Efficient Data Import in R

Michael Otterstatter

BCCDC Biostats Session

Feb 15, 2019

Outline

- Data import tools in tidyverse
- Reading csv files
- Formatting columns
- Dealing with missing values
- Dealing with dates
- Reading other file types

Importing data

- Regardless of the analytic task, we almost always begin by taking data from an existing source (file, webpage, database) and importing it into an object in R, SAS, etc. for further manipulation and analysis

Importing data with `readr`

- `readr` can import from 7 different file formats:
  - `read_csv()`: comma separated (csv) files
  - `read_tsv()`: tab separated files
  - `read_delim()`: general delimited files
  - `read_fwf()`: fixed width files
  - `read_table()`: tabular files with columns separated by white-space.
  - `read_log()`: web log files

All have similar form and syntax, so we will focus on `read_csv` the most commonly used example

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)

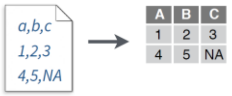
String vector to file, one element per line
write_lines(x,path, na = "NA", append = FALSE)

Object to RDS file
write_rds(x, path, compress = c("none", "gz", "bz2", "xz"), ...)

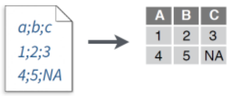
Tab delimited files
write_tsv(x, path, na = "NA", append = FALSE, col_names = !append)

Read Tabular Data - These functions share the common arguments:

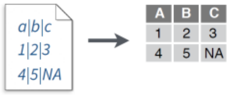
```
read_*(file, col_names = TRUE, col_types = NULL, locale = default_locale(), na = c("", "NA"),
  quoted_na = TRUE, comment = "", trim_ws = TRUE, skip = 0, n_max = Inf, guess_max = min(1000,
  n_max), progress = interactive())
```



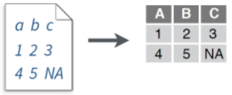
Comma Delimited Files
read_csv("file.csv")
To make file.csv run:
write_file(x = "a,b,c\n1,2,3\n4,5,NA", path = "file.csv")



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



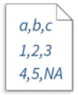
Files with Any Delimiter
read_delim("file.txt", delim = "|")
write_file(x = "a|b|c\n1|2|3\n4|5|NA", path = "file.txt")



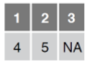
Fixed Width Files
read_fwf("file.fwf", col_positions = c(1, 3, 5))
write_file(x = "a b c\n1 2 3\n4 5 NA", path = "file.fwf")

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")

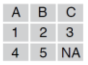
USEFUL ARGUMENTS



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

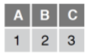


Skip lines
read_csv(f, **skip = 1**)

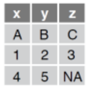


A B C
1 2 3
4 5 NA

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

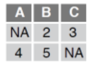


Read in a subset
read_csv(f, **n_max = 1**)



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 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())

Read Apache style log files
read_log(file, col_names = FALSE, col_types = NULL, skip = 0, n_max = -1, 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()**

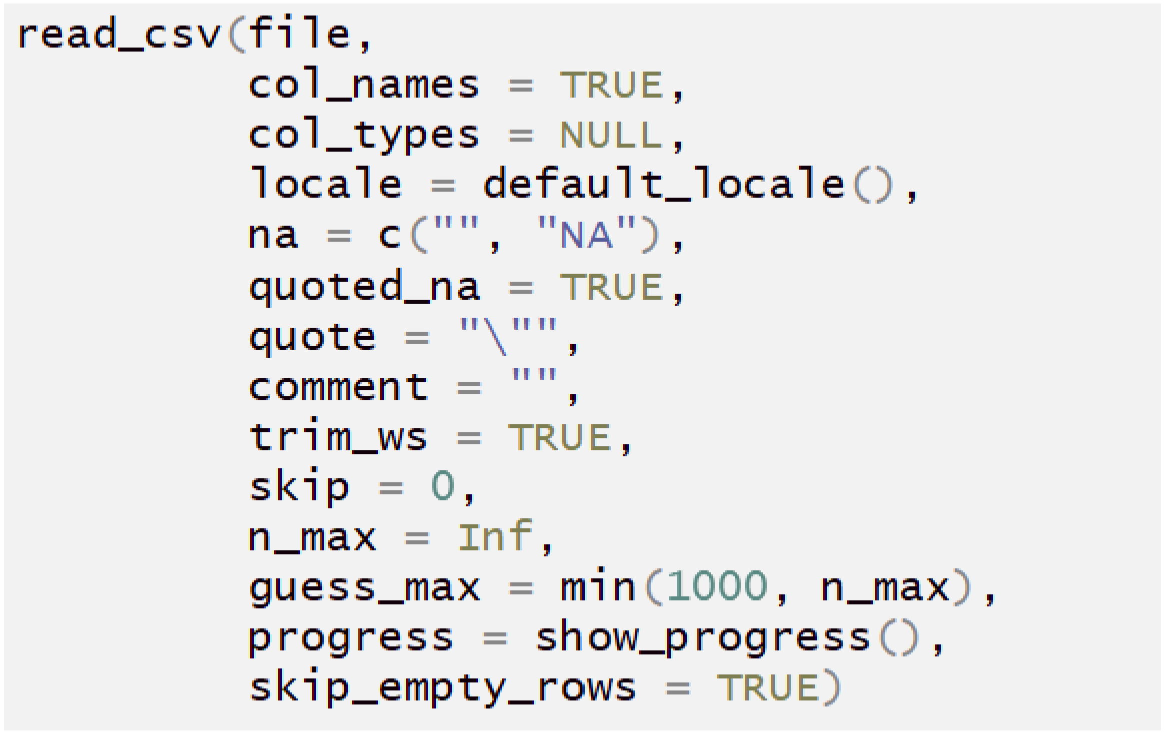
x$A <- parse_number(x$A)

R Studio

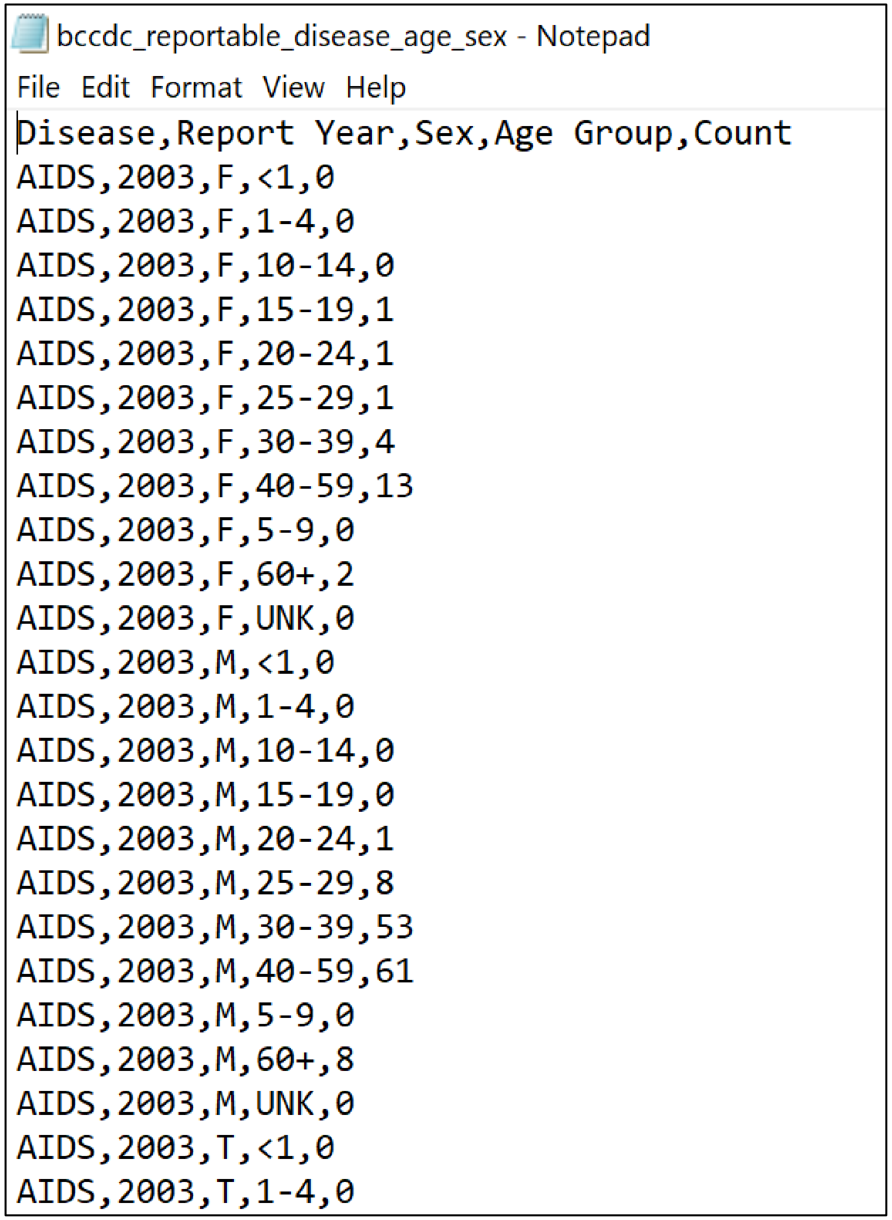
https://www.rstudio.com/resources/cheatsheets/

