II. Working with data in R (presentation)

Data Science Lab, University of Copenhagen

18 October, 2021

Tidyverse package

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

We must install and load the R package **tidyverse** before we have access to the functions.

- Install package: One option is to go via the *Tools* menu: *Tools* → *Install packages* → write `tidyverse` in the field called *Packages*. This only has to be done once. Otherwise use the `install.packages` function as shown here:

```
install.packages("tidyverse", repos = "https://mirrors.dotsrc.org/cran/")
```

```
##
## The downloaded binary packages are in
##  /var/folders/ch/mtlydjm96n7cqq2d3b3xjdww0000gp/T//RtmpV92mVM/downloaded_packages
```

- Load package: Use the `library` command below (preferred), or go to the *Packages* menu in the bottom right window, find **tidyverse** in the list, and click it. This has to be done in every R-session where you use the package.

```
library(tidyverse)
```

People with SCIENCE PC's (Windows) sometimes have problems with the installation step because R tries to install files to a place, where the user doesn't have permissions to save and edit files. You can try this instead:

- When you start RStudio, right-click the icon and choose *Run as administrator*. Perhaps you can now install packages by clicking *Tools* and *Include Packages* as above.
- If not, then the problem may be that RStudio is trying to install to your science drive (H: or \a00143.science.domain). If so, try the command `.libPaths()`. If it shows two folders - one at the science drive and one locally one on your computer (`C:`) - then try the command `install.packages("tidyverse", lib=.libPaths()[2])`.

About the working directory

When working on a project, it is important to know *where you are*. The working directory is the path on your computer that R will *try* to access files from.

There are several helpful commands that help you navigate.

```
# show current working directory (cwd)
getwd()

# show folders in cwd
```

```r
list.dirs(path = ".", recursive = FALSE)

# relative path, go one folder deeper into Presentations
setwd('./Presentations')

# go one step back in the directory
setwd('..')

# absolute path
setwd("~/Documents/work/Projects/FromExceltoR_2021/Presentations")
```

Import data

Data from Excel files can be imported via the *Import Dataset* facility. You may get the message that the package **readxl** should be installed. If so, then install it as explained for **tidyverse** above.

- Find *Import Data* in the upper right window in RStudio, and choose *From Excel* in the dropdown menu.
- A new window opens. Browse for the relevant Excel file; then a preview of the dataset is shown. Check that it looks OK, and click *Import*.
- Three things happened: Three lines of code was generated (and executed) in the Console, a new dataset now appears in the Environment window, and the dataset is shown in the top left window. Check again that it looks OK.
- Copy the first two lines of code into your R script (or into an R chunk in your Markdown document), but delete line starting with `View` and write instead the name of the dataset, here **downloads**. Then the first 10 lines of the data set are printed.

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

```
## # A tibble: 147,035 x 6
##    machineName userID  size  time date                month
##    <chr>        <dbl> <dbl> <dbl> <dttm>              <chr>
##  1 cs18        146579  2464 0.493 1995-04-24 00:00:00 1995-04
##  2 cs18        995988  7745 0.326 1995-04-24 00:00:00 1995-04
##  3 cs18        317649  6727 0.314 1995-04-24 00:00:00 1995-04
##  4 cs18        748501 13049 0.583 1995-04-24 00:00:00 1995-04
##  5 cs18        955815   356 0.259 1995-04-24 00:00:00 1995-04
##  6 cs18        444174     0 0     1995-04-24 00:00:00 1995-04
##  7 cs18        446911     0 0     1995-04-24 00:00:00 1995-04
##  8 cs18        449552     0 0     1995-04-24 00:00:00 1995-04
##  9 cs18        456142     0 0     1995-04-24 00:00:00 1995-04
## 10 cs18        458942     0 0     1995-04-24 00:00:00 1995-04
## # ... with 147,025 more rows
```

R has stored the data in a so-called *tibble*, a type of data frame. Rows are referred to as *observations* or *data lines*, columns as *variables*. The data rows appear in the order as in the Excel file.

