

---

# Lesson notes | Selecting and renaming columns

Created by the GRAPH Courses team

January 2023

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, a non-profit headquartered at the University of Geneva Global Health Institute, and supported by the World Health Organization (WHO) and other partners



Introduction .....	
Learning objectives .....	
The Yaounde COVID-19 dataset .....	
Introducing <code>select()</code> .....	
Selecting column ranges with <code>:</code> .....	
Excluding columns with <code>!</code> .....	
Helper functions for <code>select()</code> .....	
<code>starts_with()</code> and <code>ends_with()</code> .....	
<code>contains()</code> .....	
<code>everything()</code> .....	
Change column names with <code>rename()</code> .....	
Rename within <code>select()</code> .....	
Wrap Up ! .....	

---

## Introduction

Today we will begin our exploration of the `{dplyr}` package! Our first verb on the list is `select` which allows to keep or drop variables from your dataframe. Choosing your variables is the first step in cleaning your data.

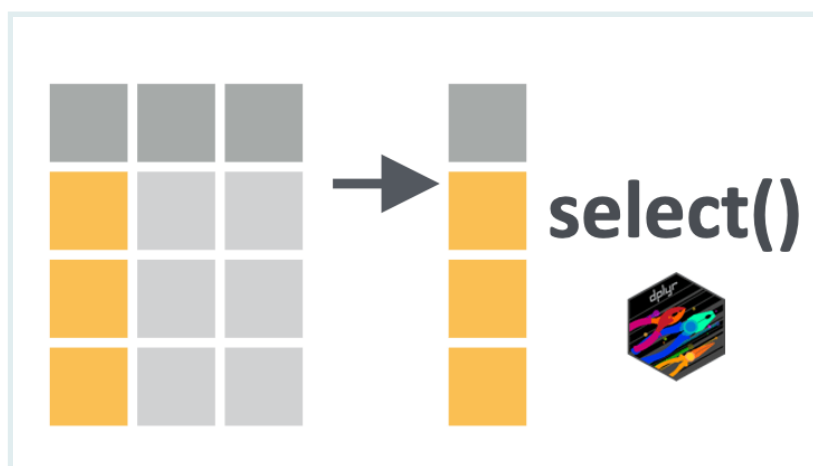


Fig: the `select()` function.

Let's go !

---

## Learning objectives

- You can keep or drop columns from a dataframe using the `dplyr::select()` function from the `{dplyr}` package.

- You can select a range or combination of columns using operators like the colon (:), the exclamation mark (!), and the `c()` function.
- You can select columns based on patterns in their names with helper functions like `starts_with()`, `ends_with()`, `contains()`, and `everything()`.
- You can use `rename()` and `select()` to change column names.

## The Yaounde COVID-19 dataset

In this lesson, we analyse results from a COVID-19 serological survey conducted in Yaounde, Cameroon in late 2020. The survey estimated how many people had been infected with COVID-19 in the region, by testing for IgG and IgM antibodies. The full dataset can be obtained from [Zenodo](#), and the paper can be viewed [here](#).

Spend some time browsing through this dataset. Each line corresponds to one patient surveyed. There are some demographic, socio-economic and COVID-related variables. The results of the IgG and IgM antibody tests are in the columns `igg_result` and `igm_result`.

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

```
## # A tibble: 5 × 53
##   id                date_surveyed  age age_category
##   <chr>              <date>         <dbl> <chr>
## 1 BRIQUETERIE_000_0001 2020-10-22      45 45 - 64
## 2 BRIQUETERIE_000_0002 2020-10-24      55 45 - 64
## 3 BRIQUETERIE_000_0003 2020-10-24      23 15 - 29
## 4 BRIQUETERIE_002_0001 2020-10-22      20 15 - 29
## 5 BRIQUETERIE_002_0002 2020-10-22      55 45 - 64
## # ... with 49 more variables: age_category_3 <chr>,
## #   sex <chr>, highest_education <chr>, occupation <chr>, ...
```



Left: the Yaounde survey team. Right: an antibody test being administered.