Introducing `read_csv`

- Many options can be specified in `read_csv` (although usually don’t need to be)



```
read_csv(file,
         col_names = TRUE,
         col_types = NULL,
         locale = default_locale(),
         na = c("", "NA"),
         quoted_na = TRUE,
         quote = "\"",
         comment = "",
         trim_ws = TRUE,
         skip = 0,
         n_max = Inf,
         guess_max = min(1000, n_max),
         progress = show_progress(),
         skip_empty_rows = TRUE)
```


bccdc_reportable_disease_age_sex - Notepad

File Edit Format View Help

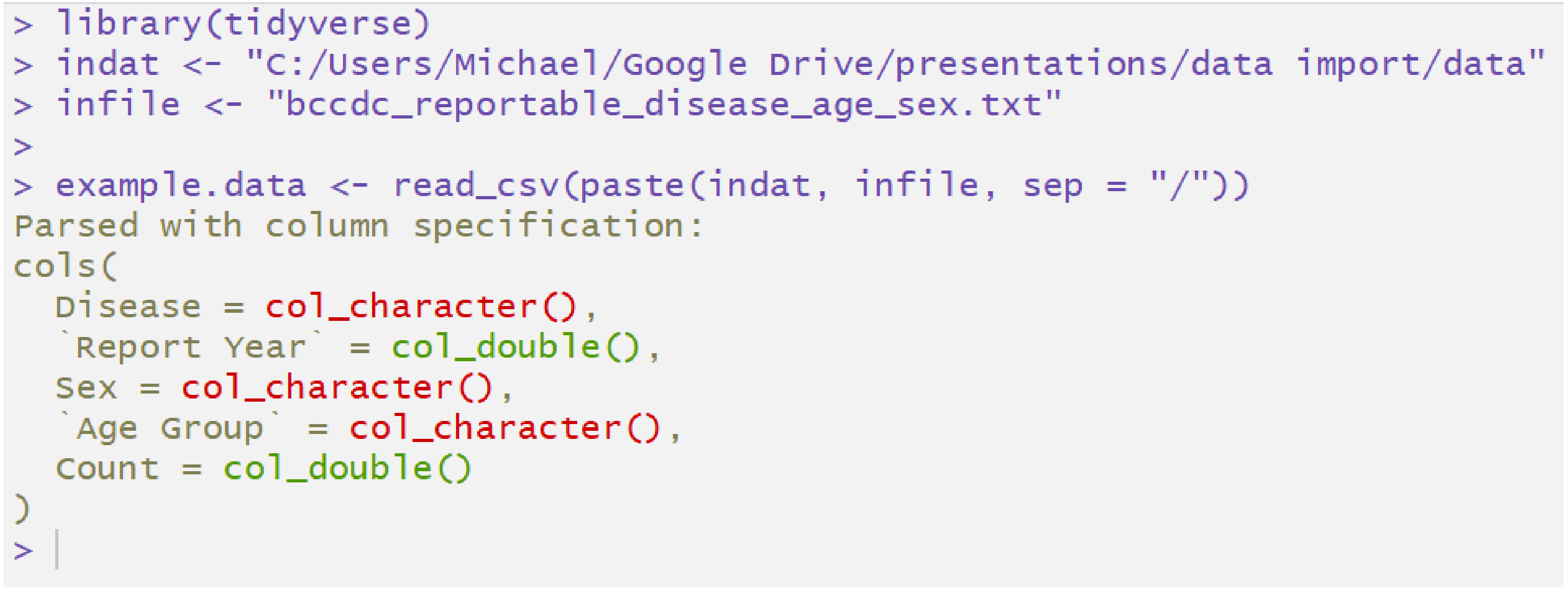
```
Disease,Report Year,Sex,Age Group,Count
AIDS,2003,F,<1,0
AIDS,2003,F,1-4,0
AIDS,2003,F,10-14,0
AIDS,2003,F,15-19,1
AIDS,2003,F,20-24,1
AIDS,2003,F,25-29,1
AIDS,2003,F,30-39,4
AIDS,2003,F,40-59,13
AIDS,2003,F,5-9,0
AIDS,2003,F,60+,2
AIDS,2003,F,UNK,0
AIDS,2003,M,<1,0
AIDS,2003,M,1-4,0
AIDS,2003,M,10-14,0
AIDS,2003,M,15-19,0
AIDS,2003,M,20-24,1
AIDS,2003,M,25-29,8
AIDS,2003,M,30-39,53
AIDS,2003,M,40-59,61
AIDS,2003,M,5-9,0
AIDS,2003,M,60+,8
AIDS,2003,M,UNK,0
AIDS,2003,T,<1,0
AIDS,2003,T,1-4,0
```

Let’s use the BCCDC reportable disease dashboard data by age and sex as an example

http://www.bccdc.ca//health-professionals/data-reports/reportable-diseases-data-dashboard

Importing data with `read_csv`

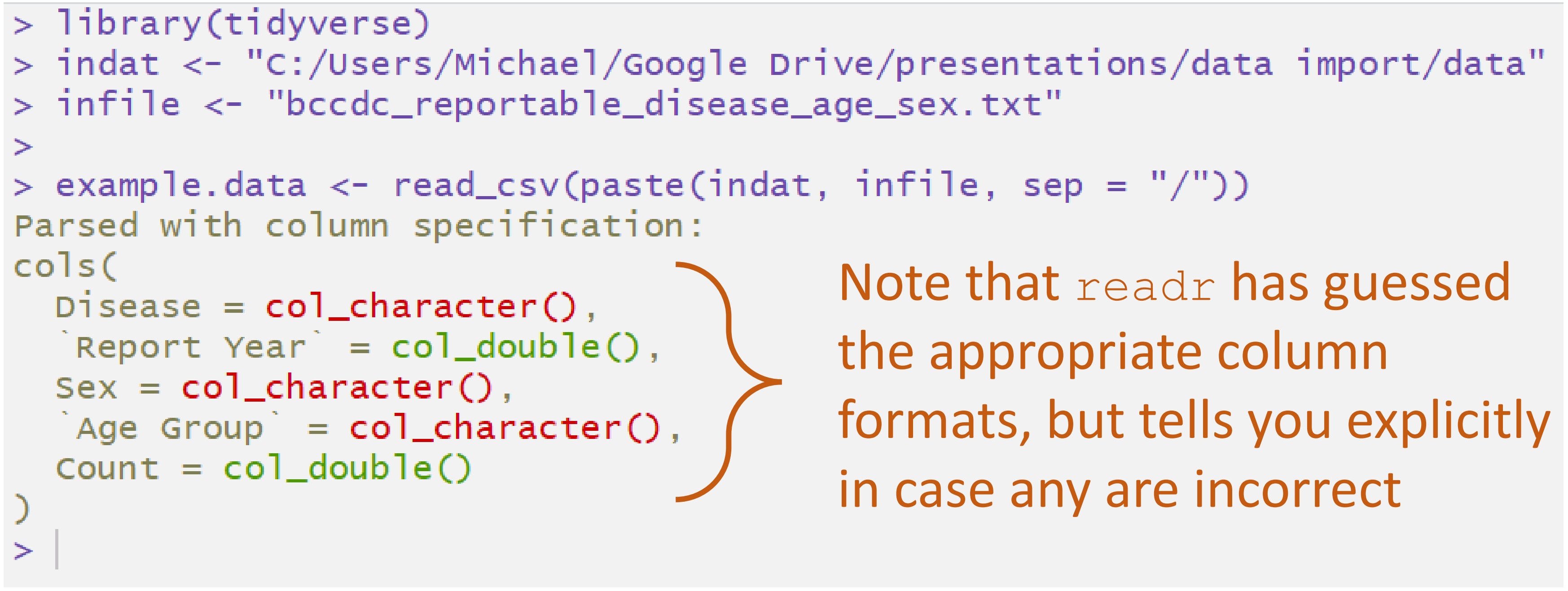
- Most of the time, you only need to specify the file location and `read_csv` will helpfully do the rest



```
> library(tidyverse)
> indat <- "C:/Users/Michael/Google Drive/presentations/data import/data"
> infile <- "bccdc_reportable_disease_age_sex.txt"
>
> example.data <- read_csv(paste(indat, infile, sep = "/"))
Parsed with column specification:
cols(
  Disease = col_character(),
  `Report Year` = col_double(),
  Sex = col_character(),
  `Age Group` = col_character(),
  Count = col_double()
)
> 
```

Importing data with `read_csv`

- Most of the time, you only need to specify the file location and `read_csv` will helpfully do the rest

