Mutating columns

Intro

You now know how to keep or drop columns and rows from your dataset. Today you will learn how to modify existing variables or create new ones, using the `mutate()` verb from {dplyr}. This is an essential step in most data analysis projects.

Let's go!

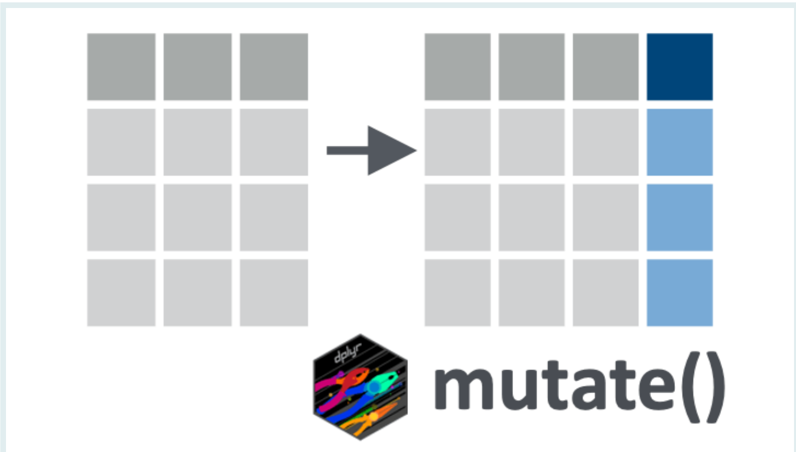


Fig: the `mutate()` verb.

Learning objectives

1. You can use the `mutate()` function from the `{dplyr}` package to create new variables or modify existing variables.

2. You can create new numeric, character, factor, and boolean variables

Packages

This lesson will require the packages loaded below:

```
if(!require(pacman)) install.packages("pacman")
pacman::p_load(here,
               janitor,
               tidyverse)
```

Datasets

In this lesson, we will again use the data from the COVID-19 serological survey conducted in Yaounde, Cameroon. Below, we import the dataset `yaounde` and create a smaller subset called `yao`. Note that this dataset is slightly different from the one used in the previous lesson.

```
yaounde <- read_csv(here::here('data/yaounde_data.csv'))

## a smaller subset of variables
yao <- yaounde %>% select(date_surveyed,
                          age,
                          weight_kg, height_cm,
                          symptoms, is_smoker)

yao
```

date_surv...	age	weight_kg	height_cm	symptoms	is_smoker
2020-10-22	45	95	169	Muscle pain	Non-smoker
2020-10-24	55	96	185	No sympto...	Ex-smoker
2020-10-24	23	74	180	No sympto...	Smoker
2020-10-22	20	70	164	Rhinitis--Sn...	Non-smoker
2020-10-22	55	67	147	No sympto...	Non-smoker
2020-10-25	17	65	162	Fever--Cou...	Non-smoker
2020-10-25	13	65	150	Sneezing	Non-smoker
2020-10-24	28	62	173	Headache	Non-smoker
2020-10-24	30	73	170	Fever--Rhin...	Non-smoker
2020-10-24	13	56	153	No sympto...	Non-smoker

1-10 of 971 rows Previous **1** 2 3 4 5 ... 98 Next

We will also use a dataset from a cross-sectional study that aimed to determine the prevalence of sarcopenia in the elderly population (>60 years) in in Karnataka, India. Sarcopenia is a condition that is common in elderly people and is characterized by progressive and generalized loss of skeletal muscle mass and strength. The data was obtained from Zenodo here, and the source publication can be found here.

