3. Handling Data with dplyr

Ali Ataullah and Hang Le

01/04/2020

Contents

1 Introduction

Cleaning and preparing data for analysis is one of the most time consuming aspect of research. But with right tools, the process can be made efficient and transparent. This session provides a brief introduction to a few function from the `dplyr()` package, perhaps the most widely used package to tidy data.

2 Intended Learning Outcomes

By the end of this chapter, the reader should be able to

1. use a variety of functions from the `dplyr` package to prepare data for their own analysis; and
2. save data as an `R` object that can be used for analysis at a later stage.

3 Data Wrangling with `dplyr`

As noted in the previous session, `dplyr` is perhaps the most widely used `R` package. It has several very useful functions to prepare data for analysis. `dplyr` is a part of the `tidyverse`. So, by loading `tidyverse` you also load `dplyr`. More information about `tidyverse` is available here.

```
library(tidyverse)
```

4 Our Data

We will work with two sets of data to practice functions in `dplyr`. Both data are in the `csv` format. The first is the data on employment and wages in the US from the Occupational Employment Statistics (the Bureau of Labour Statistics). Second data is for stock market indices. Before you begin, please download the two files, "oes2017.csv" and "indices.csv", in the directory that you are working on.

4.1 Example

1. We first load the data using the `read_csv()` function from the `readr` package. This package is in the `tidyverse`.

```
df1 <- read_csv(file="oes2017.csv")
```

4.2 Exercise

1. Load the stock index data "indices.csv" as `df2`.

```
# Your code
```

5 View the Raw Data

5.1 Example

1. The first step in data analysis is almost always to view the data as we would like to see what is in the data and what we need to do with it.

```
view(df1)
```

5.2 Exercise

1. View the stock index data stored as `df2`.

```
# Your code
```

GSPC is the value of S&P 500, while N100 is the Euronext index. The data is obtained from `Yahoo! Finance`.

6 Selecting Variables for Analysis

Our raw data can be quite messy. Analysts may have to spend a lot of time tidying the data before they can start preparing it for their analysis. The two two data frames that we have a quite *tidy* in the sense that each column contains a variable and each row contains an observation. Columns are also reasonably labelled.

6.1 Example

Let us work with `df1`. Suppose we need the following variables for our analysis: industry code (NAICS), occupation code (OCC_CODE), occupation title (OCC_TITLE), classification of the occupation (OCC_GROUP), total number of employees in each occupation (TOT_EMP), the number of employees in each occupation as a percentage of the total employees in the NAICS industry (PCT_TOTAL) and the average hourly wage of the employees in the occupation (H_MEAN). We can use the `select()` function from `dplyr` as follows. Note the use of `%>%`.

```
df1 <- df1 %>%
  select(NAICS,OCC_CODE,OCC_TITLE,
         OCC_GROUP, TOT_EMP, PCT_TOTAL, H_MEAN)
```

6.2 Exercise

Use `df2` and select the following columns: date, GSPC (this is S&P 500) and N100.

```
# Your code
```

GSPC is the value of S&P 500, while N100 is the Euronext index. The data is obtained from `Yahoo! Finance`.

7 Filtering Observations

7.1 Example

If you look at the data, you will see that many observations in this dataset are coded as `#` or `*`. Suppose we think that observations where `H_MEAN` is `*` or `#` should be removed. We can filter them using the `filter()` function.

```
df1 <- df1 %>%
  filter(H_MEAN != "#" & H_MEAN != "*")
```

7.2 Exercise

1. Your `df2` contains data for two stock indices. The data covers the period from 2000-12-01 to 2020-03-27. Suppose your analysis will cover time period after 2005-12-31. Use the `filter()` function to remove observation before 2007-01-01.

```
# Hint: First check the structure of the df2 using str() function.
# You will need to put date variable in quotations.
# For example "2020-03-01".


# Your code
```

8 Renaming Variables using `rename()`

8.1 Example

We regularly rename variables in our data. Let us change the labels for `H_MEAN` using the `rename()` function. It is a good idea to assign informative names to your variables.

```r
df1 <- df1 %>% rename(H_WAGE_MEAN = H_MEAN)
```

9 Exercise

Change the name of `N100` column to `EuroNext` and `GSPC` to `SP500`.

```r
# Your code
```

