R: Tidy Data & Data Import 2

Sakol Suethanapornkul

23 December 2020

Quiz

Import Item.csv file into R. Some things you need to consider:

[1] The file doesn't contain column names;

[2] The first column, which looks like IDs, should be treated as characters;

[3] The rest should be doubles.

Answers

Here are two possible options to import Item.csv:

```r
library(tidyverse)

item <- read_csv(file      = "./Data/Item.csv",
                 col_names = FALSE,
                 col_types = cols(
                   .default = col_double(),
                   X1       = col_character()
                   )
                 )
```

Answers

Here are two possible options to import Item.csv:

```r
item <- read_delim(file      = "./Data/Item.csv",
                   delim     = ",",
                   col_names = FALSE,
                   col_types = cols(
                     .default = col_double(),
                     X1       = col_character()
                   )
                 )
```

NOTE: You can specify column names by providing a vector of column names to `col_names`.

Answers

Here is an example:

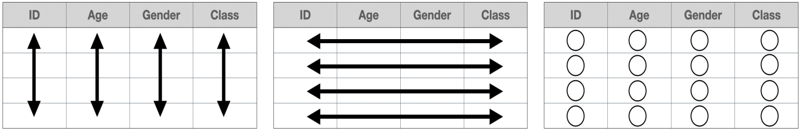
```
column <- c("id",
            "i1_1", "i1_2", "i1_3", "i1_4",
            "i2_1", "i2_2", "i2_3", "i2_4",
            "i3_1", "i3_2", "i3_3", "i3_4",
            "i4_1", "i4_2", "i4_3", "i4_4",
            "i5_1", "i5_2", "i5_3", "i5_4")

item <- read_csv(file      = "./Data/Item.csv",
                 col_names = column,
                 col_types = cols(
                     .default = col_double(),
                     id       = col_character()
                     )
                 )
```

Recap

Principles of tidy data:

- Each variable must have its own column;
- Each observation must have its own row; and
- Each value must have its own cell.



Credit: R for Data Science

Part I: `pivot_longer()`

EXERCISE: Let's take a close look at the data frame `item`. Is it tidy? Why or why not?

Part I: `pivot_longer()`

ANSWER: The answer is no.

Perhaps the biggest problem in `item` is the fact that one variable takes up more than one column.

Luckily, we can “tidy” this data frame with two functions: `pivot_longer()` and `pivot_wider()`

NOTE: You may have seen `gather()` and `spread()`. These two are now superseded by `pivot_longer()` and `pivot_wider()`, respectively.

Part I: pivot_longer()

For now, let's look at columns i1_1 through to i1_4:

```r
item %>%
  select(id:i1_4)
```

These four columns are in fact values of another variable (e.g., questions, items, tests, etc.)!

Part I: `pivot_longer()`

So we need to combine these four columns. This can be done easily with `pivot_longer()`. This function has four main arguments:

```
pivot_longer(data = ,          cols = ,
             names_to = " ",    values_to = " "
             )
```

Part I: pivot_longer()

So we need to combine these four columns. This can be done easily with `pivot_longer()`. This function has four main arguments:

```
pivot_longer(data = ,              cols = ,
             names_to = " ",     values_to = " "
             )
```

So, let's give `pivot_longer()` the information it needs:

```
item %>%
  select(id:i1_4) %>%
  pivot_longer(cols = i1_1:i1_4,
               names_to  = "items",
               values_to = "points"
               )
```

Part I: pivot_longer()

We can select columns to pivot using dplyr::select style. So, this also works:

```
item %>%
  select(id:i1_4) %>%
  pivot_longer(cols = !id,
               names_to  = "items",
               values_to = "points"
               )
```

EXERCISE: What are some other ways to pick these columns?

Part I: `pivot_longer()`

Let's look back at how we name our columns:

```r
item %>%
  select(id:i1_4)
```

Each item starts with "i" (which stands for "item"), followed by question numbers and options.

Now that `i1_1`, `i1_2` are values of the variable `items`, we run into one problem: These two numbers code different information.

Part I: `pivot_longer()`

We can “break” these values (e.g., `i1_1`, `i1_2`) into smaller parts with the help of the following arguments:

```r
item %>%
  select(id:i1_4) %>%
  pivot_longer(cols = !id,
               names_to     = c("items", "choices"),
               names_sep    = "_",
               names_prefix = "i",
               values_to    = "points"
               )
```

NOTE: `names_prefix` is used to remove matching strings from the start of a variable name.

Part I: pivot_longer()

A slightly more complex take on the previous slide, with names_pattern:

```r
item %>%
  select(id:i1_4) %>%
  pivot_longer(cols = !id,
               names_to      = c("items", "choices"),
               names_pattern = "i(.)_(.)",
               values_to     = "points"
               )
```

Part I: `pivot_longer()`

But if this is a bit too much for you, there's always a helper. `seperate()` does the trick:

```r
item_sm <- item %>%
  select(id:i1_4) %>%
  pivot_longer(cols = !id,
               names_to      = "items",
               values_to     = "points"
               ) %>%
  separate(col  = items,
           into = c("questions", "options") )
```

Part I: pivot_longer()

The opposite of separate() is of course unite():

```r
item_sm %>%
  unite(col = "items", c(questions, options),
        sep = "."
        )
```

Part II: pivot_wider()

While pivot_longer() lengthens data, pivot_wider() does the opposite. It “widens” data! Previously:

```
item_ln <- item %>%
  select(id:i1_4) %>%
  pivot_longer(cols = !id,
               names_to  = "items",
               values_to = "points")
```

Now, how can we widen item_ln which is in a long format?