A slight digression: If data are saved in a csv file (comma separated values), possibly generated via an Excel sheet, then data can be read with the read_csv function. For example, if the data file is called `mydata.csv` and values are separated with commas, then the command

```
mydata <- read.csv("mydata.csv", sep=",")
```

creates a data frame in R with the data. The data frame is *not* a tibble and some of the commands below would not work for such a data frame.

About the data

The dataset is from Boston University and is about www data transfers from November 1994 to May 1995, see http://ita.ee.lbl.gov/html/contrib/BU-Web-Client.html.

- It has 147,035 data lines and 6 variables
- *size* is the download size in bytes, and *time* is the download time in seconds.

Extracting variables, simple summary statistics

Variables can be extracted with the $-syntax, and we can use squared brackets to show only the first 40, say, values.

```
time_vector <- downloads$time
time_vector[1:40]
```

```
##  [1] 0.493030 0.325608 0.313704 0.582537 0.259252 0.000000 0.000000 0.000000
##  [9] 0.000000 0.000000 0.000000 0.335502 0.284853 0.000000 0.000000 0.000000
## [17] 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
## [25] 0.285665 0.397111 3.410561 0.267474 0.842364 0.903005 2.784645 2.806157
## [33] 0.990092 0.477629 0.000000 0.000000 0.000000 0.000000 0.988944 0.000000
```

Summary statistics like mean, standard deviation, median are easily computed for a vector.

Examples of R functions for computing summary statistics: `length`, `mean`, `median`, `sd`, `var`, `sum`, `quantile`, `min`, `max`, `IQR`.

```
length(time_vector)
```

```
## [1] 147035
```

```
mean(time_vector)
```

```
## [1] 0.9539674
```

```
sd(time_vector)
```

```
## [1] 14.22557
```

```
median(time_vector)
```

```
## [1] 0
```

```
min(time_vector)
```

```
## [1] 0
```

Notice that more than half the observations have time equal to zero (median is zero).

Filtering data (selecting rows): `filter`

The `filter` function is used to make sub-datasets where only certain datalines (rows) are maintained. It's described with *logical expressions* which datalines should be kept in the dataset.

Say that we only want observations with download time larger than 1000 seconds; there happens to be eight such observations:

```
filter(downloads, time > 1000)
```

```
## # A tibble: 8 x 6
##   machineName userID     size  time date                month
##   <chr>        <dbl>    <dbl> <dbl> <dttm>              <chr>
## 1 cs18        502807  4055821 1275. 1994-12-02 00:00:00 1994-12
## 2 cs18         16653  2573336 1335. 1994-11-22 00:00:00 1994-11
## 3 cs18        957883  2743516 1151. 1994-11-22 00:00:00 1994-11
## 4 cs18         47910  4720220 1749. 1994-11-22 00:00:00 1994-11
## 5 tweetie     223655   245003 1214. 1995-04-13 00:00:00 1995-04
## 6 kermit      576790 14518894 1380. 1995-04-20 00:00:00 1995-04
## 7 kermit      139654  1079731 1129. 1995-02-23 00:00:00 1995-02
## 8 pluto       337530  8674562 1878. 1995-03-13 00:00:00 1995-03
```