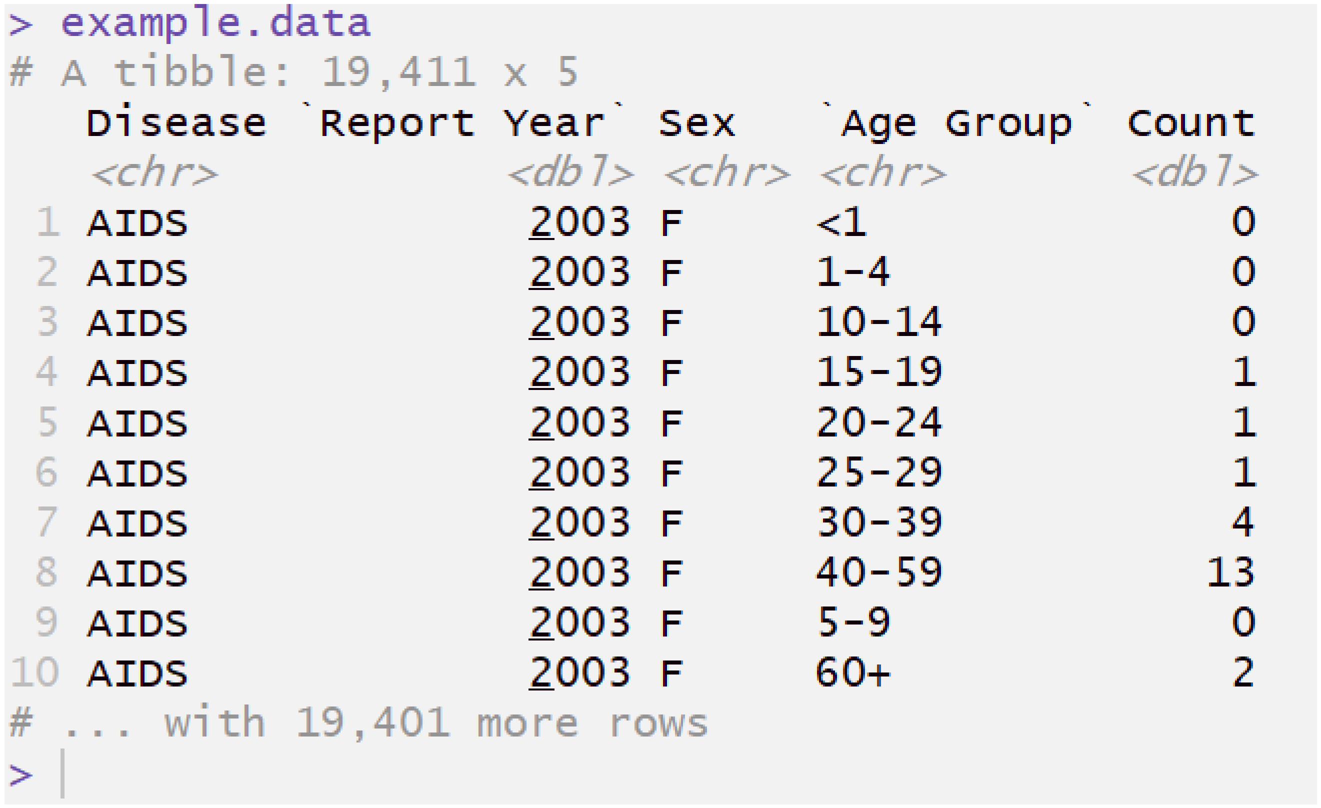


```
> library(tidyverse)
> indat <- "C:/Users/Michael/Google Drive/presentations/data import/data"
> infile <- "bccdc_reportable_disease_age_sex.txt"
>
> example.data <- read_csv(paste(indat, infile, sep = "/"))
Parsed with column specification:
cols(
  Disease = col_character(),
  `Report Year` = col_double(),
  Sex = col_character(),
  `Age Group` = col_character(),
  Count = col_double()
)
>
```

Note that `readr` has guessed the appropriate column formats, but tells you explicitly in case any are incorrect

Importing data with `read_csv`

- What did `read_csv` do?



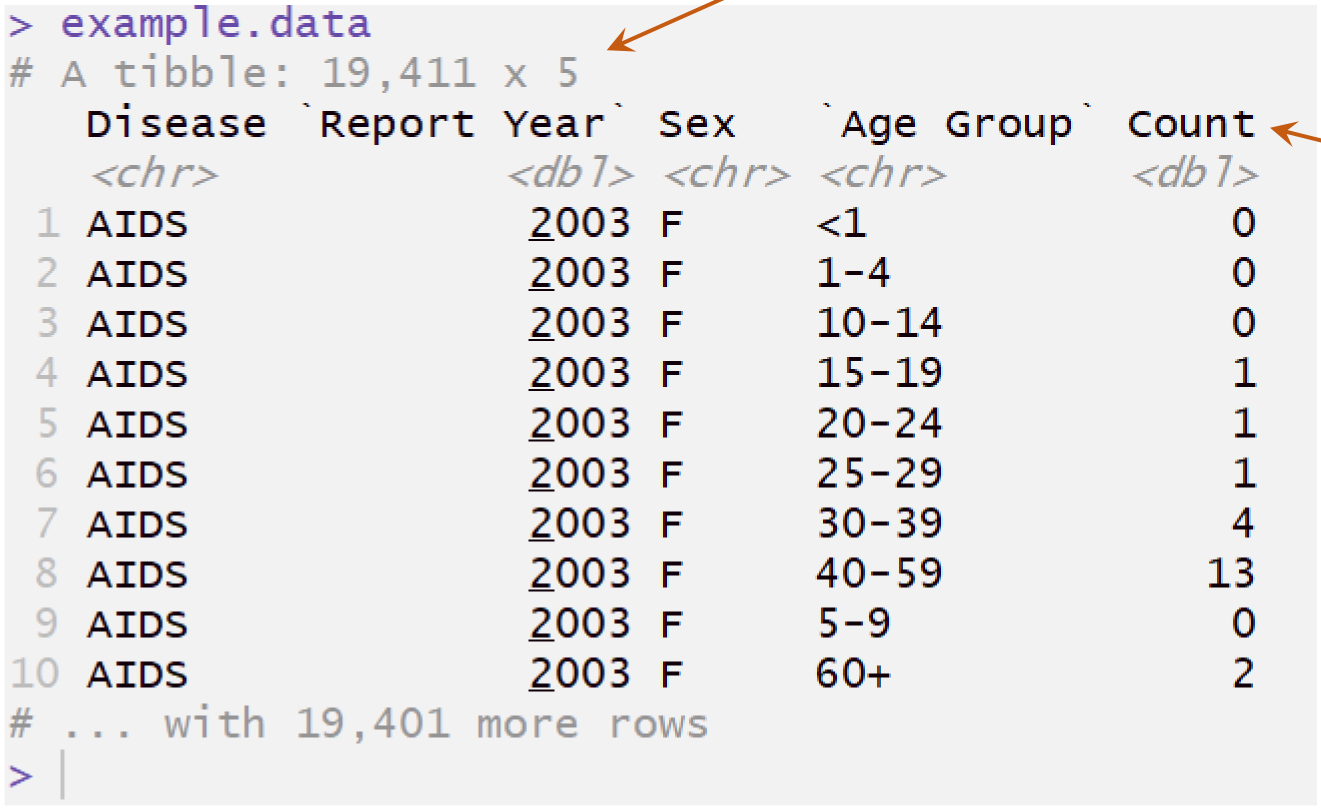
```
> example.data
# A tibble: 19,411 x 5
   Disease `Report Year` Sex   `Age Group` Count
   <chr>           <dbl> <chr> <chr>       <dbl>
 1 AIDS             2003 F     <1              0
 2 AIDS             2003 F     1-4             0
 3 AIDS             2003 F     10-14           0
 4 AIDS             2003 F     15-19           1
 5 AIDS             2003 F     20-24           1
 6 AIDS             2003 F     25-29           1
 7 AIDS             2003 F     30-39           4
 8 AIDS             2003 F     40-59          13
 9 AIDS             2003 F     5-9             0
10 AIDS             2003 F     60+             2
# ... with 19,401 more rows
> 
```

Importing data with `read_csv`

- What did `read_csv` do?

Imports data as a 'tibble' (user friendly version of a data frame

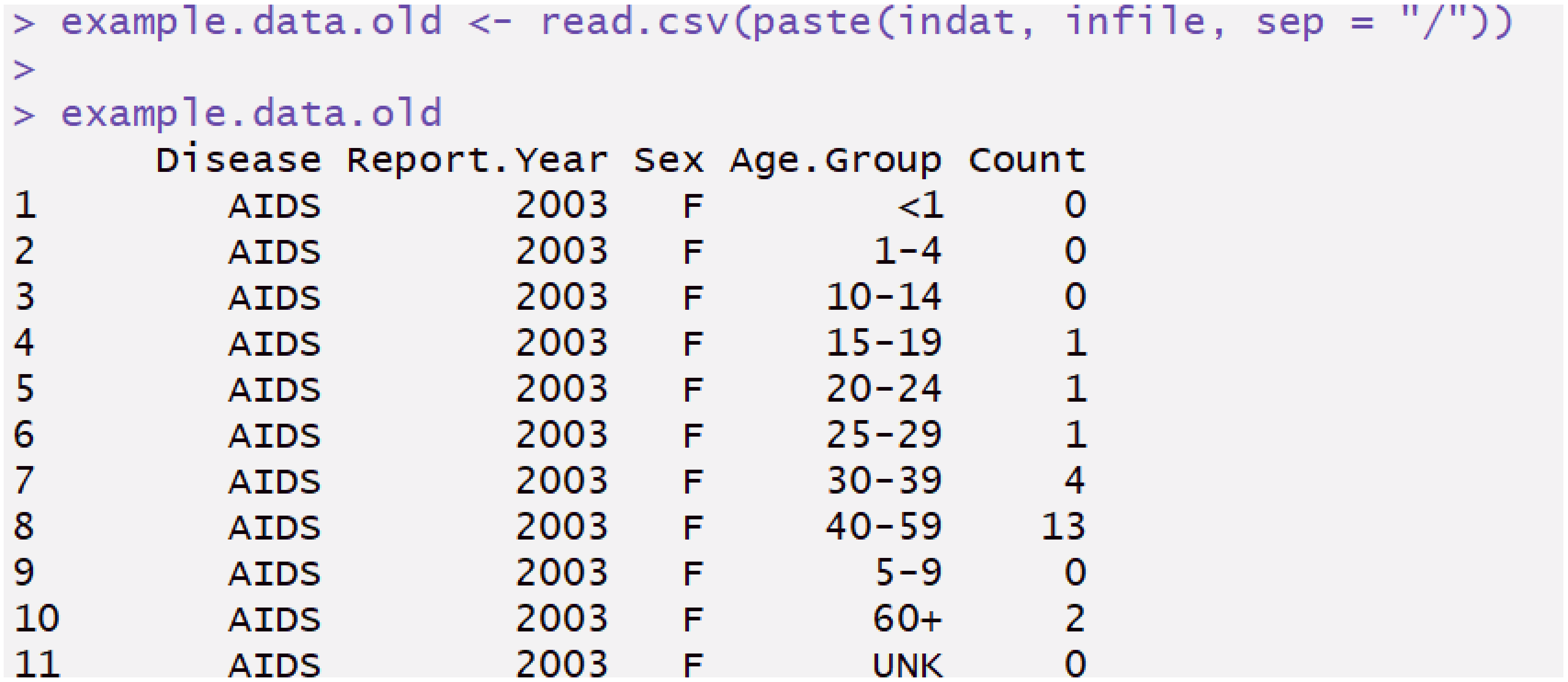
Preserves column names



```
> example.data
# A tibble: 19,411 x 5
   Disease `Report Year` Sex   `Age Group` Count
   <chr>           <dbl> <chr> <chr>       <dbl>
 1 AIDS             2003 F     <1              0
 2 AIDS             2003 F     1-4             0
 3 AIDS             2003 F     10-14           0
 4 AIDS             2003 F     15-19           1
 5 AIDS             2003 F     20-24           1
 6 AIDS             2003 F     25-29           1
 7 AIDS             2003 F     30-39           4
 8 AIDS             2003 F     40-59          13
 9 AIDS             2003 F     5-9             0
10 AIDS             2003 F     60+             2
# ... with 19,401 more rows
> 
```

What about `read.csv`?

