Working with data in R

August 2017

Importing data

```r
library(readxl)
downloads <- read_excel("~/data/downloads.xlsx")
downloads
```

<p>| A tibble: 147,835 x 6 |
| machine name | user id | size | time | date | month |</p>
<table>
<thead>
<tr>
<th>&lt;chr&gt;</th>
<th>&lt;dbl&gt;</th>
<th>&lt;dbl&gt;</th>
<th>&lt;dbl&gt;</th>
<th>&lt;dttm&gt;</th>
<th>&lt;chr&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>cs18</td>
<td>146579</td>
<td>2464</td>
<td>0.493030</td>
<td>1995-04-24</td>
<td>1995-04</td>
</tr>
<tr>
<td>cs18</td>
<td>995988</td>
<td>7745</td>
<td>0.325608</td>
<td>1995-04-24</td>
<td>1995-04</td>
</tr>
<tr>
<td>cs18</td>
<td>317649</td>
<td>6727</td>
<td>0.313704</td>
<td>1995-04-24</td>
<td>1995-04</td>
</tr>
<tr>
<td>cs18</td>
<td>748581</td>
<td>356</td>
<td>0.259252</td>
<td>1995-04-24</td>
<td>1995-04</td>
</tr>
<tr>
<td>cs18</td>
<td>444174</td>
<td>0</td>
<td>0.000000</td>
<td>1995-04-24</td>
<td>1995-04</td>
</tr>
</tbody>
</table>

# ... with 1.47e+05 more rows


size: Download size in bytes
time: Download time in seconds

R stores the data in a *tibble*, a type of *data frame*.

The data rows appear in the order that they were read into R, i.e. in the order that they appeared in the Excel file.

This is a "tidy" data set. Rows are referred to as *observations*, columns as *variables*.

(Tidyverse) packages

```r
library("tidyverse")
```

Loading tidyverse: ggplot2
Loading tidyverse: tibble
Loading tidyverse: tidyr
Loading tidyverse: readr
Loading tidyverse: purrr
Loading tidyverse: dplyr

Conflicts with tidy packages ----------------------------------------------

filter(): dplyr, stats
lag():    dplyr, stats

The tidyverse, see https://www.tidyverse.org/, is a collection of R packages which, among other things, facilitate data handling and data transformation in R.

Loading the R package tidyverse loads the core packages from this collection.
Sorting data

The `arrange` function can be used to sort the data according to one or more columns.

The sorted data set is printed on-screen. The original data set assigned to `downloads` has not been changed.

Filtering data

Here, only observations with strictly positive download sizes are kept.

The result is assigned to `downloads`. This modifies/overwrites the data imported from the Excel file.

Filtering requires logical predicates. These are expressions in terms of columns, which evaluate to either `TRUE` or `FALSE` for each row.

Combining filters

Rows not from kermit, and with size greater than 200000 bytes are kept.

Comparisons: `==`, `!=`, `<`, `>`, `<=`, `>=`, `%in%`, `is.na`.

Logical operations: `!` (not), `|` (or), `&` (and).
Selecting variables

downloads <- select(downloads, `machine name`, size, time)
downloads

# A tibble: 36,788 x 3
  `machine name` size    time
  <chr>    <dbl>  <dbl>
1       cs18  2464 0.493030
2       cs18  7745 0.325608
3       cs18  6727 0.313704
4       cs18 13849 0.582537
5       cs18   356 0.259252
6       cs18 15063 0.335502
# ... with 3.67e+04 more rows

Here, the three variables `machine name`, `size`, and `time` are kept. The rest are discarded from the data set.

The `select` function can be used to select only a subset of the variables from the original data set.

Transforming data

downloads <- mutate(downloads, speed = size / time)
downloads

# A tibble: 36,788 x 4
  `machine name` size    time  speed
  <chr>    <dbl>  <dbl>  <dbl>
1       cs18  2464 0.493030  4997.667
2       cs18  7745 0.325608  23786.271
3       cs18  6727 0.313704  21443.781
4       cs18 13049 0.582537  22400.294
5       cs18   356 0.259252   1373.181
6       cs18 15063 0.335502  44896.901
# ... with 3.67e+04 more rows

Here, `speed` is computed as a transformation of two existing variables in the data set.

Using `mutate`, new columns with transformed data can be added to the data set, and existing columns can be modified.

Use `transmute` to keep the transformed columns only (here: `speed`).

More transformation of data

downloads <- mutate(downloads, slow = ifelse(speed < 150, "Yes", "No"))
arrange(downloads, desc(speed))

# A tibble: 36,788 x 5
  `machine name` size    time  speed  slow
  <chr>    <dbl>  <dbl>  <dbl> <chr>
1       cs18 2691689 0.833677 3228695    No
2    tweetie 1236599 0.785326 1574631    No
3    piglet 1877229 0.546841 1440062    No
4    tweetie 1488452 1.069772 1391373    No
5    piglet    961943 0.882681 1089797    No
6    tweetie    961943 0.882681 1089797    No
# ... with 3.67e+04 more rows

Here, a new categorial variable `slow`, a transformation of the numerical variable `speed`, is added to the data set.