Below, we import and view this dataset:

```
sarcopenia <- read_csv(here::here('data/sarcopenia_elderly.csv'))

sarcopenia
```

number	age	age_group	sex_male…	marital_s…	height_m…	weig
7	60.8	Sixties	0	married	1.57	58
8	72.3	Seventies	1	married	1.65	72
9	62.6	Sixties	0	married	1.59	64
12	72	Seventies	0	widow	1.473	54.5
13	60.1	Sixties	0	married	1.55	47
19	60.6	Sixties	0	married	1.422	64
45	60.1	Sixties	1	widower	1.68	60
46	60.2	Sixties	0	married	1.8	64.6
51	63	Sixties	0	married	1.6	57.8
56	60.4	Sixties	0	married	1.6	71.8

1-10 of 239 rows Previous **1** 2 3 4 5 … 24 Next

Introducing `mutate()`

The `mutate()` function. (Drawing adapted from Allison Horst)

We use `dplyr::mutate()` to create new variables or modify existing variables. The syntax is quite intuitive, and generally looks like `df %>% mutate(new_column_name =`

`what_it_contains)`.

Let's see a quick example.

The `yaounde` dataset currently contains a column called `height_cm`, which shows the height, in centimeters, of survey respondents. Let's create a data frame, `yao_height`, with just this column, for easy illustration:

```
yao_height <- yaounde %>% select(height_cm)
yao_height
```

```
## # A tibble: 5 × 1
##   height_cm
##       <dbl>
## 1       169
## 2       185
## 3       180
## 4       164
## 5       147
```

What if you wanted to **create a new variable**, called `height_meters` where heights are converted to meters? You can use `mutate()` for this, with the argument `height_meters = height_cm/100`:

```
yao_height %>%
  mutate(height_meters = height_cm/100)
```

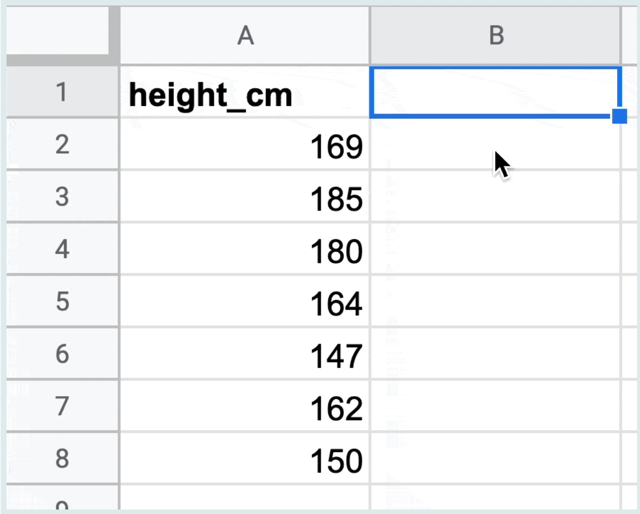
```
## # A tibble: 5 × 2
##   height_cm height_meters
##       <dbl>         <dbl>
## 1       169          1.69
## 2       185          1.85
## 3       180          1.8
## 4       164          1.64
## 5       147          1.47
```

Great. The syntax is beautifully simple, isn't it?

SIDE NOTE

Sometimes it is helpful to think of data manipulation functions in the context of familiar spreadsheet software. Here is what the R command `mutate(height_m = height_cm/100)` would be equivalent to in Google Sheets:

SIDE NOTE



	A	B
1	**height_cm**	
2	169	
3	185	
4	180	
5	164	
6	147	
7	162	
8	150	

Now, imagine there was a small error in the equipment used to measure respondent heights, and all heights are 5cm too small. You therefore like to add 5cm to all heights in the dataset. To do this, rather than creating a new variable as you did before, you can **modify the existing variable** with mutate:

```
yao_height %>%
  mutate(height_cm = height_cm + 5)
```

```
## # A tibble: 5 × 1
##   height_cm
##       <dbl>
## 1       174
## 2       190
## 3       185
## 4       169
## 5       152
```

Again, very easy to do!

PRACTICE

(in RMD)

The `sarcopenia` data frame has a variable `weight_kg`, which contains respondents' weights in kilograms. Create a new column, called `weight_grams`, with respondents' weights in grams. Store your answer in the `Q_weight_to_g` object. (1 kg equals 1000 grams.)

```
# Complete the code with your answer:
Q_weight_to_g <-
  sarcopenia %>%
  ______________________
```

Hopefully you now see that the mutate function is quite user-friendly. In theory, we could end the lesson here, because you now know how to use `mutate()` 😃 . But of course, the devil will be in the details—the interesting thing is not `mutate()` itself but what goes *inside* the `mutate()` call.

