R Tutorial at the WZB

5 - Transforming data with R I

Markus Konrad

November 22, 2018

Introductory note

For this session there are no slides, because I'm not at the WZB to hold a presentation. Instead I provide you with a full document for self-study. As always, there are some tasks at the end of the document which you should complete. I recommend that you not only read the document, but also try out some of the code examples and experiment with them.

Solution for tasks #4

The solutions for tasks #4 are now online on https://wzbsocialsciencecenter.github.io/wzb_r_tutorial/ (https://wzbsocialsciencecenter.github.io/wzb_r_tutorial/).

1 Data transformation using the *tidyverse*

After being introduced to the basics of R programming using "traditional" R (or: *base R*) concepts, we will now focus on a more modern way of R programming using the *tidyverse philosophy*. Don't worry, you will still apply concepts that you learnt in the last sessions with base R as you will soon notice! The modern R programming approach in the tidyverse builds on top of the basic R concepts and provides a common, clearly defined "grammar" for all the main aspects of data analysis including data transformation, visualization and modeling. The tidyverse is organized into several R packages, each serving a distinct purpose. You can inform yourself about the individual packages on https://www.tidyverse.org/packages/ (https://www.tidyverse.org/packages/).

The concept of the tidyverse and the underlying software packages are mainly developed by Hadley Wickham. Together with Garrett Grolemund, he wrote the book *R for Data Science*. The book is freely available online on http://r4ds.had.co.nz/ (http://r4ds.had.co.nz/). We will use this book to study the main concepts of modern data transformation in R.

Before diving into the tidyverse, make sure you've installed the *tidyverse* package via RStudio's package manager or the `install.packages()` function. **You also need to install *nycflights13* package in order to complete the exercises.**

1.1 Filtering rows with `filter()`

Read sections 5.1 and 5.2 (http://r4ds.had.co.nz/transform.html#introduction-2) on filtering (i.e. subsetting) observations with `filter()` from the tidyverse package *dplyr*. Notice how the concepts for comparisons and logical operators are similar to what we've learnt. They are only applied now in a different context.

1.2 Filtering for distinct rows with `distinct()`

Sometimes, you need to get all *distinct* observations of a vector or data frame. This means, that the resulting vector or data frame does not include duplicate observations. Only *unique* observations will be left.

In base R, there is the function `unique()` which will remove duplicate values in vectors:

```
colors <- c('red', 'green', 'red', 'yellow', 'red', 'yellow')
unique(colors)
```

```
## [1] "red"    "green"  "yellow"
```

To filter data frames for unique observations, we can use `distinct()` from the tidyverse package *dplyr*. With this, for example, we can find all unique carrier codes in the `flights` data set:

```
library(nycflights13)
library(tidyverse)

distinct(flights, carrier)
```

```
## # A tibble: 16 x 1
##    carrier
##    <chr>
##  1 UA
##  2 AA
##  3 B6
##  4 DL
##  5 EV
##  6 MQ
##  7 US
##  8 WN
##  9 VX
## 10 FL
## 11 AS
## 12 9E
## 13 F9
## 14 HA
## 15 YV
## 16 OO
```

We could do the same with `unique()` , however, notice that `distinct()` always returns a data frame (the above having only a single column) while `unique()` returns a vector:

```
unique(flights$carrier)
```

```
##  [1] "UA" "AA" "B6" "DL" "EV" "MQ" "US" "WN" "VX" "FL" "AS" "9E" "F9" "HA"
## [15] "YV" "OO"
```

So far we only created distinct observations of a single column, but you can also find the distinct observations using several (or all) columns in a data frame. With this, for example, we can find all distinct flight routes in the `flights` data set:

```
distinct(flights, origin, dest)
```

```
## # A tibble: 224 x 2
##    origin dest
##    <chr>  <chr>
##  1 EWR    IAH
##  2 LGA    IAH
##  3 JFK    MIA
##  4 JFK    BQN
##  5 LGA    ATL
##  6 EWR    ORD
##  7 EWR    FLL
##  8 LGA    IAD
##  9 JFK    MCO
## 10 LGA    ORD
## # ... with 214 more rows
```

1.3 Sorting rows with `arrange()`

Read sections 5.3 (http://r4ds.had.co.nz/transform.html#arrange-rows-with-arrange) on sorting observations with `arrange()` from the tidyverse package *dplyr*. We already got to know the function `sort()` for sorting vectors. `arrange()` allows us to sort data frames, even using several variables to sort along.

1.4 Selecting and renaming columns with `select()` and `rename()`

Read sections 5.4 (http://r4ds.had.co.nz/transform.html#select) on selecting and renaming variables in data frames with `select()` and `rename()` from the tidyverse package *dplyr*.

