Introduction to R

Tidyverse-II

Sumit Mishra

Krea University | WSDS002

Contents

tidyr

Intro

`tidyr` helps you format the data for your analysis.

The main verbs of `tidyr` that we will discuss are: `pivot_longer()`, `pivot_wider()`, `separate()`, and `unite()`.

`pivot_longer()`

You can reshape a 'wide' format data into a long format. Let me explain this via an example[1].

```r
# create a tibble:
classDB <- tibble(name = c("Jaya", "Sushma", "Arun", "Uruj"),
                  test1 = c(12,20,14,16),
                  test2 = c(20,15,19,18),
                  midterm = c(40,47,48,50),
                  endterm = c(30,24,29,28))
```

[1] the older version of `pivot_longer()` is called `gather()`

name	test1	test2	midterm	endterm
Jaya	12	20	40	30
Sushma	20	15	47	24
Arun	14	19	48	29
Uruj	16	18	50	28

The data frame `classDB` is in wide format. There is one row per each student, and each assessment type (test, and exam) has its own column. What if you want to redo the same table such that we have one row per each assessment type?

To do this, we will use `pivot_longer()` which has three arguments: `pivot_long(data, cols, names_to = "new column name" , values_to = "value")`

- `data`: the data that you wish to reshape.
- `cols`: columns to be 'pivoted'.
- `names_to=`: the column that you want to create to store the old column names.
- `values_to=`: the column where you will store the values for the old columns.

```
classDB_long <- classDB %>%
  pivot_longer(test1:endterm,
               names_to = "assessment",  # columns stored here
               values_to = "score") %>%  # values stored here
  relocate(assessment, name) %>% #cleanup (arrange columns)
  arrange(assessment,name)       #cleanup (sort by assessment, and name)
```

assessment	name	score
endterm	Arun	29
endterm	Jaya	30
endterm	Sushma	24
endterm	Uruj	28
midterm	Arun	48
midterm	Jaya	40
midterm	Sushma	47
midterm	Uruj	50
test1	Arun	14
test1	Jaya	12
test1	Sushma	20
test1	Uruj	16
test2	Arun	19
test2	Jaya	20
test2	Sushma	15
test2	Uruj	18

`pivot_wider()`

`pivot_wider()` will transform a dataset from long to wide. Take the following example. You have data on returns for two stocks for many different dates. So, for each stock, you have multiple rows. Should you wish to transform this data such that you have one row per stock, `pivot_wider()` will come handy[2].

`pivot_wider(data, names_from = name, values_from = value)`

The main arguments of this command are:

[2] the older version of `pivot_wider()` is called `spread()`

- `data`: the data frame.
- `names_from = name`: the column that will get you the variable name.
- `values_from = value`: the column that will get you the values.

Time to do this in `R`.

```
stocks <- tibble(
  year   = c(2018, 2019, 2018, 2019),
  name    = c("X",  "X",   "Y",   "Y"),
  return = c(1.88, 0.59, 0.92, 0.17)
)
```

year	name	return
2018	X	1.88
2019	X	0.59
2018	Y	0.92
2019	Y	0.17

In the above table, there are two rows per stock. We can try to reshape the data such that we have one row per stock and returns corresponding to each year appear as separate columns.

```
stocks_wide <- stocks %>%
  pivot_wider(names_from = year, # you want years to be column-ized
              values_from = return)
```

name	2018	2019
X	1.88	0.59
Y	0.92	0.17

Here's a GIF that summarizes `pivot()` functions[3].

```
## Please check the HTML version to view this GIF.
```

`separate()`

A problem that people who work with data routinely encounter is that of a column containing information that can be spread out into many columns. For example, consider the university address: `5655 Central Expressway, Sri City, Chittoor, Pincode:517646`. This address contains four different variables: `road`, `area`, `district`, and `pincode`. We will use `separate()` to extract these variables. The function has the following arguments:

- `data`: the data frame
- `col` = the column name
- `into` = fill in the new column names
- `sep = "[^[:alnum:]]+"`: choose the separator (comma, semicolon, etc.)
- `remove = TRUE` (will remove the column by default)
- `convert = FALSE` (will preserve the column type)

```
address <- "5655 Central Expressway, Sri City, Chittoor, Pincode:517646"
# create a tibble
```

[3] This and other GIFs in this set of notes are from Garrick Aden-Buie's excellent `tidyexplain` package. Link: https://github.com/gadenbuie/tidyexplain

```
uni.address <- tibble(name = "IFMR", address = address)
print(uni.address)
```

```
## # A tibble: 1 x 2
##   name  address
##   <chr> <chr>
## 1 IFMR  5655 Central Expressway, Sri City, Chittoor, Pincode:517646
```

```
# use separate to create the columns
uni.address %>% separate(address,
                         into = c("road", "area", "district", "pincode"), sep = ",")
```

```
## # A tibble: 1 x 5
##   name  road                    area        district    pincode
##   <chr> <chr>                   <chr>       <chr>       <chr>
## 1 IFMR  5655 Central Expressway " Sri City" " Chittoor" " Pincode:517646"
```

The column `address` has been broken into four new variables. Notice that the column itself has gone missing in the process. If you wish to retain the variable, you should add `remove = F` to the command.