The rest of the lesson will go through a few use cases for the `mutate()` verb. In the process, we'll touch on several new functions you have not yet encountered.

Creating a Boolean variable

You can use `mutate()` to create a Boolean variable to categorize part of your population.

Below we create a Boolean variable, `is_child` which is either `TRUE` if the subject is a child or `FALSE` if the subject is an adult (first, we select just the `age` variable so it's easy to see what is being done; you will likely not need this pre-selection for your own analyses).

```
yao %>%
  select(age) %>%
  mutate(is_child = age <= 18)
```

```
## # A tibble: 5 × 2
##     age is_child
##   <dbl> <lgl>
## 1    45 FALSE
## 2    55 FALSE
## 3    23 FALSE
## 4    20 FALSE
## 5    55 FALSE
```

The code `age <= 18` evaluates whether each age is less than or equal to 18. Ages that match that condition (ages 18 and under) are `TRUE` and those that fail the condition are `FALSE`.

Such a variable is useful to, for example, count the number of children in the dataset. The code below does this with the `janitor::tabyl()` function:

```
yao %>%
  mutate(is_child = age <= 18) %>%
  tabyl(is_child)
```

```
##  is_child   n   percent
##     FALSE 662 0.6817714
##      TRUE 309 0.3182286
```

You can observe that 31.8% (0.318…) of respondents in the dataset are children.

Let's see one more example, since the concept of Boolean variables can be a bit confusing. The `symptoms` variable reports any respiratory symptoms experienced by the patient:

```
yao %>%
  select(symptoms)
```

```
## # A tibble: 5 × 1
##   symptoms
##   <chr>
## 1 Muscle pain
## 2 No symptoms
## 3 No symptoms
## 4 Rhinitis--Sneezing--Anosmia or ageusia
## 5 No symptoms
```

You could create a Boolean variable, called `has_no_symptoms`, that is set to `TRUE` if the respondent reported no symptoms:

```
yao %>%
  select(symptoms) %>%
  mutate(has_no_symptoms = symptoms == "No symptoms")
```

```
## # A tibble: 5 × 2
##   symptoms                               has_no_symptoms
##   <chr>                                  <lgl>
## 1 Muscle pain                            FALSE
## 2 No symptoms                            TRUE
## 3 No symptoms                            TRUE
## 4 Rhinitis--Sneezing--Anosmia or ageusia FALSE
## 5 No symptoms                            TRUE
```

Similarly, you could create a Boolean variable called `has_any_symptoms` that is set to `TRUE` if the respondent reported any symptoms. For this, you'd simply swap the `symptoms == "No symptoms"` code for `symptoms != "No symptoms"`:

```
yao %>%
  select(symptoms) %>%
  mutate(has_any_symptoms = symptoms != "No symptoms")
```

```
## # A tibble: 5 × 2
##   symptoms                               has_any_symptoms
##   <chr>                                  <lgl>
## 1 Muscle pain                            TRUE
## 2 No symptoms                            FALSE
## 3 No symptoms                            FALSE
```

```
## 4 Rhinitis--Sneezing--Anosmia or ageusia TRUE
## 5 No symptoms                             FALSE
```

Still confused by the Boolean examples? That's normal. Pause and play with the code above a little. Then try the practice question below

Women with a grip strength below 20kg are considered to have low grip strength. With a female subset of the `sarcopenia` data frame, add a variable called `low_grip_strength` that is `TRUE` for women with a grip strength < 20 kg and FALSE for other women.

```
# Complete the code with your answer:
Q_women_low_grip_strength <-
  sarcopenia %>%
  filter(sex_male_1_female_0 == 0) # first we filter the dataset
         to only women
  # mutate code here
```

What percentage of women surveyed have a low grip strength according to the definition above? Enter your answer as a number without quotes (e.g. 43.3 or 12.2), to one decimal place.

```
Q_prop_women_low_grip_strength <- YOUR_ANSWER_HERE
```