- Base R uses data import function `read.csv`

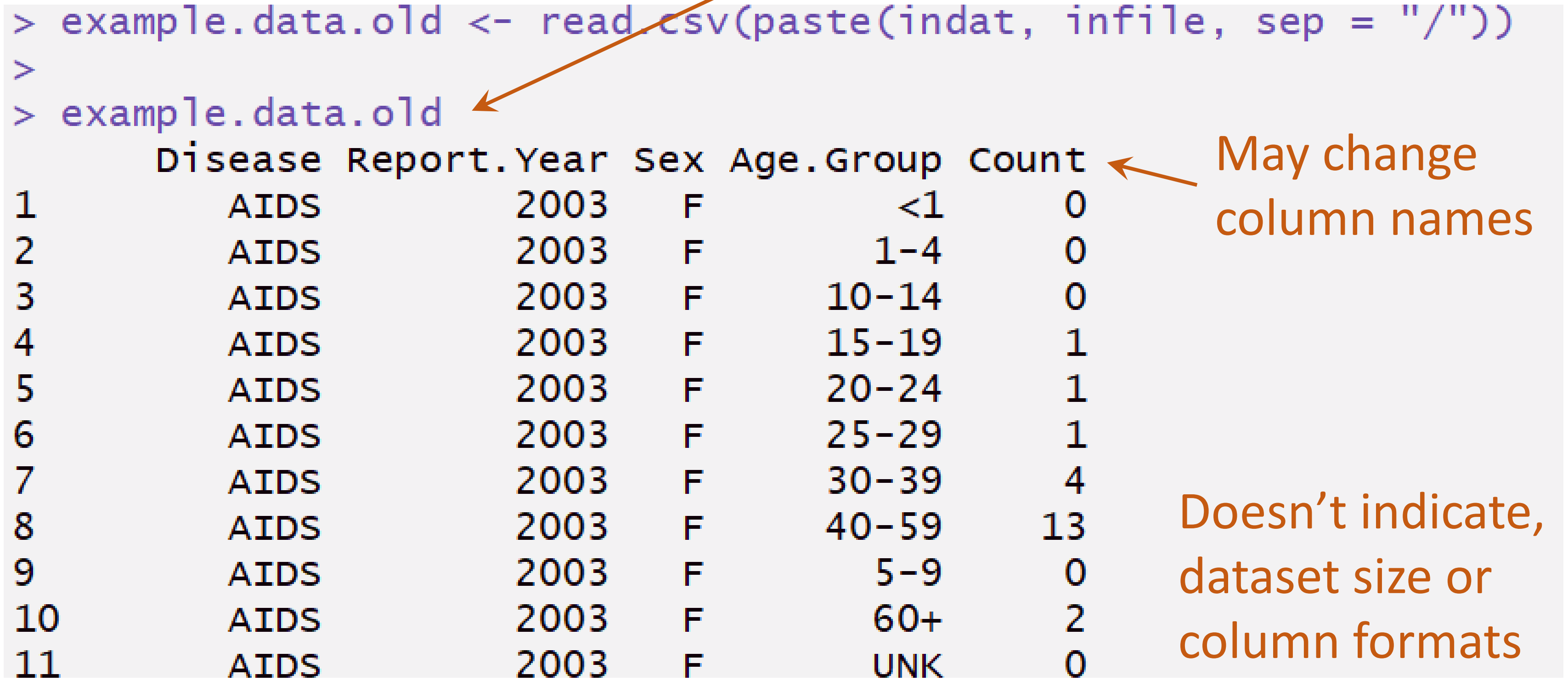


```
> example.data.old <- read.csv(paste(indat, infile, sep = "/"))
>
> example.data.old
      Disease Report.Year Sex Age.Group Count
1        AIDS        2003   F        <1     0
2        AIDS        2003   F       1-4     0
3        AIDS        2003   F     10-14     0
4        AIDS        2003   F     15-19     1
5        AIDS        2003   F     20-24     1
6        AIDS        2003   F     25-29     1
7        AIDS        2003   F     30-39     4
8        AIDS        2003   F     40-59    13
9        AIDS        2003   F       5-9     0
10       AIDS        2003   F       60+     2
11       AIDS        2003   F       UNK     0
```

What about `read.csv`?

- What did `read.csv` do?

Imports data as a 'data frame'



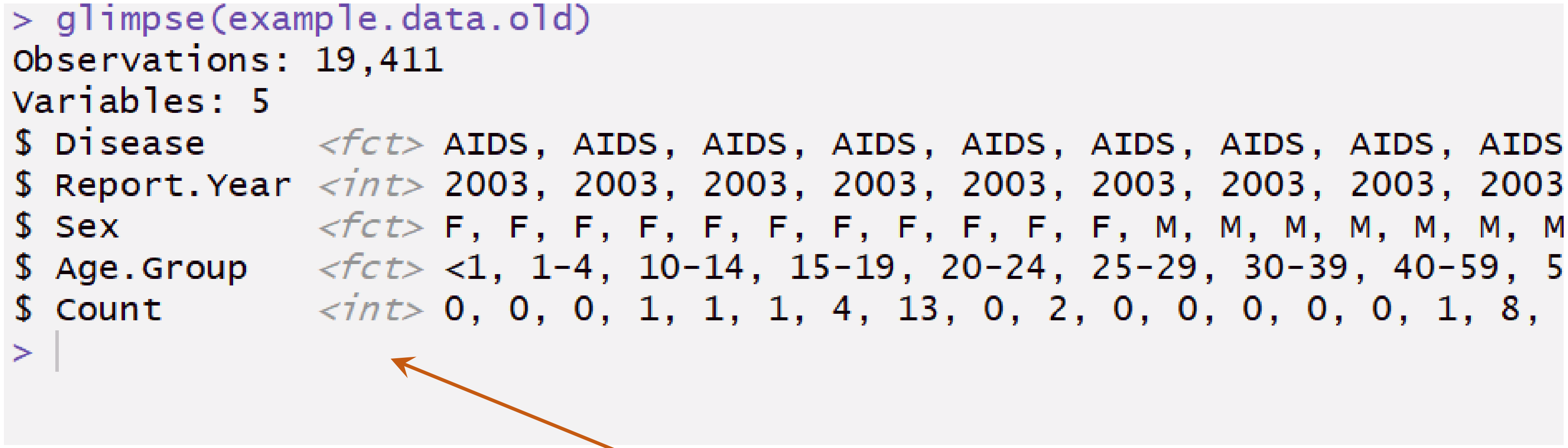
```
> example.data.old <- read.csv(paste(indat, infile, sep = "/"))
>
> example.data.old
     Disease Report.Year Sex Age.Group Count
1       AIDS        2003   F        <1     0
2       AIDS        2003   F       1-4     0
3       AIDS        2003   F     10-14     0
4       AIDS        2003   F     15-19     1
5       AIDS        2003   F     20-24     1
6       AIDS        2003   F     25-29     1
7       AIDS        2003   F     30-39     4
8       AIDS        2003   F     40-59    13
9       AIDS        2003   F       5-9     0
10      AIDS        2003   F       60+     2
11      AIDS        2003   F       UNK     0
```

May change column names

Doesn't indicate, dataset size or column formats

What about `read.csv`?

- What did `read.csv` do?