`unite()`

`unite()` does the exact opposite of what `separate()` does: it combines multiple columns into one column. Imagine that you have a database with columns for the first names and the last names of students at IFMR. `unite()` will help you create a new variable for the full names of the students.

```
itr_stu <- tibble(first.name = c("Gunjan","Rehan","Simran"),
                  last.name  = c("Agarwal", "Asdaque", "Heerekar"),
                  rollno = c("062","087","156"))
print(itr_stu)
```

```
## # A tibble: 3 x 3
##   first.name last.name rollno
##   <chr>      <chr>     <chr>
## 1 Gunjan     Agarwal   062
## 2 Rehan      Asdaque   087
## 3 Simran     Heerekar  156
```

```
# create a new column called full.name

itr_stu %>% unite(full.name, c(first.name, last.name))
```

```
## # A tibble: 3 x 2
##   full.name       rollno
##   <chr>           <chr>
## 1 Gunjan_Agarwal  062
## 2 Rehan_Asdaque   087
## 3 Simran_Heerekar 156
```

We see that we have been able to create a new column called `full.name`, but the names contain underscore, and the old columns are all gone. We can add the option `sep = " "` and define a separator. We can also keep the old columns by adding the argument `remove = F` into the command.

```
itr_stu %>% unite(full.name, c(first.name, last.name),
                  sep = " ",
                  remove = F)
```

```
## # A tibble: 3 x 4
##   full.name       first.name last.name rollno
##   <chr>           <chr>      <chr>     <chr>
## 1 Gunjan Agarwal  Gunjan     Agarwal   062
## 2 Rehan Asdaque   Rehan      Asdaque   087
## 3 Simran Heerekar Simran     Heerekar  156
```

dplyr Revisited

Finally, we wrap this section up by learning how to merge two datasets in `R`[4]. The material that we need to combine any two data frames is known as keys. Keys are nothing but the common columns in the two datasets that you want to match. For any two datasets `df1` and `df2` and a key `var`, we can do

- `inner_join(df1, df2, by = var)`: keep the matched rows.
- `left_join(df1, df2, by = var)`: keep the matched rows plus the unmatched rows from `df1`
- `right_join(df1, df2, by = var)`: keep the matched rows plus the unmatched rows from `df2`.
- `full_join()`: keep all the rows from the two datasets.

Example

Dataset 1 (Columns: `ID` and `x`)

```
df1
```

```
## # A tibble: 5 x 2
##      ID x
##   <int> <chr>
## 1     1 x1
## 2     2 x2
## 3     3 x3
## 4     4 x4
## 5     5 x5
```

Dataset 2 (Columns: `ID` and `y`)

```
df2
```

```
## # A tibble: 5 x 2
##      ID y
##   <dbl> <chr>
## 1     2 y1
## 2     4 y2
## 3     6 y3
## 4     8 y4
## 5    10 y5
```

We will use the column `ID` as the key to match the two data frames.

`inner_join()`

```
inner_join(df1,df2, by = "ID")
```

```
## # A tibble: 2 x 3
##      ID x     y
##   <dbl> <chr> <chr>
## 1     2 x2    y1
## 2     4 x4    y2
```

```
## Please check the HTML version to view this GIF.
```

Check the dimension of the matched data frame. There are two rows, and three columns. The original data frames had five rows apiece, but only two of those five matched on the variable `ID`.

[4] `merge()` command will also do the job for you, but I am going to stick to dplyr verbs in this course

`left_join()`

```
## Please check the HTML version to view this GIF.
```

```r
left_join(df1,df2, by = "ID")
```

```
## # A tibble: 5 x 3
##      ID x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    <NA> 
## 2     2 x2    y1   
## 3     3 x3    <NA> 
## 4     4 x4    y2   
## 5     5 x5    <NA> 
```

Check the dimension of the matched data frame. There are five rows, and three columns, but notice that the new data frame retains all the rows from `df1` and fills the umatched rows for the column `y` as NAs.

`right_join()`

```
## Please check the HTML version to view this GIF.
```

```r
right_join(df1,df2, by = "ID")
```

```
## # A tibble: 5 x 3
##      ID x     y    
##   <dbl> <chr> <chr>
## 1     2 x2    y1   
## 2     4 x4    y2   
## 3     6 <NA>  y3   
## 4     8 <NA>  y4   
## 5    10 <NA>  y5   
```

Check the dimension of the matched data frame. There are five rows, and three columns, but notice that the new data frame retains all the rows from `df2` and fills the umatched rows for the column `x` as NAs.

`full_join()`

```
## Please check the HTML version to view this GIF.
```

```r
full_join(df1,df2, by = "ID")
```

```
## # A tibble: 8 x 3
##      ID x     y    
##   <dbl> <chr> <chr>
## 1     1 x1    <NA> 
## 2     2 x2    y1   
## 3     3 x3    <NA> 
## 4     4 x4    y2   
## 5     5 x5    <NA> 
## 6     6 <NA>  y3   
## 7     8 <NA>  y4   
## 8    10 <NA>  y5   
```

Check the dimension of the matched data frame. There are eight rows, and three columns, but notice that the new data frame retains all the rows from `df1` as well as those from `df2`, filling NAs in the unmatched rows for both `x` and `y`.

Real-life datasets hardly ever come with nicely-defined keys as seen in the example above. A good deal of time is usually spent on creating or identifying the key while trying to combine datasets.

Done for the day

```
## Sorry, this silly GIF is only available in the the HTML version of the notes.
```