10 Creating New Variables using `mutate()`

10.1 Example

Suppose we want to create new, called `log_wage`, which is the natural logarithm of mean hourly wage, `H_WAGE_MEAN`. The function that we need is `mutate()`. However, we must first check if the variable is numeric.

```r
str(df1)
```

```
## Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 15809 obs. of  7 variables:
##  $ NAICS      : num  221111 221111 221111 221111 221111 ...
##  $ OCC_CODE   : chr  "00-0000" "11-0000" "11-1000" "11-1020" ...
##  $ OCC_TITLE  : chr  "Industry Total" "Management Occupations" "Top Executives" "General and Operati
##  $ OCC_GROUP  : chr  "total" "major" "minor" "broad" ...
##  $ TOT_EMP    : chr  "5,610" "250" "180" "180" ...
##  $ PCT_TOTAL  : chr  "100.00" "4.53" "3.15" "3.12" ...
##  $ H_WAGE_MEAN: chr  "34.23" "60.20" "59.23" "58.33" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   NAICS = col_double(),
##   ..   NAICS_TITLE = col_character(),
##   ..   OCC_CODE = col_character(),
##   ..   OCC_TITLE = col_character(),
##   ..   OCC_GROUP = col_character(),
```

```
##   ..   TOT_EMP = col_character(),
##   ..   EMP_PRSE = col_character(),
##   ..   PCT_TOTAL = col_character(),
##   ..   PCT_RPT = col_character(),
##   ..   H_MEAN = col_character(),
##   ..   A_MEAN = col_character(),
##   ..   MEAN_PRSE = col_character(),
##   ..   H_PCT10 = col_character(),
##   ..   H_PCT25 = col_character(),
##   ..   H_MEDIAN = col_character(),
##   ..   H_PCT75 = col_character(),
##   ..   H_PCT90 = col_character(),
##   ..   A_PCT10 = col_character(),
##   ..   A_PCT25 = col_character(),
##   ..   A_MEDIAN = col_character(),
##   ..   A_PCT75 = col_character(),
##   ..   A_PCT90 = col_character(),
##   ..   ANNUAL = col_logical(),
##   ..   HOURLY = col_logical()
##   .. )
```

The above output shows that `H_WAGE_MEAN` is a character vector. So, we will use `mutate()` along with `as.numeric()` function to convert `H_WAGE_MEAN` to log of mean hourly wage, `log_wage`. We will also convert `TOT_EMP` (total employment) and `PCT_TOTAL` (percentage of total employment) into numeric vectors.

```r
# Note how we have as.numeric() within log().

df1 <- df1 %>% mutate(log_wage = log(as.numeric(H_WAGE_MEAN)),
                    TOT_EMP = as.numeric(TOT_EMP),
                    PCT_TOTAL = as.numeric(PCT_TOTAL))
```

```
## Warning: NAs introduced by coercion
```

```
## Warning: NAs introduced by coercion
```

11 Exercise

The `lag()` function in `dplyr` can be used to obtain the lagged value of a variable. We know that the **log return** on an asset is $log(P_t/P_{t-1})$. Use the `mutate()` function and the `lag()` function to compute log return for S&P100 and EuroNext. Assign return for S&P 500 to `RtSP500` and return for EuroNext to `RtEuro`.

```
# Your code
```

It is important to note that the stock index data is arrange according to the date variable (i.e. from earliest to the latest). If the data is not arranged, you can arrange it using the `arrange()` function in `dplyr`

12 Removing `NA`

12.1 Example

The `df1` now has missing observations (why?). To remove all rows with `NA`, we use `drop_na()` function below. Please note that `drop_na()` is from a package called `tidy_r()`, which is also a part of the `tidyverse`.

```
df1 <- df1 %>% drop_na()
```

Let us check that there are no `NA`s in our data.

```
any(is.na(df1))
```

```
## [1] FALSE
```

12.2 Exercise

Remove rows with `NA` from `df2`.

13 Extracting Information from a Cell

13.1 Example

In `df1`, `NAICS` contain 5-digit industry code. Suppose we wish to create a new column that contains the first 2-digits of `NAICS`. This can be done by using the `substr()` function as follows.

```
df1 <- df1 %>%
  mutate(NAICS2 = substr(NAICS, start = 1, stop = 2))
```

13.2 Exercise

Create a new column `year` in `df2` that contains the year extracted from the `date` column.

```
# Your code
```

14 Operation on Groups using `group_by()`

14.1 Example

Sometimes we want to summarise the data by a group, for example, the mean hourly wage for each industry (NAICS). We can use the `group_by()` function.

```
df1_grouped <- df1 %>% select(NAICS, H_WAGE_MEAN) %>%
  group_by(NAICS) %>%
  summarise(grouped_mean = mean(as.numeric(H_WAGE_MEAN))) %>%
  ungroup()

# Note our use of as.numeric() - Why?
```

14.2 Exercise

Create a new column in `df2` that contains the average daily return for S&P 500 and EuroNext for each year.

```r
# Your code
```

15 Saving Data as an `R` Object

15.1 Example

We can save the data that we have prepared in R format as follows.

```r
saveRDS(df1, file = "df1.rds")
```

15.2 Exercise

Save `df2` in `rds` format.

```r
# Your code
```

16 Next Steps

Data wrangling is often a very time consuming aspect of research. But with right tools (e.g. `R`), the process can be made less painful (even enjoyable if you think about the pleasure of problem solving). This session has provided a brief introduction to data wrangling using the `dplyr` package. You will need more practice to really appreciate the value of using `R`, especially when you are dealing with large datasets. Our next step is exploratory data analysis. We will focus on visualising our data to get some preliminary insights about our potential research questions.