Or say that only want observations with strictly positive download size:

```
downloads2 <- filter(downloads, size > 0)
downloads2
```

```
## # A tibble: 36,708 x 6
##    machineName userID  size  time date                month
##    <chr>        <dbl> <dbl> <dbl> <dttm>              <chr>
##  1 cs18        146579  2464 0.493 1995-04-24 00:00:00 1995-04
##  2 cs18        995988  7745 0.326 1995-04-24 00:00:00 1995-04
##  3 cs18        317649  6727 0.314 1995-04-24 00:00:00 1995-04
##  4 cs18        748501 13049 0.583 1995-04-24 00:00:00 1995-04
##  5 cs18        955815   356 0.259 1995-04-24 00:00:00 1995-04
##  6 cs18        596819 15063 0.336 1995-04-24 00:00:00 1995-04
##  7 cs18        169424  2548 0.285 1995-04-24 00:00:00 1995-04
##  8 cs18        386686  1932 0.286 1995-04-24 00:00:00 1995-04
##  9 cs18        783767  7294 0.397 1995-04-24 00:00:00 1995-04
## 10 cs18        788633  4470 3.41  1995-04-24 00:00:00 1995-04
## # ... with 36,698 more rows
```

Notice that this result is assigned to **downloads2**. It has 36,708 data lines. The original data called **downloads** still exists with 147,035 data lines.

Filtering requires *logical predicates*. These are expressions in terms of columns, which evaluate to either `TRUE` or `FALSE` for each row. Logical expressions can be combined with logical operations.

- Comparisons: `==`, `!=`, `<`, `>`, `<=`, `>=`, `%in%`, `is.na`
- Logical operations: `!` (not), `|` (or), `&` (and). A comma can be used instead of `&`

Here comes two sub-datasets:

```
# Rows from kermit, and with size greater than 200000 bytes are kept.
filter(downloads2, machineName == "kermit", size > 200000)
```

```
## # A tibble: 98 x 6
##    machineName userID     size     time date                    month
##    <chr>        <dbl>    <dbl>    <dbl> <dttm>                  <chr>
```

```
##   1 kermit      157161   498325    0.629 1995-04-13 00:00:00 1995-04
##   2 kermit      734988   271058   17.3   1995-04-22 00:00:00 1995-04
##   3 kermit      388066   435923   29.2   1995-04-22 00:00:00 1995-04
##   4 kermit       34030   642771    4.80  1995-04-12 00:00:00 1995-04
##   5 kermit      327021   724757    4.98  1995-04-12 00:00:00 1995-04
##   6 kermit       38016   561762    9.75  1995-04-05 00:00:00 1995-04
##   7 kermit      277395   404209   11.3   1995-04-05 00:00:00 1995-04
##   8 kermit      576790 14518894 1380.    1995-04-20 00:00:00 1995-04
##   9 kermit       17623   489473   21.2   1995-04-20 00:00:00 1995-04
##  10 kermit      198041   355963   15.3   1995-04-20 00:00:00 1995-04
## # ... with 88 more rows
```

```
# Rows NOT from kermit, and with size greater than 200000 bytes are kept.
filter(downloads2, machineName != "kermit" & size > 200000)
```

```
## # A tibble: 220 x 6
##    machineName userID    size    time date                month
##    <chr>        <dbl>   <dbl>   <dbl> <dttm>              <chr>
##  1 cs18        204764 2691689   0.834 1995-04-26 00:00:00 1994-04
##  2 cs18        397405  215045   1.10  1994-12-15 00:00:00 1994-12
##  3 cs18        809091  226586   3.92  1994-12-15 00:00:00 1994-12
##  4 cs18        779032 1080472 156.    1994-12-11 00:00:00 1994-12
##  5 cs18        688294  748705  93.1   1994-12-11 00:00:00 1994-12
##  6 cs18        447740 6360764 863.    1994-12-11 00:00:00 1994-12
##  7 cs18        708452  204918   7.07  1994-12-18 00:00:00 1994-12
##  8 cs18        598668  204918  12.7   1994-12-18 00:00:00 1994-12
##  9 cs18        288167  204918   4.98  1994-12-18 00:00:00 1994-12
## 10 cs18        974956  203714   6.13  1994-12-16 00:00:00 1994-12
## # ... with 210 more rows
```

A helpful function to know which machine names are valid can be:

```
# get unique machineName values in downloads2
distinct(downloads2, machineName)
```

```
## # A tibble: 5 x 1
##   machineName
##   <chr>
## 1 cs18
## 2 piglet
## 3 kermit
## 4 tweetie
## 5 pluto
```

