Parsers

David Gerard

2019-02-15

Learning Objectives

- Change character vectors into other types using parsers.
- Parsers and reader.
- Chapter 11 of RDS

Motivation

- Suppose you have the following data frame

```
suppressPackageStartupMessages(library(tidyverse))
dfdat <- tribble(
  ~date,        ~time,      ~number, ~factor, ~logical,
  ##----------  ----------  -------  -------  --------
  "12-01-1988", "10:10:02", "2",     "A",     "TRUE",
  "11-12-1987", "11:10:57", "4",     "A",     "TRUE",
  "02-03-1989", "10:10:25", "6",     "B",     "FALSE",
  "06-03-1982", "22:10:55", "2",     "B",     "TRUE",
  "09-21-1981", "10:10:02", "1",     "A",     "FALSE"
  )
dfdat
```

```
## # A tibble: 5 x 5
##   date       time     number factor logical
##   <chr>      <chr>    <chr>  <chr>  <chr>
## 1 12-01-1988 10:10:02 2      A      TRUE
## 2 11-12-1987 11:10:57 4      A      TRUE
## 3 02-03-1989 10:10:25 6      B      FALSE
## 4 06-03-1982 22:10:55 2      B      TRUE
## 5 09-21-1981 10:10:02 1      A      FALSE
```

- How do we convert the characters to the types we want? Parse!

Parsing dates and times

Parsing dates with `parse_date()` and `parse_date_time()`.

- `parse_date()` and `parse_datetime()` are very similar, but internally count the time from 1970-01-01 in terms of either days or seconds.

```
parse_date("2018-01-02")
```

```
## [1] "2018-01-02"
```

```r
parse_datetime("2018-01-02")
```

```
## [1] "2018-01-02 UTC"
```

- They expect the format `"YYYY-MM-DD"`. If your date is in a different format, you need to use the `format` argument.

```r
## Parsing Failure
parse_date("02/01/2018")
```

```
## Warning: 1 parsing failure.
## row col   expected     actual
##   1  -- date like  02/01/2018

## [1] NA
```

```r
## Parsing Success!
parse_date("02/01/2018", format = "%m/%d/%Y")
```

```
## [1] "2018-02-01"
```

- We added slashes so that R can know how the date is formatted.
- Format options:
  - `%d`: 2-digit representation of day (but can recognize single digits sometimes)
  - `%m`: 2-digit representation of month
  - `%b`: Abbreviation of month ("Jan")
  - `%B`: Full month name ("January")
  - `%y`: 2-digit representation of year
  - `%Y`: 4-digit representation of year
- Another example:

```r
parse_date("January 1, 2018", format = "%B %d, %Y")
```

```
## [1] "2018-01-01"
```

- Our example:

```r
dfdat %>%
  mutate(date = parse_date(date, format = "%m-%d-%Y"))
```

```
## # A tibble: 5 x 5
##   date       time     number factor logical
##   <date>     <chr>    <chr>  <chr>  <chr>
## 1 1988-12-01 10:10:02 2      A      TRUE
## 2 1987-11-12 11:10:57 4      A      TRUE
## 3 1989-02-03 10:10:25 6      B      FALSE
## 4 1982-06-03 22:10:55 2      B      TRUE
## 5 1981-09-21 10:10:02 1      A      FALSE
```

- **Exercise**: Parse the following strings to be dates:

```
"01, January 2018"
"01-January/2000"
"1 Jan 19"
```

Parsing times with `parse_time()`

- `parse_time()` is very similar to `parse_date()` except the format argument.
  - `%H`: Hour in 0-23 format
  - `%I`: Hour in 0-12 format
  - `%p`: am/pm
  - `%M`: minutes
  - `%S`: integer seconds
  - `%OS`: double seconds
  - `%Z`: Time zone (need nuance here)
- Example:

```
dfdat %>%
  mutate(time = parse_time(time, format = "%H:%M:%S"))
```

```
## # A tibble: 5 x 5
##   date       time   number factor logical
##   <chr>      <time> <chr>  <chr>  <chr>
## 1 12-01-1988 10:10  2      A      TRUE
## 2 11-12-1987 11:10  4      A      TRUE
## 3 02-03-1989 10:10  6      B      FALSE
## 4 06-03-1982 22:10  2      B      TRUE
## 5 09-21-1981 10:10  1      A      FALSE
```

