Data Summarization and Transformation - 2

One should look for what is and not what he thinks should be. (Albert Einstein)

Module completion checklist

Topic	Complete
Transform messy data to tidy data using tidyr package	
Manipulate columns by using the separate and unite functions	

Using tidyr to tidy data

- Tidy data is defined by three main properties:
  - Each **variable** must have its own **column**
  - Each **observation** must have its own **row**
  - Each **value** must have its own **cell**
- Using the tidyr package can help us tidy our wrangled data
- The most important functions for restructuring rows and columns involve a `pivot`

pivot_longer()

- `pivot_longer()` pulls multiple columns into one new variable

```
?tidyr::pivot_longer

pivot_longer(cols,          #<- columns to convert
             names_to,      #<- name of new key
column
             values_to)     #<- name of new value
column
```

pivot_longer {tidyr} R Documentation

Pivot data from wide to long

Description

lifecycle maturing

`pivot_longer()` "lengthens" data, increasing the number of rows and decreasing the number of columns. The inverse transformation is `pivot_wider()`

Learn more in `vignette("pivot")`.

Usage

```
pivot_longer(
  data,
  cols,
  names_to = "name",
  names_prefix = NULL,
  names_sep = NULL,
  names_pattern = NULL,
  names_ptypes = list(),
  names_transform = list(),
  names_repair = "check_unique",
  values_to = "value",
  values_drop_na = FALSE,
  values_ptypes = list(),
  values_transform = list(),
  ...
)
```

pivot_longer parameters

We need three parameters for `pivot_longer()`:

- The **set of columns** that represent the values
- The **name of the variable** (which we decide upon) that represents those values, or the `key`
- The **name of the variable** (which we decide upon) that represents the values that are currently within the value columns, or the `value`

pivot_longer example

- Let's look at `table4a` data from the WHO tuberculosis dataset

```
table4a
```

```
# A tibble: 3 x 3
  country     `1999` `2000`
* <chr>        <int>  <int>
1 Afghanistan    745   2666
2 Brazil       37737  80488
3 China       212258 213766
```

- Notice that the second and third column are actually both values (1999 and 2000) of a single hypothetical variable, year
- We can use pivot_longer to combine these two columns into a single column, year
- We can also create a new column, cases, to contain the count that currently appears in the 1999 and 2000 columns

pivot_longer example contd

- Let's pivot the `table4a` data frame from a wide format to a long format

```
# Gather the `table4a` data frame to make it tidy.
table4a %>%    #<- set the data frame and use pipe to use it as input into `pivot_longer`
  pivot_longer(cols = `1999`:`2000`, #<- set columns to convert
               names_to = "year",    #<- set `year` column as a key
               values_to = "cases")  #<- set `cases` column for the values
```

```
# A tibble: 6 x 3
  country     year   cases
  <chr>       <chr>  <int>
1 Afghanistan 1999     745
2 Afghanistan 2000    2666
3 Brazil      1999   37737
4 Brazil      2000   80488
5 China       1999  212258
6 China       2000  213766
```

- **Reminder:** You can use the pipe (`%>%`) in all the packages within `tidyverse`!

pivot_longer example specifying a range

- Let's modify the same example, this time using the colon syntax to specify range
- Note that the code substitutes `2:3` instead of column names

```
# Gather the `table4a` data frame to make it tidy.
table4a %>%    #<- set the data frame and use pipe to use it as input into `pivot_longer`
  pivot_longer(cols = 2:3,           #<- set columns to convert
               names_to = "year",    #<- set `year` column as a key
               values_to = "cases")  #<- set `cases` column for the values
```

```
# A tibble: 6 x 3
  country     year   cases
  <chr>       <chr>  <int>
1 Afghanistan 1999     745
2 Afghanistan 2000    2666
3 Brazil      1999   37737
4 Brazil      2000   80488
5 China       1999  212258
6 China       2000  213766
```

pivot_wider()

- You use `pivot_wider()` when an observation is scattered across multiple rows
- `pivot_wider()` spreads one column into multiple variables
- `pivot_wider()` is the opposite of `pivot_longer()`

```
?tidyr::pivot_wider()

pivot_wider(names_from,    #<- name of current key column
            values_from)   #<- name of current value column
```

pivot_wider {tidyr} R Documentation

Pivot data from long to wide

Description

lifecycle maturing

`pivot_wider()` "widens" data, increasing the number of columns and decreasing the number of rows. The inverse transformation is `pivot_longer()`.

