Applied Statistics with R

A Very Brief Introduction to **dplyr**

Kaloyan Ganev

2022/2023

Lecture Contents

Introduction

What is **dplyr**?

- A powerful package for manipulation of datasets
- An element of the **tidyverse** ecosystem
- Facilitates operations on datasets by speeding and simplifying them compared to base R
- Looks similar to operations on databases (e.g. through SQL)
- Allows filtering and rearranging rows, selecting variables (columns), data summaries, etc.

The **eurostat** package

The **eurostat** Package: An Interlude

- In order to demonstrate the capabilities of **dplyr**, we will need a dataset
- A nice option are Eurostat datasets
- **eurostat** is a specialized package for working with the Eurostat database
- Naturally, needs to be installed and loaded

```
install.packages("eurostat")
library(eurostat)
```

- We can search for data in the Eurostat database as follows (assume we are looking for GDP data):

```
query <- search_eurostat("GDP")
```

- This will create a data frame containing the search results

The **eurostat** Package: An Interlude (2)

- Assume we need annual data so we download the relevant dataset

```
gdp_all <- get_eurostat("nama_10_gdp")
```

  . . . and inspect the data frame

dplyr functions

The **dplyr** functions

- Now, filter the dataset to leave only Bulgarian data:

```
    gdp_bg <- gdp_all %>%
    filter(geo == "BG")
```

- Two things to pay attention to: the double equality (stands for a logical check), and the %>% (called the *pipe* operator)
- How about if we wish to filter by more than one variable?

```
    gdp_bg <- gdp_all %>%
    filter(geo == "BG" & unit == "CLV10_MEUR" & na_item == "B1GQ")
```

The **dplyr** functions (2)

- The `time` variable does not look OK (just peculiarities of R's Date format)
- We can change this

```
    gdp_bg <- gdp_bg %>%
      mutate(time = as.integer(substr(time, 1, 4)))
```

- Now, we have columns that we don't need; we can get rid of the redundant ones by selecting only the relevant ones:

```
    gdp_bg <- gdp_bg %>%
      select(time:values)
```

- Note the colon; could also be a comma

The **dplyr** functions (3)

- We might wish to sort the data by the `time` variable in ascending order

```
    gdp_bg <- gdp_bg %>%
      arrange(time)
```

(to use descending order, write `desc(time)`

- How about if we need only the period 2000-2009 and the year 2015? We can take a slice of rows

```
    gdp_bg <- gdp_bg %>%
      slice(6:15, 21)
```

(Of course, you can also use `select` and logical relations to do the same)

The **dplyr** functions (4)

- An example with not so tidy data (from Bulgarian NSI)
- Inspect first in MS Excel, and then load into R

```r
income_by_source <- read_excel("income_by_source_en.xlsx",
    sheet = 1, skip = 2, na = c("-", "/"))
colnames(income_by_source)[1:2] <- c("region", "source")
```

- Perform some stuff to improve on the situation

```r
income_by_source <- income_by_source %>%
  mutate_if(is.character,
    str_replace_all,
    pattern = "\\(",
    replacement = "") %>%
  mutate_if(is.character,
    str_replace_all,
    pattern = "\\)",
    replacement = "") %>%
  select(region, source, contains("per capita")) %>%
  slice(1:560)
```

The **dplyr** functions (5)

- Convert some columns to numeric

```
income_by_source[,3:ncol(income_by_source)] <- sapply(income_by
    _source[,3:ncol(income_by_source)], as.numeric)
```

- Rename columns

```
colnames(income_by_source) <- c("region", "source", 2008:(ncol(
    income_by_source)+2008-3))
```

- Leave only years from 2000 onward

```
income_by_source <- income_by_source %>%
select(-contains("200"))
```

- Some more:
  https://dplyr.tidyverse.org/reference/group_by.html;
  https://dplyr.tidyverse.org/reference/summarise.html
- etc.