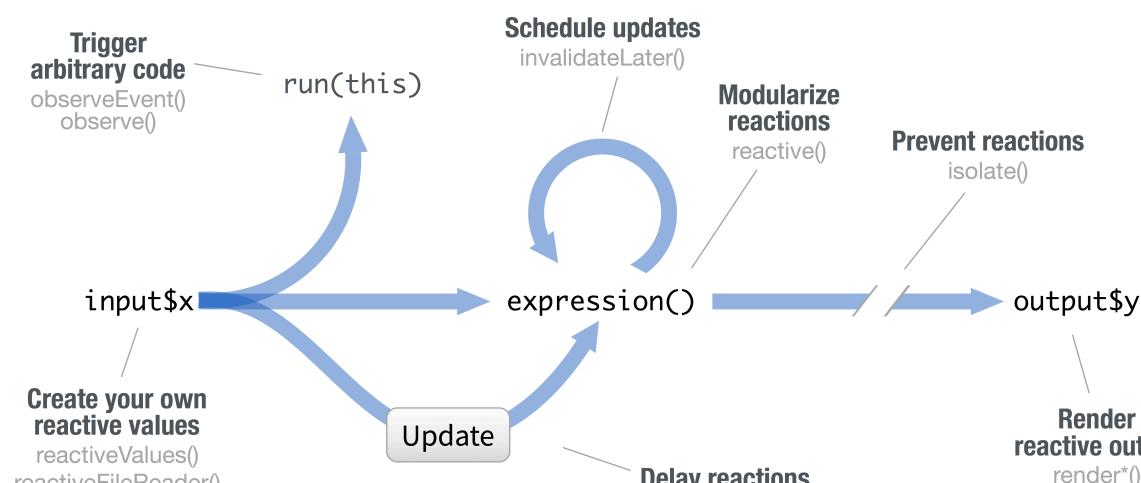
Enter text

`submitButton(text, icon)`  
(Prevents reactions across entire app)

`textInput(inputId, label, value)`

# Reactivity

Reactive values work together with reactive functions. Call a reactive value from within the arguments of one of these functions to avoid the error **Operation not allowed without an active reactive context**.



## CREATE YOUR OWN REACTIVE VALUES

```
example snippets
ui <- fluidPage(
 textInput("a","","A"))

server <-
function(input,output){
 rv <- reactiveValues()
 rv$number <- 5
}

reactiveValues() creates a
list of reactive values
whose values you can set.
```

## PREVENT REACTIONS

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b"))

server <-
function(input,output){
 output$b <-
 renderText({
 isolate({input$a})
 })
}

shinyApp(ui, server)
```

**isolate(expr)**  
Runs a code block.  
Returns a **non-reactive** copy of the results.

## RENDERS REACTIVE OUTPUT

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 textOutput("b"))

server <-
function(input,output){
 output$b <-
 renderText({
 input$a
 })
}

shinyApp(ui, server)
```

**render\*() functions**  
(see front page)

Builds an object to display. Will rerun code in body to rebuild the object whenever a reactive value in the code changes.  
Save the results to **output\$<outputId>**

## TRIGGER ARBITRARY CODE

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go"))

server <-
function(input,output){
 observeEvent(input$go,{
 print(input$a)
 })
}

shinyApp(ui, server)
```

**observeEvent(eventExpr, handlerExpr, event.env, event.quoted, handler.env, handler.quoted, label, suspended, priority, domain, autoDestroy, ignoreNULL)**

Runs code in 2nd argument when reactive values in 1st argument change. See **observe()** for alternative.

## MODULARIZE REACTIONS

```
ui <- fluidPage(
 textInput("a","","A"),
 textInput("z","","Z"),
 textOutput("b"))

server <-
function(input,output){
 re <- reactive({
 paste(input$a,input$z)
 })
 output$b <- renderText({
 re()
 })
}

shinyApp(ui, server)
```

**reactive(x, env, quoted, label, domain)**  
Creates a **reactive expression** that  

- caches its value to reduce computation
- can be called by other code
- notifies its dependencies when it has been invalidated

Call the expression with function syntax, e.g. **re()**

## DELAY REACTIONS

```
library(shiny)
ui <- fluidPage(
 textInput("a","","A"),
 actionButton("go","Go"),
 textOutput("b"))

server <-
function(input,output){
 re <- eventReactive(
 input$go,{input$a})
 output$b <- renderText({
 re()
 })
}

shinyApp(ui, server)
```

**eventReactive(eventExpr, valueExpr, event.env, event.quoted, value.env, value.quoted, label, domain, ignoreNULL)**

Creates reactive expression with code in 2nd argument that only invalidates when reactive values in 1st argument change.

## UI

An app's UI is an HTML document.

Use Shiny's functions to assemble this HTML with R.

```
fluidPage(
 textInput("a",""))
<div class="container-fluid">
<div class="form-group shiny-input-container">
<label for="a"></label>
<input id="a" type="text"
class="form-control" value="" />
</div>
</div>
```



Add static HTML elements with **tags**, a list of functions that parallel common HTML tags, e.g. **tags\$a()**. Unnamed arguments will be passed into the tag; named arguments will become tag attributes.

|                  |                  |              |                |                |
|------------------|------------------|--------------|----------------|----------------|
| tags\$a          | tags\$data       | tags\$h6     | tags\$nav      | tags\$span     |
| tags\$abbr       | tags\$datalist   | tags\$head   | tags\$noscript | tags\$strong   |
| tags\$address    | tags\$dd         | tags\$header | tags\$object   | tags\$style    |
| tags\$area       | tags\$del        | tags\$hgroup | tags\$ol       | tags\$sub      |
| tags\$article    | tags\$details    | tags\$hrg    | tags\$optgroup | tags\$summary  |
| tags\$aside      | tags\$dfn        | tags\$HTML   | tags\$option   | tags\$sup      |
| tags\$audio      | tags\$div        | tags\$si     | tags\$output   | tags\$table    |
| tags\$b          | tags\$dl         | tags\$iframe | tags\$p        | tags\$tbody    |
| tags\$base       | tags\$dt         | tags\$img    | tags\$param    | tags\$td       |
| tags\$bdi        | tags\$em         | tags\$input  | tags\$pre      | tags\$textarea |
| tags\$bdo        | tags\$embed      | tags\$ins    | tags\$progress | tags\$tfoot    |
| tags\$blockquote | tags\$eventsouce | tags\$kbd    | tags\$q        | tags\$th       |
| tags\$body       | tags\$fieldset   | tags\$keygen | tags\$ruby     | tags\$thead    |
| tags\$br         | tags\$figcaption | tags\$label  | tags\$rp       | tags\$time     |
| tags\$button     | tags\$figure     | tags\$legend | tags\$rt       | tags\$title    |
| tags\$canvas     | tags\$footer     | tags\$li     | tags\$ss       | tags\$tr       |
| tags\$caption    | tags\$form       | tags\$link   | tags\$amp      | tags\$track    |
| tags\$cite       | tags\$h1         | tags\$mark   | tags\$script   | tags\$u        |
| tags\$code       | tags\$h2         | tags\$map    | tags\$section  | tags\$ul       |
| tags\$col        | tags\$h3         | tags\$menu   | tags\$select   | tags\$var      |
| tags\$colgroup   | tags\$h4         | tags\$meta   | tags\$small    | tags\$video    |
| tags\$command    | tags\$h5         | tags\$meter  | tags\$source   | tags\$wbr      |

The most common tags have wrapper functions. You do not need to prefix their names with **tags\$**

```
ui <- fluidPage(
 h1("Header 1"),
 hr(),
 br(),
 p(strong("bold")),
 p(em("italic")),
 p(code("code")),
 a(href="", "link"),
 HTML("<p>Raw html</p>"))
```

**Header 1**

bold  
italic  
code  
link  
Raw html

To include a CSS file, use **includeCSS()**, or  
1. Place the file in the **www** subdirectory  
2. Link to it with

```
tags$head(tags$link(rel = "stylesheet",
 type = "text/css", href = "<file name>"))
```

To include JavaScript, use **includeScript()** or  
1. Place the file in the **www** subdirectory  
2. Link to it with

```
tags$head(tags$script(src = "<file name>"))
```

**IMAGES** To include an image  
1. Place the file in the **www** subdirectory  
2. Link to it with **img(src=<file name>")**

## Layouts



Combine multiple elements into a "single element" that has its own properties with a panel function, e.g.

```
wellPanel(dateInput("a", ""),
 submitButton())
)
```

2015-06-10
Apply Changes

absolutePanel()  
conditionalPanel()  
fixedPanel()  
headerPanel()  
inputPanel()  
mainPanel()
navlistPanel()  
sidebarPanel()  
tabPanel()  
tabsetPanel()  
titlePanel()  
wellPanel()

Organize panels and elements into a layout with a layout function. Add elements as arguments of the layout functions.

**fluidRow()**

column
col
column

```
ui <- fluidPage(
 fluidRow(column(width = 4),
 column(width = 2, offset = 3)),
 fluidRow(column(width = 12))
)
```

**flowLayout()**

object 1
object 2
object 3
object 1

```
ui <- fluidPage(
 flowLayout(# object 1,
 # object 2,
 # object 3
)
)
```

**sidebarLayout()**

side panel
main panel

```
ui <- fluidPage(
 sidebarLayout(
 sidebarPanel(),
 mainPanel()
)
)
```

**splitLayout()**

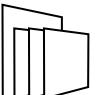
object 1
object 2

```
ui <- fluidPage(
 splitLayout(# object 1,
 # object 2
)
)
```

**verticalLayout()**

object 1
object 2
object 3

```
ui <- fluidPage(
 verticalLayout(# object 1,
 # object 2,
 # object 3
)
)
```



Layer tabPanels on top of each other, and navigate between them, with:

```
ui <- fluidPage(tabsetPanel(
 tabPanel("tab 1", "contents"),
 tabPanel("tab 2", "contents"),
 tabPanel("tab 3", "contents")
))
```

```
ui <- fluidPage(navlistPanel(
 tabPanel("tab 1", "contents"),
 tabPanel("tab 2", "contents"),
 tabPanel("tab 3", "contents")
))
```

```
ui <- navbarPage(title = "Page",
 tabPanel("tab 1", "contents"),
 tabPanel("tab 2", "contents"),
 tabPanel("tab 3", "contents")
)
```

# Data & Variable Transformation with sjmisc Cheat Sheet



sjmisc complements dplyr, and helps with data transformation tasks and recoding *variables*.

sjmisc works together seamlessly with dplyr and pipes. All functions are designed to support labelled data.



## Design Philosophy

The design of sjmisc functions follows the tidyverse-approach: first argument is always the data (either a *data frame* or *vector*), followed by variable names to be processed by the functions.

The returned object for each function *equals the type of the data-argument*.

### Vector input

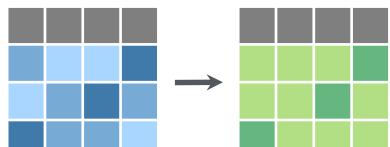
- If the data-argument is a *vector*, functions return a *vector*.



```
rec(mtcars$carb, rec = "1,2=1; 3,4=2; else=3")
```

### Data frame input

- If the data-argument is a *data frame*, functions return a *data frame*.



```
rec(mtcars, carb, rec = "1,2=1; 3,4=2; else=3")
```

## The ...-ellipses Argument

Apply functions to a single variable, selected variables or to a complete data frame.

Variable selection is powered by dplyr's `select()`: Separate variables with comma, or use dplyr's select-helpers to select variables, e.g. `?rec`:

```
rec(mtcars, one_of(c("gear", "carb")),
 rec = "min:3=1; 4:max=2")
```

```
rec(mtcars, gear, carb, rec = "min:3=1; 4:max=2")
```

## Descriptives and Summaries

Most of the sjmisc functions (including recode-functions) also work on grouped data frames:

```
library(dplyr)
efc %>%
 group_by(e16sex, c172code) %>%
 frq(e42dep)
```

### Frequency Tables

```
frq(x, ..., sort.frq = c("none", "asc", "desc"),
 weight.by = NULL, auto.grp = NULL, ...)
```

Print frequency tables of (labelled) vectors. Uses variable labels as table header.

```
data(efc); frq(efc, e42dep, c161sex)
```

Use this data set in examples!

```
flat_table(data, ..., margin = c("counts",
 "cell", "row", "col"), digits = 2,
 show.values = FALSE)
```

Print contingency tables of (labelled) vectors. Uses value labels.

```
flat_table(efc, e42dep, c172code, e16sex)
```

```
count_na(x, ...)
```

Print frequency table of tagged NA values.

```
library(haven); x <- labelled(c(1:3,
 tagged_na("a", "a", "z")), labels =
 c("Refused" = tagged_na("a"), "N/A" =
 tagged_na("z")))
count_na(x)
```

### Descriptive Summary

```
descr(x, ..., max.length = NULL)
```

Descriptive summary of data frames, including variable labels in output.

```
descr(efc, contains("cop"), max.length = 20)
```

## Finding Variables in a Data Frame

Use `find_var()` to search for variables by names, value or variable labels. Returns vector/data frame.

```
variables with "cop" in names and variable labels
find_var(efc, pattern = "cop", out = "df")
```

```
variables with "level" in names and value labels
find_var(efc, "level", search = "name_value")
```

## Recode and Transform Variables

Recode functions add a *suffix* to new variables, so original variables are preserved.

By default, original input data frame and new created variables are returned. Use `append = FALSE` to return the recoded variables only.

```
rec(x, ..., rec, as.num = TRUE, var.label =
 NULL, val.labels = NULL, append = TRUE,
 suffix = "_r")
```

Recode values, return result as numeric, character or categorical (factor).

```
rec(mtcars, carb, rec = "1,2=1; 3,4=2; else=3")
```

```
dicho(x, ..., dich.by = "median", as.num =
 FALSE, var.label = NULL, val.labels = NULL,
 append = TRUE, suffix = "_d")
```

Dichotomise variable by median, mean or specific value.