Learn more in `vignette("pivot")`.

Usage

```
pivot_wider(
  data,
  id_cols = NULL,
  names_from = name,
  names_prefix = "",
  names_sep = "_",
  names_glue = NULL,
  names_sort = FALSE,
  names_repair = "check_unique",
  values_from = value,
  values_fill = NULL,
  values_fn = NULL,
  ...
)
```

pivot_wider parameters

There are two parameters we need to pay attention to when using `pivot_wider`:

- The **column** that contains the variable names, the `names_from` column
- The **column** that contains the values for the multiple variables, the `values_from` column

pivot_wider example

- Use `table2` as the initial data frame
- The type column currently contains multiple categories of information, `cases` and `population`
- To reshape and tidy this table, we might create a `cases` column and a `population` column

```
# Let's look at `table2` inbuilt data.
table2
```

```
# A tibble: 12 x 4
   country      year type            count
   <chr>       <int> <chr>           <int>
 1 Afghanistan  1999 cases             745
 2 Afghanistan  1999 population   19987071
 3 Afghanistan  2000 cases            2666
 4 Afghanistan  2000 population   20595360
 5 Brazil       1999 cases           37737
 6 Brazil       1999 population  172006362
 7 Brazil       2000 cases           80488
 8 Brazil       2000 population  174504898
 9 China        1999 cases          212258
10 China        1999 population 1272915272
11 China        2000 cases          213766
12 China        2000 population 1280428583
```

pivot_wider example contd

- Let's pivot the `table2` data frame from a long format to a wide format
- Use `pivot_wider()` with 2 main parameters:
  - The `type`, which contains the variables
  - The `count`, which contains the values for each of the rows of the variables in the `type` column

```
# Spread the data
# Pass data to spread with pipe.
table2 %>%
  pivot_wider(names_from = type,
              values_from = count)
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

Module completion checklist

Topic	Complete
Transform messy data to tidy data using tidyr package	✔
Manipulate columns by using the separate and unite functions	

Separating and uniting

- We can also use **tidyr** to address a single variable, a single column, containing multiple pieces of information