## Introducing `select()`

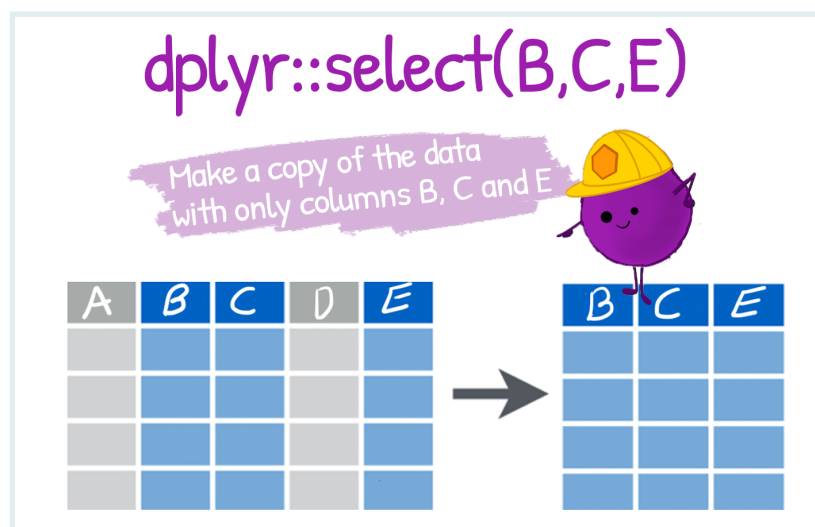


Fig: the `select()` function. (Drawing adapted from Allison Horst).

`dplyr::select()` lets us pick which columns (variables) to keep or drop.

We can select a column **by name**:

```
yaounde %>% select(age)
```

```
## # A tibble: 5 × 1
##   age
```

```
##      <dbl>
## 1      45
## 2      55
## 3      23
## 4      20
## 5      55
```

Or we can select a column **by position**:

```
yaounde %>% select(3) # `age` is the 3rd column
```

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

To select **multiple variables**, we separate them with commas:

```
yaounde %>% select(age, sex, igg_result)
```

```
## # A tibble: 971 × 3
##       age sex    igg_result
##   <dbl> <chr>  <chr>
## 1     45 Female Negative
## 2     55 Male   Positive
## 3     23 Male   Negative
## 4     20 Female Positive
## 5     55 Female Positive
## 6     17 Female Negative
## 7     13 Female Positive
## 8     28 Male   Negative
## 9     30 Male   Negative
## 10    13 Female Positive
## # ... with 961 more rows
```

#### PRACTICE



(in RMD)

- Select the weight and height variables in the `yaounde` data frame.
- Select the 16th and 22nd columns in the `yaounde` data frame.

For the next part of the tutorial, let's create a smaller subset of the data, called `yao`.

```
yao <-  
  yaounde %>% select(age,  
                     sex,  
                     highest_education,  
                     occupation,  
                     is_smoker,  
                     is_pregnant,  
                     igg_result,  
                     igm_result)  
yao
```

```
## # A tibble: 5 × 8  
##   age sex    highest_education occupation    is_smoker  
##   <dbl> <chr>  <chr>                <chr>      <chr>  
## 1    45 Female Secondary      Informal worker Non-smoker  
## 2    55 Male   University    Salaried worker Ex-smoker  
## 3    23 Male   University    Student        Smoker  
## 4    20 Female Secondary      Student        Non-smoker  
## 5    55 Female Primary        Trader--Farmer Non-smoker  
## # ... with 3 more variables: is_pregnant <chr>,  
## #   igg_result <chr>, igm_result <chr>
```

## Selecting column ranges with `:`

The `:` operator selects a **range of consecutive variables**:

```
yao %>% select(age:occupation) # Select all columns from `age` to `occupation`
```

```
## # A tibble: 5 × 4  
##   age sex    highest_education occupation  
##   <dbl> <chr>  <chr>                <chr>  
## 1    45 Female Secondary      Informal worker  
## 2    55 Male   University    Salaried worker  
## 3    23 Male   University    Student  
## 4    20 Female Secondary      Student  
## 5    55 Female Primary        Trader--Farmer
```