And if you are looking for multiple values for a given variable:

```
downloads2 %>% filter(machineName %in% c("kermit","pluto"), size > 2000000)
```

```
## # A tibble: 8 x 6
##   machineName userID     size   time date                month
##   <chr>        <dbl>    <dbl>  <dbl> <dttm>              <chr>
## 1 kermit      576790 14518894 1380.  1995-04-20 00:00:00 1995-04
## 2 kermit      756949  4418124  439.  1995-04-20 00:00:00 1995-04
## 3 kermit      287308  6935603   88.2 1995-04-24 00:00:00 1995-04
## 4 kermit      928227  9523767  171.  1995-02-08 00:00:00 1995-02
## 5 kermit      128147  2743816  216.  1995-02-23 00:00:00 1995-02
## 6 pluto       867173  4670973  230.  1995-03-14 00:00:00 1995-03
```

```
## 7 kermit      456524  2836135  127.   1995-03-31 00:00:00 1995-03
## 8 pluto       337530  8674562 1878.   1995-03-13 00:00:00 1995-03
```

Selecting variables: `select`

Sometimes, datasets has many variables of which only some are relevant for the analysis. Variables can be selected or skipped with the `select` function.

```
# Without the date variable
select(downloads2, -date)
```

```
## # A tibble: 36,708 x 5
##    machineName userID  size  time month
##    <chr>        <dbl> <dbl> <dbl> <chr>
##  1 cs18        146579  2464 0.493 1995-04
##  2 cs18        995988  7745 0.326 1995-04
##  3 cs18        317649  6727 0.314 1995-04
##  4 cs18        748501 13049 0.583 1995-04
##  5 cs18        955815   356 0.259 1995-04
##  6 cs18        596819 15063 0.336 1995-04
##  7 cs18        169424  2548 0.285 1995-04
##  8 cs18        386686  1932 0.286 1995-04
##  9 cs18        783767  7294 0.397 1995-04
## 10 cs18        788633  4470 3.41  1995-04
## # ... with 36,698 more rows
```

```
# Only include the three mentioned variable names
downloads3 <- select(downloads2, machineName, size, time)
downloads3
```

```
## # A tibble: 36,708 x 3
##    machineName  size  time
##    <chr>       <dbl> <dbl>
##  1 cs18         2464 0.493
##  2 cs18         7745 0.326
##  3 cs18         6727 0.314
##  4 cs18        13049 0.583
##  5 cs18          356 0.259
##  6 cs18        15063 0.336
##  7 cs18         2548 0.285
##  8 cs18         1932 0.286
##  9 cs18         7294 0.397
## 10 cs18         4470 3.41
## # ... with 36,698 more rows
```

Notice that we have made a new dataframe, **downloads3** with only three variables.

Transformations of data

Tranformations of existing variables in the data set can be computed and included in the data set with the `mutate` function.

We first compute two new variables, download speed (**speed**) and the logarithm of the download size (**logSize**):

```
downloads3 <- mutate(downloads3, speed = size / time, logSize = log10(size))
downloads3
```

```
## # A tibble: 36,708 x 5
##    machineName  size  time  speed logSize
##    <chr>       <dbl> <dbl>  <dbl>   <dbl>
##  1 cs18         2464 0.493  4998.    3.39
##  2 cs18         7745 0.326 23786.    3.89
##  3 cs18         6727 0.314 21444.    3.83
##  4 cs18        13049 0.583 22400.    4.12
##  5 cs18          356 0.259  1373.    2.55
##  6 cs18        15063 0.336 44897.    4.18
##  7 cs18         2548 0.285  8945.    3.41
##  8 cs18         1932 0.286  6763.    3.29
##  9 cs18         7294 0.397 18368.    3.86
## 10 cs18         4470 3.41   1311.    3.65
## # ... with 36,698 more rows
```