```
table3
```

```
# A tibble: 6 x 3
  country      year rate
* <chr>       <int> <chr>
1 Afghanistan  1999 745/19987071
2 Afghanistan  2000 2666/20595360
3 Brazil       1999 37737/172006362
4 Brazil       2000 80488/174504898
5 China        1999 212258/1272915272
6 China        2000 213766/1280428583
```

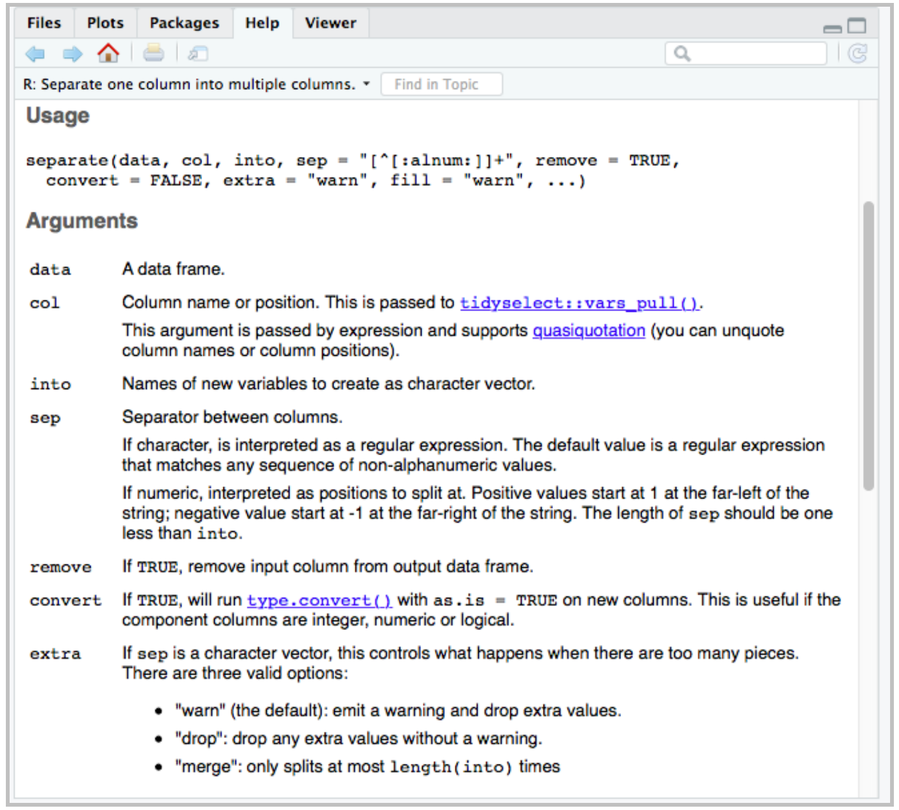
- For instance, `table3` shows both cases and population as a single rate
- We can use the function `separate()` to extract the values into separate columns
- We will also practice using the complementary function, `unite()`

separate() parameters

- `separate()` separates a single character column into multiple columns and takes two arguments:
  - The column to be separated
  - What to separate the variable into, like into = c("var_1", "var_2")

```
?tidyr::separate

separate(df,    #<- data frame
         col,   #<- name of column to separate
         into)  #<- name of new variables to
                #   create as a character vector
```



separate example

- Let's try out `separate()` on `table3` to create `cases` and `population` columns using the pipe operator

```
# Using `table3` separate its `rate` column into two.
table3 %>%                    #<- set data frame and pass it to next function with pipe
  separate(rate,                    #<- separate `rate`
           into = c("cases",        #<- into column `cases`, and
                    "population")) #<-      column `population`
```

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <int> <chr>  <chr>
1 Afghanistan  1999 745    19987071
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

separate example using `sep`

- By default, `separate()` will operate on any non alpha-numeric character
- However, it is also possible to specify the character by which to split using the `sep=` parameter

```
# Using `table3` separate its `rate` column into two.
table3 %>%
  separate(rate,
           into = c("cases",
                    "population"),
           sep = "/")              #<- set the separating character to `/`
```

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <int> <chr>  <chr>
1 Afghanistan  1999 745    19987071
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

separate example using `sep`

- Next, let's use the `sep` parameter to separate the year column on the **character index**
- If we split the `year` column at index 2, we can create two new columns: `century` and `year`

```
# Using `table3` separate its `year` column into two.
table3 %>%
  separate(year,                #<- separate `year`
           into= c("century",  #<- into two columns: `century`, and
                   "year"),    #<-                    `year`
           sep = 2)            #<- set the separator at index = 2
```

```
# A tibble: 6 x 4
  country     century year  rate
  <chr>       <chr>   <chr> <chr>
1 Afghanistan 19      99    745/19987071
2 Afghanistan 20      00    2666/20595360
3 Brazil      19      99    37737/172006362
4 Brazil      20      00    80488/174504898
5 China       19      99    212258/1272915272
6 China       20      00    213766/1280428583
```

Data type conversions using separate()

- When using `separate()`, the data type of the original column will be preserved

```
# The new columns
# are now also characters.
table3 %>%
  separate(rate, into = c("cases", "population"))
```

