# 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**)**

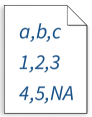
**String vector to file, one element per line**
**write_lines(**x,path, na = "NA", append = FALSE**)**

**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**)**

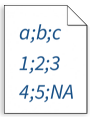
## Read Tabular Data - These functions share the common arguments:

**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()**)**



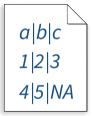

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | NA |

**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")



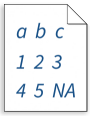

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | NA |

**Semi-colon Delimited Files**
**read_csv2(**"file2.csv"**)**
write_file(x = "a;b;c\n1;2;3\n4;5;NA", path = "file2.csv")




| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | NA |

**Files with Any Delimiter**
**read_delim(**"file.txt", delim = "|"**)**
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")

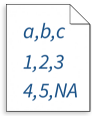



| A | B | C |
|---|---|---|
| 1 | 2 | 3 |
| 4 | 5 | NA |

**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




**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 |

**No header**
read_csv(f, **col_names = FALSE**)

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

**Provide header**
read_csv(f, **col_names = c("x", "y", "z")**)

| 1 | 2 | 3 |
|---|---|---|
| 4 | 5 | NA |

**Skip lines**
read_csv(f, **skip = 1**)

| A | B | C |
|---|---|---|
| 1 | 2 | 3 |

**Read in a subset**
read_csv(f, **n_max = 1**)

| 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
earn is a double (numeric)

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()**

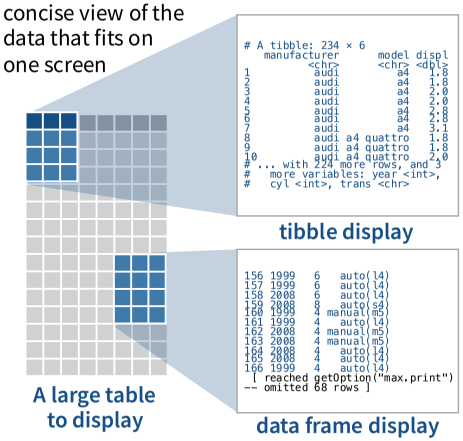
*x$A <- parse_number(x$A)*

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01

# 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




- Control the default appearance with options:

  **options(**tibble.print_max = n, tibble.print_min = m, tibble.width = Inf**)**

- View full data set with **View()** or **glimpse()**
- Revert to data frame with **as.data.frame()**

## CONSTRUCT A TIBBLE IN TWO WAYS

**tibble**(…)
Construct by columns.
*tibble(x = 1:3, y = c("a", "b", "c"))*

**tribble**(…)
Construct by rows.
*tribble(~x, ~y, 1, "a", 2, "b", 3, "c")*

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

**as_tibble**(x, …) Convert data frame to tibble.

**enframe**(x, name = "name", value = "value")
Convert named vector to a tibble

**is_tibble**(x) Test whether x is a tibble.

# Tidy Data with tidyr

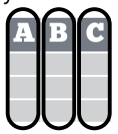
**Tidy data** is a way to organize tabular data. It provides a consistent data structure across packages.

A table is tidy if:

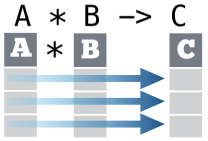
Each **variable** is in its own **column**

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

Tidy data:



Makes variables easy to access as vectors




Preserves cases during vectorized operations

# Reshape Data - change the layout of values in a table

Use **gather()** and **spread()** to reorganize the values of a table into a new layout.

**gather**(data, key, value, ..., na.rm = FALSE, convert = FALSE, factor_key = FALSE**)**

gather() moves column names into a **key** column, gathering the column values into a single **value** column.

table4a

| country | 1999 | 2000 |
|---|---|---|
| A | 0.7K | 2K |
| B | 37K | 80K |
| C | 212K | 213K |

→

| country | year | cases |
|---|---|---|
| A | 1999 | 0.7K |
| B | 1999 | 37K |
| C | 1999 | 212K |
| A | 2000 | 2K |
| B | 2000 | 80K |
| C | 2000 | 213K |

key value

*gather(table4a, `1999`, `2000`, key = "year", value = "cases")*

**spread**(data, key, value, fill = NA, convert = FALSE, drop = TRUE, sep = NULL**)**

spread() moves the unique values of a **key** column into the column names, spreading the values of a **value** column across the new columns.

table2

| country | year | type | count |
|---|---|---|---|
| A | 1999 | cases | 0.7K |
| A | 1999 | pop | 19M |
| A | 2000 | cases | 2K |
| A | 2000 | pop | 20M |
| B | 1999 | cases | 37K |
| B | 1999 | pop | 172M |
| B | 2000 | cases | 80K |
| B | 2000 | pop | 174M |
| C | 1999 | cases | 212K |
| C | 1999 | pop | 1T |
| C | 2000 | cases | 213K |
| C | 2000 | pop | 1T |

