R403: Probabilistic and Statistical Computations with R

Topic 08: Data Management with the **dplyr** Package

Kaloyan Ganev

2022/2023

Lecture Contents

Introduction

Introduction

- Working with data frames in the 'traditional' way can be cumbersome
- Luckily, additional packages that save a lot of time and trouble exist
- Two notable alternatives: the **dplyr** package, and the **data.table** package
- We will not discuss the latter (it is left to your curiosity)
- Our focus will be on the former
- Before that, we will discuss the 'ecosystem' that it belongs to: the **tidyverse**
- Note: A whole book available here in HTML format: `http://r4ds.had.co.nz/`

The tidyverse

The **tidyverse**

- A group of packages that "share an underlying design philosophy, grammar, and data structures" (https://www.tidyverse.org/)
- This is what in fact makes the "ecosystem"
- Main purpose: data science, but naturally appropriate for other purposes, too
- Take a look at the package list; we will pay some attention to a couple of them initially, then we will switch to **dplyr**
- Over the course of the module, we will have thorough sessions on some other among them (**ggplot2** in particular)
- To install **tidyverse**:

```
install.packages("tidyverse")
```

The **readr** Package

- Load **tidyverse**:

```
library(tidyverse)
```

- **readr** has similar wrappers to the ones present in base R (recall the `read.table()` function):
  - `read_csv()`
  - `read_csv2()`
  - `read_tsv()`
  - `read_delim()`
  - `read_fwf()`
  - `read_table()`: for reading fixed-width files, columns separated by white space

The **readr** Package (2)

- An example:[1]

```
data1 <- read_csv("Demographic_Statistics_By_Zip_Code.csv")
```

- Read now the help on `read_csv()` to see what options are available
- Check:

```
class(data1)
```

- Note that the latter does not produce just `"data.frame"`
- The object that is created is a modern version of the data frame called *tibble*
- The **tibble** package is also a member of the **tidyverse** family

[1] Taken from https://catalog.data.gov/dataset?res_format=CSV

The **tibble** package

- Advantages of tibbles:
  - Better looking when printed in the console
  - Printing options can be controlled
  - Referencing (slicing) works analogically to data frames
  - Does not change the data type of inputs
  - Does not change variable names
  - Does not create row names
- To convert a data frame to a tibble:

```
df1 <- as.data.frame(matrix(1:9, nrow = 3))
tb1 <- as_tibble(df1)
```

Working with **dplyr**

Working with **dplyr**

- We will use the dataset that the book works with: flights from New York City in 2013

```
install.packages("nycflights13")
library(nycflights13)
print(flights)
```

- Obviously, the data are organized in a tibble
- We can now, for example, select specific rows based on specified values
- In order to select all flights that took place on 27 July,

```
july27 <- filter(flights, month == 7, day == 27)
```

- Mind the double equality signs, they are logical checks!

Working with **dplyr** (2)

- Boolean operators that are used: `&` (and), `|` (or), and `!` (not)
- An `or` example:

```
july7or27 <- filter(flights, (day == 7 | day == 27) & month == 7)
```

- A `not` example:

```
not_in_jan <- filter(flights, month != 1)
```

- ...or another one:

```
not_in_winter <- filter(flights, !(month %in% c(1,2,12)))
```

- Note in the latter the `%in%` function that serves as a convenient shorthand

Working with **dplyr** (3)

- Data can be sorted in the following way:

```
sort1 <- arrange(flights, dep_delay, carrier)
```

- In this example, the data will be sorted first according to departure delay, and then by carrier
- Sorting in performed in ascending order
- To sort in descending order:

```
sort2 <- arrange(filter(flights, month == 1, day == 1), desc(dep_
    delay))
```

Working with **dplyr** (4)

- To select variables (columns) from a tibble/data frame:

```
new_tbl1 <- select(flights, year, month, day, flight)
```

- Shortcut for selecting adjacent columns:

```
new_tbl2 <- select(flights, year:day, flight)
```

- Select all columns with the exception of some:

```
new_tbl3 <- select(flights, year:day, -c(hour, minute))
```

- Options to the `select()` function such as `starts_with()`, `ends_with()`, `contains()`, etc. can be used, e.g.

```
new_tbl4 <- select(flights, starts_with("flig"))
```

Working with **dplyr** (5)

- Renaming variables is also simple (new name comes first):

```
flights1 <- rename(flights, airline = carrier)
```

- A way to rearrange columns:

```
flights2 <- select(flights, time_hour, everything())
```

- Here, the `time_hour` variable is moved to the first position, `everything()` plays the role of everything else
- New variables are created with

```
flights <-  mutate(flights, travel_time = arr_time - dep_time)
```

Working with **dplyr** (6)

- Grouped summaries:

```
by_airline <- group_by(flights, carrier)
summarize(by_airline, delay = mean(dep_delay, na.rm = TRUE))
```

- In this example, the average departure delay by airline is calculated
- Grouping can be performed by more than one variable
- In such cases, summaries can be calculated per level of grouping
- Instead of performing operations one by one, the so-called pipe can be used, e.g.:

```
avg_delay <- flights %>%
  group_by(carrier) %>%
  summarize(mean(dep_delay, na.rm = TRUE))
```