We then make a new categorial variable, **slow**, which is "Yes" is speed < 150 and "No" otherwise

```
downloads3 <- mutate(downloads3, slow = ifelse(speed < 150, "Yes", "No"))
downloads3
```

```
## # A tibble: 36,708 x 6
##    machineName  size  time  speed logSize slow
##    <chr>       <dbl> <dbl>  <dbl>   <dbl> <chr>
##  1 cs18         2464 0.493  4998.    3.39 No
##  2 cs18         7745 0.326 23786.    3.89 No
##  3 cs18         6727 0.314 21444.    3.83 No
##  4 cs18        13049 0.583 22400.    4.12 No
##  5 cs18          356 0.259  1373.    2.55 No
##  6 cs18        15063 0.336 44897.    4.18 No
##  7 cs18         2548 0.285  8945.    3.41 No
##  8 cs18         1932 0.286  6763.    3.29 No
##  9 cs18         7294 0.397 18368.    3.86 No
## 10 cs18         4470 3.41   1311.    3.65 No
## # ... with 36,698 more rows
```

Counting, tabulation of categorical variables: `count`

The `count` function is useful for counting data datalines, possibly according to certain criteria or for the different levels of categorical values.

```
# Total number of observations in the current dataset
count(downloads3)
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1 36708
```

```
# Number of observations from each machine
count(downloads3, machineName)
```

```
## # A tibble: 5 x 2
##   machineName     n
```

```
##   <chr>       <int>
## 1 cs18         3814
## 2 kermit       9094
## 3 piglet      11200
## 4 pluto        5253
## 5 tweetie      7347
```

```
# Number of observations which have/have not size larger than 5000
count(downloads3, size>5000)
```

```
## # A tibble: 2 x 2
##   `size > 5000`     n
##   <lgl>         <int>
## 1 FALSE         25865
## 2 TRUE          10843
```

```
# Number of observations for each combination of machine name and the *slow* variable.
count(downloads3, machineName, slow)
```

```
## # A tibble: 10 x 3
##    machineName 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
```

Sorting data: `arrange`

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

Let's sort the data according to download size (ascending order). The first lines of the sorted data set is printed on-screen, but the dataset **downloads3** has *not* been changed.

```
arrange(downloads3, size)
```

```
## # A tibble: 36,708 x 6
##    machineName  size  time speed logSize slow
##    <chr>       <dbl> <dbl> <dbl>   <dbl> <chr>
##  1 cs18            3  3.73 0.804   0.477 Yes
##  2 piglet          3  1.53 1.96    0.477 Yes
##  3 piglet          3  1.53 1.96    0.477 Yes
##  4 tweetie         3  1.11 2.71    0.477 Yes
##  5 kermit          3  1.12 2.69    0.477 Yes
##  6 pluto           3  8.60 0.349   0.477 Yes
##  7 pluto           3  9.87 0.304   0.477 Yes
##  8 pluto           3  3.78 0.793   0.477 Yes
##  9 pluto           3  4.68 0.641   0.477 Yes
## 10 pluto           3  4.93 0.608   0.477 Yes
```

```
## # ... with 36,698 more rows
```

