

ASP 460 2.0 Special topics in Statistics: Data Wrangling

13/05/2020

Packages

Install tidyverse into your computer.

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

We are going to work with datasets in EDAWR.

Installation of EDAWR is bit different. Use the following command.

Step 1

```
install.packages("devtools")
```

Step 2

```
devtools::install_github("rstudio/EDAWR")
```

Load packages

```
library(tidyverse)
```

```
## -- Attaching packages ----- tidyverse 1.3.0 --
```

```
## v ggplot2 3.3.0      v purrr   0.3.4
## v tibble  3.0.1      v dplyr   0.8.5
## v tidyr   1.0.3      v stringr 1.4.0
## v readr   1.3.1      v forcats 0.5.0
```

```
## -- Conflicts ----- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()     masks stats::lag()
```

```
library(EDAWR) # load data
```

```
##
## Attaching package: 'EDAWR'
```

```
## The following object is masked from 'package:dplyr':
##
##     storms

## The following objects are masked from 'package:tidyr':
##
##     population, who
```

Pipe operator, dplyr, and tidyr

- dplyr is a package for data wrangling, with several key verbs (functions).
- slice() and filter(): subset rows based on numbers or conditions.
- select() and pull(): select columns or a single column as a vector.
- arrange(): order rows by one or multiple columns.
- rename() and mutate(): rename or create columns.
- mutate_at(): apply a function to given columns.

Recall: Pipe operator (%>%)

See the slides in STA 326 2.0

Link: : https://hellor.netlify.app/slides/17_intro_tidyverse.html#43

```
iris %>%
  filter(Sepal.Length >= 7)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	7.0	3.2	4.7	1.4	versicolor
2	7.1	3.0	5.9	2.1	virginica
3	7.6	3.0	6.6	2.1	virginica
4	7.3	2.9	6.3	1.8	virginica
5	7.2	3.6	6.1	2.5	virginica
6	7.7	3.8	6.7	2.2	virginica
7	7.7	2.6	6.9	2.3	virginica
8	7.7	2.8	6.7	2.0	virginica
9	7.2	3.2	6.0	1.8	virginica
10	7.2	3.0	5.8	1.6	virginica
11	7.4	2.8	6.1	1.9	virginica
12	7.9	3.8	6.4	2.0	virginica
13	7.7	3.0	6.1	2.3	virginica

```
iris %>%
  filter(Sepal.Length >= 7) %>%
  head(2)
```

	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	7.0	3.2	4.7	1.4	versicolor
2	7.1	3.0	5.9	2.1	virginica

tidyr functions (They are also called tidy R verbs)

Main verbs (functions) in tidyr:

- `pivot_longer()`: makes datasets longer by increasing the number of rows and decreasing the number of columns.
- `pivot_wider()`: is the opposite of `pivot_longer()` : it makes a dataset wider by increasing the number of columns and decreasing the number of rows.
- `separate()`: splits a single column into multiple columns.
- `unite()`: combines multiple columns into a single column.

`pivot_longer()`

```
# EDAWR::cases means cases dataset in EDAWR
EDAWR::cases %>%
  head(3)
```

```
##   country 2011 2012 2013
## 1      FR 7000 6900 7000
## 2      DE 5800 6000 6200
## 3      US 15000 14000 13000
```

```
EDAWR::cases %>%
  pivot_longer(names_to = "year", values_to = "n", cols = 2:4) %>%
  head(5)
```

```
## # A tibble: 5 x 3
##   country year      n
##   <chr>   <chr> <dbl>
## 1 FR     2011  7000
## 2 FR     2012  6900
## 3 FR     2013  7000
## 4 DE     2011  5800
## 5 DE     2012  6000
```

- Here the columns 2:4 are transposed into a `year` column.
- We put the corresponding count values into a column called `n`.

Other approaches to do the same thing.

