Data Manipulation with dplyr

This tutorial is partially based on **Data Analysis and Visualization in R for Ecologists** by François Michonneau & Auriel Fournier.

Data for the tutorial is taken from `dslab` R package

Introduction

Bracket subsetting is handy, but it can be cumbersome and difficult to read, especially for complicated operations. Enter `dplyr`. `dplyr` is a package for helping with tabular data manipulation. It pairs nicely with `tidyr` which enables you to swiftly convert between different data formats for plotting and analysis.

The `tidyverse` package is an "umbrella-package" that installs `tidyr`, `dplyr`, and several other useful packages for data analysis, such as `ggplot2`, `tibble`, etc.

The `tidyverse` package tries to address 3 common issues that arise when doing data analysis in R:

1. The results from a base R function sometimes depend on the type of data.
2. R expressions are used in a non standard way, which can be confusing for new learners.
3. The existence of hidden arguments having default operations that new learners are not aware of.

To use `tidyverse` package, you can type `install.packages("tidyverse")` straight into the console to install it. Then, type `library(tidyverse)` to load the package.

What is `dplyr`

`dplyr` is a grammar of data manipulation, providing a consistent set of verbs that help you solve the most common data manipulation challenges:

- `mutate()` adds new variables that are functions of existing variables
- `select()` picks variables based on their names.

- `filter()` picks cases based on their values.
- `summarise()` reduces multiple values down to a single summary.
- `arrange()` changes the ordering of the rows.

Preparing data

Loading libraries:

```r
library(dplyr)
library(dslabs)
```

Preparing and preview data

```r
data(gapminder)
gapminder <- gapminder |> as_tibble()
gapminder
```

```
# A tibble: 10,545 x 9
   country           year infan~1 life_~2 ferti~3 popul~4      gdp conti~5 region
   <fct>            <int>   <dbl>   <dbl>   <dbl>   <dbl>    <dbl> <fct>   <fct>
 1 Albania           1960   115.     62.9    6.19 1.64e6 NA        Europe  South~
 2 Algeria           1960   148.     47.5    7.65 1.11e7  1.38e10 Africa  North~
 3 Angola            1960   208      36.0    7.32 5.27e6 NA        Africa  Middl~
 4 Antigua and Ba~   1960    NA      63.0    4.43 5.47e4 NA        Americ~ Carib~
 5 Argentina         1960    59.9    65.4    3.11 2.06e7  1.08e11 Americ~ South~
 6 Armenia           1960    NA      66.9    4.55 1.87e6 NA        Asia    Weste~
 7 Aruba             1960    NA      65.7    4.82 5.42e4 NA        Americ~ Carib~
 8 Australia         1960    20.3    70.9    3.45 1.03e7  9.67e10 Oceania Austr~
 9 Austria           1960    37.3    68.8    2.7  7.07e6  5.24e10 Europe  Weste~
10 Azerbaijan        1960    NA      61.3    5.57 3.90e6 NA        Asia    Weste~
# ... with 10,535 more rows, and abbreviated variable names
#   1: infant_mortality, 2: life_expectancy, 3: fertility, 4: population,
#   5: continent
```

Inspecting data

```r
glimpse(gapminder)
```

```
Rows: 10,545
Columns: 9
```

```
$ country          <fct> "Albania", "Algeria", "Angola", "Antigua and Barbuda"~
$ year             <int> 1960, 1960, 1960, 1960, 1960, 1960, 1960, 1960, 1960,~
$ infant_mortality <dbl> 115.40, 148.20, 208.00, NA, 59.87, NA, NA, 20.30, 37.~
$ life_expectancy  <dbl> 62.87, 47.50, 35.98, 62.97, 65.39, 66.86, 65.66, 70.8~
$ fertility        <dbl> 6.19, 7.65, 7.32, 4.43, 3.11, 4.55, 4.82, 3.45, 2.70,~
$ population       <dbl> 1636054, 11124892, 5270844, 54681, 20619075, 1867396,~
$ gdp              <dbl> NA, 13828152297, NA, NA, 108322326649, NA, NA, 966778~
$ continent        <fct> Europe, Africa, Africa, Americas, Americas, Asia, Ame~
$ region           <fct> Southern Europe, Northern Africa, Middle Africa, Cari~
```