Two different examples:

```r
# According to download size in descending order
arrange(downloads3, desc(size))
```

```
## # A tibble: 36,708 x 6
##    machineName     size   time   speed logSize slow
##    <chr>          <dbl>  <dbl>   <dbl>   <dbl> <chr>
##  1 kermit      14518894 1380.   10522.    7.16 No
##  2 piglet      14158123  123.  115169.    7.15 No
##  3 kermit       9523767  171.   55562.    6.98 No
##  4 piglet       9384067   80.0 117309.    6.97 No
##  5 pluto        8674562 1878.    4619.    6.94 No
##  6 kermit       6935603   88.2  78655.    6.84 No
##  7 cs18         6360764  863.    7374.    6.80 No
##  8 piglet       5143062  597.    8611.    6.71 No
##  9 piglet       4812334  215.   22345.    6.68 No
## 10 cs18         4720220 1749.    2700.    6.67 No
## # ... with 36,698 more rows
```

```r
# After machine name and then according to download size in descending order
arrange(downloads3, machineName, desc(size))
```

```
## # A tibble: 36,708 x 6
##    machineName    size     time    speed logSize slow
##    <chr>         <dbl>    <dbl>    <dbl>   <dbl> <chr>
##  1 cs18        6360764  863.       7374.    6.80 No
##  2 cs18        4720220 1749.       2700.    6.67 No
##  3 cs18        4055821 1275.       3180.    6.61 No
##  4 cs18        3047343   20.9    146038.    6.48 No
##  5 cs18        2952381  318.       9289.    6.47 No
##  6 cs18        2743516 1151.       2383.    6.44 No
##  7 cs18        2691689    0.834 3228695.    6.43 No
##  8 cs18        2613025   18.5    140959.    6.42 No
##  9 cs18        2573336 1335.       1928.    6.41 No
## 10 cs18        1931453  186.      10388.    6.29 No
## # ... with 36,698 more rows
```

Grouping: `group_by`

We can group the dataset by one or more categorical variables with `group_by`. The dataset is not changed as such, but - as we will see - grouping can be useful for computation of summary statistics and graphics.

Here we group after machine name (first) *and* the slow variable (second). The only way we can see it at this point is in the second line in the output (`# Groups:`):

```r
# Group according to machine
group_by(downloads3, machineName)
```

```
## # A tibble: 36,708 x 6
## # Groups:   machineName [5]
##    machineName  size  time  speed logSize slow
##    <chr>       <dbl> <dbl>  <dbl>   <dbl> <chr>
##  1 cs18         2464 0.493  4998.    3.39 No
```

```
##   2 cs18          7745 0.326 23786.    3.89 No
##   3 cs18          6727 0.314 21444.    3.83 No
##   4 cs18         13049 0.583 22400.    4.12 No
##   5 cs18           356 0.259  1373.    2.55 No
##   6 cs18         15063 0.336 44897.    4.18 No
##   7 cs18          2548 0.285  8945.    3.41 No
##   8 cs18          1932 0.286  6763.    3.29 No
##   9 cs18          7294 0.397 18368.    3.86 No
##  10 cs18          4470 3.41   1311.    3.65 No
## # ... with 36,698 more rows
```

```
# Group according to machine and slow
group_by(downloads3, machineName, slow)
```

```
## # A tibble: 36,708 x 6
## # Groups:   machineName, slow [10]
##    machineName  size  time  speed logSize slow
##    <chr>       <dbl> <dbl>  <dbl>   <dbl> <chr>
##   1 cs18          2464 0.493  4998.    3.39 No
##   2 cs18          7745 0.326 23786.    3.89 No
##   3 cs18          6727 0.314 21444.    3.83 No
##   4 cs18         13049 0.583 22400.    4.12 No
##   5 cs18           356 0.259  1373.    2.55 No
##   6 cs18         15063 0.336 44897.    4.18 No
##   7 cs18          2548 0.285  8945.    3.41 No
##   8 cs18          1932 0.286  6763.    3.29 No
##   9 cs18          7294 0.397 18368.    3.86 No
##  10 cs18          4470 3.41   1311.    3.65 No
## # ... with 36,698 more rows
```

Summary statistics, revisited: `summarize`

Recall how we could compute summary statistics for a single variable in a dataset, e.g.

```
mean(downloads3$size)
```

```
## [1] 16638.36
```

```
max(downloads3$size)
```

```
## [1] 14518894
```

With `summarize` we can compute summary statistics for a variable for each level of a grouping variable or for each combination of several grouping variables.