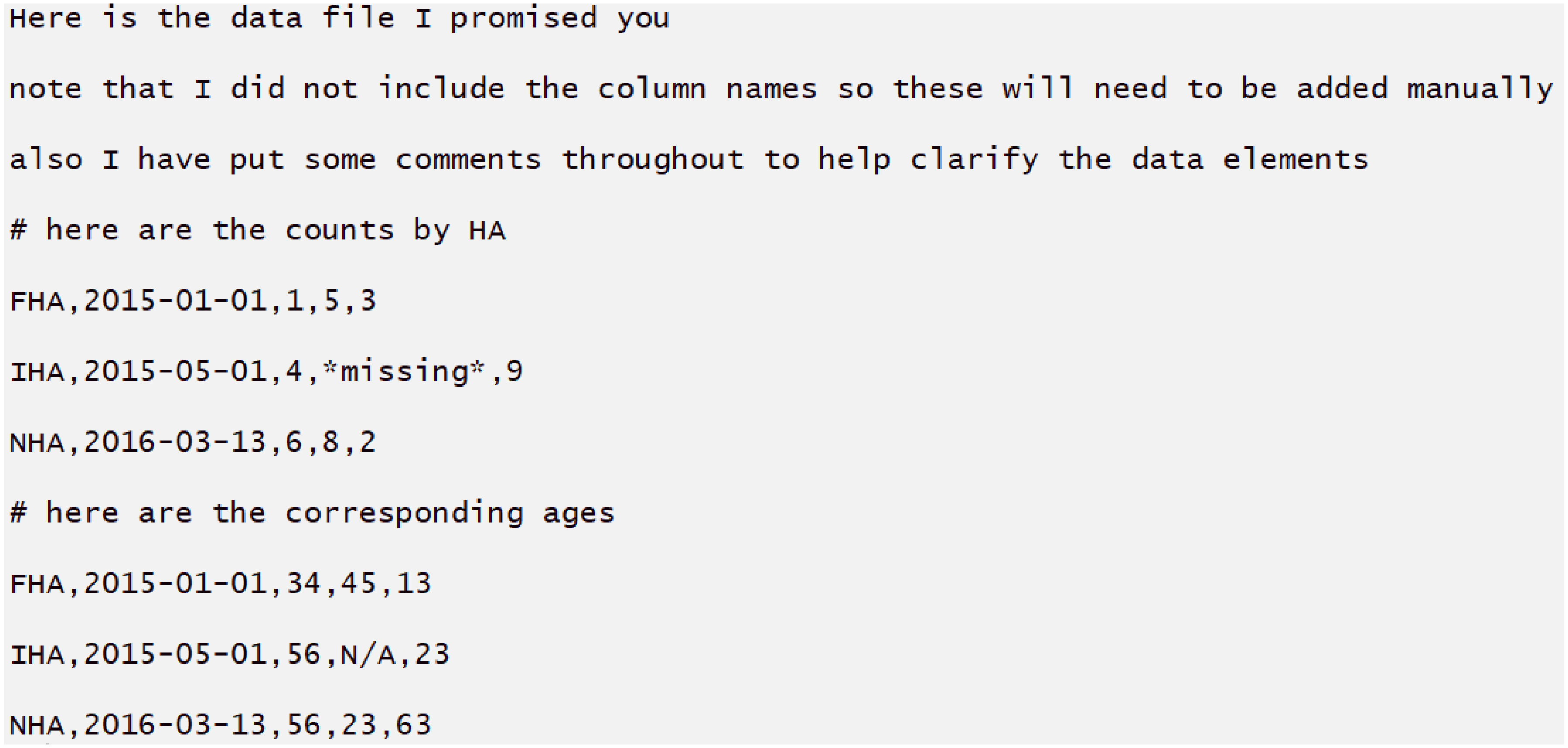
```
> glimpse(example.data.old)
Observations: 19,411
Variables: 5
$ Disease     <fct> AIDS, AIDS, AIDS, AIDS, AIDS, AIDS, AIDS, AIDS, AIDS
$ Report.Year <int> 2003, 2003, 2003, 2003, 2003, 2003, 2003, 2003, 2003
$ Sex         <fct> F, F, F, F, F, F, F, F, F, F, F, M, M, M, M, M, M, M
$ Age.Group   <fct> <1, 1-4, 10-14, 15-19, 20-24, 25-29, 30-39, 40-59, 5
$ Count       <int> 0, 0, 0, 1, 1, 1, 4, 13, 0, 2, 0, 0, 0, 0, 0, 1, 8,
> 
```

Imported character variables as factors (which can cause problems later and should generally be avoided)

What about `read.csv`?

- Compared to the base R function `read.csv`, `read_csv`
  - retains original column names
  - is much faster (up to 10x)
  - does not convert character strings to factors by default
  - automatically parses common date/time formats
  - has a progress bar if loading is slow

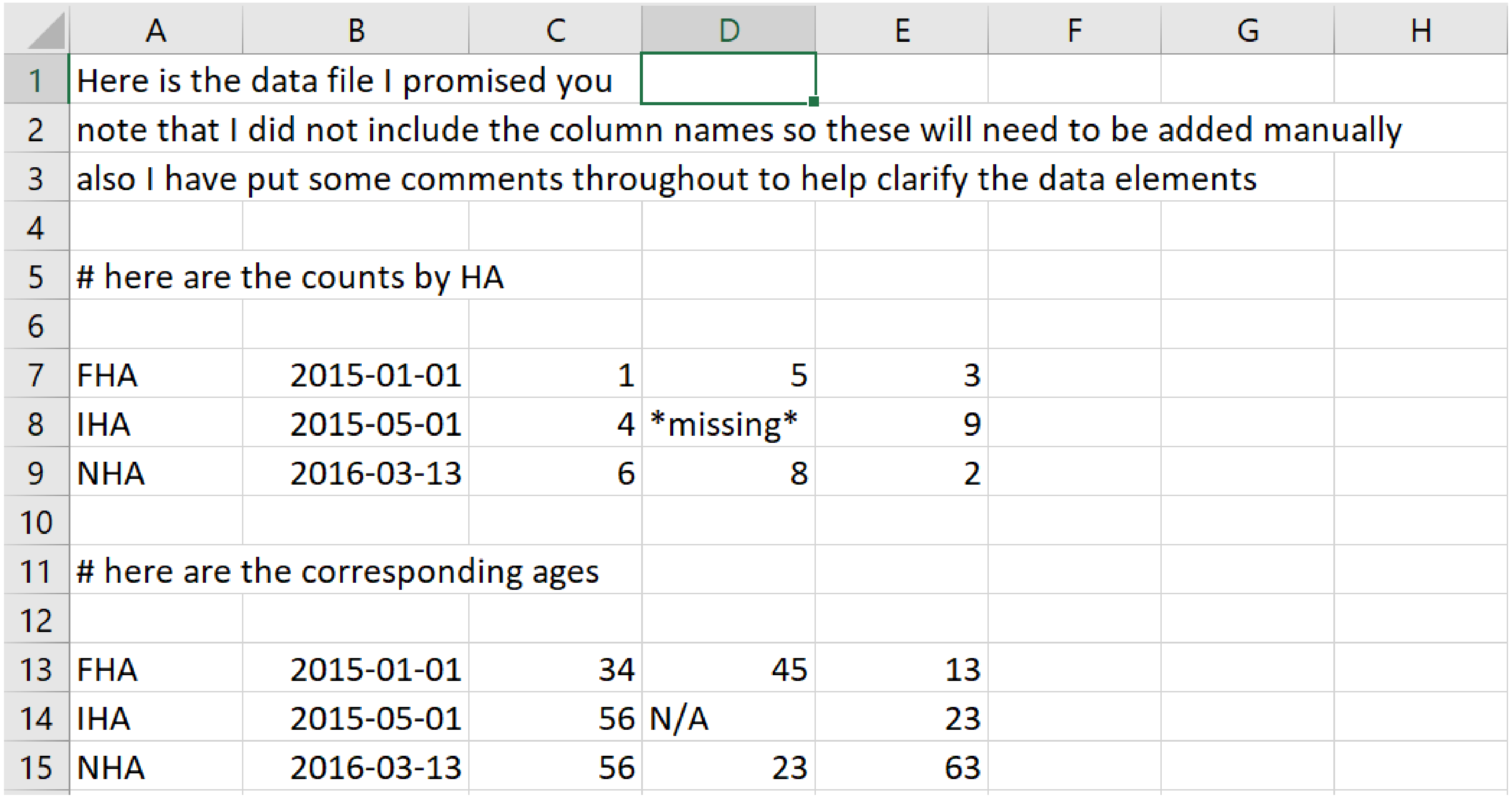
A messier example



```
Here is the data file I promised you

note that I did not include the column names so these will need to be added manually

also I have put some comments throughout to help clarify the data elements

# here are the counts by HA

FHA,2015-01-01,1,5,3

IHA,2015-05-01,4,*missing*,9

NHA,2016-03-13,6,8,2

# here are the corresponding ages

FHA,2015-01-01,34,45,13

IHA,2015-05-01,56,N/A,23

NHA,2016-03-13,56,23,63
```

What if we receive the data as a text file or spreadsheet with unneeded text and comments interspersed with the data?