Selecting columns and filtering rows

To select columns of a data frame, use `select()`. The first argument to this function is the data frame (`gapminder`), and the subsequent arguments are the columns to keep.

```
select(gapminder, country, year, population)
```

```
# A tibble: 10,545 x 3
   country              year population
   <fct>               <int>      <dbl>
 1 Albania              1960    1636054
 2 Algeria              1960   11124892
 3 Angola               1960    5270844
 4 Antigua and Barbuda  1960      54681
 5 Argentina            1960   20619075
 6 Armenia              1960    1867396
 7 Aruba                1960      54208
 8 Australia            1960   10292328
 9 Austria              1960    7065525
10 Azerbaijan           1960    3897889
# ... with 10,535 more rows
```

To select all columns *except* certain ones, put a "-" in front of the variable to exclude it.

```
select(gapminder, -c(gdp, continent, region))
```

```
# A tibble: 10,545 x 6
   country              year infant_mortality life_expectancy fertility popula~1
   <fct>               <int>            <dbl>           <dbl>     <dbl>    <dbl>
 1 Albania              1960            115.             62.9      6.19  1636054
 2 Algeria              1960            148.             47.5      7.65 11124892
 3 Angola               1960            208              36.0      7.32  5270844
 4 Antigua and Barbuda  1960             NA              63.0      4.43    54681
 5 Argentina            1960             59.9            65.4      3.11 20619075
 6 Armenia              1960             NA              66.9      4.55  1867396
 7 Aruba                1960             NA              65.7      4.82    54208
 8 Australia            1960             20.3            70.9      3.45 10292328
 9 Austria              1960             37.3            68.8      2.7   7065525
10 Azerbaijan           1960             NA              61.3      5.57  3897889
# ... with 10,535 more rows, and abbreviated variable name 1: population
```

This will select all the variables in `gapminder` except `gdp`, `continent` and `region`.

To choose rows based on a specific criterion, use `filter()`:

```
filter(gapminder, year == 2016)
```

```
# A tibble: 185 x 9
   country              year infan~1 life_~2 ferti~3 popul~4   gdp conti~5 region
   <fct>               <int>   <dbl>   <dbl>   <dbl>   <dbl> <dbl> <fct>   <fct>
 1 Albania              2016      NA    78.1      NA      NA    NA Europe  South~
 2 Algeria              2016      NA    76.5      NA      NA    NA Africa  North~
 3 Angola               2016      NA    60        NA      NA    NA Africa  Middl~
 4 Antigua and Barbu~  2016      NA    76.5      NA      NA    NA Americ~ Carib~
 5 Argentina            2016      NA    76.7      NA      NA    NA Americ~ South~
 6 Armenia              2016      NA    74.9      NA      NA    NA Asia    Weste~
 7 Aruba                2016      NA    75.8      NA      NA    NA Americ~ Carib~
 8 Australia            2016      NA    82.3      NA      NA    NA Oceania Austr~
 9 Austria              2016      NA    81.4      NA      NA    NA Europe  Weste~
10 Azerbaijan           2016      NA    73.3      NA      NA    NA Asia    Weste~
# ... with 175 more rows, and abbreviated variable names 1: infant_mortality,
#   2: life_expectancy, 3: fertility, 4: population, 5: continent
```

Pipes

What if you want to select and filter at the same time? There are three ways to do this: use intermediate steps, nested functions, or pipes.

With intermediate steps, you create a temporary data frame and use that as input to the next function, like this:

```
df_1 <- filter(gapminder, continent == "Europe")
gapminder_sml <- select(df_1, country, year, gdp)
```

This is readable, but can clutter up your workspace with lots of objects that you have to name individually. With multiple steps, that can be hard to keep track of.

You can also nest functions (i.e. one function inside of another), like this:

```
surveys_sml <- select(filter(gapminder, continent == "Europe"), country, year, gdp)
```

This is handy, but can be difficult to read if too many functions are nested, as R evaluates the expression from the inside out (in this case, filtering, then selecting).