First, a bunch of summaries for the size variable for each machine name, where we give explicit names for the new variables:

```
downloads.grp1 <- group_by(downloads3, machineName)
summarize(downloads.grp1,
          avg = mean(size),
          med = median(size),
          stdev = sd(size),
          total = sum(size),
          n = n())
```

```
## # A tibble: 5 x 6
```

```
##   machineName    avg   med   stdev     total     n
##   <chr>        <dbl> <dbl>   <dbl>     <dbl> <int>
## 1 cs18        26375. 1990. 208915. 100593281  3814
## 2 kermit      19247. 2466  213985. 175032552  9094
## 3 piglet      14121. 2146. 188340. 158149841 11200
## 4 pluto       13822. 2069  144425.  72605544  5253
## 5 tweetie     14207. 2197   94318. 104379794  7347
```

Second, the same thing but for each combination of machine name and the slow variable:

```
downloads.grp2 <- group_by(downloads3, machineName, slow)
summarize(downloads.grp2,
          avg = mean(size),
          med = median(size),
          stdev = sd(size),
          total = sum(size),
          n = n())
```

```
## # A tibble: 10 x 7
## # Groups:   machineName [5]
##    machineName slow     avg   med   stdev     total     n
##    <chr>       <chr>  <dbl> <dbl>   <dbl>     <dbl> <int>
##  1 cs18        No    27445. 2092. 213140. 100503042  3662
##  2 cs18        Yes     594.  368.    614.     90239   152
##  3 kermit      No    20030. 2598  218529. 174602282  8717
##  4 kermit      Yes    1141.  541    3049.    430270   377
##  5 piglet      No    14687. 2264  192365. 157650747 10734
##  6 piglet      Yes    1071.  416.   1934.    499094   466
##  7 pluto       No    14564. 2164  148551.  72280790  4963
##  8 pluto       Yes    1120.  413    2108.    324754   290
##  9 tweetie     No    14894. 2373   96694. 104001733  6983
## 10 tweetie     Yes    1039.  471    2603.    378061   364
```

Third, mean and standard deviation for several variables:

```
summarize_at(downloads.grp2, c("time", "size"), list(ave=mean,stdev=sd))
```

```
## # A tibble: 10 x 6
## # Groups:   machineName [5]
##    machineName slow  time_ave size_ave time_stdev size_stdev
##    <chr>       <chr>    <dbl>    <dbl>      <dbl>      <dbl>
##  1 cs18        No        5.17   27445.       57.1    213140.
##  2 cs18        Yes       9.63     594.       17.8       614.
##  3 kermit      No        3.41   20030.       25.3    218529.
##  4 kermit      Yes      20.7     1141.       47.8      3049.
##  5 piglet      No        2.33   14687.       13.8    192365.
##  6 piglet      Yes      19.4     1071.       40.2      1934.
##  7 pluto       No        3.40   14564.       30.4    148551.
##  8 pluto       Yes      21.7     1120.       46.3      2108.
##  9 tweetie     No        2.68   14894.       17.3     96694.
## 10 tweetie     Yes      17.8     1039.       34.5      2603.
```

The datasets with summaries can be saves as datasets themselves, for example to be used as the basis for certain graphs.

The pipe operator: `%>%`

Two or more function calls can be evaluated sequentially using the so-called pipe operator, `%>%`. Nesting of function calls becomes more readable, and intermediate assignments are avoided.

Let's try it to do a bunch of things in one go, starting with the original dataset:

```
downloads %>%
  filter(size>0) %>% # Subset of data
  group_by(machineName) %>% # Grouping
  summarize(avg = mean(size)) %>% # Compute mean
  arrange(avg) # Sort after mean
```

```
## # A tibble: 5 x 2
##   machineName    avg
##   <chr>        <dbl>
## 1 pluto       13822.
## 2 piglet      14121.
## 3 tweetie     14207.
## 4 kermit      19247.
## 5 cs18        26375.
```