```
dicho(mtcars, disp)
```

```
split_var(x, ..., n, as.num = FALSE,
 val.labels = NULL, var.label = NULL,
 inclusive = FALSE, append = TRUE,
 suffix = "_g")
```

Split variable into equal sized groups. Unlike `dplyr::ntile()`, does not split original categories into different values (see examples in `?split_var`).

```
split_var(mtcars, mpg, disp, n = 3)
```

```
group_var(x, ..., size = 5, as.num = TRUE,
 right.interval = FALSE, n = 30, append =
 TRUE, suffix = "_gr")
```

Split variable into groups with equal value range, or into a max. # of groups (value range per group is adjusted to match # of groups).

```
group_var(mtcars, mpg, disp, size = 5)
```

```
group_var(mtcars, mpg, size = "auto", n = 4)
```

```
std(x, ..., robust = "sd", include.fac = FALSE,
 append = TRUE, suffix = "_z")
```

Z-standardise variables. Also `center()`.

```
std(efc, e17age, c160age)
```

```
recode_to(x, ..., lowest = 0, highest = -1,
 append = TRUE, suffix = "_r0")
```

Shift ("renumber") categories or values.

```
recode_to(mtcars$gear)
```

## Summarise Variables and Cases

The summary functions mostly mimic base R equivalents, but are designed to work together with pipes and dplyr.



```
row_sums(x, ..., na.rm = TRUE, var =
 "rowsums", append = FALSE)
```

Row sums of data frames.

```
row_sums(efc, c82cop1:c90cop9)
```

```
row_means(x, ..., n, var = "rowmeans",
 append = FALSE)
```

Row means, for at least `n` valid (non-NA) values.

```
row_means(efc, c82cop1:c90cop9, n = 7)
```

```
row_count(x, ..., count, var = "rowcount",
 append = FALSE)
```

Row-wise count # of values in data frames.

```
row_count(efc, c82cop1:c90cop9, count = 2)
```

## Other Useful Functions

- `add_columns()` and `replace_columns()` to combine data frames, but either replace or preserve existing columns.

- `set_na()` and `replace_na()` to convert regular into missing values, or vice versa. `replace_na()` also replaces specific `tagged NA` values only.

- `remove_var()` and `var_rename()` to remove variables from data frames, or rename variables.

- `group_str()` to group similar string values. Useful for variables with similar, but not identically spelled string values that should be "merged".

- `merge_df()` to full join data frames and preserve value and variable labels.

- `to_long()` to gather multiple columns in data frames from wide into long format.

## Use with %>% and dplyr

```
use sjmisc-functions in pipes
mtcars %>% select(gear, carb) %>%
 rec(rec = "min:3=1; 4:max=2")
```

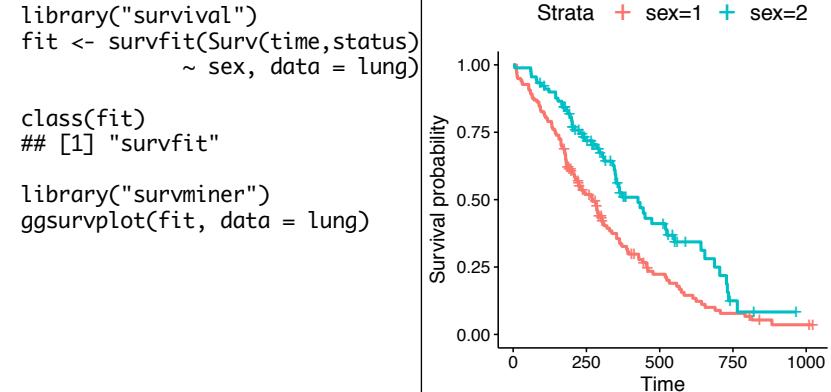
```
use sjmisc-function inside mutate
mtcars %>% select(gear, carb) %>% mutate(
 carb2 = rec(carb, rec = "1,2=0;3:8=1"),
 gear2 = rec(gear, rec = "3=1;4:max=2"))
```

# Creating Survival Plots

## Informative and Elegant with survminer

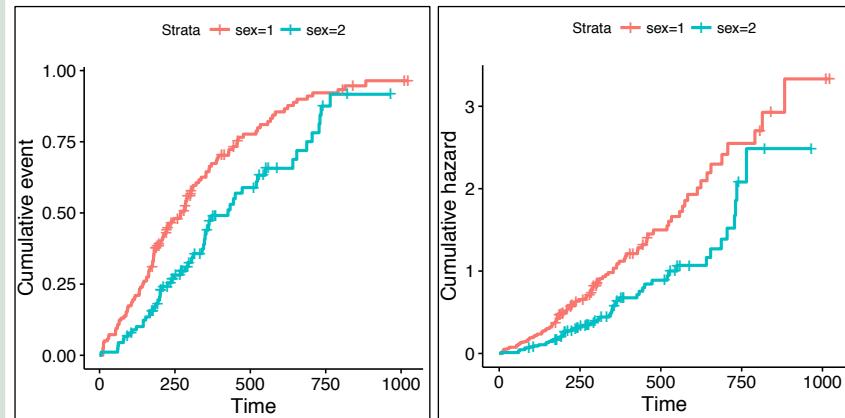
### Survival Curves

The **ggsurvplot()** function creates **ggplot2** plots from **survfit** objects.



Use the **fun** argument to set the transformation of the survival curve. E.g. "**event**" for cumulative events, "**cumhaz**" for the cumulative hazard function or "**pct**" for survival probability in percentage.

```
ggsurvplot(fit, data = lung, fun = "event")
ggsurvplot(fit, data = lung, fun = "cumhaz")
```



With lots of graphical parameters you have full control over look and feel of the survival plots; position and content of the legend; additional annotations like p-value, title, subtitle.

```
ggsurvplot(fit, data = lung,
conf.int = TRUE,
pval = TRUE,
fun = "pct",
risk.table = TRUE,
size = 1,
linetype = "strata",
palette = c("#E7B800",
"#2E9FDF"),
legend = "bottom",
legend.title = "Sex",
legend.labs = c("Male",
"Female"))
```

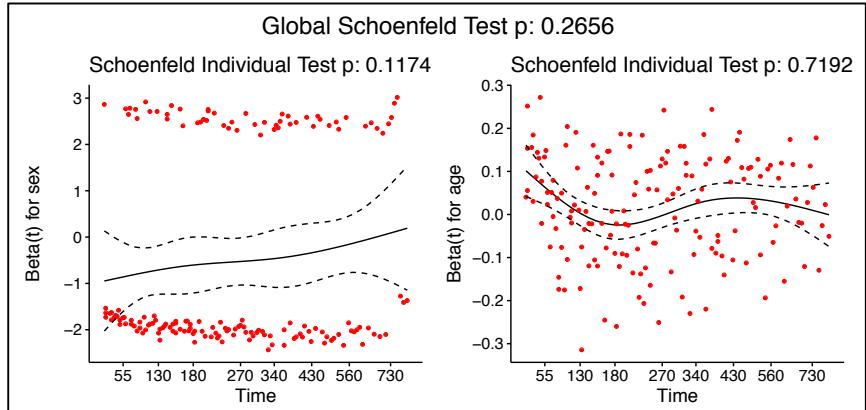
### Diagnostics of Cox Model

The function **cox.zph()** from **survival** package may be used to test the proportional hazards assumption for a Cox regression model fit. The graphical verification of this assumption may be performed with the function **ggcooxzph()** from the **survminer** package. For each covariate it produces plots with scaled Schoenfeld residuals against the time.

```
library("survival")
fit <- coxph(Surv(time, status) ~ sex + age, data = lung)
ftest <- cox.zph(fit)
ftest
```

|           | rho     | chisq | p     |
|-----------|---------|-------|-------|
| ## sex    | 0.1236  | 2.452 | 0.117 |
| ## age    | -0.0275 | 0.129 | 0.719 |
| ## GLOBAL | NA      | 2.651 | 0.266 |

```
library("survminer")
ggcooxzph(ftest)
```



The function **ggcoxdiagnostics()** plots different types of residuals as a function of time, linear predictor or observation id. The type of residual is selected with **type** argument. Possible values are "martingale", "deviance", "score", "schoenfeld", "dfbeta", "dfbetas", and "scaledsch".

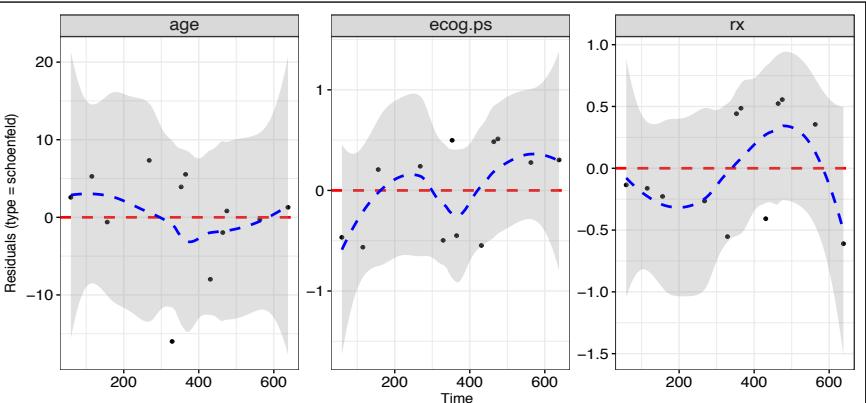
The **ox.scale** argument defines what shall be plotted on the OX axis. Possible values are "linear.predictions", "observation.id", "time".

Logical arguments **hline** and **sline** may be used to add horizontal line or smooth line to the plot.

```
library("survival")
library("survminer")
fit <- coxph(Surv(time, status) ~ sex + age, data = lung)
```

```
ggcoxdiagnostics(fit,
type = "deviance",
ox.scale = "linear.predictions")
```

```
ggcoxdiagnostics(fit,
type = "schoenfeld",
ox.scale = "time")
```



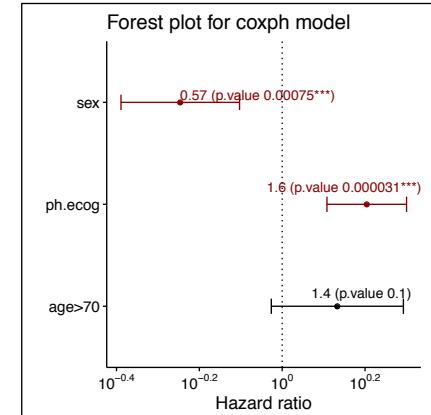
### Summary of Cox Model

The function **ggforest()** from the **survminer** package creates a forest plot for a Cox regression model fit. Hazard ratio estimates along with confidence intervals and p-values are plotted for each variable.

```
library("survival")
library("survminer")
lung$age <- ifelse(lung$age > 70, ">70", "<= 70")
fit <- coxph(Surv(time, status) ~ sex + ph.ecog + age, data = lung)
fit
```

```
Call:
coxph(formula = Surv(time, status) ~ sex+ph.ecog+age, data=lung)
##
coef exp(coef) se(coef) z p
sex -0.567 0.567 0.168 -3.37 0.00075
ph.ecog 0.470 1.600 0.113 4.16 3.1e-05
age>70 0.307 1.359 0.187 1.64 0.10175
##
Likelihood ratio test=31.6 on
n= 227, number of events= 164
```

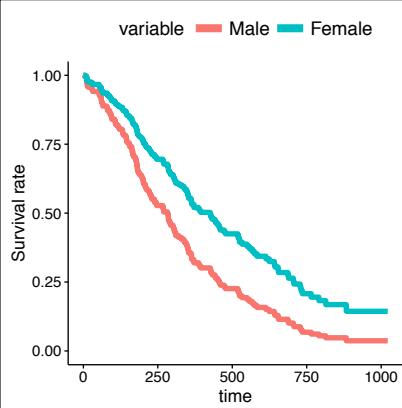
```
ggforest(fit)
```



The function **ggcoxaadjustedcurves()** from the **survminer** package plots Adjusted Survival Curves for Cox Proportional Hazards Model. Adjusted Survival Curves show how a selected factor influences survival estimated from a Cox model.

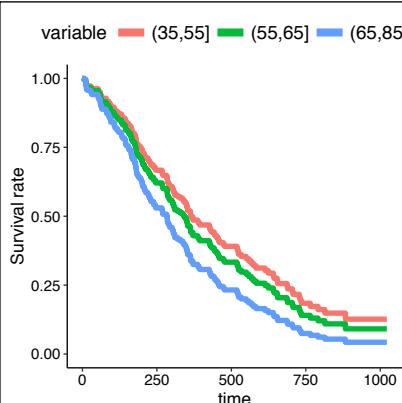
Note that these curves differ from Kaplan Meier estimates since they present expected survival based on given Cox model.

```
library("survival")
library("survminer")
lung$sex <- ifelse(lung$sex == 1,
"Male", "Female")
fit <- coxph(Surv(time, status) ~ sex + ph.ecog + age,
data = lung)
ggcoxaadjustedcurves(fit, data=lung,
variable=lung$sex)
```



Note that it is not necessary to include the grouping factor in the Cox model. Survival curves are estimated from Cox model for each group defined by the factor independently.