Creating a numeric variable based on a formula

Now, let's look at an example of creating a numeric variable, the body mass index (BMI), which a commonly used health indicator. The formula for the body mass index can be written as:

$$BMI = \frac{weight(kilograms)}{height(meters)^2}$$

You can use `mutate()` to calculate BMI in the `yao` dataset as follows:

```
yao %>%
  select(weight_kg,  height_cm) %>%

  # first obtain the height in meters
  mutate(height_meters = height_cm/100) %>%

  # then use the BMI formula
  mutate(bmi = weight_kg / (height_meters)^2)
```

```
## # A tibble: 5 × 4
##   weight_kg height_cm height_meters   bmi
##       <dbl>     <dbl>         <dbl> <dbl>
## 1        95       169          1.69  33.3
## 2        96       185          1.85  28.0
## 3        74       180          1.8   22.8
## 4        70       164          1.64  26.0
## 5        67       147          1.47  31.0
```

Let's save the data frame with BMIs for later. We will use it in the next section.

```
yao_bmi <-
  yao %>%
  select(weight_kg,  height_cm) %>%
  # first obtain the height in meters
  mutate(height_meters = height_cm/100) %>%
  # then use the BMI formula
  mutate(bmi = weight_kg / (height_meters)^2)
```

PRACTICE
(in RMD)

Appendicular muscle mass (ASM), a useful health indicator, is the sum of muscle mass in all 4 limbs. It can predicted with the following formula, called Lee's equation:

$$ASM(kg) = (0.244 \times weight(kg)) + (7.8 \times height(m)) + (6.6 \times sex) - (0.098 \times ag$$

The `sex` variable in the formula assumes that men are coded as 1 and women are coded as 0 (which is already the case for our `sarcopenia` dataset.) The `- 4.5` at the end is a constant used for Asians.

Calculate the ASM value for all individuals in the `sarcopenia` dataset. This value should be in a new column called `asm`

```
# Complete the code with your answer:
Q_asm_calculation <-
  sarcopenia #_____
  #__________________
```

Changing a variable's type

In your data analysis workflow, you often need to redefine variable *types*. You can do so with functions like `as.integer()`, `as.factor()`, `as.character()` and `as.Date()` within your `mutate()` call. Let's see one example of this.

Integer: `as.integer`

`as.integer()` converts any numeric values to integers:

```
yao_bmi %>%
  mutate(bmi_integer = as.integer(bmi))
```

```
## # A tibble: 5 × 5
##   weight_kg height_cm height_meters   bmi bmi_integer
##       <dbl>     <dbl>         <dbl> <dbl>       <int>
## 1        95       169          1.69  33.3          33
## 2        96       185          1.85  28.0          28
## 3        74       180          1.8   22.8          22
## 4        70       164          1.64  26.0          26
## 5        67       147          1.47  31.0          31
```

Note that this *truncates* integers rather than rounding them up or down, as you might expect. For example the BMI 22.8 in the third row is truncated to 22. If you want rounded numbers, you can use the `round` function from base R

PRO TIP

Using `as.integer()` on a factor variable is a fast way of encoding strings into numbers. It can be essential to do so for some machine learning data processing.

```
yao_bmi %>%
  mutate(bmi_integer = as.integer(bmi),
         bmi_rounded = round(bmi))
```

```
## # A tibble: 5 × 6
##   weight_kg height_cm height_meters   bmi bmi_integer
##       <dbl>     <dbl>         <dbl> <dbl>       <int>
## 1        95       169          1.69  33.3          33
## 2        96       185          1.85  28.0          28
## 3        74       180          1.8   22.8          22
## 4        70       164          1.64  26.0          26
## 5        67       147          1.47  31.0          31
## # … with 1 more variable: bmi_rounded <dbl>
```

SIDE NOTE

The base R `round()` function rounds "half down". That is, the number 3.5, for example, is rounded down to 3 by `round()`. This is weird. Most people expect 3.5 to be rounded *up* to 4, not down to 3. So most of the time, you'll actually want to use the `round_half_up()` function from janitor.