We can also specify a range with column numbers:

```
yao %>% select(1:4) # Select columns 1 to 4
```

```
## # A tibble: 5 × 4  
##   age sex    highest_education occupation  
##   <dbl> <chr>  <chr>                <chr>  
## 1    45 Female Secondary      Informal worker  
## 2    55 Male   University    Salaried worker
```

```
## 3    23 Male   University      Student
## 4    20 Female Secondary      Student
## 5    55 Female Primary        Trader--Farmer
```

## PRACTICE



(in RMD)

- With the `yaounde` data frame, select the columns between `symptoms` and `sequelae`, inclusive. ("Inclusive" means you should also include `symptoms` and `sequelae` in the selection.)

## Excluding columns with !

The **exclamation point** negates a selection:

```
yao %>% select(!age) # Select all columns except `age`
```

```
## # A tibble: 5 × 7
##   sex      highest_education occupation      is_smoker
##   <chr>    <chr>              <chr>        <chr>
## 1 Female Secondary          Informal worker Non-smoker
## 2 Male    University          Salaried worker Ex-smoker
## 3 Male    University          Student         Smoker
## 4 Female Secondary          Student         Non-smoker
## 5 Female Primary            Trader--Farmer  Non-smoker
## # ... with 3 more variables: is_pregnant <chr>,
## #   igg_result <chr>, igm_result <chr>
```

To drop a range of consecutive columns, we use, for example, `!age:occupation`:

```
yao %>% select(!age:occupation) # Drop columns from `age` to `occupation`
```

```
## # A tibble: 5 × 4
##   is_smoker is_pregnant igg_result igm_result
##   <chr>      <chr>      <chr>      <chr>
## 1 Non-smoker No          Negative   Negative
## 2 Ex-smoker  <NA>          Positive   Negative
## 3 Smoker      <NA>          Negative   Negative
## 4 Non-smoker No          Positive   Negative
## 5 Non-smoker No          Positive   Negative
```

To drop several non-consecutive columns, place them inside `!c()`:

```
yao %>% select(!c(age, sex, igg_result))
```

```
## # A tibble: 5 × 5
##   highest_education occupation      is_smoker is_pregnant
```



```
##   <chr>                <chr>                <chr>                <chr>
## 1 Secondary            Informal worker Non-smoker No
## 2 University          Salaried worker Ex-smoker <NA>
## 3 University          Student           Smoker      <NA>
## 4 Secondary            Student           Non-smoker No
## 5 Primary             Trader--Farmer   Non-smoker No
## # ... with 1 more variable: igm_result <chr>
```

## PRACTICE



- From the `yaounde` data frame, **remove** all columns between `highest_education` and `consultation`, inclusive.

## Helper functions for `select()`

`dplyr` has a number of helper functions to make selecting easier by using patterns from the column names. Let's take a look at some of these.

`starts_with()` and `ends_with()`

These two helpers work exactly as their names suggest!

```
yao %>% select(starts_with("is_")) # Columns that start with "is"
```

```
## # A tibble: 5 × 2
##   is_smoker is_pregnant
##   <chr>     <chr>
## 1 Non-smoker No
## 2 Ex-smoker <NA>
## 3 Smoker    <NA>
## 4 Non-smoker No
## 5 Non-smoker No
```

```
yao %>% select(ends_with("_result")) # Columns that end with "result"
```

```
## # A tibble: 5 × 2
##   igg_result igm_result
##   <chr>     <chr>
## 1 Negative  Negative
## 2 Positive  Negative
## 3 Negative  Negative
## 4 Positive  Negative
## 5 Positive  Negative
```

`contains()`

`contains()` helps select columns that contain a certain string:

```
yaounde %>% select(contains("drug")) # Columns that contain the string "drug"
```

```
## # A tibble: 5 × 12
##   drugsource      is_drug_parac is_drug_antibio
##   <chr>          <dbl>          <dbl>
## 1 Self or familial      1              0
## 2 <NA>                 NA              NA
## 3 <NA>                 NA              NA
## 4 Self or familial      0              1
## 5 <NA>                 NA              NA
## # ... with 9 more variables: is_drug_hydrocortisone <dbl>,
## #   is_drug_other_anti_inflam <dbl>, ...
```