- **Exercise**: Parse the following times:

```
"10:40 pm"
"23:40-22"
```

Parsing Numbers

- `parse_double()` and `parse_integer()` expect strict numbers and will fail if there is anything non-number-like.

```
parse_double("2.11")
```

```
## [1] 2.11
```

```
parse_double("$2.11")
```

```
## Warning: 1 parsing failure.
## row col expected actual
##   1  -- a double  $2.11
## [1] NA
## attr(,"problems")
## # A tibble: 1 x 4
##     row   col expected actual
##   <int> <int> <chr>    <chr>
## 1     1    NA a double $2.11
```

```
parse_integer("2")
```

```
## [1] 2
```

```
parse_integer("2%")
```

```
## Warning: 1 parsing failure.
## row col               expected actual
##   1  -- no trailing characters      %
```

```
## [1] NA
## attr(,"problems")
## # A tibble: 1 x 4
##     row   col expected               actual
##   <int> <int> <chr>                  <chr>
## 1     1    NA no trailing characters %
```

- `parse_number()` removes non-numeric characters.

```
parse_number("$2.11")
```

```
## [1] 2.11
```

```
parse_number("2%")
```

```
## [1] 2
```

- You can change the grouping variable from "," to "." with

```
parse_number("2.555,11",
             locale = locale(grouping_mark = ".",
                             decimal_mark = ","))
```

```
## [1] 2555
```

- Example:

```
dfdat %>%
  mutate(number = parse_number(number))
```

```
## # A tibble: 5 x 5
##   date       time     number factor logical
##   <chr>      <chr>     <dbl> <chr>  <chr>
## 1 12-01-1988 10:10:02      2 A      TRUE
## 2 11-12-1987 11:10:57      4 A      TRUE
## 3 02-03-1989 10:10:25      6 B      FALSE
## 4 06-03-1982 22:10:55      2 B      TRUE
## 5 09-21-1981 10:10:02      1 A      FALSE
```

Parsing other types

- `parse_logical()` and `parse_factor()` and `parse_string()` are pretty self-explanatory.

```
dfdat %>%
  mutate(factor = parse_factor(factor))
```

```
## # A tibble: 5 x 5
##   date       time     number factor logical
##   <chr>      <chr>    <chr>  <fct>  <chr>
## 1 12-01-1988 10:10:02 2      A      TRUE
## 2 11-12-1987 11:10:57 4      A      TRUE
## 3 02-03-1989 10:10:25 6      B      FALSE
## 4 06-03-1982 22:10:55 2      B      TRUE
## 5 09-21-1981 10:10:02 1      A      FALSE
```

```
dfdat %>%
  mutate(logical = parse_logical(logical))
```

```
## # A tibble: 5 x 5
##   date       time     number factor logical
##   <chr>      <chr>    <chr>  <chr>  <lgl>
## 1 12-01-1988 10:10:02 2      A      TRUE
## 2 11-12-1987 11:10:57 4      A      TRUE
## 3 02-03-1989 10:10:25 6      B      FALSE
## 4 06-03-1982 22:10:55 2      B      TRUE
## 5 09-21-1981 10:10:02 1      A      FALSE
```

Parsing and readr

- When you specify `col_types` in `read_csv()`, those are wrappers for parsers.

```
read_csv("../../data/estate.csv",
         col_types = cols(
           Price   = col_double(),
           Area    = col_double(),
           Bed     = col_double(),
           Bath    = col_double(),
           AC      = col_logical(),
           Garage  = col_double(),
           Pool    = col_logical(),
           Year    = col_double(),
           Quality = col_factor(),
           Style   = col_factor(),
           Lot     = col_double(),
           Highway = col_logical()
           )) ->
  estate
estate
```

```
## # A tibble: 522 x 12
##     Price  Area   Bed  Bath AC    Garage Pool   Year Quality Style   Lot
##     <dbl> <dbl> <dbl> <dbl> <lgl>  <dbl> <lgl> <dbl> <fct>   <fct> <dbl>
##  1 360000  3032     4     4 TRUE       2 FALSE  1972 Medium  1     22221
##  2 340000  2058     4     2 TRUE       2 FALSE  1976 Medium  1     22912
##  3 250000  1780     4     3 TRUE       2 FALSE  1980 Medium  1     21345
##  4 205500  1638     4     2 TRUE       2 FALSE  1963 Medium  1     17342
##  5 275500  2196     4     3 TRUE       2 FALSE  1968 Medium  7     21786
##  6 248000  1966     4     3 TRUE       5 TRUE   1972 Medium  1     18902
##  7 229900  2216     3     2 TRUE       2 FALSE  1972 Medium  7     18639
##  8 150000  1597     2     1 TRUE       1 FALSE  1955 Medium  1     22112
##  9 195000  1622     3     2 TRUE       2 FALSE  1975 Low     1     14321
## 10 160000  1976     3     3 FALSE      1 FALSE  1918 Low     1     32358
## # ... with 512 more rows, and 1 more variable: Highway <lgl>
```