Then, the data are sorted in descending order by download speed.

Counting observations

count(downloads)

# A tibble: 1 x 1
  n
*<int>
1 36788

Here, a simple application of `count` counts the number of observations (rows) currently in the data set.
Counting missing values

```r
# A tibble: 1 x 2
  `is.na(time)`     n
  <lgl> <int>
1  FALSE 36708
```

Here, `is.na` returns TRUE for the observations which are missing a download time, and FALSE otherwise.

Counting the number of observations for which `is.na(time)` is TRUE, respectively FALSE, reveals that `time` has no missing values.

Tabulation of categorical variables

```r
# A tibble: 10 x 3
  `machine name`  slow     n
  <chr> <chr> <int>
1           cs18    No  3662
2           cs18   Yes   152
3         kermit    No  8717
4         kermit   Yes   377
5         piglet    No 10734
6         piglet   Yes   466
7          pluto    No  4963
8          pluto   Yes   290
9        tweetie    No  6983
10        tweetie   Yes   364
```

The `count` function is useful for tabulation of categorical variables. Here, the number of observations for each combination of `machine name` and `slow` are counted.

Extracting the values of a variable

```r
size_vector <- downloads$size
size_vector[1:100]
```

A vector with the values of a variable can be extracted from the data set using the `$` operator.

Here, the first hundred download sizes in the vector are shown.

```r
[   2464  7746  6727 13849  356 15663  2548 1933  7294  4470  269
[  12]  1872  4799  1872  24694  2769  513  6441  2865  27815  1681  465
[  23]  3425 18564  275  9385 14684  2336  2280  1862  4559  725  750
[  34]  754  737  753  687  718  713  788  717  440  440  440
[  45]  440  440  440  514  514  514  585  468  394  391  408
[  56]  6511 2752  417  6466  539  715  1094  455  1897  3007  2769
[  67]  663  6067  513  755  31614  833  21639  877  7975  938  33127
[  78]  2242  786  533  3316 12855  504  297  558  382  557  559
[  89]  455  535  2466  7747  3903  1805  717  23947  743  2948  23757
[ 100]  23757
```

A summary of the download sizes is:

```r
mean(size_vector)
```

```
16638.36
```

```r
sd(size_vector)
```

```
177424.2
```

```r
sum(size_vector)
```

```
610761012
```

Here, the average, standard deviation, and sum of the download sizes are computed from the vector of sizes.

Examples of R functions for computing summary statistics: mean, median, sd, var, sum, quantile, min, max, IQR.
Summary stats with summarize

```r
summarize(downloads, 
  avg = mean(size), 
  med = median(size), 
  stdev = sd(size), 
  total = sum(size), 
  n = n())
```

Here, summary statistics for the download sizes are computed from the `size` variable in the data set, and presented in a tibble.

Observe that the number of observations is counted using `n()`.

Grouping

```r
downloads <- group_by(downloads, `machine name`)
downloads
```

Here, `group_by` is used to indicate that the observations in the data set should be grouped by `machine name`. Notice the line

```
# Groups:   machine name 
```

in the output above.

Grouped summary statistics

```r
## After grouping data by `machine name`
summarize(downloads, 
  avg = mean(size), 
  med = median(size), 
  stdev = sd(size), 
  total = sum(size), 
  n = n())
```

Now `summarize` generates group summary statistics for the data set.

The number of observations in each group is counted using `n()`.

Grouping by several variables

```r
group_by(downloads, `machine name`, slow)
downloads
```

Data can be grouped by more than one variable.

For example, data could be grouped by both `machine name` and `slow`.

```
# Groups:   machine name, slow 
```

```r
summarize(downloads, 
  avg = mean(size), 
  med = median(size), 
  stdev = sd(size), 
  total = sum(size), 
  n = n())
```

Now `summarize` generates group summary statistics for the data set.

The number of observations in each group is counted using `n()`.
The pipe operator: %>%

```
# Data grouped (only) by `machine name`
downloads %>%
  filter(`machine name` %in% c("cs18", "pluto", "tweetie")) %>%
  summarize(avg = mean(size)) %>%
  arrange(avg)
```

```
# A tibble: 3 x 2
  `machine name` avg
  <chr>    <dbl>
1 pluto    13821.73
2 tweetie 14207.13
3 cs18    26374.75
```

Two or more function calls can be evaluated sequentially using %>%.

The default behaviour is that, e.g., \( x \%>\% f(y) \) can be used in place of \( f(x, y) \), see more at http://magrittr.tidyverse.org/.

Nesting of function calls becomes more readable, and intermediate assignments are avoided.

Where to read more

- **About the tidyverse packages**
  - *R for Data Science* by Garrett Grolemund & Hadley Wickham
    - Book available online for free: http://r4ds.had.co.nz/
  - The tidyverse website: https://www.tidyverse.org/
- **About R Markdown**
  - The RStudio webpage on R markdown: http://rmarkdown.rstudio.com/