`everything()`

Another helper function, `everything()`, matches all variables that have not yet been selected.

```
# First, `is_pregnant`, then every other column.
yao %>% select(is_pregnant, everything())
```

```
## # A tibble: 5 × 8
##   is_pregnant  age sex  highest_education occupation
##   <chr>      <dbl> <chr> <chr>              <chr>
## 1 No         45 Female Secondary         Informal worker
## 2 <NA>       55 Male  University         Salaried worker
## 3 <NA>       23 Male  University         Student
## 4 No         20 Female Secondary         Student
## 5 No         55 Female Primary          Trader--Farmer
## # ... with 3 more variables: is_smoker <chr>,
## #   igg_result <chr>, igm_result <chr>
```

It is often useful for establishing the order of columns.

Say we wanted to bring the `is_pregnant` column to the start of the `yao` data frame, we could type out all the column names manually:

```
yao %>% select(is_pregnant,
               age,
               sex,
               highest_education,
               occupation,
               is_smoker,
               igg_result,
               igm_result)
```

```
## # A tibble: 5 × 8
##   is_pregnant age sex highest_education occupation
##   <chr>      <dbl> <chr> <chr> <chr>
## 1 No        45 Female Secondary Informal worker
## 2 <NA>      55 Male University Salaried worker
## 3 <NA>      23 Male University Student
## 4 No        20 Female Secondary Student
## 5 No        55 Female Primary Trader--Farmer
## # ... with 3 more variables: is_smoker <chr>,
## # igg_result <chr>, igm_result <chr>
```

But this would be painful for larger data frames, such as our original `yaounde` data frame. In such a case, we can use `everything()`:

```
# Bring `is_pregnant` to the front of the data frame
yaounde %>% select(is_pregnant, everything())
```

```
## # A tibble: 5 × 53
##   is_pregnant id date_surveyed age
##   <chr>      <chr> <date> <dbl>
## 1 No        BRIQUETERIE_000_0001 2020-10-22 45
## 2 <NA>      BRIQUETERIE_000_0002 2020-10-24 55
## 3 <NA>      BRIQUETERIE_000_0003 2020-10-24 23
## 4 No        BRIQUETERIE_002_0001 2020-10-22 20
## 5 No        BRIQUETERIE_002_0002 2020-10-22 55
## # ... with 49 more variables: age_category <chr>,
## # age_category_3 <chr>, sex <chr>, ...
```

This helper can be combined with many others.

```
# Bring columns that end with "result" to the front of the data frame
yaounde %>% select(ends_with("result"), everything())
```

```
## # A tibble: 5 × 53
##   igg_result igm_result id date_surveyed age
##   <chr>      <chr> <chr> <date> <dbl>
## 1 Negative Negative BRIQUETERIE_000... 2020-10-22 45
## 2 Positive Negative BRIQUETERIE_000... 2020-10-24 55
## 3 Negative Negative BRIQUETERIE_000... 2020-10-24 23
## 4 Positive Negative BRIQUETERIE_002... 2020-10-22 20
```

```
## 5 Positive    Negative    BRIQUETERIE_002... 2020-10-22      55
## # ... with 48 more variables: age_category <chr>,
## #   age_category_3 <chr>, sex <chr>, ...
```

### PRACTICE

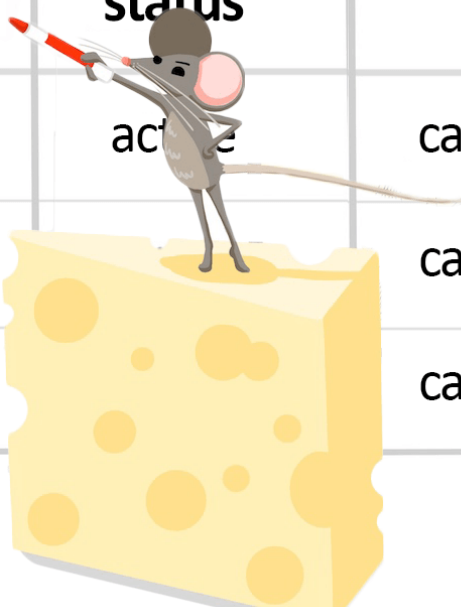


- Select all columns in the `yaounde` data frame that start with “is\_”.
- Move the columns that start with “is\_” to the beginning of the `yaounde` data frame.