The last option, *pipes*. Pipes let you take the output of one function and send it directly to the next, which is useful when you need to do many things to the same dataset. Pipes in R look like `|>` or `%>%` with **`dplyr`**. If you use RStudio, you can type the pipe with Ctrl + Shift + M if you have a PC or Cmd + Shift + M if you have a Mac.

```
gapminder |>
  filter(continent == "Europe") |>
  select(country, year, gdp)
```

```
# A tibble: 2,223 x 3
   country                 year         gdp
   <fct>                  <int>       <dbl>
 1 Albania                 1960          NA
 2 Austria                 1960 52392699681
 3 Belarus                 1960          NA
 4 Belgium                 1960 68236665814
 5 Bosnia and Herzegovina  1960          NA
 6 Bulgaria                1960          NA
 7 Croatia                 1960          NA
 8 Czech Republic          1960          NA
 9 Denmark                 1960 52164745342
10 Estonia                 1960          NA
# ... with 2,213 more rows
```

If we want to create a new object with this smaller version of the data, we can assign it a new name:

```
gapminder_sml <- gapminder |>
  filter(continent == "Europe") |>
```

```
    select(country, year, gdp)

  gapminder_sml
```

```
# A tibble: 2,223 x 3
   country                 year         gdp
   <fct>                  <int>       <dbl>
 1 Albania                 1960          NA
 2 Austria                 1960 52392699681
 3 Belarus                 1960          NA
 4 Belgium                 1960 68236665814
 5 Bosnia and Herzegovina  1960          NA
 6 Bulgaria                1960          NA
 7 Croatia                 1960          NA
 8 Czech Republic          1960          NA
 9 Denmark                 1960 52164745342
10 Estonia                 1960          NA
# ... with 2,213 more rows
```

Note that the final data frame is the leftmost part of this expression.

Mutate

Frequently you'll want to create new columns based on the values in existing columns, for example to do unit conversions, or to find the ratio of values in two columns. For this we'll use `mutate()`.

To create a new column of weight in kg:

```
gapminder %>%
  mutate(population_mln = round(population / 1000000, 2)) |>
  select(country, year, population, population_mln)
```

```
# A tibble: 10,545 x 4
   country              year population population_mln
   <fct>               <int>      <dbl>          <dbl>
 1 Albania              1960    1636054           1.64
 2 Algeria              1960   11124892          11.1
 3 Angola               1960    5270844           5.27
 4 Antigua and Barbuda  1960      54681           0.05
 5 Argentina           1960   20619075          20.6
 6 Armenia             1960    1867396           1.87
 7 Aruba               1960      54208           0.05
 8 Australia           1960   10292328          10.3
 9 Austria             1960    7065525           7.07
10 Azerbaijan          1960    3897889           3.9
# ... with 10,535 more rows
```

Split-apply-combine data analysis with the `group_by()` and `summarize()` functions

Many data analysis tasks can be approached using the *split-apply-combine* paradigm: split the data into groups, apply some analysis to each group, and then combine the results. Key functions of **`dplyr`** for this workflow are `group_by()` and `summarize()`.

`group_by()` is often used together with `summarize()`, which collapses each group into a single-row summary of that group. `group_by()` takes as arguments the column names that contain the **categorical** variables for which you want to calculate the summary statistics. So to compute the mean `population` by `country`:

```
gapminder |>
  group_by(country) |>
  summarize(mean_population = mean(population, na.rm = TRUE)) |>
  mutate_if(is.numeric, round, 0)
```

```
# A tibble: 185 x 2
   country             mean_population
   <fct>                         <dbl>
 1 Albania                     2708629
 2 Algeria                    24231378
 3 Angola                     11909433
 4 Antigua and Barbuda           71053
 5 Argentina                  31638376
 6 Armenia                     2925011
 7 Aruba                         74148
 8 Australia                  16601155
 9 Austria                     7800180
10 Azerbaijan                  6897604
# ... with 175 more rows
```

Counting

When working with data, we often want to know the number of observations found for each factor or combination of factors. For this task, **`dplyr`** provides `count()`. For example, if we wanted to count the number of rows of data for each country, we would do:

```r
gapminder %>%
    count(country)
```

```
# A tibble: 185 x 2
   country                 n
   <fct>               <int>
 1 Albania                57
 2 Algeria                57
 3 Angola                 57
 4 Antigua and Barbuda    57
 5 Argentina              57
 6 Armenia                57
 7 Aruba                  57
 8 Australia              57
 9 Austria                57
10 Azerbaijan             57
# ... with 175 more rows
```