A messier example

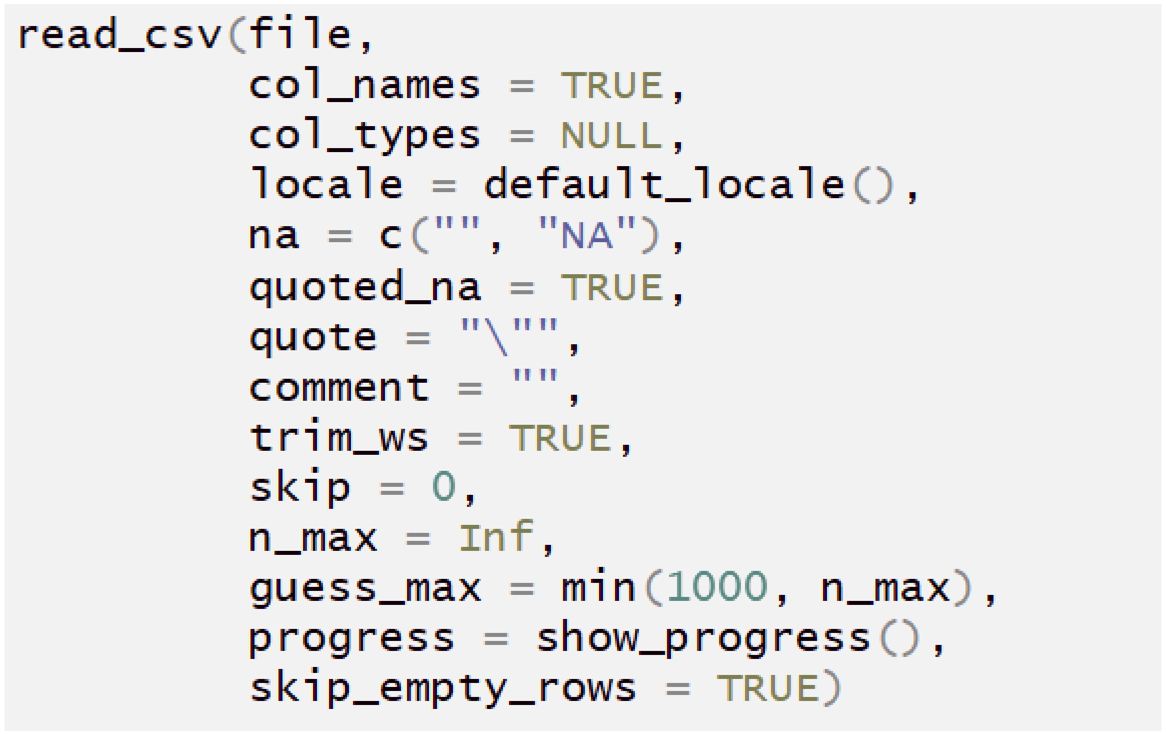


	A	B	C	D	E	F	G	H
1	Here is the data file I promised you							
2	note that I did not include the column names so these will need to be added manually							
3	also I have put some comments throughout to help clarify the data elements							
4								
5	# here are the counts by HA							
6								
7	FHA	2015-01-01	1	5	3			
8	IHA	2015-05-01	4	*missing*	9			
9	NHA	2016-03-13	6	8	2			
10								
11	# here are the corresponding ages							
12								
13	FHA	2015-01-01	34	45	13			
14	IHA	2015-05-01	56	N/A	23			
15	NHA	2016-03-13	56	23	63			

What if we receive the data as a text file or spreadsheet with unneeded text and comments interspersed with the data?

A messier example

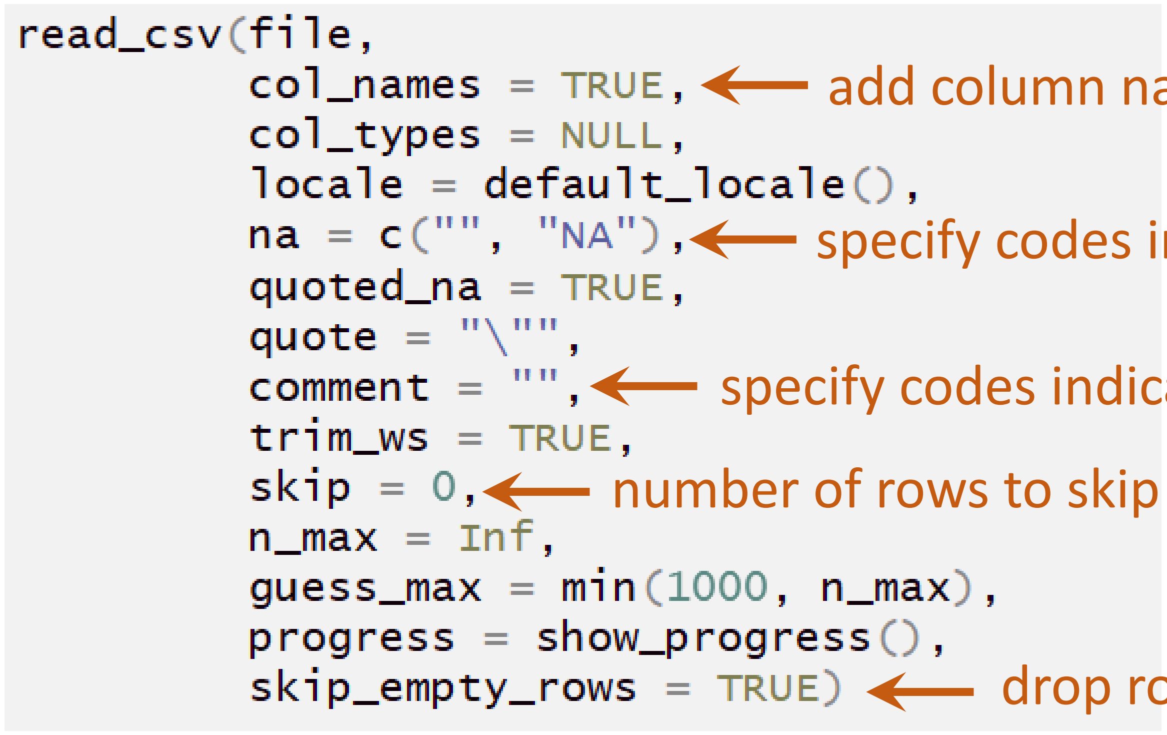
- We can make use of optional settings in `read_csv`



```
read_csv(file,
         col_names = TRUE,
         col_types = NULL,
         locale = default_locale(),
         na = c("", "NA"),
         quoted_na = TRUE,
         quote = "\"",
         comment = "",
         trim_ws = TRUE,
         skip = 0,
         n_max = Inf,
         guess_max = min(1000, n_max),
         progress = show_progress(),
         skip_empty_rows = TRUE)
```

A messier example

- We can make use of optional settings in `read_csv`



```
read_csv(file,
         col_names = TRUE,
         col_types = NULL,
         locale = default_locale(),
         na = c("", "NA"),
         quoted_na = TRUE,
         quote = "\"",
         comment = "",
         trim_ws = TRUE,
         skip = 0,
         n_max = Inf,
         guess_max = min(1000, n_max),
         progress = show_progress(),
         skip_empty_rows = TRUE)
```