```
# Method 2
EDAWR::cases %>%
  pivot_longer(names_to = "year", values_to = "n", -country) %>%
  head(5)
```

```
## # A tibble: 5 x 3
##   country year      n
##   <chr>   <chr> <dbl>
## 1 FR      2011    7000
## 2 FR      2012    6900
## 3 FR      2013    7000
## 4 DE      2011    5800
## 5 DE      2012    6000
```

```
# Method 3
# EDAWR::cases %>%
#   pivot_longer(names_to = "year", values_to = "n", c(`2011`, `2012`, `2013`))
# Method 4
# EDAWR::cases %>%
#   pivot_longer(names_to = "year", values_to = "n", `2011`:`2013`)
```

pivot_wider(): Makes longer data formats wider.

```
EDAWR::pollution %>%
  head(5)
```

```
##       city size amount
## 1 New York large     23
## 2 New York small    14
## 3  London large     22
## 4  London small     16
## 5  Beijing large    121
```

```
EDAWR::pollution %>%
  pivot_wider(names_from = "size",
              values_from = "amount")
```

```
## # A tibble: 3 x 3
##   city      large small
##   <chr>   <dbl> <dbl>
## 1 New York     23     14
## 2 London       22     16
## 3 Beijing    121     56
```

When could I use these operations?

- Data visualization with `ggplot2`.

Read: <https://tidyr.tidyverse.org/articles/pivot.html#manual-specs>

separate()

To separate a character column into multiple columns using a regular expression separator.

The following code separates date into multiple columns. “-” is used to separate between words.

```
EDAWR::storms %>%  
  head(3)
```

```
##      storm wind pressure      date  
## 1 Alberto  110      1007 2000-08-03  
## 2   Alex   45      1009 1998-07-27  
## 3 Allison 65      1005 1995-06-03
```

```
storms2 <- EDAWR::storms %>%  
  separate(date, c("y", "m", "d"), sep="-") # sep = "-"  
storms2
```

```
## # A tibble: 6 x 6  
##   storm   wind pressure y     m     d  
##   <chr>  <int>    <int> <chr> <chr> <chr>  
## 1 Alberto  110      1007 2000  08    03  
## 2 Alex     45      1009 1998  07    27  
## 3 Allison  65      1005 1995  06    03  
## 4 Ana      40      1013 1997  06    30  
## 5 Arlene   50      1010 1999  06    11  
## 6 Arthur   45      1010 1996  06    17
```

unite()

Paste together multiple columns into one.

The following code combines y, m and d in `storm2` using “-”.

```
storms2 %>%  
  unite(date, y, m, d, sep = "-")
```

```
## # A tibble: 6 x 4  
##   storm   wind pressure date  
##   <chr>  <int>    <int> <chr>  
## 1 Alberto  110      1007 2000-08-03  
## 2 Alex     45      1009 1998-07-27  
## 3 Allison  65      1005 1995-06-03  
## 4 Ana      40      1013 1997-06-30  
## 5 Arlene   50      1010 1999-06-11  
## 6 Arthur   45      1010 1996-06-17
```

Note that `unite()` and `separate()` are inverse operations.

dplyr package

group_by()

To define a grouping of rows based on a column:

```
iris %>%  
  group_by(Species) %>%  
  head(4)
```

```
## # A tibble: 4 x 5  
## # Groups:   Species [1]  
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species  
##         <dbl>      <dbl>      <dbl>      <dbl> <fct>  
## 1         5.1         3.5         1.4         0.2 setosa  
## 2         4.9         3         1.4         0.2 setosa  
## 3         4.7         3.2         1.3         0.2 setosa  
## 4         4.6         3.1         1.5         0.2 setosa
```

```
iris %>%  
  group_by(Species) %>%  
  head(4) %>% class
```

```
## [1] "grouped_df" "tbl_df"      "tbl"        "data.frame"
```

- This doesn't actually change anything in the output.
- The only difference is that when it prints, we're told about the groups.
- But it will play a big role in how other dplyr functions work.

summarize() (American) or summarise() (British)

summarize() or summarise() in dplyr gives you single numerical summaries.