```
lung$age3 <- cut(lung$age,
c(35, 55, 65, 85))
ggcoxaadjustedcurves(fit, data=lung,
variable=lung$age3)
```



# caret Package

## Cheat Sheet

### Specifying the Model

Possible syntaxes for specifying the variables in the model:

```
train(y ~ x1 + x2, data = dat, ...)
train(x = predictor_df, y = outcome_vector, ...)
train(recipe_object, data = dat, ...)
```

- `rfe`, `sbf`, `gafs`, and `safs` only have the `x/y` interface.
- The `train` formula method will **always** create dummy variables.
- The `x/y` interface to `train` will not create dummy variables (but the underlying model function might).

**Remember** to:

- Have column names in your data.
- Use factors for a classification outcome (not 0/1 or integers).
- Have valid R names for class levels (not "0"/"1")
- Set the random number seed prior to calling `train` repeatedly to get the same resamples across calls.
- Use the `train` option `na.action = na.pass` if you will be imputing missing data. Also, use this option when predicting new data containing missing values.

To pass options to the underlying model function, you can pass them to `train` via the ellipses:

```
train(y ~ ., data = dat, method = "rf",
 # options to `randomForest`:
 importance = TRUE)
```

### Parallel Processing

The `foreach` package is used to run models in parallel. The `train` code does not change but a "`do`" package must be called first.

```
on Mac OS or Linux # on Windows
library(doMC) library(doParallel)
registerDoMC(cores=4) cl <- makeCluster(2)
 registerDoParallel(cl)
```

The function `parallel::detectCores` can help too.

### Preprocessing

Transformations, filters, and other operations can be applied to the *predictors* with the `preProc` option.

```
train(..., preProc = c("method1", "method2"), ...)
```

Methods include:

- `"center"`, `"scale"`, and `"range"` to normalize predictors.
- `"BoxCox"`, `"YeoJohnson"`, or `"expoTrans"` to transform predictors.
- `"knnImpute"`, `"bagImpute"`, or `"medianImpute"` to impute.
- `"corr"`, `"nzv"`, `"zv"`, and `"conditionalX"` to filter.
- `"pca"`, `"ica"`, or `"spatialSign"` to transform groups.

`train` determines the order of operations; the order that the methods are declared does not matter.

The `recipes` package has a more extensive list of preprocessing operations.

### Adding Options

Many `train` options can be specified using the `trainControl` function:

```
train(y ~ ., data = dat, method = "cubist",
 trControl = trainControl(<options>))
```

### Resampling Options

`trainControl` is used to choose a resampling method:

```
trainControl(method = <method>, <options>)
```

Methods and options are:

- `"cv"` for K-fold cross-validation (`number` sets the # folds).
- `"repeatedcv"` for repeated cross-validation (`repeats` for # repeats).
- `"boot"` for bootstrap (`number` sets the iterations).
- `"LGOCV"` for leave-group-out (`number` and `p` are options).
- `"L0O"` for leave-one-out cross-validation.
- `"oob"` for out-of-bag resampling (only for some models).
- `"timeslice"` for time-series data (options are `initialWindow`, `horizon`, `fixedWindow`, and `skip`).

### Performance Metrics

To choose how to summarize a model, the `trainControl` function is used again.

```
trainControl(summaryFunction = <R function>,
 classProbs = <logical>)
```

Custom R functions can be used but `caret` includes several: `defaultSummary` (for accuracy, RMSE, etc), `twoClassSummary` (for ROC curves), and `prSummary` (for information retrieval). For the last two functions, the option `classProbs` must be set to `TRUE`.

### Grid Search

To let `train` determine the values of the tuning parameter(s), the `tuneLength` option controls how many values `per tuning` parameter to evaluate.

Alternatively, specific values of the tuning parameters can be declared using the `tuneGrid` argument:

```
grid <- expand.grid(alpha = c(0.1, 0.5, 0.9),
 lambda = c(0.001, 0.01))
```

```
train(x = x, y = y, method = "glmnet",
 preProc = c("center", "scale"),
 tuneGrid = grid)
```

### Random Search

For tuning, `train` can also generate random tuning parameter combinations over a wide range. `tuneLength` controls the total number of combinations to evaluate. To use random search:

```
trainControl(search = "random")
```

### Subsampling

With a large class imbalance, `train` can subsample the data to balance the classes them prior to model fitting.

```
trainControl(sampling = "down")
```

Other values are `"up"`, `"smote"`, or `"rose"`. The latter two may require additional package installs.

# The eurostat package

## R tools to access open data from Eurostat database

### Search and download

Data in the Eurostat database is stored in tables. Each table has an identifier, a short table\_code, and a description (e.g. tsdtr420 - People killed in road accidents).

Key eurostat functions allow to find the table\_code, download the eurostat table and polish labels in the table.

### Find the table code

The `search_eurostat(pattern, ...)` function scans the directory of Eurostat tables and returns codes and descriptions of tables that match pattern.

```
library("eurostat")
query <- search_eurostat("road", type = "table")
query[1:3,1:2]
title code
1 Goods transport by road ttr00005
2 People killed in road accidents tsdtr420
3 Enterprises with broadband access tin00090
```

### Download the table

The `get_eurostat(id, time_format = "date", filters = "none", type = "code", cache = TRUE, ...)` function downloads the requested table from the *Eurostat bulk download facility* or from *The Eurostat Web Services JSON API* (if `filters` are defined). Downloaded data is cached (if `cache=TRUE`). Additional arguments define how to read the time column (`time_format`) and if table dimensions shall be kept as codes or converted to labels (`type`).

```
dat <- get_eurostat(id="tsdtr420", time_format="num")
head(dat)
unit sex geo time values
1 NR T AT 1999 1079
2 NR T BE 1999 1397
3 NR T CZ 1999 1455
4 NR T DK 1999 514
5 NR T EL 1999 2116
6 NR T ES 1999 5738
```

### Add labels

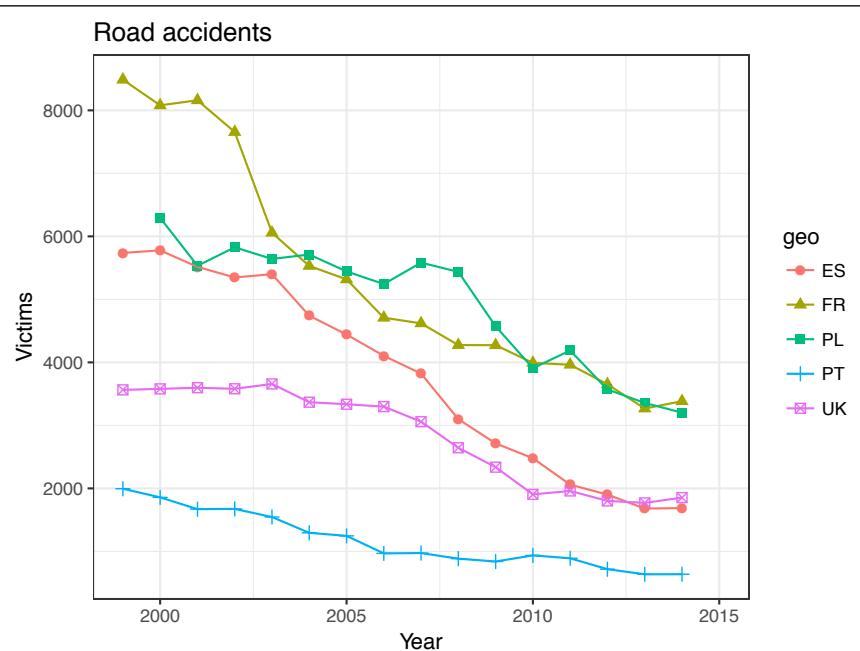
The `label_eurostat(x, lang = "en", ...)` gets definitions for Eurostat codes and replace them with labels in given language ("en", "fr" or "de").

```
dat <- label_eurostat(dat)
head(dat)
unit sex geo time values
1 Number Total Austria 1999 1079
2 Number Total Belgium 1999 1397
3 Number Total Czech Republic 1999 1455
4 Number Total Denmark 1999 514
5 Number Total Greece 1999 2116
6 Number Total Spain 1999 5738
```

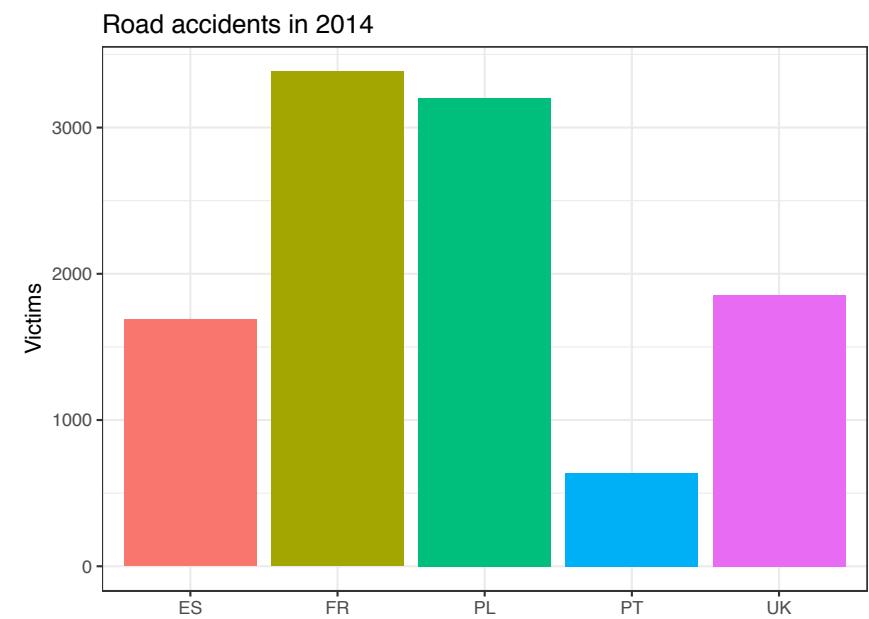
### eurostat and plots

The `get_eurostat()` function returns tibbles in the long format. Packages `dplyr` and `tidyverse` are well suited to transform these objects. The `ggplot2` package is well suited to plot these objects.

```
t1 <- get_eurostat("tsdtr420", filters =
 list(geo = c("UK", "FR", "PL", "ES", "PT")))
library("ggplot2")
ggplot(t1, aes(x = time, y = values, color = geo,
 group = geo, shape = geo)) +
 geom_point(size = 2) +
 geom_line() + theme_bw() +
 labs(title = "Road accidents", x = "Year", y = "Victims")
```



```
library("dplyr")
t2 <- t1 %>% filter(time == "2014-01-01")
ggplot(t2, aes(geo, values, fill = geo)) +
 geom_bar(stat = "identity") + theme_bw() +
 theme(legend.position = "none") +
 labs(title = "Road accidents in 2014", x = "", y = "Victims")
```



### eurostat and maps

#### Fetch and process data

There are three function to work with geospatial data from GISCO. The `get_eurostat_geospatial()` returns preprocessed spatial data as sp-objects or as data frames. The `merge_eurostat_geospatial()` both downloads and merges the geospatial data with a preloaded tabular data. The `cut_to_classes()` is a wrapper for `cut()` - function and is used for categorizing data for maps with tidy labels.

```
library("eurostat")
library("dplyr")
fertility <- get_eurostat("demo_r_frate3") %>%
 filter(time == "2014-01-01") %>%
 mutate(cat = cut_to_classes(values, n = 7, decimals = 1))

mapdata <- merge_eurostat_geodata(fertility,
 resolution = "20")
```

```
head(select(mapdata, geo, values, cat, long, lat, order, id))
geo values cat long lat order id
1 AT124 1.39 ~< 1.5 15.54245 48.90770 214 10
2 AT124 1.39 ~< 1.5 15.75363 48.85218 215 10
3 AT124 1.39 ~< 1.5 15.88763 48.78511 216 10
4 AT124 1.39 ~< 1.5 15.81535 48.69270 217 10
5 AT124 1.39 ~< 1.5 15.94094 48.67173 218 10
6 AT124 1.39 ~< 1.5 15.90833 48.59815 219 10
```

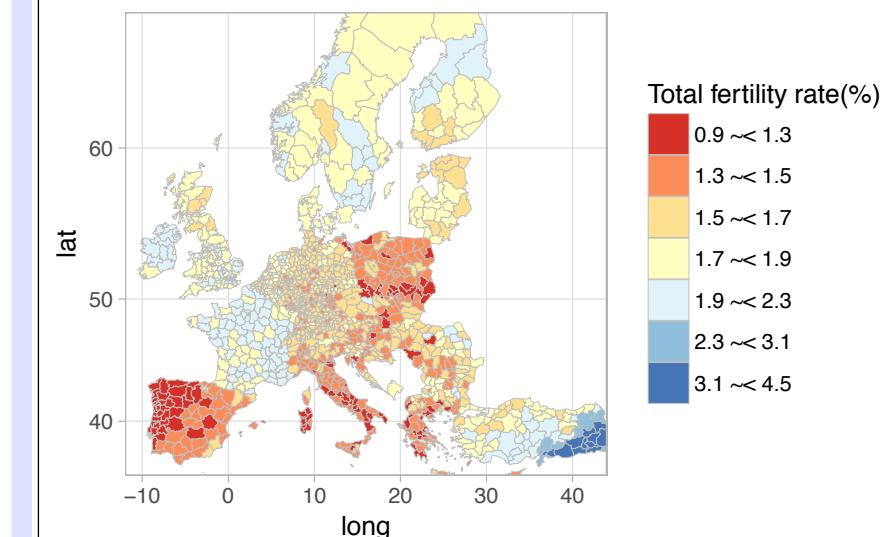
#### Draw a cartogram

The object returned by `merge_eurostat_geospatial()` are ready to be plotted with ggplot2 package. The `coord_map()` function is useful to set the projection while `labs()` adds annotations o the plot.