add column names

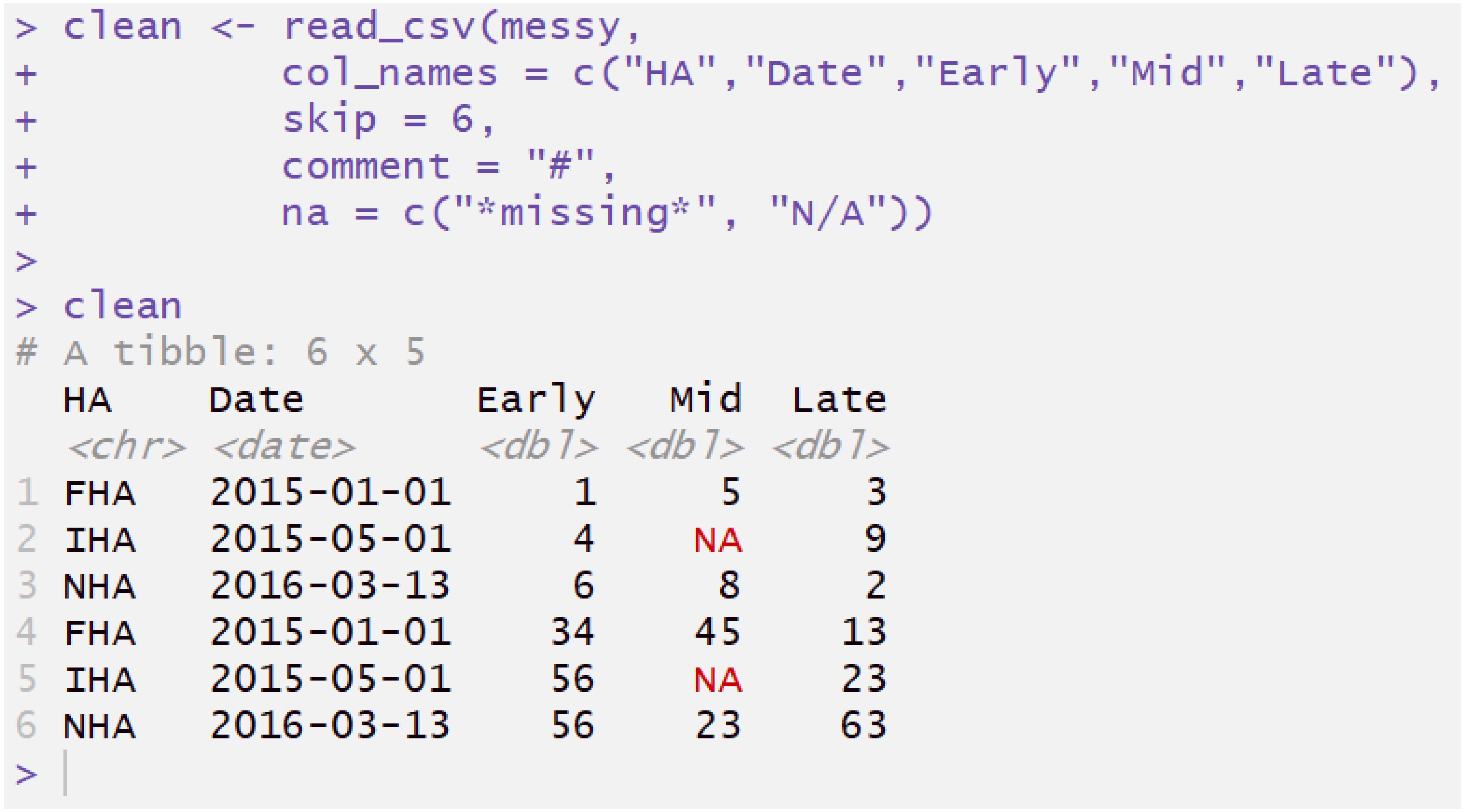
specify codes indicating missing data

specify codes indicating comments

number of rows to skip at beginning

drop rows with no data

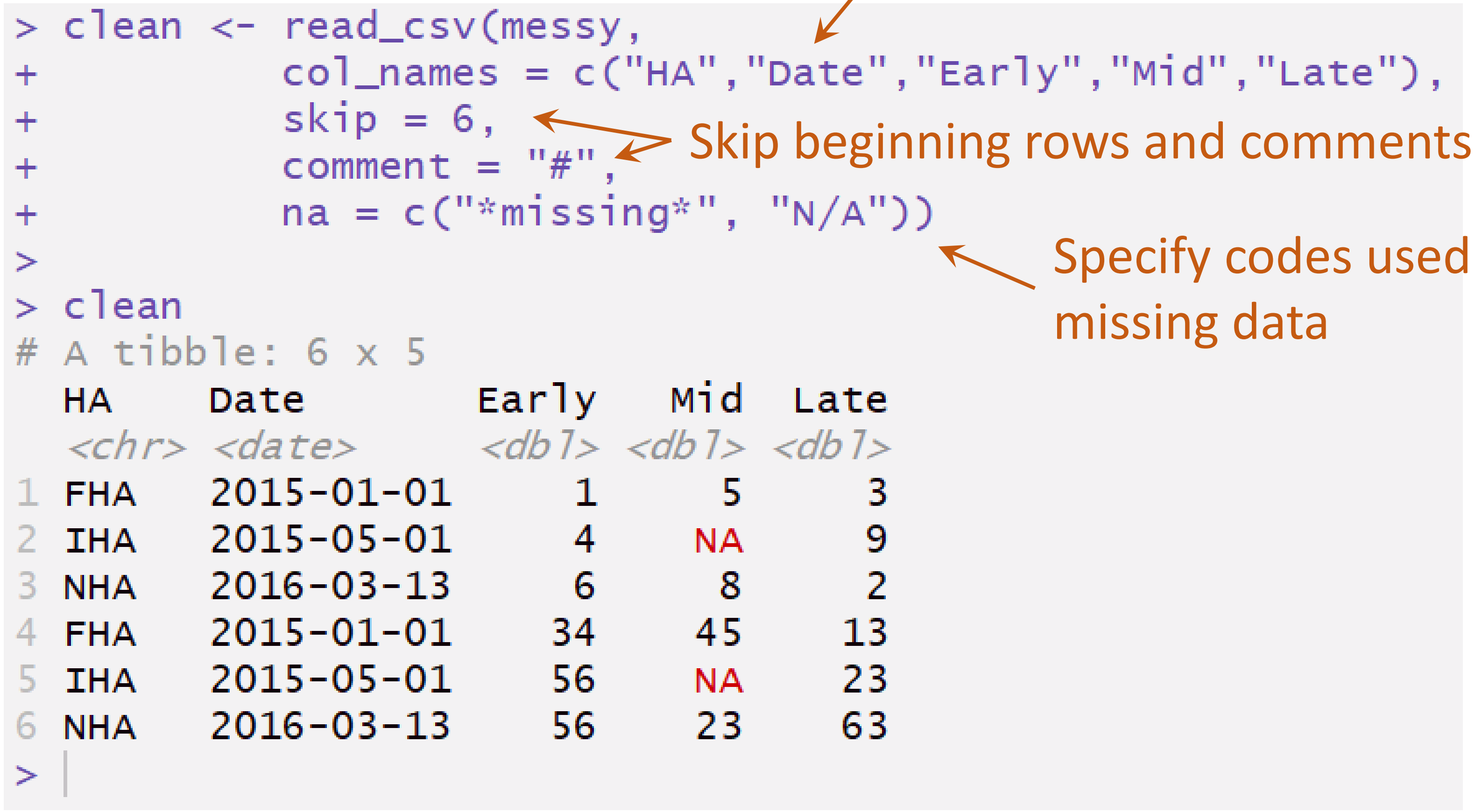
A messier example



```
> clean <- read_csv(messy,
+               col_names = c("HA","Date","Early","Mid","Late"),
+               skip = 6,
+               comment = "#",
+               na = c("*missing*", "N/A"))
>
> clean
# A tibble: 6 x 5
  HA    Date        Early   Mid  Late
  <chr> <date>      <dbl> <dbl> <dbl>
1 FHA   2015-01-01      1     5     3
2 IHA   2015-05-01      4    NA     9
3 NHA   2016-03-13      6     8     2
4 FHA   2015-01-01     34    45    13
5 IHA   2015-05-01     56    NA    23
6 NHA   2016-03-13     56    23    63
> 
```

A messier example

Specify column names



```
> clean <- read_csv(messy,
+                   col_names = c("HA","Date","Early","Mid","Late"),
+                   skip = 6,
+                   comment = "#",
+                   na = c("*missing*", "N/A"))
>
> clean
# A tibble: 6 x 5
  HA    Date       Early   Mid  Late
  <chr> <date>     <dbl> <dbl> <dbl>
1 FHA   2015-01-01     1     5     3
2 IHA   2015-05-01     4    NA     9
3 NHA   2016-03-13     6     8     2
4 FHA   2015-01-01    34    45    13
5 IHA   2015-05-01    56    NA    23
6 NHA   2016-03-13    56    23    63
> 
```

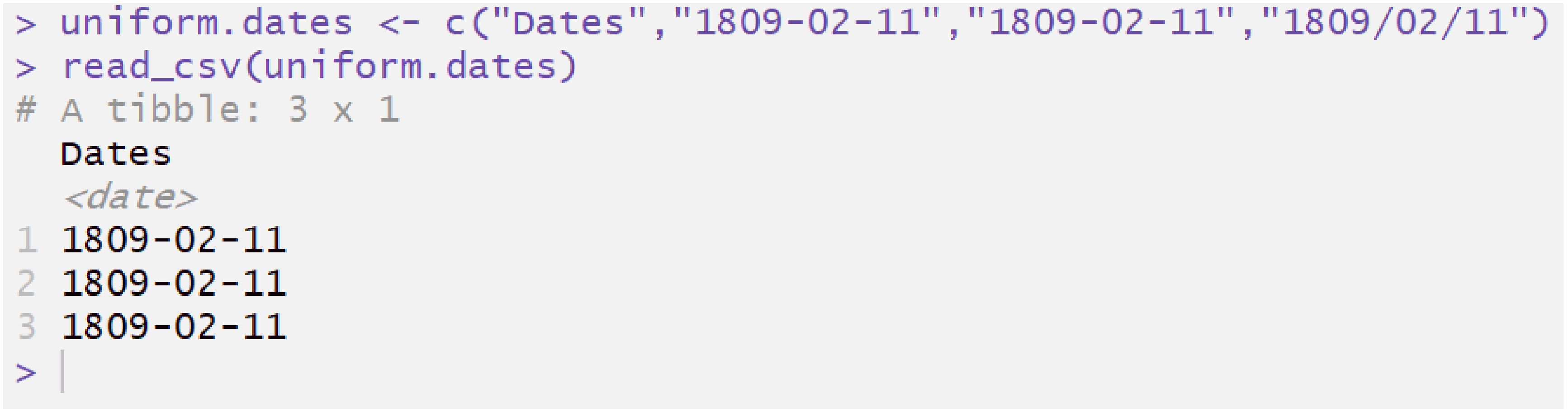
Skip beginning rows and comments

Specify codes used for missing data

Also, by default `read_csv` automatically skipped empty rows

Dealing with dates

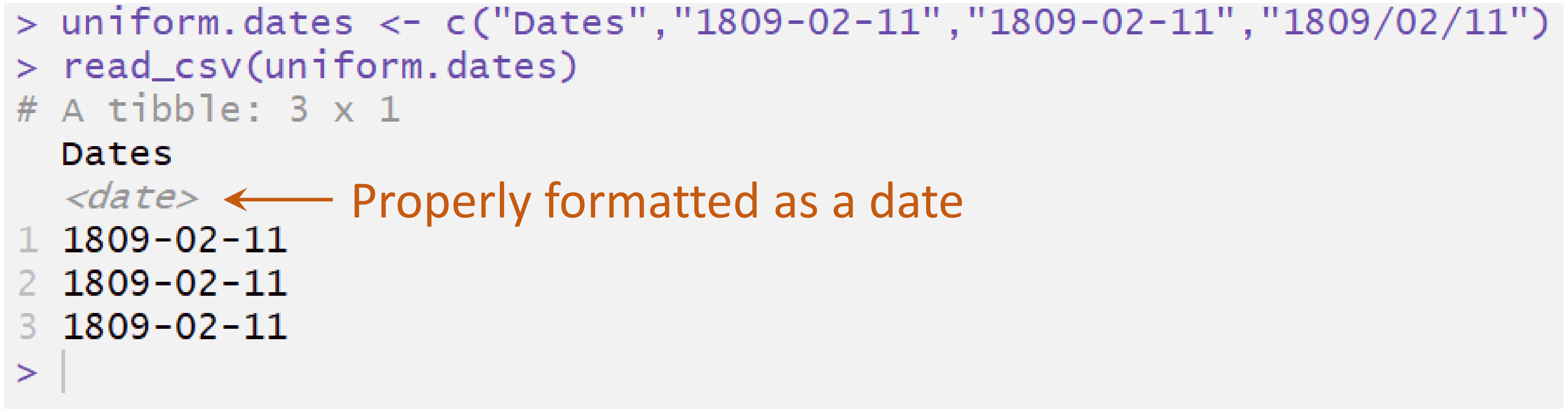
- `read_csv` recognizes dates that are uniformly presented (e.g., all in the form year, month, day)



```
> uniform.dates <- c("Dates","1809-02-11","1809-02-11","1809/02/11")
> read_csv(uniform.dates)
# A tibble: 3 x 1
  Dates
  <date>
1 1809-02-11
2 1809-02-11
3 1809-02-11
> 
```

