The verb `separate` is used to split a compound column into two or more variables via a given separator, which is a character string that tells the function where to make the split.

If you haven't already, open your project in RStudio and then load the messy TB notifications data (instructions were provided at the beginning of the Gather (https://www.futurelearn.com/courses/data-science-wrangling-and-workflow/1/steps/525040) section). You can then make a start on the following exercise.

Give it a go!

There's more to keeping it tidy. In the previous steps, you transformed the messy tuberculosis (TB) notifications data into long form using **`gather`**, but it's still **not** completely tidy.

To make a start, copy and then run the following code chunk in RStudio.

```
tb_long <- gather(tb_messy,
                  key = "sex_agegroup", value = "count",
                  -country, -year, -iso3)
tb_long
```

```
## # A tibble: 157,820 x 5
##    country     iso3   year sex_agegroup count
##    <chr>       <chr> <dbl> <chr>        <dbl>
##  1 Afghanistan AFG    1980 new_sp_m04      NA
##  2 Afghanistan AFG    1981 new_sp_m04      NA
##  3 Afghanistan AFG    1982 new_sp_m04      NA
##  4 Afghanistan AFG    1983 new_sp_m04      NA
##  5 Afghanistan AFG    1984 new_sp_m04      NA
##  6 Afghanistan AFG    1985 new_sp_m04      NA
##  7 Afghanistan AFG    1986 new_sp_m04      NA
##  8 Afghanistan AFG    1987 new_sp_m04      NA
##  9 Afghanistan AFG    1988 new_sp_m04      NA
## 10 Afghanistan AFG    1989 new_sp_m04      NA
## # ... with 157,810 more rows
```

To make it completely tidy, you'll need to separate the column **"sex_agegroup"** into **two** columns.

Using separate

By default, the separator is all the characters that are not numbers or letters. You can use **`separate`** to split the type of TB diagnosis method from **sex** and **agegroup**.

This will requires you to use a regular expression (https://en.wikipedia.org/wiki/Regular_expression), which tells the function where to make the split. The following code chunk will separate **"sex_agegroup"**.

When you're ready, copy the code chunk and then run it.

```
tb_long2 <- separate(tb_long, "sex_agegroup",
                     c("type", "sex_agegroup"),
                     sep = "_(?=[mf])"
)
tb_long2
```

```
## # A tibble: 157,820 x 6
##    country     iso3   year type   sex_agegroup count
##    <chr>       <chr> <dbl> <chr>  <chr>        <dbl>
##  1 Afghanistan AFG    1980 new_sp m04             NA
##  2 Afghanistan AFG    1981 new_sp m04             NA
##  3 Afghanistan AFG    1982 new_sp m04             NA
##  4 Afghanistan AFG    1983 new_sp m04             NA
##  5 Afghanistan AFG    1984 new_sp m04             NA
##  6 Afghanistan AFG    1985 new_sp m04             NA
##  7 Afghanistan AFG    1986 new_sp m04             NA
##  8 Afghanistan AFG    1987 new_sp m04             NA
##  9 Afghanistan AFG    1988 new_sp m04             NA
## 10 Afghanistan AFG    1989 new_sp m04             NA
## # ... with 157,810 more rows
```

This creates two columns: one called **"type"** that contains the string *new_sp* and another called **"sex_agegroup"** that contains the sex and age group (as a combined string). The regular expression tells the function to make the split at the underscore that is followed by the characters **"m"** or **"f"**. The underscore is removed from the values, but the **"m"** and **"f"** are retained. Clearly, there's more work to do as we need to split up the remaining column into **"sex"** and **"age_group"**.

Using 'extract'

A companion verb to `separate` is `extract`. It works similarly to `separate` but allows you to split columns when there is **not** an obvious separating character.

In this case, we use a regular expression that contains two groups; the **first** says match characters **"m"** or **"f"** and the **second** says match anything that comes after.

This results in the new **"sex"** column containing either **"m"** or **"f"**, and the **"age_group"** variable representing the age categories.

```
tb_long3 <- extract(tb_long2,
                    "sex_agegroup",
                    c("sex", "age_group"),
                    regex = "([mf])(.*)")
tb_long3
```

```
## # A tibble: 157,820 x 7
##    country     iso3   year type   sex   age_group count
##    <chr>       <chr> <dbl> <chr>  <chr> <chr>     <dbl>
##  1 Afghanistan AFG    1980 new_sp m     04           NA
##  2 Afghanistan AFG    1981 new_sp m     04           NA
##  3 Afghanistan AFG    1982 new_sp m     04           NA
##  4 Afghanistan AFG    1983 new_sp m     04           NA
##  5 Afghanistan AFG    1984 new_sp m     04           NA
##  6 Afghanistan AFG    1985 new_sp m     04           NA
##  7 Afghanistan AFG    1986 new_sp m     04           NA
##  8 Afghanistan AFG    1987 new_sp m     04           NA
##  9 Afghanistan AFG    1988 new_sp m     04           NA
## 10 Afghanistan AFG    1989 new_sp m     04           NA
## # ... with 157,810 more rows
```

In the next step, you'll continue wrangling this data using more verbs from the `tidyverse`.

Tell us how you went

Within the **Comments**, share with other learners your experience using the code chunks on this step.

In what way do you think you could use `separate` and `extract` to tidy data from a project you're working on?

Don't forget to contribute to the discussion by reviewing the comments made by other learners, making sure you provide constructive feedback and commentary.

Find out more

If you'd like to find out more about writing regular expressions, consider exploring RegExr (https://regexr.com/) or the `stringr` package (https://stringr.tidyverse.org/) for additional resources on how to manipulate text. We hope you find them useful.