→

| country | year | cases | pop |
|---|---|---|---|
| A | 1999 | 0.7K | 19M |
| A | 2000 | 2K | 20M |
| B | 1999 | 37K | 172M |
| B | 2000 | 80K | 174M |
| C | 1999 | 212K | 1T |
| C | 2000 | 213K | 1T |

key value

*spread(table2, type, count)*

# Handle Missing Values

**drop_na**(data, ...)
Drop rows containing NA's in … columns.

x

| x1 | x2 |
|---|---|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

→

| x1 | x2 |
|---|---|
| A | 1 |
| D | 3 |

*drop_na(x, x2)*

**fill**(data, ..., .direction = c("down", "up"))
Fill in NA's in … columns with most recent non-NA values.

x

| x1 | x2 |
|---|---|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

→

| x1 | x2 |
|---|---|
| A | 1 |
| B | 1 |
| C | 1 |
| D | 3 |
| E | 3 |

*fill(x, x2)*

**replace_na**(data, replace = list(), ...)
Replace NA's by column.

x

| x1 | x2 |
|---|---|
| A | 1 |
| B | NA |
| C | NA |
| D | 3 |
| E | NA |

→

| x1 | x2 |
|---|---|
| A | 1 |
| B | 2 |
| C | 2 |
| D | 3 |
| E | 2 |

*replace_na(x, list(x2 = 2))*

# Expand Tables - quickly create tables with combinations of values

**complete**(data, ..., fill = list())
Adds to the data missing combinations of the values of the variables listed in …
*complete(mtcars, cyl, gear, carb)*

**expand**(data, ...)
Create new tibble with all possible combinations of the values of the variables listed in …
*expand(mtcars, cyl, gear, carb)*

# Split Cells

Use these functions to split or combine cells into individual, isolated values.

**separate**(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", ...**)**

Separate each cell in a column to make several columns.

table3

| country | year | rate |
|---|---|---|
| A | 1999 | 0.7K/19M |
| A | 2000 | 2K/20M |
| B | 1999 | 37K/172M |
| B | 2000 | 80K/174M |
| C | 1999 | 212K/1T |
| C | 2000 | 213K/1T |

→

| country | year | cases | pop |
|---|---|---|---|
| A | 1999 | 0.7K | 19M |
| A | 2000 | 2K | 20M |
| B | 1999 | 37K | 172 |
| B | 2000 | 80K | 174 |
| C | 1999 | 212K | 1T |
| C | 2000 | 213K | 1T |

*separate(table3, rate, into = c("cases", "pop"))*

**separate_rows**(data, ..., sep = "[^[:alnum:].]+", convert = FALSE**)**

Separate each cell in a column to make several rows. Also **separate_rows_()**.

table3

| country | year | rate |
|---|---|---|
| A | 1999 | 0.7K/19M |
| A | 2000 | 2K/20M |
| B | 1999 | 37K/172M |
| B | 2000 | 80K/174M |
| C | 1999 | 212K/1T |
| C | 2000 | 213K/1T |

→

| country | year | rate |
|---|---|---|
| A | 1999 | 0.7K |
| A | 1999 | 19M |
| A | 2000 | 2K |
| A | 2000 | 20M |
| B | 1999 | 37K |
| B | 1999 | 172M |
| B | 2000 | 80K |
| B | 2000 | 174M |
| C | 1999 | 212K |
| C | 1999 | 1T |
| C | 2000 | 213K |
| C | 2000 | 1T |

*separate_rows(table3, rate)*

**unite**(data, col, ..., sep = "_", remove = TRUE**)**

Collapse cells across several columns to make a single column.

table5

| country | century | year |
|---|---|---|
| Afghan | 19 | 99 |
| Afghan | 20 | 0 |
| Brazil | 19 | 99 |
| Brazil | 20 | 0 |
| China | 19 | 99 |
| China | 20 | 0 |

→

| country | year |
|---|---|
| Afghan | 1999 |
| Afghan | 2000 |
| Brazil | 1999 |
| Brazil | 2000 |
| China | 1999 |
| China | 2000 |

*unite(table5, century, year, col = "year", sep = "")*

RStudio® is a trademark of RStudio, Inc. • CC BY SA RStudio • info@rstudio.com • 844-448-1212 • rstudio.com • Learn more at tidyverse.org • readr 1.1.0 • tibble 1.2.12 • tidyr 0.6.0 • Updated: 2017-01