Dealing with dates

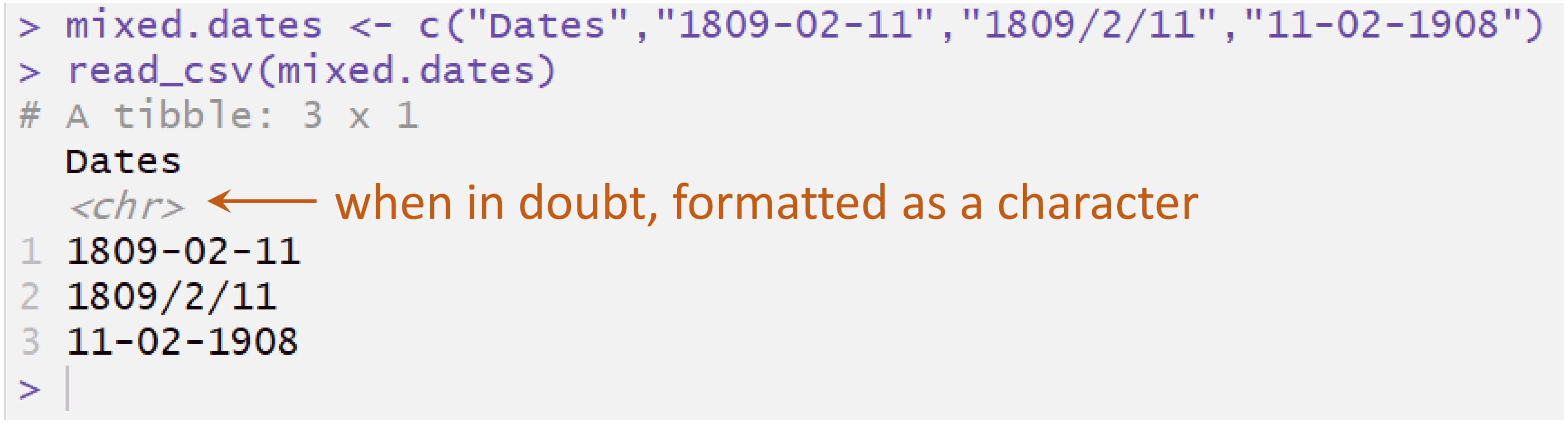
- `read_csv` recognizes dates that are uniformly presented (e.g., all in the form year, month, day)



```
> uniform.dates <- c("Dates","1809-02-11","1809-02-11","1809/02/11")
> read_csv(uniform.dates)
# A tibble: 3 x 1
  Dates
  <date>
1 1809-02-11
2 1809-02-11
3 1809-02-11
>
```

Dealing with dates

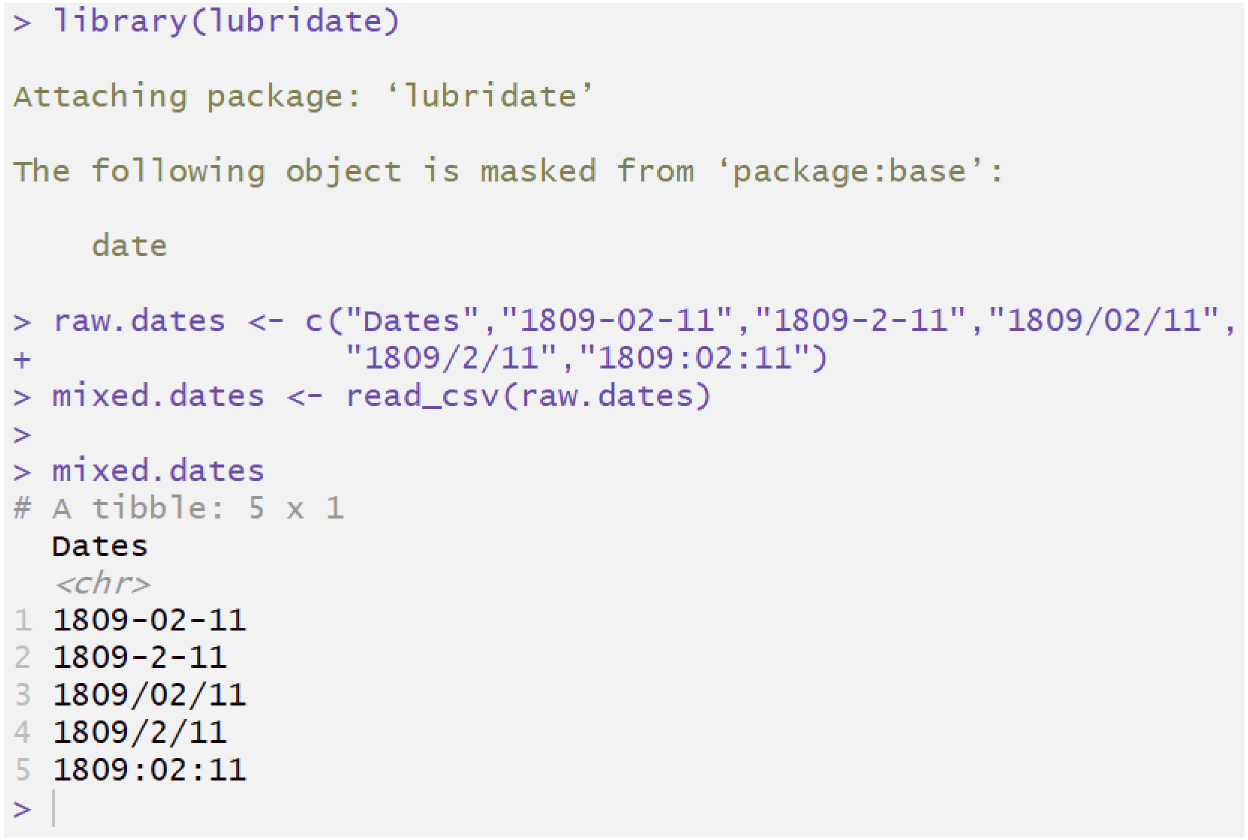
- When dates are not uniformly presented, `read_csv` simply formats as character (i.e., preserves the original data) and let's you do the clean-up



```
> mixed.dates <- c("Dates","1809-02-11","1809/2/11","11-02-1908")
> read_csv(mixed.dates)
# A tibble: 3 x 1
  Dates
  <chr>
1 1809-02-11
2 1809/2/11
3 11-02-1908
> 
```

Dealing with dates

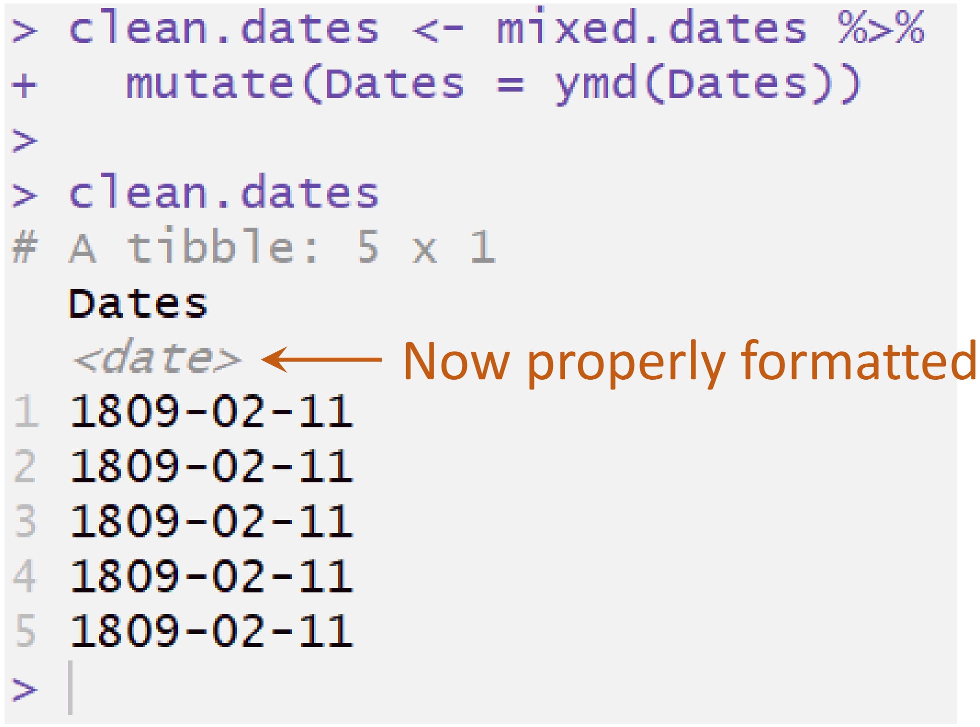
- Fortunately, with tidy data tools like the `lubridate` package, working with dates is easy



```
> library(lubridate)

Attaching package: ‘lubridate’

The following object is masked from ‘package:base’:

    date

> raw.dates <- c("Dates","1809-02-11","1809-2-11","1809/02/11",
+                   "1809/2/11","1809:02:11")
> mixed.dates <- read_csv(raw.dates)
>
> mixed.dates
# A tibble: 5 x 1
  Dates
  <chr>
1 1809-02-11
2 1809-2-11
3 1809/02/11
4 1809/2/11
5 1809:02:11
> |
```

Dealing with dates

- We only need specify the ordering of the date components (e.g., `ymd` indicates year, month, day order), `lubridate` does the rest



```
> clean.dates <- mixed.dates %>%
+   mutate(Dates = ymd(Dates))
>
> clean.dates
# A tibble: 5 x 1
  Dates
  <date>
1 1809-02-11
2 1809-02-11
3 1809-02-11
4 1809-02-11
5 1809-02-11
> 
```

Now properly formatted as a date