```
library("ggplot2")
ggplot(mapdata, aes(x = long, y = lat, group = group)) +
 geom_polygon(aes(fill = cat), color = "grey", size = .1) +
 scale_fill_brewer(palette = "RdYlBu") +
 labs(title = "Fertility rate, by NUTS-3 regions, 2014",
 subtitle = "Avg. number of live births per woman",
 fill = "Total fertility rate(%)") + theme_light() +
 coord_map(xlim = c(-12, 44), ylim = c(35, 67))
```

#### Fertility rate, by NUTS-3 regions, 2014

Avg. number of live births per woman



# Leaflet Cheat Sheet



an open-source JavaScript library for mobile-friendly interactive maps

## Quick Start

### Installation

Use `install.packages("leaflet")` to install the package or directly from Github `devtools::install_github("rstudio/leaflet")`.

### First Map

```
m <- leaflet() %>%
 addTiles() %>%
 addMarkers(lng = 174.768, lat = -36.852, popup = "The birthplace of R")
add a single point layer
```



## Map Widget

### Initialization

|                                                             |                                      |
|-------------------------------------------------------------|--------------------------------------|
| <code>m &lt;- leaflet(options = leafletOptions(...))</code> | Initial geographic center of the map |
| <code>center</code>                                         | Initial map zoom level               |
| <code>zoom</code>                                           | Minimum zoom level of the map        |
| <code>minZoom</code>                                        | Maximum zoom level of the map        |
| <code>maxZoom</code>                                        |                                      |

### Map Methods

```
m %>% setView(lng, lat, zoom, options = list())
Set the view of the map (center and zoom level)
m %>% fitBounds(lng1, lat1, lng2, lat2)
Fit the view into the rectangle [lng1, lat1] - [lng2, lat2]
m %>% clearBounds()
Clear the bound, automatically determine from the map elements
```

### Data Object

Both `leaflet()` and the `map` layers have an optional data parameter that is designed to receive spatial data with the following formats:

#### Base R

The arguments of all layers take normal R objects:

```
df <- data.frame(lat = ..., lng = ...)
```

```
leaflet(df) %>% addTiles() %>% addCircles()
```

library(sp) Useful functions:

SpatialPoints, SpatialLines, SpatialPolygons, ...

library(maps) Build a map of states with colors:

```
mapStates <- map("state", fill = TRUE, plot = FALSE)
```

```
leaflet(mapStates) %>% addTiles() %>%
```

```
addPolygons(fillColor = topo.colors(10, alpha = NULL), stroke = FALSE)
```

## Markers

Use markers to call out points, express locations with latitude/longitude coordinates, appear as icons or as circles.

Data come from vectors or assigned data frame, or `sp` package objects.

### Icon Markers

Regular Icons: default and simple

```
addMarkers(lng, lat, popup, label) add basic icon markers
```

```
makeIcon(Icons(iconUrl, iconWidth, iconHeight, iconAnchorX, iconAnchorY,
 shadowUrl, shadowWidth, shadowHeight, ...)) customize marker icons
```

```
iconList() create a list of icons
```

Awesome Icons: customizable with colors and icons

```
addAwesomeMarkers, makeAwesomeIcon, awesomeIcons, awesomeIconList
```

Marker Clusters: option of `addMarkers()`

```
clusterOptions = markerClusterOptions()
```

```
freezeAtZoom Freeze the cluster at assigned zoom level
```

### Circle Markers

```
addCircleMarkers(color, radius, stroke, opacity, ...)
```

Customize their color, radius, stroke, opacity

## Popups and Labels

`addPopups(lng, lat, ...content..., options)` Add standalone popups

```
options = popupOptions(closeButton=FALSE)
```

`addMarkers(..., popup, ...)` Show popups with markers or shapes

`addMarkers(..., label, labelOptions...)` Show labels with markers or shapes

```
labelOptions = labelOptions(noHide, textOnly, textSize, direction, style)
```

`addLabelOnlyMarkers()` Add labels without markers

## Lines and Shapes

### Polygons and Polylines

`addPolygons(color, weight=1, smoothFactor=0.5, opacity=1.0, fillOpacity=0.5,`  
`fillColor= ~colorQuantile("YlOrRd", ALAND)(ALAND), highlightOptions, ... )`

`highlightOptions(color, weight=2, bringToFront=TRUE)` highlight shapes

Use `rmapshaper::ms_simplify` to simplify complex shapes

`Circles addCircles(lng, lat, weight=1, radius, ...)`

`Rectangles addRectangles(lng1, lat1, lng2, lat2, fillColor="transparent", ... )`

## Basemaps

`addTiles()`

Default Tiles

Use `addTiles()` to add a custom map tile URL template, use `addWMSTiles()` to add WMS (Web Map Service) tiles

`providers$Stamen.Toner, CartoDB.Positron, Esri.NatGeoWorldMap`

Third-Party Tiles

`addProviderTiles()`

## GeoJSON and TopoJSON

There are two options to use the GeoJSON/TopoJSON data:

- \* To read into `sp` objects with the `geojsonio` or `rgdal` package:  
`geojsonio::geojson_read(..., what="sp") rgdal::readOGR(..., "OGRGeoJSON")`

- \* Or to use the `addGeoJSON()` and `addTopoJSON()` functions:  
`addTopoJSON/addGeoJSON(... weight, color, fill, opacity, fillOpacity...)`

Styles can also be tuned separately with a `style: {}` object.  
Other packages including `RJSONIO` and `jsonlite` can help fast parse or generate the data needed.

## Shiny Integration

To integrate a Leaflet map into an app:

- \* In the UI, call `leafletOutput("name")`

- \* On the server side, assign a `renderLeaflet(...)` call to the output

- \* Inside the `renderLeaflet` expression, return a Leaflet map object

### Modification

To modify an existing map or add incremental changes to the map, you can use `leafletProxy()`. This should be performed in an observer on the server side.

Other useful functions to edit your map:

`fitBounds(o, 0, 11, 11)` similar to `setView`

fit the view to within these bounds

`addCircles(1:10, 1:10, layerId = LETTERS[1:10])`

create circles with layerIds of "A", "B", "C"...

`removeShape(c("B", "F"))` remove some of the circles

`clearShapes()` clear all circles (and other shapes)

### Inputs/Events

#### Object Events

Object event names generally use this pattern:

`inputs$MAPID_OBJCATEGORY_EVENTNAME`.

Triger an event changes the value of the Shiny input at this variable.

Valid values for `OBJCATEGORY` are `marker`, `shape`, `geojson` and `topojson`.

Valid values for `EVENTNAME` are `click`, `mouseover` and `mouseout`.

All of these events are set to either `NULL` if the event has never happened, or a `list()` that includes:

- \* `lat` The latitude of the object, if available; otherwise, the mouse cursor

- \* `lng` The longitude of the object, if available; otherwise, the mouse cursor

- \* `id` The layerId, if any

GeoJSON events also include additional properties:

- \* `featureId` The feature ID, if any

- \* `properties` The feature properties

#### Map Events

`inputs$MAPID_click` when the map background or basemap is clicked

`value -- a list with lat and lng`

`inputs$MAPID_bounds` provide the lat/long bounds of the visible map area

`value -- a list with north, east, south and west`

`inputs$MAPID_zoom` an integer indicates the zoom level

# Basic Regular Expressions in R

## Cheat Sheet

### Character Classes

|                                            |                                                               |
|--------------------------------------------|---------------------------------------------------------------|
| <code>[:digit:]</code> or <code>\d</code>  | Digits; [0-9]                                                 |
| <code>\D</code>                            | Non-digits; [^0-9]                                            |
| <code>[:lower:]</code>                     | Lower-case letters; [a-z]                                     |
| <code>[:upper:]</code>                     | Upper-case letters; [A-Z]                                     |
| <code>[:alpha:]</code>                     | Alphabetic characters; [A-z]                                  |
| <code>[:alnum:]</code>                     | Alphanumeric characters [A-z0-9]                              |
| <code>\w</code>                            | Word characters; [A-z0-9_]                                    |
| <code>\W</code>                            | Non-word characters                                           |
| <code>[:xdigit:]</code> or <code>\x</code> | Hexadec. digits; [0-9A-Fa-f]                                  |
| <code>[:blank:]</code>                     | Space and tab                                                 |
| <code>[:space:]</code> or <code>\s</code>  | Space, tab, vertical tab, newline, form feed, carriage return |
| <code>\S</code>                            | Not space; [^[:space:]]                                       |
| <code>[:punct:]</code>                     | Punctuation characters; !#\$%&(*+,-./;:<=>?[@]^_`{ }~         |
| <code>[:graph:]</code>                     | Graphical char.; [:alnum:][:punct:]\s                         |
| <code>[:print:]</code>                     | Printable characters; [:alnum:][:punct:]\s                    |
| <code>[:cntrl:]</code> or <code>\c</code>  | Control characters; \n, \r etc.                               |

### Special Metacharacters

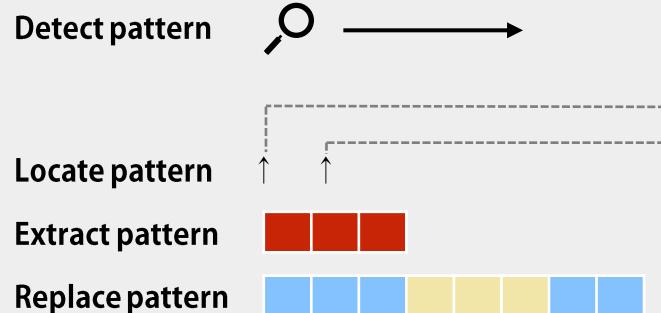
|                 |                 |
|-----------------|-----------------|
| <code>\n</code> | New line        |
| <code>\r</code> | Carriage return |
| <code>\t</code> | Tab             |
| <code>\v</code> | Vertical tab    |
| <code>\f</code> | Form feed       |

### Lookarounds and Conditionals\*

|                             |                                                                                  |
|-----------------------------|----------------------------------------------------------------------------------|
| <code>(?=)</code>           | Lookahead (requires PERL = TRUE), e.g. (?=yx): position followed by 'xy'         |
| <code>(?!)</code>           | Negative lookahead (PERL = TRUE); position NOT followed by pattern               |
| <code>(?&lt;=)</code>       | Lookbehind (PERL = TRUE), e.g. (?<=yx): position following 'xy'                  |
| <code>(?&lt;!)</code>       | Negative lookbehind (PERL = TRUE); position NOT following pattern                |
| <code>?(if)then</code>      | If-then-condition (PERL = TRUE); use lookaheads, optional char. etc in if-clause |
| <code>?(if)then else</code> | If-then-else-condition (PERL = TRUE)                                             |

\*see, e.g. <http://www.regular-expressions.info/lookaround.html>  
<http://www.regular-expressions.info/conditional.html>

## Functions for Pattern Matching



```
> string <- c("Hipopopotamus", "Rhymenoceros", "time for bottomless lyrics")
> pattern <- "t.m"
```

### Detect Patterns

```
grep(pattern, string)
[1] 1 3
grep(pattern, string, value = TRUE)
[1] "Hipopopotamus"
[2] "time for bottomless lyrics"
grepl(pattern, string)
[1] TRUE FALSE TRUE
stringr::str_detect(string, pattern)
[1] TRUE FALSE TRUE
```

### Split a String using a Pattern

```
strsplit(string, pattern) or stringr::str_split(string, pattern)
```

### Locate Patterns

```
regexpr(pattern, string)
find starting position and length of first match
gregexpr(pattern, string)
find starting position and length of all matches
stringr::str_locate(string, pattern)
find starting and end position of first match
stringr::str_locate_all(string, pattern)
find starting and end position of all matches
```

### Extract Patterns

```
regmatches(string, regexpr(pattern, string))
extract first match [1] "tam" "tim"
regmatches(string, gregexpr(pattern, string))
extracts all matches, outputs a list [[1]] "tam" [[2]] character(0) [[3]] "tim" "tom"
stringr::str_extract(string, pattern)
extract first match [1] "tam" NA "tim"
stringr::str_extract_all(string, pattern)
extract all matches, outputs a list
stringr::str_extract_all(string, pattern, simplify = TRUE)
extract all matches, outputs a matrix
stringr::str_match(string, pattern)
extract first match + individual character groups
stringr::str_match_all(string, pattern)
extract all matches + individual character groups
```

### Replace Patterns