Change column names with `rename()`

# RENAME COLUMNS

`dplyr::rename(enemies = species)`



<del>species</del> enemies	status	diet
Dog	active	carnivore
House cat		carnivore
Osprey		carnivore

Fig: the `rename()` function. (Drawing adapted from Allison Horst)

`dplyr::rename()` is used to change column names:

```
# Rename `age` and `sex` to `patient_age` and `patient_sex`  
yaounde %>%  
  rename(patient_age = age,  
         patient_sex = sex)
```

```
## # A tibble: 5 × 53  
##   id                date_surveyed patient_age age_category  
##   <chr>              <date>          <dbl> <chr>  
## 1 BRIQUETERIE_000_00... 2020-10-22          45 45 - 64  
## 2 BRIQUETERIE_000_00... 2020-10-24          55 45 - 64
```

```
## 3 BRIQUETERIE_000_00... 2020-10-24      23 15 - 29
## 4 BRIQUETERIE_002_00... 2020-10-22      20 15 - 29
## 5 BRIQUETERIE_002_00... 2020-10-22      55 45 - 64
## # ... with 49 more variables: age_category_3 <chr>,
## #   patient_sex <chr>, highest_education <chr>, ...
```

#### WATCH OUT



The fact that the new name comes first in the function `(rename(NEWNAME = OLDNAME))` is sometimes confusing. You should get used to this with time.

### Rename within `select()`

You can also rename columns while selecting them:

```
# Select `age` and `sex`, and rename them to `patient_age` and `patient_sex`
yaounde %>%
  select(patient_age = age,
         patient_sex = sex)
```

```
## # A tibble: 5 × 2
##   patient_age patient_sex
##   <dbl> <chr>
## 1      45 Female
## 2      55 Male
## 3      23 Male
## 4      20 Female
## 5      55 Female
```

## Wrap Up !

I hope this first lesson has allowed you to see how intuitive and useful the {dplyr} verbs are! This is the first of a series of basic data wrangling verbs: see you in the next lesson to learn more.

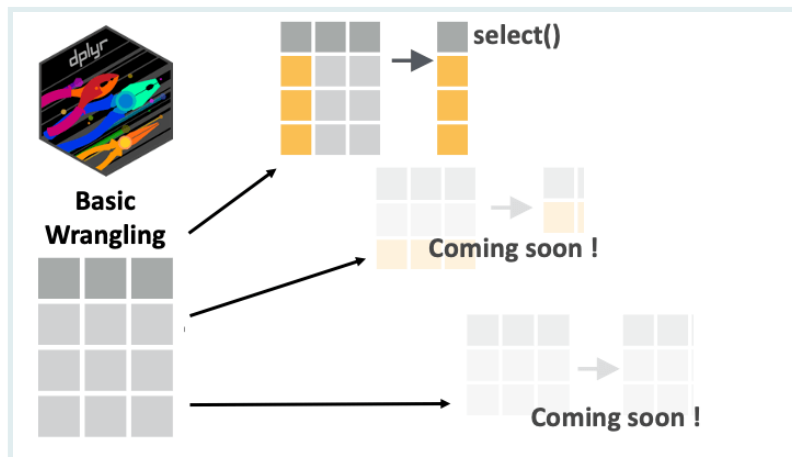


Fig: Basic Data Wrangling Dplyr Verbs.

---

## 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 world improvement

---

## References

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

- Horst, A. (2021). *Dplyr-learnr*. <https://github.com/allisonhorst/dplyr-learnr> (Original work published 2020)

- 
- *Subset columns using their names and types–Select*. (n.d.). Retrieved 31 December 2021, from <https://dplyr.tidyverse.org/reference/select.html>

Artwork was adapted from:

- Horst, A. (2021). *R & stats illustrations by Allison Horst*. <https://github.com/allisonhorst/stats-illustrations> (Original work published 2018)