```
# A tibble: 6 x 4
  country      year cases  population
  <chr>       <int> <chr>  <chr>
1 Afghanistan  1999 745    19987071
2 Afghanistan  2000 2666   20595360
3 Brazil       1999 37737  172006362
4 Brazil       2000 80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

- However, `separate()` can be instructed to convert to what it thinks the data types of new columns should be

```
table3 %>%
  separate(rate, into = c("cases", "population"),
convert = TRUE)
```

```
# A tibble: 6 x 4
  country      year  cases population
  <chr>       <int>  <int>      <int>
1 Afghanistan  1999    745   19987071
2 Afghanistan  2000   2666   20595360
3 Brazil       1999  37737  172006362
4 Brazil       2000  80488  174504898
5 China        1999 212258 1272915272
6 China        2000 213766 1280428583
```

unite()

- `unite()` combines multiple character columns into a single column
- `unite()` is the inverse of separate

```
?tidyr::unite

unite(df,  #<- data frame
      col, #<- name of column to unite
      sep) #<- separator to use
```

unite {tidyr} R Documentation

Unite multiple columns into one by pasting strings together

Description

Convenience function to paste together multiple columns into one.

Usage

```
unite(data, col, ..., sep = "_", remove = TRUE, na.rm = FALSE)
```

Arguments

`data`	A data frame.
`col`	The name of the new column, as a string or symbol. This argument is passed by expression and supports quasiquotation (you can unquote strings and symbols). The name is captured from the expression with `rlang::ensym()` (note that this kind of interface where symbols do not represent actual objects is now discouraged in the tidyverse; we support it here for backward compatibility).
`...`	<`tidy-select`> Columns to unite
`sep`	Separator to use between values.
`remove`	If `TRUE`, remove input columns from output data frame.
`na.rm`	If `TRUE`, missing values will be remove prior to uniting each value.

unite parameters

- `unite()` separates a single character column into multiple columns and takes two arguments:
  - The name of the new column
  - Columns to unite, ex c(“var_1”, “var_2”)

unite example

- To demonstrate how `unite()` works, let's undo the separation of century and year in `table3`

```
# Let's separate the `table3`'s `year` column
into `century` and `year` first.
ex_table = table3 %>%
  separate(year, into = c("century", "year"), sep
= 2)
ex_table
```

```
# A tibble: 6 x 4
  country     century year  rate
  <chr>       <chr>   <chr> <chr>
1 Afghanistan 19      99    745/19987071
2 Afghanistan 20      00    2666/20595360
3 Brazil      19      99    37737/172006362
4 Brazil      20      00    80488/174504898
5 China       19      99    212258/1272915272
6 China       20      00    213766/1280428583
```

```
# Now we use `unite` to combine the two new
columns back into one.
# By default, unite will combine columns using
`_` so we can use `sep` to specify that we
# do not want anything between the two columns
when combined into one cell.
ex_table %>%      #<- specify the data frame to
pipe into `unite`
  unite(time,     #<- set the column `time` for
combined values
        century,  #<- 1st column to unite
        year,     #<- 2nd column to unite
        sep = "") #<- set the separator to an
empty string
```

```
# A tibble: 6 x 3
  country     time  rate
  <chr>       <chr> <chr>
1 Afghanistan 1999  745/19987071
2 Afghanistan 2000  2666/20595360
3 Brazil      1999  37737/172006362
4 Brazil      2000  80488/174504898
5 China       1999  212258/1272915272
6 China       2000  213766/1280428583
```

Knowledge check

Exercise

You are now ready to try Tasks 4-9 in the Exercise for this topic

Module completion checklist

Topic	Complete
Transform messy data to tidy data using tidyr package	✔
Manipulate columns by using the separate and unite functions	✔

Data Summarization and Transformation: Topic summary

In this part of the course, we have covered:

- Tidy data best practices
- Transform data with tidyr

Congratulations on completing this module!