```
sub(pattern, replacement, string)
replace first match
gsub(pattern, replacement, string)
replace all matches
stringr::str_replace(string, pattern, replacement)
replace first match
stringr::str_replace_all(string, pattern, replacement)
replace all matches
```

### Character Classes and Groups

|        |                                                                    |
|--------|--------------------------------------------------------------------|
| .      | Any character except \n                                            |
|        | Or, e.g. (a b)                                                     |
| [...]  | List permitted characters, e.g. [abc]                              |
| [a-z]  | Specify character ranges                                           |
| [^...] | List excluded characters                                           |
| (...)  | Grouping, enables back referencing using \\N where N is an integer |

### Anchors

|    |                                       |
|----|---------------------------------------|
| ^  | Start of the string                   |
| \$ | End of the string                     |
| \b | Empty string at either edge of a word |
| \B | NOT the edge of a word                |
| \B | Beginning of a word                   |
| \B | End of a word                         |

### Quantifiers

|       |                                         |
|-------|-----------------------------------------|
| *     | Matches at least 0 times                |
| +     | Matches at least 1 time                 |
| ?     | Matches at most 1 time; optional string |
| {n}   | Matches exactly n times                 |
| {n,}  | Matches at least n times                |
| {,n}  | Matches at most n times                 |
| {n,m} | Matches between n and m times           |

### General Modes

By default R uses *POSIX extended regular expressions*. You can switch to *PCRE regular expressions* using PERL = TRUE for base or by wrapping patterns with perl() for stringr.

All functions can be used with literal searches using fixed = TRUE for base or by wrapping patterns with fixed() for stringr.

All base functions can be made case insensitive by specifying ignore.cases = TRUE.

### Escaping Characters

Metacharacters (. \* + etc.) can be used as literal characters by escaping them. Characters can be escaped using \\ or by enclosing them in \Q...\\E.

### Case Conversions

Regular expressions can be made case insensitive using (?i). In backreferences, the strings can be converted to lower or upper case using \\L or \\U (e.g. \\L\\1). This requires PERL = TRUE.

### Greedy Matching

By default the asterisk \* is greedy, i.e. it always matches the longest possible string. It can be used in lazy mode by adding ?, i.e. \*?.

Greedy mode can be turned off using (?U). This switches the syntax, so that (?U)a\* is lazy and (?U)a\*? is greedy.

### Note

Regular expressions can conveniently be created using `rex::rex()`.

# Advanced R

## Cheat Sheet

Created by: Arianne Colton and Sean Chen

### Environment Basics

Environment – **Data structure** (with two components below) that powers lexical scoping

Create environment: `env1<-new.env()`

1. **Named list** (“Bag of names”) – each name points to an object stored elsewhere in memory.

If an object has no names pointing to it, it gets automatically deleted by the garbage collector.

- Access with: `ls('env1')`

2. **Parent environment** – used to implement lexical scoping. If a name is not found in an environment, then R will look in its parent (and so on).

- Access with: `parent.env('env1')`

### Four special environments

1. **Empty environment** – ultimate ancestor of all environments
  - Parent: none
  - Access with: `emptyenv()`

2. **Base environment** - environment of the base package
  - Parent: empty environment
  - Access with: `baseenv()`

3. **Global environment** – the interactive workspace that you normally work in
  - Parent: environment of last attached package
  - Access with: `globalenv()`

4. **Current environment** – environment that R is currently working in (may be any of the above and others)
  - Parent: empty environment
  - Access with: `environment()`

## Environments

### Search Path

**Search path** – mechanism to look up objects, particularly functions.

- Access with : `search()` – lists all parents of the global environment (see Figure 1)
- Access any environment on the search path:  
`as.environment('package:base')`

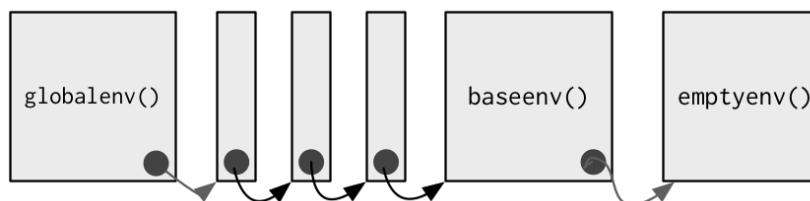


Figure 1 – The Search Path

- Mechanism : always start the search from global environment, then inside the latest attached package environment.
  - New package loading with `library()`/`require()` : new package is attached right after global environment. (See Figure 2)
  - Name conflict in two different package : functions with the same name, latest package function will get called.

`search()`:

```
'.GlobalEnv' ... 'Autoloads' 'package:base'
library(reshape2); search()
.GlobalEnv' 'package:reshape2' ... 'Autoloads' 'package:base'
```

**NOTE:** Autoloads : special environment used for saving memory by only loading package objects (like big datasets) when needed

Figure 2 – Package Attachment

### Binding Names to Values

**Assignment** – act of binding (or rebinding) a name to a value in an environment.

1. `<-` (Regular assignment arrow) – always creates a variable in the current environment
2. `<--` (Deep assignment arrow) - modifies an existing variable found by walking up the parent environments

**Warning:** If `<--` doesn't find an existing variable, it will create one in the global environment.

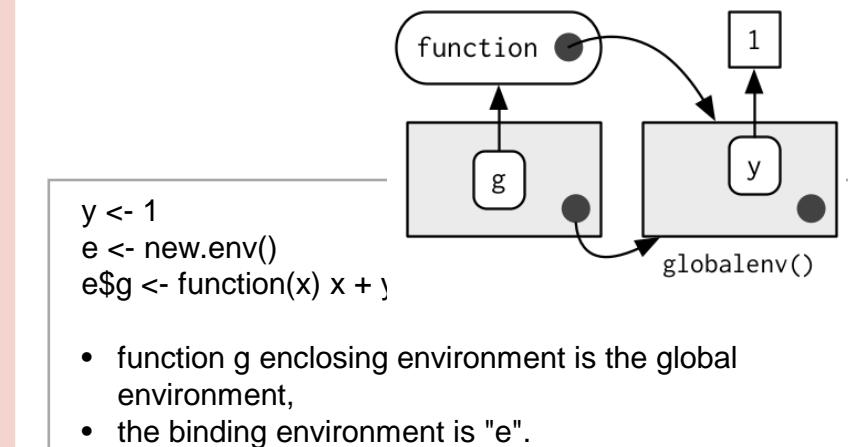
### Function Environments

1. **Enclosing environment** - an environment where the function is created. It determines how function finds value.
  - Enclosing environment never changes, even if the function is moved to a different environment.
  - Access with: `environment('func1')`

2. **Binding environment** - all environments that the function has a binding to. It determines how we find the function.

- Access with: `pryr::where('func1')`

**Example** (for enclosing and binding environment):



- function g enclosing environment is the global environment,
- the binding environment is "e".

3. **Execution environment** - new created environments to host a function call execution.

- Two parents :
  - I. Enclosing environment of the function
  - II. Calling environment of the function
- Execution environment is thrown away once the function has completed.

4. **Calling environment** - environments where the function was called.

- Access with: `parent.frame('func1')`
- Dynamic scoping :
  - About : look up variables in the calling environment rather than in the enclosing environment
  - Usage : most useful for developing functions that aid interactive data analysis

# Data Structures

|    | Homogeneous   | Heterogeneous |
|----|---------------|---------------|
| 1d | Atomic vector | List          |
| 2d | Matrix        | Data frame    |
| nd | Array         |               |

**Note:** R has no 0-dimensional or scalar types. Individual numbers or strings, are actually vectors of length one, NOT scalars.

Human readable description of any R data structure :

```
str(variable)
```

Every **Object** has a mode and a class

1. **Mode**: represents how an object is stored in memory

- 'type' of the object from R's point of view
- Access with: **typeof()**

2. **Class**: represents the object's abstract type

- 'type' of the object from R's object-oriented programming point of view
- Access with: **class()**

|                              | typeof()  | class()    |
|------------------------------|-----------|------------|
| strings or vector of strings | character | character  |
| numbers or vector of numbers | numeric   | numeric    |
| list                         | list      | list       |
| data.frame                   | list      | data.frame |

## Factors

1. Factors are built on top of integer vectors using two attributes :

```
class(x) -> 'factor'
```

```
levels(x) # defines the set of allowed values
```

2. Useful when you know the possible values a variable may take, even if you don't see all values in a given dataset.

### Warning on Factor Usage:

1. Factors look and often behave like character vectors, they are actually integers. Be careful when treating them like strings.
2. Most data loading functions automatically convert character vectors to factors. (Use argument `stringAsFactors = FALSE` to suppress this behavior)

# Object Oriented (OO) Field Guide

## Object Oriented Systems

R has three object oriented systems :

1. **S3** is a very casual system. It has no formal definition of classes. It implements generic function OO.
  - **Generic-function OO** - a special type of function called a generic function decides which method to call.

|           |                          |
|-----------|--------------------------|
| Example:  | drawRect(canvas, 'blue') |
| Language: | R                        |

- **Message-passing OO** - messages (methods) are sent to objects and the object determines which function to call.

|           |                         |
|-----------|-------------------------|
| Example:  | canvas.drawRect('blue') |
| Language: | Java, C++, and C#       |

2. **S4** works similarly to S3, but is more formal. Two major differences to S3 :

- **Formal class definitions** - describe the representation and inheritance for each class, and has special helper functions for defining generics and methods.
- **Multiple dispatch** - generic functions can pick methods based on the class of any number of arguments, not just one.

3. **Reference classes** are very different from S3 and S4:

- **Implements message-passing OO** - methods belong to classes, not functions.
- **Notation** - \$ is used to separate objects and methods, so method calls look like `canvas$drawRect('blue')`.

## S3

### 1. About S3 :

- R's first and simplest OO system
- Only OO system used in the base and stats package
- Methods belong to functions, not to objects or classes.

### 2. Notation :

- **generic.class()**

|             |                                      |
|-------------|--------------------------------------|
| mean.Date() | Date method for the generic - mean() |
|-------------|--------------------------------------|

### 3. Useful 'Generic' Operations

- Get all methods that belong to the 'mean' generic:
  - **Methods('mean')**
- List all generics that have a method for the 'Date' class :
  - **methods(class = 'Date')**

### 4. S3 objects

are usually built on top of lists, or atomic vectors with attributes.

- Factor and data frame are S3 class
- Useful operations:

|                                                |                                                              |
|------------------------------------------------|--------------------------------------------------------------|
| Check if object is an S3 object                | <code>is.object(x) &amp; !isS4(x) or pryr::oGetType()</code> |
| Check if object inherits from a specific class | <code>inherits(x, 'classname')</code>                        |
| Determine class of any object                  | <code>class(x)</code>                                        |

## Base Type (C Structure)

R base types - the internal C-level types that underlie the above OO systems.

- **Includes** : atomic vectors, list, functions, environments, etc.
- **Useful operation** : Determine if an object is a base type (Not S3, S4 or RC) `is.object(x)` returns FALSE

- **Internal representation** : C structure (or struct) that includes :

- Contents of the object
- Memory Management Information
- Type
  - Access with: **typeof()**

# Functions

## Function Basics

**Functions** – objects in their own right

All R functions have three parts:

|               |                                                                                 |
|---------------|---------------------------------------------------------------------------------|
| body()        | code inside the function                                                        |
| formals()     | list of arguments which controls how you can call the function                  |
| environment() | “map” of the location of the function’s variables (see “Enclosing Environment”) |

Every operation is a function call

- +, for, if, [, \$, { ...
- x + y is the same as `+`(x, y)

**Note:** the backtick (`), lets you refer to functions or variables that have otherwise reserved or illegal names.

## Lexical Scoping

### What is Lexical Scoping?

- Looks up value of a symbol. (see “Enclosing Environment”)
- **findGlobals()** - lists all the external dependencies of a function

```
f <- function() x + 1
codetools::findGlobals(f)
> '+' 'x'

environment(f) <- emptyenv()
f()

error in f(): could not find function "+"
```

- R relies on lexical scoping to find everything, even the + operator.

## Function Arguments

**Arguments** – passed by reference and copied on modify

1. Arguments are matched first by exact name (perfect matching), then by prefix matching, and finally by position.
2. Check if an argument was supplied : **missing()**

```
i <- function(a, b) {
 missing(a) -> # return true or false
}
```

3. Lazy evaluation – since x is not used **stop("This is an error!")** never get evaluated.

```
f <- function(x) {
 10
}
f(stop('This is an error!')) -> 10
```

4. Force evaluation

```
f <- function(x) {
 force(x)
 10
}
```

5. Default arguments evaluation

```
f <- function(x = ls()) {
 a <- 1
 x
}
```

|                |                                      |
|----------------|--------------------------------------|
| f() -> 'a' 'x' | ls() evaluated inside f              |
| f(ls())        | ls() evaluated in global environment |

## Return Values

- **Last expression evaluated or explicit return()**. Only use explicit return() when returning early.
- **Return ONLY single object**. Workaround is to return a list containing any number of objects.
- **Invisible return object value** - not printed out by default when you call the function.

```
f1 <- function() invisible(1)
```

## Primitive Functions

### What are Primitive Functions?

1. Call C code directly with **.Primitive()** and contain no R code

```
print(sum) :
> function (... , na.rm = FALSE) .Primitive('sum')
```

2. **formals()**, **body()**, and **environment()** are all NULL

3. Only found in base package

4. More efficient since they operate at a low level

## Influx Functions

### What are Influx Functions?

1. Function name comes in between its arguments, like + or -
2. All user-created infix functions must start and end with %.

```
'%+%' <- function(a, b) paste0(a, b)
'new' %+%'string'
```

3. Useful way of providing a default value in case the output of another function is NULL:

```
'%||%' <- function(a, b) if (!is.null(a)) a else b
function_that_might_return_null() %||% default value
```

## Replacement Functions

### What are Replacement Functions?

1. Act like they modify their arguments in place, and have the special name xxx <-
2. Actually create a modified copy. Can use **pryr::address()** to find the memory address of the underlying object

```
second<- <- function(x, value) {
 x[2] <- value
 x
}
x <- 1:10
second(x) <- 5L
```

# Subsetting

Subsetting returns a copy of the original data, NOT copy-on modified

## Simplifying vs. Preserving Subsetting

### 1. Simplifying subsetting

- Returns the **simplest** possible data structure that can represent the output

### 2. Preserving subsetting

- Keeps the structure of the output the **same** as the input.
- When you use drop = FALSE, it's preserving

|            | Simplifying*     | Preserving                           |
|------------|------------------|--------------------------------------|
| Vector     | x[[1]]           | x[1]                                 |
| List       | x[[1]]           | x[1]                                 |
| Factor     | x[1:4, drop = T] | x[1:4]                               |
| Array      | x[1, ] or x[, 1] | x[1, , drop = F] or x[, 1, drop = F] |
| Data frame | x[, 1] or x[[1]] | x[, 1, drop = F] or x[1]             |

Simplifying behavior varies slightly between different data types:

### 1. Atomic Vector

- x[[1]] is the same as x[1]

### 2. List

- [ ] always returns a list
- Use [[ ]] to get list contents, this returns a single value piece out of a list

### 3. Factor

- Drops any unused levels but it remains a factor class

### 4. Matrix or Array

- If any of the dimensions has length 1, that dimension is dropped

### 5. Data Frame

- If output is a single column, it returns a vector instead of a data frame

## Data Frame Subsetting

**Data Frame** – possesses the **characteristics of both lists and matrices**. If you subset with a single vector, they behave like lists; if you subset with two vectors, they behave like matrices

### 1. Subset with a single vector : Behave like lists

```
df1[c('col1', 'col2')]
```

### 2. Subset with two vectors : Behave like matrices

```
df1[, c('col1', 'col2')]
```

The results are the same in the above examples, however, results are different if subsetting with only one column. (see below)

### 1. Behave like matrices

```
str(df1[, 'col1']) -> int [1:3]
```

- Result: the result is a vector

### 2. Behave like lists

```
str(df1['col1']) -> 'data.frame'
```

- Result: the result remains a data frame of 1 column

## \$ Subsetting Operator

### 1. About Subsetting Operator

- Useful shorthand for [[ combined with character subsetting

```
x$y is equivalent to x[['y', exact = FALSE]]
```

### 2. Difference vs. [[

- \$ does partial matching, [[ does not

```
x <- list(abc = 1)
x$a -> 1 # since "exact = FALSE"
x[['a']] -> # would be an error
```

### 3. Common mistake with \$

- Using it when you have the name of a column stored in a variable

```
var <- 'cyl'
x$var
doesn't work, translated to x[['var']]
Instead use x[[var]]
```

## Examples

### 1. Lookup tables (character subsetting)

```
x <- c('m', 'f', 'u', 'f', 'f', 'm', 'm')
lookup <- c(m = 'Male', f = 'Female', u = NA)
lookup[x]
> m f u f f m m
> 'Male' 'Female' NA 'Female' 'Female' 'Male' 'Male'
unname(lookup[x])
> 'Male' 'Female' NA 'Female' 'Female' 'Male' 'Male'
```

### 2. Matching and merging by hand (integer subsetting)

Lookup table which has multiple columns of information:

```
grades <- c(1, 2, 2, 3, 1)
info <- data.frame(
 grade = 3:1,
 desc = c('Excellent', 'Good', 'Poor'),
 fail = c(F, F, T)
)
```

First Method

```
id <- match(grades, info$grade)
info[id,]
```

Second Method

```
rownames(info) <- info$grade
info[as.character(grades),]
```

### 3. Expanding aggregated counts (integer subsetting)

- Problem:** a data frame where identical rows have been collapsed into one and a count column has been added
- Solution:** rep() and integer subsetting make it easy to uncollapse the data by subsetting with a repeated row index:  
rep(x, y) rep replicates the values in x, y times.