Part II: pivot_wider()

We can use pivot_wider() to do that job:

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

So, we can revert item_ln back to its original form:

```
item_ln %>%
  pivot_wider(names_from  = items,
              values_from = points)
```

We use a <tidy-select> style to select columns (without a quotation mark).

Part II: pivot_wider()

Like pivot_longer(), the use case of pivot_wider() is much more than what you just saw!

Previously:

```
item_ln <- item %>%
  select(id:i1_4) %>%
  pivot_longer(cols = !id,
               names_to     = c("items", "choices"),
               names_sep    = "_",
               names_prefix = "i",
               values_to    = "points"
               )
```

Part II: pivot_wider()

We can combine the two columns items and choices in names_from. names_glue tells pivot_wider() how we want the two columns to be combined:

```
item_ln %>%
  pivot_wider(names_from  = c(items, choices),
              values_from = points,
              names_glue = "i{items}_{choices}"
              )
```

Part II: pivot_wider()

Let's look at a slightly more challenging case with an existing data frame that comes with the tidyverse package:

```
head(us_rent_income, n = 6)
```

```
## # A tibble: 6 x 5
##   GEOID NAME    variable estimate   moe
##   <chr> <chr>   <chr>       <dbl> <dbl>
## 1 01    Alabama income      24476   136
## 2 01    Alabama rent          747     3
## 3 02    Alaska  income      32940   508
## 4 02    Alaska  rent         1200    13
## 5 04    Arizona income      27517   148
## 6 04    Arizona rent          972     4
```

Part II: pivot_wider()

Suppose we want to widen variable with values from estimate and moe, we can do provide both columns to values_from:

```r
us_rent_income %>%
  pivot_wider(names_from  = variable,
              values_from = c(estimate, moe)
              )
```

Notice that pivot_wider() makes the column names by joining the two parts together for us.

Part II: pivot_wider()

We can change how the column names are created with our friend names_glue:

```r
us_rent_income %>%
  pivot_wider(names_from  = variable,
              values_from = c(estimate, moe),
              names_glue  = "{variable}.{.value}"
              )
```

Use .value to reference each of the two columns we provide to values_from.

What's the point in all of these?

Let's illustrate this with `geom_point()`:

```r
head(us_rent_income, n = 6)

us_rent_income %>%
  ggplot(aes(x = estimate, y = moe)) +
  geom_point()
```

This plot visualizes income and rent on the same plot because, for example, the column `estimate` happens to contain both sets of information. Ask yourself: Is this what you want?

What's the point in all of these?

If all we ever want is to plot the relationships between estimate and moe of income, we can certainly go this route:

```
us_rent_income %>%
  filter(variable == "income") %>%
  ggplot(aes(x = estimate, y = moe)) +
  geom_point()
```

What's the point in all of these?

But what if we wanted to depict the relationships between the estimates of income and those of rent? This means we need one column for `income.estimates` and the for `rent.estimates`:

```r
us_rent_income %>%
  pivot_wider(names_from  = variable,
              values_from = c(estimate, moe),
              names_glue  = "{variable}.{.value}") %>%
  ggplot(aes(x = income.estimate, y = rent.estimate)
         ) +
  geom_point()
```

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.

```r
income <- read_csv(file = "./Data/Income.csv")
```

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.

```
income_sm <- income %>%
  select(!c(‘1800‘:‘1979‘)
         ) %>%
  filter(country %in% c("China", "Japan", "Malaysia",
                        "Singapore", "Thailand")
         )
```

NOTE: Since our column names do not begin with letters, we need back ticks (‘ ‘)

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.
[4] Pivot columns of years to a new variable "`year`" and values of GDP to "`gdp`"

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.
[4] Pivot columns of years to a new variable "`year`" and values of GDP to "`gdp`"

```r
income_ln <- income_sm %>%
  pivot_longer(cols = !country,
               names_to = "year", values_to = "gdp")
```

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.
[4] Pivot columns of years to a new variable "`year`" and values of GDP to "`gdp`"
[5] **Extra** Create a line plot for each country.

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.
[4] Pivot columns of years to a new variable "`year`" and values of GDP to "`gdp`"
[5] **Extra** Create a line plot for each country.

```
income_ln %>%
  ggplot(aes(x = year, y = gdp)) +
  geom_line(aes(group = country,
                color = country) )
```

Exercise

Let's return to Income.csv. We will work through this exercise slowly.

[1] Import Income.csv and name the data frame as `income`.
[2] Create a new data frame consisting of country names and years since 1980.
[3] Select China, Japan, Malaysia, Singapore, and Thailand.
[4] Pivot columns of years to a new variable "`year`" and values of GDP to "`gdp`"
[5] **Extra** Create a line plot for each country.
[6] **Extra** Widen the data frame `income_ln` back to its original form.