```
# Ungrouped  
iris %>%  
  summarize(Sepal.Length = mean(Sepal.Length),  
            Sepal.Width = mean(Sepal.Width))
```

```
##   Sepal.Length Sepal.Width  
## 1      5.843333      3.057333
```

```
# Grouped by number of Species  
iris %>%  
  group_by(Species) %>%  
  summarize(Sepal.Length = mean(Sepal.Length),  
            Sepal.Width = mean(Sepal.Width))
```

```
## # A tibble: 3 x 3
##   Species    Sepal.Length Sepal.Width
##   <fct>      <dbl>      <dbl>
## 1 setosa      5.01      3.43
## 2 versicolor 5.94      2.77
## 3 virginica   6.59      2.97
```

```
iris %>%
  group_by(Species) %>%
  summarize(Sepal.Width_mean = mean(Sepal.Width),
            Sepal.Width_max = max(Sepal.Width),
            Sepal.Length_mean = mean(Sepal.Length),
            Sepal.Length_max = max(Sepal.Length))
```

```
## # A tibble: 3 x 5
##   Species    Sepal.Width_mean Sepal.Width_max Sepal.Length_mean Sepal.Length_max
##   <fct>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 setosa      3.43      4.4      5.01      5.8
## 2 versicolor 2.77      3.4      5.94      7
## 3 virginica   2.97      3.8      6.59      7.9
```

ungroup()

To remove groupings structure from a data frame or a tibble.

```
iris %>%
  group_by(Species) %>%
  ungroup() %>%
  summarize(Sepal.Width = mean(Sepal.Width),
            Petal.Width = mean(Petal.Width))
```

```
## # A tibble: 1 x 2
##   Sepal.Width Petal.Width
##   <dbl>      <dbl>
## 1      3.06      1.20
```

Join operations

A “join” operation combines two data sets. There are 4 types of join operations.

- **Inner join** (or just **join**): keeps just the rows each table that match the condition.
- **Left outer join** (or just **left join**): keeps all rows in the first table, and just the rows in the second table that match the condition.
- **Right outer join** (or just **right join**): keeps just the rows in the first table that match the condition, and all rows in the second table.
- **Full outer join** (or just **full join**): keeps all rows in both tables.

Note Column values that cannot be filled in are assigned NA values.

Illustration with two simple data sets.

```
tab1_age <- data.frame(name = c("Ann", "Jenny", "Andrew"),
                        age = c(70, 52, 40),
                        stringsAsFactors = FALSE)
tab2_testresult <- data.frame(name = c("Ann", "Nick", "Anderw"),
                              result = c("negative", "positive", "negative"),
                              stringsAsFactors = FALSE)
tab1_age
```

```
##      name age
## 1    Ann  70
## 2  Jenny  52
## 3 Andrew  40
```

```
tab2_testresult
```

```
##      name  result
## 1    Ann negative
## 2   Nick positive
## 3 Anderw negative
```

inner_join()

name column is common to both tab1_age and tab2_testresult. This keeps only the common rows (intersection) in both datasets.

```
inner_join(x = tab1_age, y = tab2_testresult, by = "name")
```

```
##      name age  result
## 1    Ann  70 negative
```

left_join()

This keeps all names from tab1_age.

```
left_join(x = tab1_age, y = tab2_testresult, by = c("name" = "name"))
```

```
##      name age  result
## 1    Ann  70 negative
## 2  Jenny  52    <NA>
## 3 Andrew  40    <NA>
```

right_join()

This keeps all names from tab2_testresult.


```
right_join(x = tab1_age, y = tab2_testresult, by = "name")
```

```
##      name age  result
## 1    Ann  70 negative
## 2   Nick  NA positive
## 3 Anderw NA negative
```

full_join()

This keeps all rows from both data frames.

```
full_join(x = tab1_age, y = tab2_testresult, by = "name")
```

```
##      name age  result
## 1    Ann  70 negative
## 2   Jenny  52    <NA>
## 3 Andrew  40    <NA>
## 4   Nick  NA positive
## 5 Anderw NA negative
```

Summary

- tidyr is a package for manipulating the structure of data frames
- `pivot_longer()`: make wide data longer
- `pivot_wider()`: make long data wider
- `unite()` and `separate()`: combine or split columns
- `dplyr` has advanced functionality that mirrors SQL
- `group_by()`: create groups of rows according to a condition
- `summarize()`: apply computations across groups of rows
- `*_join()` where `*` = `inner`, `left`, `right`, or `full`: join two data frames together according to common values in certain columns, and `*` indicates how many rows to keep.