```
df1$countCol is c(3, 5, 1)
rep(1:nrow(df1), df1$countCol)
> 1 1 1 2 2 2 2 2 3
```

### 4. Removing columns from data frames (character subsetting)

There are two ways to remove columns from a data frame:

|                                        |                        |
|----------------------------------------|------------------------|
| Set individual columns to NULL         | df1\$col3 <- NULL      |
| Subset to return only columns you want | df1[c('col1', 'col2')] |

### 5. Selecting rows based on a condition (logical subsetting)

- This is the most commonly used technique for extracting rows out of a data frame.

```
df1[df1$col1 == 5 & df1$col2 == 4,]
```

## Subsetting continued

### Boolean Algebra vs. Sets (Logical and Integer Subsetting)

1. **Using integer subsetting** is more effective when:

- You want to find the first (or last) TRUE.
- You have very few TRUEs and very many FALSEs; a set representation may be faster and require less storage.

2. **which()** - conversion from boolean representation to integer representation

```
which(c(T, F, T F)) -> 1 3
```

- Integer representation length : is always <= boolean representation length
- Common mistakes :
  - I. Use **x[which(y)]** instead of **x[y]**
  - II. **x[-which(y)]** is not equivalent to **x[!y]**

#### Recommendation:

Avoid switching from logical to integer subsetting unless you want, for example, the first or last TRUE value

## Subsetting with Assignment

1. All subsetting operators can be combined with assignment to modify selected values of the input vector.

```
df1$col1[df1$col1 < 8] <- 0
```

2. Subsetting with nothing in conjunction with assignment :

- Why : Preserve original object class and structure

```
df1[] <- lapply(df1, as.integer)
```

## Debugging, Condition Handling and Defensive Programming

### Debugging Methods

#### 1. traceback() or RStudio's error inspector

- Lists the sequence of calls that lead to the error

#### 2. browser() or RStudio's breakpoints tool

- Opens an interactive debug session at an arbitrary location in the code

#### 3. options(error = browser) or RStudio's "Rerun with Debug" tool

- Opens an interactive debug session where the error occurred

#### Error Options:

##### options(error = recover)

- Difference vs. 'browser': can enter environment of any of the calls in the stack

##### options(error = dump\_and\_quit)

- Equivalent to 'recover' for non-interactive mode
- Creates **last.dump.rda** in the current working directory

In batch R process :

```
dump_and_quit <- function() {
 # Save debugging info to file
 last.dump.rda
 dump.frames(to.file = TRUE)
 # Quit R with error status
 q(status = 1)
}

options(error = dump_and_quit)
```

In a later interactive session :

```
load("last.dump.rda")
debugger()
```

### Condition Handling of Expected Errors

#### 1. Communicating potential problems to users:

##### I. stop()

- Action : raise fatal error and force all execution to terminate
- Example usage : when there is no way for a function to continue

##### II. warning()

- Action : generate warnings to display potential problems
- Example usage : when some of elements of a vectorized input are invalid

##### III. message()

- Action : generate messages to give informative output
- Example usage : when you would like to print the steps of a program execution

#### 2. Handling conditions programmatically:

##### I. try()

- Action : gives you the ability to continue execution even when an error occurs

##### II. tryCatch()

- Action : lets you specify handler functions that control what happens when a condition is signaled

```
result = tryCatch(code,
 error = function(c) "error",
 warning = function(c) "warning",
 message = function(c) "message"
)
```

Use **conditionMessage(c)** or **c\$message** to extract the message associated with the original error.

## Defensive Programming

**Basic principle** : "fail fast", to raise an error as soon as something goes wrong

1. **stopifnot()** or use 'assertthat' package - check inputs are correct

2. **Avoid subset(), transform() and with()** - these are non-standard evaluation, when they fail, often fail with uninformative error messages.

3. **Avoid [ and sapply()** - functions that can return different types of output.

- Recommendation : Whenever subsetting a data frame in a function, you should always use **drop = FALSE**



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

Contents:

1. **Markdown Syntax**
2. Knitr chunk options
3. Pandoc options

## Syntax

Plain text

End a line with two spaces to start a new paragraph.

\*italics\* and \_italics\_

\*\*bold\*\* and \_\_bold\_\_

superscript<sup>2</sup>

~~strikethrough~~

[link](www.rstudio.com)

# Header 1

## Header 2

### Header 3

#### Header 4

##### Header 5

##### Header 6

endash: --

emdash: ---

ellipsis: ...

inline equation: \$A = \pi \* r^2\$

image: 

horizontal rule (or slide break):

\*\*\*

> block quote

\* unordered list  
\* item 2  
  + sub-item 1  
  + sub-item 2

1. ordered list

2. item 2  
  + sub-item 1  
  + sub-item 2

| Table Header | Second Header |
|--------------|---------------|
| Table Cell   | Cell 2        |
| Cell 3       | Cell 4        |

## Becomes

Plain text

End a line with two spaces to start a new paragraph.

*italics* and *italics*

**bold** and **bold**

superscript<sup>2</sup>

~~strikethrough~~

[link](#)

## Header 1

## Header 2

## Header 3

## Header 4

### Header 5

### Header 6

endash: –

emdash: —

ellipsis: ...

inline equation:  $A = \pi * r^2$



horizontal rule (or slide break):

block quote

- unordered list
- item 2
  - sub-item 1
  - sub-item 2

1. ordered list

2. item 2
  - sub-item 1
  - sub-item 2

**Table Header**

**Second Header**

Table Cell

Cell 2

Cell 3

Cell 4



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. Markdown Syntax
- 2. Knitr chunk options**
3. Pandoc options

## Syntax

Make a code chunk with three back ticks followed by an r in braces. End the chunk with three back ticks:

```
```{r}
paste("Hello", "World!")
```

```

Place code inline with a single back ticks. The first back tick must be followed by an R, like this `r paste("Hello", "World!")`.

Add chunk options within braces. For example, `echo=FALSE` will prevent source code from being displayed:

```
```{r eval=TRUE, echo=FALSE}
paste("Hello", "World!")
```

```

## Becomes

Make a code chunk with three back ticks followed by an r in braces. End the chunk with three back ticks:

```
paste("Hello", "World!")
```

```
[1] "Hello World!"
```

Place code inline with a single back ticks. The first back tick must be followed by an R, like this Hello World!.

Add chunk options within braces. For example, `echo=FALSE` will prevent source code from being displayed:

```
[1] "Hello World!"
```

Learn more about chunk options at <http://yihui.name/knitr/options>

## Chunk options

| option                   | default value | description                                                                                                                                                                                                                                                                     |
|--------------------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Code evaluation</b>   |               |                                                                                                                                                                                                                                                                                 |
| <code>child</code>       | NULL          | A character vector of filenames. Knitr will knit the files and place them into the main document.                                                                                                                                                                               |
| <code>code</code>        | NULL          | Set to R code. Knitr will replace the code in the chunk with the code in the code option.                                                                                                                                                                                       |
| <code>engine</code>      | 'R'           | Knitr will evaluate the chunk in the named language, e.g. <code>engine = 'python'</code> . Run <code>names(knitr::knit_engines\$get())</code> to see supported languages.                                                                                                       |
| <code>eval</code>        | TRUE          | If FALSE, knitr will not run the code in the code chunk.                                                                                                                                                                                                                        |
| <code>include</code>     | TRUE          | If FALSE, knitr will run the chunk but not include the chunk in the final document.                                                                                                                                                                                             |
| <code>purl</code>        | TRUE          | If FALSE, knitr will not include the chunk when running <code>purl()</code> to extract the source code.                                                                                                                                                                         |
| <b>Results</b>           |               |                                                                                                                                                                                                                                                                                 |
| <code>collapse</code>    | FALSE         | If TRUE, knitr will collapse all the source and output blocks created by the chunk into a single block.                                                                                                                                                                         |
| <code>echo</code>        | TRUE          | If FALSE, knitr will not display the code in the code chunk above it's results in the final document.                                                                                                                                                                           |
| <code>results</code>     | 'markup'      | If 'hide', knitr will not display the code's results in the final document. If 'hold', knitr will delay displaying all output pieces until the end of the chunk. If 'asis', knitr will pass through results without reformatting them (useful if results return raw HTML, etc.) |
| <code>error</code>       | TRUE          | If FALSE, knitr will not display any error messages generated by the code.                                                                                                                                                                                                      |
| <code>message</code>     | TRUE          | If FALSE, knitr will not display any messages generated by the code.                                                                                                                                                                                                            |
| <code>warning</code>     | TRUE          | If FALSE, knitr will not display any warning messages generated by the code.                                                                                                                                                                                                    |
| <b>Code Decoration</b>   |               |                                                                                                                                                                                                                                                                                 |
| <code>comment</code>     | '##'          | A character string. Knitr will append the string to the start of each line of results in the final document.                                                                                                                                                                    |
| <code>highlight</code>   | TRUE          | If TRUE, knitr will highlight the source code in the final output.                                                                                                                                                                                                              |
| <code>prompt</code>      | FALSE         | If TRUE, knitr will add > to the start of each line of code displayed in the final document.                                                                                                                                                                                    |
| <code>strip.white</code> | TRUE          | If TRUE, knitr will remove white spaces that appear at the beginning or end of a code chunk.                                                                                                                                                                                    |
| <code>tidy</code>        | FALSE         | If TRUE, knitr will tidy code chunks for display with the <code>tidy_source()</code> function in the <code>formatR</code> package.                                                                                                                                              |



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. Markdown Syntax
2. Knitr chunk options
3. Pandoc options

## Chunk options (Continued)

| option                             | default value   | description                                                                                                                                                                                                                                                                                            |
|------------------------------------|-----------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Chunks</b>                      |                 |                                                                                                                                                                                                                                                                                                        |
| <b>opts.label</b>                  | NULL            | The label of options set in <code>knitr:: opts_template()</code> to use with the chunk.                                                                                                                                                                                                                |
| <b>R.options</b>                   | NULL            | Local R options to use with the chunk. Options are set with <code>options()</code> at start of chunk. Defaults are restored at end.                                                                                                                                                                    |
| <b>ref.label</b>                   | NULL            | A character vector of labels of the chunks from which the code of the current chunk is inherited.                                                                                                                                                                                                      |
| <b>Cache</b>                       |                 |                                                                                                                                                                                                                                                                                                        |
| <b>autodep</b>                     | FALSE           | If <b>TRUE</b> , knitr will attempt to figure out dependencies between chunks automatically by analyzing object names.                                                                                                                                                                                 |
| <b>cache</b>                       | FALSE           | If <b>TRUE</b> , knitr will cache the results to reuse in future knits. Knitr will reuse the results until the code chunk is altered.                                                                                                                                                                  |
| <b>cache.comments</b>              | NULL            | If <b>FALSE</b> , knitr will not rerun the chunk if only a code comment has changed.                                                                                                                                                                                                                   |
| <b>cache.lazy</b>                  | TRUE            | If <b>TRUE</b> , knitr will use <code>lazylload()</code> to load objects in chunk. If <b>FALSE</b> , knitr will use <code>load()</code> to load objects in chunk.                                                                                                                                      |
| <b>cache.path</b>                  | 'cache/'        | A file path to the directory to store cached results in. Path should begin in the directory that the .Rmd file is saved in.                                                                                                                                                                            |
| <b>cache.vars</b>                  | NULL            | A character vector of object names to cache if you do not wish to cache each object in the chunk.                                                                                                                                                                                                      |
| <b>dependson</b>                   | NULL            | A character vector of chunk labels to specify which other chunks a chunk depends on. Knitr will update a cached chunk if its dependencies change.                                                                                                                                                      |
| <b>Animation</b>                   |                 |                                                                                                                                                                                                                                                                                                        |
| <b>anipots</b>                     | 'controls,loop' | Extra options for animations (see the <code>animate</code> package).                                                                                                                                                                                                                                   |
| <b>interval</b>                    | 1               | The number of seconds to pause between animation frames.                                                                                                                                                                                                                                               |
| <b>Plots</b>                       |                 |                                                                                                                                                                                                                                                                                                        |
| <b>dev</b>                         | 'png'           | The R function name that will be used as a graphical device to record plots, e.g. <code>dev='CairoPDF'</code> .                                                                                                                                                                                        |
| <b>dev.args</b>                    | NULL            | Arguments to be passed to the device, e.g. <code>dev.args=list(bg='yellow', pointsize=10)</code> .                                                                                                                                                                                                     |
| <b>dpi</b>                         | 72              | A number for knitr to use as the dots per inch (dpi) in graphics (when applicable).                                                                                                                                                                                                                    |
| <b>external</b>                    | TRUE            | If <b>TRUE</b> , knitr will externalize tikz graphics to save LaTex compilation time (only for the <code>tikzDevice::tikz()</code> device).                                                                                                                                                            |
| <b>fig.align</b>                   | 'default'       | How to align graphics in the final document. One of 'left', 'right', or 'center'.                                                                                                                                                                                                                      |
| <b>fig.cap</b>                     | NULL            | A character string to be used as a figure caption in LaTex.                                                                                                                                                                                                                                            |
| <b>fig.env</b>                     | 'figure'        | The Latex environment for figures.                                                                                                                                                                                                                                                                     |
| <b>fig.ext</b>                     | NULL            | The file extension for figure output, e.g. <code>fig.ext='png'</code> .                                                                                                                                                                                                                                |
| <b>fig.height, fig.width</b>       | 7               | The width and height to use in R for plots created by the chunk (in inches).                                                                                                                                                                                                                           |
| <b>fig.keep</b>                    | 'high'          | If 'high', knitr will merge low-level changes into high level plots. If 'all', knitr will keep all plots (low-level changes may produce new plots). If 'first', knitr will keep the first plot only. If 'last', knitr will keep the last plot only. If 'none', knitr will discard all plots.           |
| <b>fig.lp</b>                      | 'fig:'          | A prefix to be used for figure labels in latex.                                                                                                                                                                                                                                                        |
| <b>fig.path</b>                    | 'figure/'       | A file path to the directory where knitr should store the graphics files created by the chunk.                                                                                                                                                                                                         |
| <b>fig.pos</b>                     | "               | A character string to be used as the figure position arrangement in LaTex.                                                                                                                                                                                                                             |
| <b>fig.process</b>                 | NULL            | A function to post-process a figure file. Should take a filename and return a filename of a new figure source.                                                                                                                                                                                         |
| <b>fig.retina</b>                  | 1               | Dpi multiplier for displaying HTML output on retina screens.                                                                                                                                                                                                                                           |
| <b>fig.scap</b>                    | NULL            | A character string to be used as a short figure caption.                                                                                                                                                                                                                                               |
| <b>fig.subcap</b>                  | NULL            | A character string to be used as captions in sub-figures in LaTex.                                                                                                                                                                                                                                     |
| <b>fig.show</b>                    | 'asis'          | If 'hide', knitr will generate the plots created in the chunk, but not include them in the final document. If 'hold', knitr will delay displaying the plots created by the chunk until the end of the chunk. If 'animate', knitr will combine all of the plots created by the chunk into an animation. |
| <b>fig.showtext</b>                | NULL            | If <b>TRUE</b> , knitr will call <code>showtext::showtext.begin()</code> before drawing plots.                                                                                                                                                                                                         |
| <b>out.extra</b>                   | NULL            | A character string of extra options for figures to be passed to LaTex or HTML.                                                                                                                                                                                                                         |
| <b>out.height, out.width</b>       | NULL            | The width and height to scale plots to in the final output. Can be in units recognized by output, e.g. <code>8\ linewidth, 50px</code>                                                                                                                                                                 |
| <b>resize.height, resize.width</b> | NULL            | The width and height to resize like graphics in LaTex, passed to <code>\resizebox{}`{`}</code> .                                                                                                                                                                                                       |
| <b>sanitize</b>                    | FALSE           | If <b>TRUE</b> , knitr will sanitize like graphics for LaTex.                                                                                                                                                                                                                                          |



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. Markdown Syntax
2. Knitr chunk options
- 3. Pandoc options**

| Templates                                                                                                                           | Basic YAML | Template options | Latex options | Interactive Docs |
|-------------------------------------------------------------------------------------------------------------------------------------|------------|------------------|---------------|------------------|
| html_document<br>pdf_document<br>word_document<br>md_document<br>ioslides_presentation<br>slidy_presentation<br>beamer_presentation | ---        | ---              | ---           | ---              |

## Syntax for slide formats (ioslides, slidify, beamer)

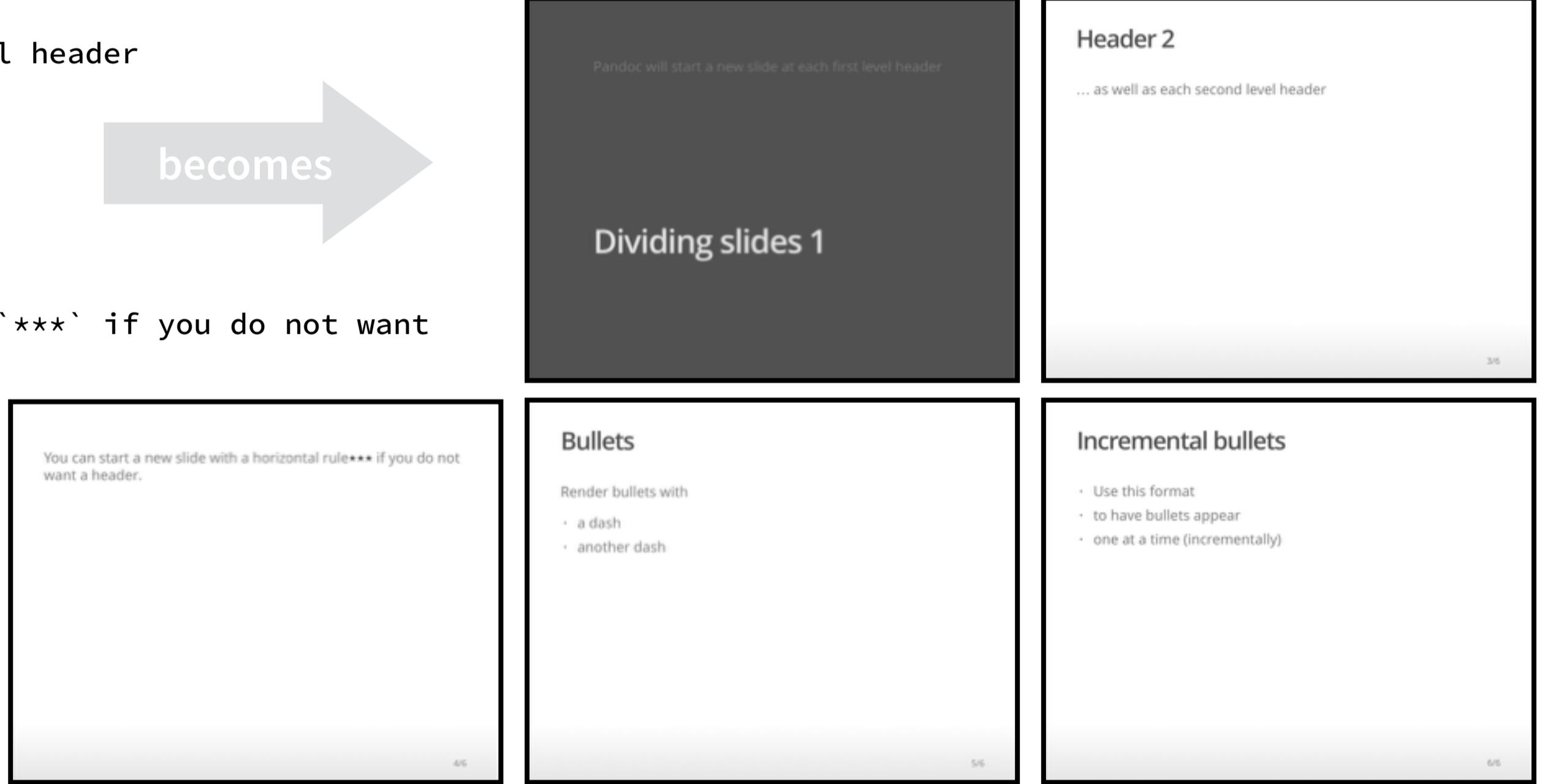
```
Dividing slides 1
Pandoc will start a new slide at each first level header
Header 2
... as well as each second level header

You can start a new slide with a horizontal rule`***` if you do not want
a header.

Bullets
Render bullets with
- a dash
- another dash

Incremental bullets
>- Use this format
>- to have bullets appear
>- one at a time (incrementally)
```

becomes



## Slide display modes

Press a key below during presentation to enter display mode. Press **esc** to exit display mode.

### ioslides

- |          |                              |
|----------|------------------------------|
| <b>f</b> | - enable fullscreen mode     |
| <b>w</b> | - toggle widescreen mode     |
| <b>o</b> | - enable overview mode       |
| <b>h</b> | - enable code highlight mode |
| <b>p</b> | - show presenter notes       |

### slidy

- |          |                                           |
|----------|-------------------------------------------|
| <b>C</b> | - show table of contents                  |
| <b>F</b> | - toggle display of the footer            |
| <b>A</b> | - toggle display of current vs all slides |
| <b>S</b> | - make fonts smaller                      |
| <b>B</b> | - make fonts bigger                       |

## Top level options to customize LaTeX (pdf) output

| option                                        | description                                                                                     |
|-----------------------------------------------|-------------------------------------------------------------------------------------------------|
| <b>lang</b>                                   | Document language code                                                                          |
| <b>fontsize</b>                               | Font size (e.g. 10pt, 11pt, 12 pt)                                                              |
| <b>documentclass</b>                          | Latex document class (e.g. article)                                                             |
| <b>classoption</b>                            | Option for document class (e.g. oneside); may be repeated                                       |
| <b>geometry</b>                               | Options for geometry class (e.g. margin=1in); may be repeated                                   |
| <b>mainfont, sansfont, monofont, mathfont</b> | Document fonts (works only with xelatex and lualatex, see the <code>latex_engine</code> option) |
| <b>linkcolor, urlcolor, citecolor</b>         | Color for internal, external, and citation links (red, green, magenta, cyan, blue, black)       |



# R Markdown Reference Guide

Learn more about R Markdown at [rmarkdown.rstudio.com](http://rmarkdown.rstudio.com)

Learn more about Interactive Docs at [shiny.rstudio.com/articles](http://shiny.rstudio.com/articles)

## Contents:

1. Markdown Syntax
2. Knitr chunk options
- 3. Pandoc options**

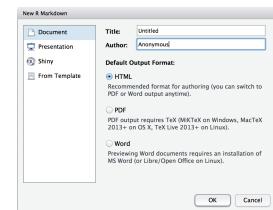
| option          | html | pdf | word | md | ioslides | slidy | beamer | description                                                                                                                                                 |
|-----------------|------|-----|------|----|----------|-------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| colortheme      |      |     |      |    |          |       | X      | Beamer color theme to use (e.g., <code>colortheme: "dolphin"</code> ).                                                                                      |
| css             | X    |     |      |    | X        | X     |        | Filepath to CSS style to use to style document (e.g., <code>css: styles.css</code> ).                                                                       |
| duration        |      |     |      |    |          | X     |        | Add a countdown timer (in minutes) to footer of slides (e.g., <code>duration: 45</code> ).                                                                  |
| fig_caption     | X    | X   | X    |    | X        | X     | X      | Should figures be rendered with captions?                                                                                                                   |
| fig_crop        |      | X   |      |    |          |       | X      | Should pdfcrop utility be automatically applied to figures (when available)?                                                                                |
| fig_height      | X    | X   | X    | X  | X        | X     | X      | Default figure height (in inches) for document.                                                                                                             |
| fig_retina      | X    |     |      | X  | X        | X     |        | Scaling to perform for retina displays (e.g., <code>fig_retina: 2</code> ).                                                                                 |
| fig_width       | X    | X   | X    | X  | X        | X     | X      | Default figure width (in inches) for document.                                                                                                              |
| font_adjustment |      |     |      |    |          | X     |        | Increase or decrease font size for entire presentation (e.g., <code>font_adjustment: -1</code> ).                                                           |
| fonttheme       |      |     |      |    |          |       | X      | Beamer font theme to use (e.g., <code>fonttheme: "structurebold"</code> ).                                                                                  |
| footer          |      |     |      |    |          | X     |        | Text to add to footer of each slide (e.g., <code>footer: "Copyright (c) 2014 RStudio"</code> ).                                                             |
| highlight       | X    | X   |      |    | X        | X     |        | Syntax highlighting style (e.g. "tango", "pygments", "kate", "zenburn", and                                                                                 |
| includes        | X    | X   |      | X  | X        | X     |        | See below                                                                                                                                                   |
| -in_header      | X    | X   |      |    | X        | X     | X      | File of content to place in document header (e.g., <code>in_header: header.html</code> ).                                                                   |
| -before_body    | X    | X   |      |    | X        | X     | X      | File of content to place before document body (e.g., <code>before_body:</code>                                                                              |
| -after_body     | X    | X   |      |    | X        | X     | X      | File of content to place after document body (e.g., <code>after_body: doc_suffix.html</code> ).                                                             |
| incremental     |      |     |      |    | X        | X     | X      | Should bullets appear one at a time (on presenter mouse clicks)?                                                                                            |
| keep_md         | X    |     |      |    | X        | X     |        | Save a copy of .md file that contains knitr output (in addition to the .Rmd and HTML files)?                                                                |
| keep_tex        |      | X   |      |    |          | X     |        | Save a copy of .tex file that contains knitr output (in addition to the .Rmd and PDF files)?                                                                |
| latex_engine    |      | X   |      |    |          |       |        | Engine to render latex. Should be one of "pdflatex", "xelatex", and "lualatex".                                                                             |
| lib_dir         | X    |     |      |    | X        | X     |        | Directory of dependency files to use (Bootstrap, MathJax, etc.) (e.g., <code>lib_dir: libs</code> ).                                                        |
| logo            |      |     |      |    | X        |       |        | File path to a logo (at least 128 x 128) to add to presentation (e.g., <code>logo: logo.png</code> ).                                                       |
| mathjax         | X    |     |      |    | X        | X     |        | Set to <code>local</code> or a URL to use a local/URL version of MathJax to render equations                                                                |
| number_section  | X    | X   |      |    |          |       |        | Add section numbering to headers (e.g., <code>number_sections: true</code> ).                                                                               |
| pandoc_args     | X    | X   | X    | X  | X        | X     | X      | Arguments to pass to Pandoc (e.g., <code>pandoc_args: ["--title-prefix", "Foo"]</code> ).                                                                   |
| preserve_yaml   |      |     |      | X  |          |       |        | Preserve YAML front matter in final document?                                                                                                               |
| reference_docx  |      |     | X    |    |          |       |        | A .docx file whose styles should be copied to use (e.g., <code>reference_docx:</code>                                                                       |
| self_contained  | X    |     |      |    | X        | X     |        | Embed dependencies into the doc? Set to <code>false</code> to keep dependencies in external files.                                                          |
| slide_level     |      |     |      |    |          | X     |        | The lowest heading level that defines individual slides (e.g., <code>slide_level: 2</code> ).                                                               |
| smaller         |      |     |      |    | X        |       |        | Use the smaller font size in the presentation?                                                                                                              |
| smart           | X    |     |      |    | X        | X     |        | Convert straight quotes to curly, dashes to em-dashes, ... to ellipses, and so on?                                                                          |
| template        | X    | X   |      |    | X        | X     |        | Pandoc template to use when rendering file (e.g., <code>template:</code>                                                                                    |
| theme           | X    |     |      |    |          | X     |        | Bootswatch or Beamer theme to use for page. Valid bootswatch themes include "cerulean", "journal", "flatly", "readable", "spacelab", "united", and "cosmo". |
| toc             | X    | X   |      | X  |          | X     |        | Add a table of contents at start of document? (e.g., <code>toc: true</code> ).                                                                              |
| toc_depth       | X    | X   |      | X  |          |       |        | The lowest level of headings to add to table of contents (e.g., <code>toc_depth: 2</code> ).                                                                |
| transition      |      |     |      |    | X        |       |        | Speed of slide transitions should be "slower", "faster" or a number in seconds.                                                                             |
| variant         |      |     | X    |    |          |       |        | The flavor of markdown to use; one of "markdown", "markdown_strict", "markdown_github", "markdown_mmd", and "markdown_phpextra"                             |
| widescreen      |      |     |      | X  |          |       |        | Display presentation in widescreen format?                                                                                                                  |

# R Markdown :: CHEAT SHEET

## What is R Markdown?

- .Rmd files** • An R Markdown (.Rmd) file is a record of your research. It contains the code that a scientist needs to reproduce your work along with the narration that a reader needs to understand your work.
- Reproducible Research** • At the click of a button, or the type of a command, you can rerun the code in an R Markdown file to reproduce your work and export the results as a finished report.
- Dynamic Documents** • You can choose to export the finished report in a variety of formats, including html, pdf, MS Word, or RTF documents; html or pdf based slides, Notebooks, and more.

## Workflow



- Open a new .Rmd file at File ► New File ► R Markdown. Use the wizard that opens to pre-populate the file with a template
- Write document by editing template
- Knit document to create report; use knit button or render() to knit
- Preview Output in IDE window
- Publish (optional) to web server
- Examine build log in R Markdown console
- Use output file that is saved along side .Rmd

The screenshot illustrates the RStudio environment for R Markdown. It shows the RStudio menu bar, a code editor with R Markdown syntax, and a browser displaying the resulting HTML output. The code editor highlights various parts of the R Markdown file, such as YAML headers, code chunks, and rendered output sections. The browser shows the final rendered HTML page with a summary of the 'cars' dataset.

## render

Use `rmarkdown::render()` to render/knit at cmd line. Important args:

**input** - file to render  
**output\_format**

**output\_options** - List of render options (as in YAML)

**output\_file**  
**output\_dir**

**params** - list of params to use

**envir** - environment to evaluate code chunks in  
**encoding** - of input file

## Embed code with knitr syntax

### INLINE CODE

Insert with `r <code>`. Results appear as text without code.

Built with `r getRVersion()` ➔ Built with 3.2.3

### CODE CHUNKS

One or more lines surrounded with `{{r}}` and `{{ }}`. Place chunk options within curly braces, after r. Insert with ➔