1.5 Adding new variables with `mutate()` and `transmute()`

At first, read sections 5.5 (http://r4ds.had.co.nz/transform.html#add-new-variables-with-mutate) on adding new variables to data frames with `mutate()` and `transmute()` from the tidyverse package *dplyr*. Then continue with the two sections below.

Note: You can gloss over section 5.5.1 “Useful creation functions” quickly – you don’t need to understand what these functions do in all detail.

1.5.1 Type conversion with `mutate()` and `transmute()`

`mutate()` and `transmute()` are often used to convert the data type of variables. Let's consider this data frame, where `smoker` is indicated with a numerical variable where *0* means *non-smoker* and *1* means *smoker*:

```
(smoker_data <- data.frame(age = c(19, 15, 24, 29, 17), smoker = c(0, 0, 1, NA, 1)))
```

```
##   age smoker
## 1  19      0
## 2  15      0
## 3  24      1
## 4  29     NA
## 5  17      1
```

We can convert this to a logical `TRUE` / `FALSE` vector with the conversion function `as.logical` :

```
(smoker_data <- mutate(smoker_data, smoker = as.logical(smoker)))
```

```
##   age smoker
## 1  19  FALSE
## 2  15  FALSE
## 3  24   TRUE
## 4  29     NA
## 5  17   TRUE
```

1.5.2 `mutate()` and `transmute()` in combination with `ifelse()`

`mutate()` and `transmute()` are often used in combination with the function `ifelse(cond, true-value, false-value)` . This function allows to set one value `true-value` if a condition `cond` is `TRUE` and another value `false-value` if this condition is `FALSE` .

Imagine that NA values were coded as `-1` in the `smoker` variable:

```
(smoker_data <- data.frame(age = c(19, 15, 24, 29, 17), smoker = c(0, 0, 1, -1, 1)))
```

```
##   age smoker
## 1  19      0
## 2  15      0
## 3  24      1
## 4  29     -1
## 5  17      1
```

Now if we tried to convert that to a logical vector, we would run into trouble, because everything that is not *0* is considered `TRUE` , hence *-1* also becomes `TRUE` :

```
mutate(smoker_data, smoker = as.logical(smoker))
```

```
##   age smoker
## 1  19  FALSE
## 2  15  FALSE
## 3  24   TRUE
## 4  29   TRUE
## 5  17   TRUE
```

In order to fix that, we can use `ifelse()` . We set the condition `smoker == -1` and pass `NA` as value that should be taken whenever the condition is `TRUE` . On all other occasions, we want to retain the original value, hence we pass `smoker` . After converting the output to `as.logical()` we get the correct result:

```
mutate(smoker_data, smoker = as.logical(ifelse(smoker == -1, NA, smoker)))
```

```
##   age smoker
## 1  19  FALSE
## 2  15  FALSE
## 3  24   TRUE
## 4  29     NA
## 5  17   TRUE
```

2 Tasks

Note: You need to install the packages *nycflights13* and *tidyverse* in order to complete the exercises.

1. Complete exercises 1 to 3 from section 5.2.4 in *R for Data Science* (http://r4ds.had.co.nz/transform.html#exercises-7). **For exercise 1, choose only 3 out of the 7 tasks (you can choose any task you want).**
2. Complete exercises 1 to 3 from section 5.3.1 in *R for Data Science* (http://r4ds.had.co.nz/transform.html#exercises-8)
3. Complete exercises 1 and 2 from section 5.4.1 in *R for Data Science* (http://r4ds.had.co.nz/transform.html#exercises-9)
4. Download and unzip `05transform1-resources.zip` from the course website (https://wzbsocialsciencecenter.github.io/wzb_r_tutorial/), read `codebook.txt` and complete the following tasks:
    1. Load the CSV file `schulen_potsdam.csv` into R without specifying further parameters for `read.csv()` . Have a look at the data using functions like `str()` and `head()` . Do you spot any potential problems?
    2. Load the data again, this time specifying two additional parameters for `read.csv()` : `stringsAsFactors = FALSE, colClasses = c(plz = "character")` . Inspect the result. What is the effect of these parameters? Are all problems now fixed and do the variable types match the specifications in the codebook (see `codebook.txt` also contained in the zip-file)? If not, convert these variables to the correct data type using `mutate()` and `as.factor()` .
    3. Create a new variable `full_address` using `mutate()` . This new variable should contain the full address consisting of street name, zip code and city name, e.g. *"Carl-von-Ossietzky-Straße 37 01570 Potsdam"*. You can combine several strings to form one string using the function `paste()` .
5. **Optional**: Complete exercises 1 and 2 from section 5.5.2 in *R for Data Science* (http://r4ds.had.co.nz/transform.html#exercises-10)