CHALLENGE

In future lessons, you will discover how to manipulate dates and how to convert to a date type using `as.Date()`.

PRACTICE (in RMD)

Use `as_integer()` to convert the ages of respondents in the `sarcopenia` dataset to integers (truncating them in the process). This should go in a new column called `age_integer`

```
# Complete the code with your answer:
Q_age_integer <-
  sarcopenia #_____
  #________________
```

Wrap up

As you can imagine, transforming data is an essential step in any data analysis workflow. It is often required to clean data and to prepare it for further statistical analysis or for making plots. And as you have seen, it is quite simple to transform data with dplyr's `mutate()` function, although certain transformations are trickier to achieve than others.

Congrats on making it through.

But your data wrangling journey isn't over yet! In our next lessons, we will learn how to create complex data summaries and how to create and work with data frame groups. Intrigued? See you in the next lesson.

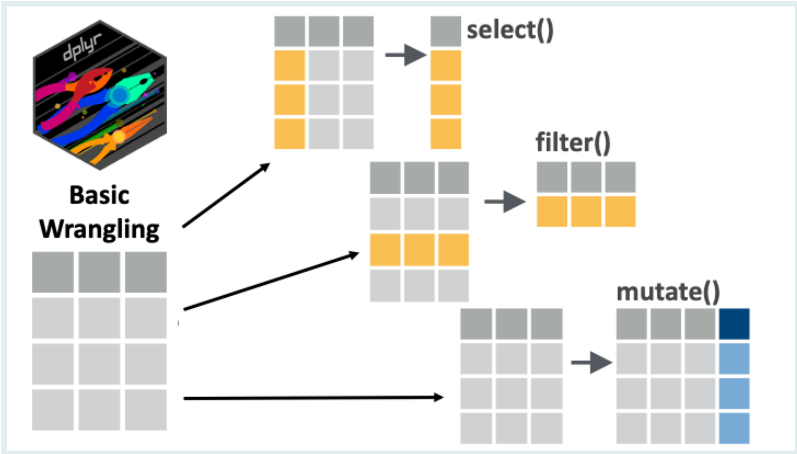


Fig: Basic Data Wrangling with `select()`, `filter()`, and `mutate()`.

Contributors

The following team members contributed to this lesson:

LAURE VANCAUWENBERGHE

Data analyst, the GRAPH Network
A firm believer in science for good, striving to ally programming, health and education

ANDREE VALLE CAMPOS

R Developer and Instructor, the GRAPH Network
Motivated by reproducible science and education

KENE DAVID NWOSU

Data analyst, the GRAPH Network
Passionate about education

References

Some material in this lesson was adapted from the following sources:

- Horst, A. (2022). *Dplyr-learnr*. https://github.com/allisonhorst/dplyr-learnr (Original work published 2020)

- *Create, modify, and delete columns — Mutate*. (n.d.). Retrieved 21 February 2022, from https://dplyr.tidyverse.org/reference/mutate.html

- *Apply a function (or functions) across multiple columns — Across*. (n.d.). Retrieved 21 February 2022, from https://dplyr.tidyverse.org/reference/across.html

Artwork was adapted from:

- Horst, A. (2022). *R & stats illustrations by Allison Horst*. https://github.com/allisonhorst/stats-illustrations (Original work published 2018)

Other references:

- Lee, Robert C, ZiMian Wang, Moonseong Heo, Robert Ross, Ian Janssen, and Steven B Heymsfield. "Total-Body Skeletal Muscle Mass: Development and Cross-Validation of Anthropometric Prediction Models." *The American Journal of Clinical Nutrition* 72, no. 3 (2000): 796-803. https://doi.org/10.1093/ajcn/72.3.796.

title: "Mutating values" credits: "This document serves as an accompaniment for a lesson found on https://thegraphcourses.org.

The GRAPH Courses is a project of the Global Research and Analyses for Public Health (GRAPH) Network, with the support of the World Health Organization (WHO) and other partners" date: "November 2022" author: "The GRAPH Courses team" —