```{r echo=TRUE}  
getRVersion()
```

### GLOBAL OPTIONS

Set with `knitr::opts_chunk$set()`, e.g.

```{r include=FALSE}  
knitr::opts_chunk\$set(echo = TRUE)
```

### IMPORTANT CHUNK OPTIONS

**cache** - cache results for future knits (default = FALSE)

**cache.path** - directory to save cached results in (default = "cache/")

**child** - file(s) to knit and then include (default = NULL)

**collapse** - collapse all output into single block (default = FALSE)

**comment** - prefix for each line of results (default = "#")

**dependson** - chunk dependencies for caching (default = NULL)

**echo** - Display code in output document (default = TRUE)

**engine** - code language used in chunk (default = 'R')

**error** - Display error messages in doc (TRUE) or stop render when errors occur (FALSE) (default = FALSE)

**eval** - Run code in chunk (default = TRUE)

**fig.align** - 'left', 'right', or 'center' (default = 'default')

**fig.cap** - figure caption as character string (default = NULL)

**fig.height, fig.width** - Dimensions of plots in inches

**highlight** - highlight source code (default = TRUE)

**include** - Include chunk in doc after running (default = TRUE)

**message** - display code messages in document (default = TRUE)

**results** (default = 'markup')

'asis' - passthrough results

'hide' - do not display results

'hold' - put all results below all code

**tidy** - tidy code for display (default = FALSE)

**warning** - display code warnings in document (default = TRUE)

Options not listed above: `R.options`, `aniopts`, `autodep`, `background`, `cache.comments`, `cache.lazy`, `cache.rebuild`, `cache.vars`, `dev`, `dev.args`, `dpi`, `engine.opts`, `engine.path`, `fig.asp`, `fig.env`, `fig.ext`, `fig.keep`, `fig.lp`, `fig.path`, `fig.pos`, `fig.process`, `fig.retina`, `fig.scap`, `fig.show`, `fig.showtext`, `fig.subcap`, `interval`, `out.extra`, `out.height`, `out.width`, `prompt`, `purl`, `ref.label`, `render`, `size`, `split`, `tidy.opts`



## .rmd Structure

### YAML Header

Optional section of render (e.g. pandoc) options written as key:value pairs (YAML).

At start of file

Between lines of ---

### Text

Narration formatted with markdown, mixed with:

### Code Chunks

Chunks of embedded code. Each chunk:

Begins with `{{r}}`

ends with `{{ }}`

R Markdown will run the code and append the results to the doc.

It will use the location of the .Rmd file as the **working directory**

## Parameters

Parameterize your documents to reuse with different inputs (e.g., data, values, etc.)

- Add parameters • Create and set parameters in the header as sub-values of params

```

params:
 n: 100
 d: ! Sys.Date()

```

- Call parameters • Call parameter values in code as `params$<name>`

Today's date is `r params\$d`

- Set parameters • Set values with Knit with parameters or the params argument of render():
 

```
render("doc.Rmd", params = list(n = 1, d = as.Date("2015-01-01")))
```

Knit to HTML  
Knit to PDF  
Knit to Word  
Knit with Parameters...

## Interactive Documents

Turn your report into an interactive Shiny document in 4 steps

- Add runtime: shiny to the YAML header.
- Call Shiny input functions to embed input objects.
- Call Shiny render functions to embed reactive output.
- Render with `rmarkdown::run` or click Run Document in RStudio IDE

```

output: html_document
runtime: shiny

```{r, echo = FALSE}  
numericInput("n",  
  "How many cars?", 5)  
renderTable({  
  head(cars, input$n)  
})``
```

How many cars?
5
speed dist
1 4.00 2.00
2 4.00 10.00
3 7.00 4.00
4 7.00 22.00
5 8.00 16.00

Embed a complete app into your document with `shiny::shinyAppDir()`

NOTE: Your report will be rendered as a Shiny app, which means you must choose an html output format, like `html_document`, and serve it with an active R Session.



Pandoc's Markdown

Write with syntax on the left to create effect on right (after render)

```
Plain text
End a line with two spaces
to start a new paragraph.
*italics* and **bold**
`verbatim` code
sub/superscript22
~~strikethrough~~
escaped: `*` \\
endash: --, emdash: ---
equation: $A = \pi * r^2$
```

```
equation block:
```

```
$$E = mc^2$$
```

```
> block quote
```

```
# Header1 {#anchor}
```

```
## Header 2 {#css_id}
```

```
### Header 3 {.css_class}
```

```
#### Header 4
```

```
##### Header 5
```

```
##### Header 6
```

```
<!--Text comment-->
```

```
\textbf{Text ignored in HTML}
<em>HTML ignored in pdfs</em>
```

```
<http://www.rstudio.com>
[link](www.rstudio.com)
Jump to [Header 1]{#anchor}
image:
```

```
Plain text
End a line with two spaces
to start a new paragraph.
*italics* and **bold**
`verbatim` code
sub/superscript22
~~strikethrough~~
escaped: `*` \\
endash: --, emdash: ---
equation: $A = \pi * r^2$
```

```
E = mc2
```

```
block quote
```

Header1

Header 2

Header 3

Header 4

Header 5

Header 6

HTML ignored in pdfs

http://www.rstudio.com

link

Jump to Header 1

image:



Caption

- unordered list
 - sub-item 1
 - sub-item 2
 - sub-sub-item 1
- item 2

Continued (indent 4 spaces)

1. ordered list

2. item 2

- i) sub-item 1
 - A. sub-sub-item 1

(@) A list whose numbering

continues after

continues after

- 2. an interruption

Term 1

Definition 1

Right	Left	Default	Center
12	12	12	12
123	123	123	123
1	1	1	1

- slide bullet 1
 - slide bullet 2

(>- to have bullets appear on click)

horizontal rule/slide break:

A footnote^[1]

[^1]: Here is the footnote.

A footnote¹

1. Here is the footnote.¹

Set render options with YAML

When you render, R Markdown

1. runs the R code, embeds results and text into .md file with knitr
2. then converts the .md file into the finished format with pandoc



Set a document's default output format in the YAML header:

```
---  
output: html_document  
---  
# Body
```

output value

creates

html_document	html
pdf_document	pdf (requires Tex)
word_document	Microsoft Word (.docx)
odt_document	OpenDocument Text
rtf_document	Rich Text Format
md_document	Markdown
github_document	Github compatible markdown
ioslides_presentation	ioslides HTML slides
slidy_presentation	slidy HTML slides
beamer_presentation	Beamer pdf slides (requires Tex)

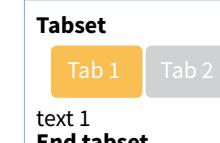
Customize output with sub-options (listed to the right):

```
---  
output: html_document:  
  code_folding: hide  
  toc_float: TRUE  
---  
# Body
```

html tabs

Use tablet css class to place sub-headers into tabs

```
# Tabset {.tabset .tabset-fade .tabset-pills}  
## Tab 1  
text 1  
## Tab 2  
text 2  
### End tabset
```



Create a Reusable Template

1. **Create a new package** with a `inst/rmarkdown/templates` directory

2. In the directory, **Place a folder** that contains:

template.yaml (see below)

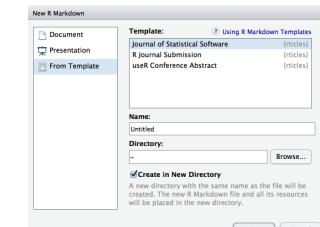
skeleton.Rmd (contents of the template)

any supporting files

3. **Install the package**

4. **Access template** in wizard at File ▶ New File ▶ R Markdown template.yaml

```
---  
name: My Template  
---
```



sub-option **description**

citation_package The LaTeX package to process citations, natbib, biblatex or none

	html	pdf	word	odt	rtf	md	gitbook	ioslides	slidy	beamer
X				X						X

code_folding Let readers to toggle the display of R code, "none", "hide", or "show"

X

colortheme Beamer color theme to use

--

css CSS file to use to style document

X	X	X
---	---	---

dev Graphics device to use for figure output (e.g. "png")

X	X	X	X	X
---	---	---	---	---

duration Add a countdown timer (in minutes) to footer of slides

--

fig_caption Should figures be rendered with captions?

X	X	X	X	X
---	---	---	---	---

fig_height, fig_width Default figure height and width (in inches) for document

X	X	X	X	X
---	---	---	---	---

highlight Syntax highlighting: "tango", "pygments", "kate", "zenburn", "textmate"

X	X	X
---	---	---

includes File of content to place in document (in_header, before_body, after_body)

X	X	X	X	X
---	---	---	---	---

incremental Should bullets appear one at a time (on presenter mouse clicks)?

X	X	X
---	---	---

keep_md Save a copy of .md file that contains knitr output

X	X	X	X
---	---	---	---

keep_tex Save a copy of .tex file that contains knitr output

X

latex_engine Engine to render latex, "pdflatex", "xelatex", or "lualatex"

X

lib_dir Directory of dependency files to use (Bootstrap, MathJax, etc.)

X	X
---	---

mathjax Set to local or a URL to use a local/URL version of MathJax to render equations

X	X
---	---

md_extensions Markdown extensions to add to default definition or R Markdown

X	X	X	X	X
</tbl