A tutorial for Exploratory Analysis and Data Wrangling in R

Version 1.0.0

Konstantinos I. Bougioukas

07/10/2021

Contents

Objectives of the lessons

- Select rows in a data frame according to filtering conditions with the dplyr function `filter()`.
- Select columns in a data frame with the dplyr function `select()`.
- Direct the output of one dplyr function to the input of another function with the "pipe" operator `%>%`.
- Add new columns to a data frame that are functions of existing columns with `mutate()`.
- Use `summarize()` with `across()` to calculate summary statistics for multiple variables, and `group_by()` to split a data frame into groups of observations, apply summary statistics for each group, and then combine the results.

1 Explorartion of the data frame

1.1 Packages to download

We need to install the following packages to execute the full tutorial:

- `{skimr}`
- `{dlookr}`
- `{naniar}`
- `{tidyverse}`
- `{here}`
- `{tidycovid19}` (from github)
- `{EnvStats}`

1.2 Dataset description

In this tutorial we will work on Covid-19 data. The `download_merged_data()` from `tidycovid19` package downloads all data sources and creates a merged country-day panel.

```
library(tidyverse)
library(lubridate)
library(here)

#library(remotes)
#remotes::install_github("joachim-gassen/tidycovid19")
# https://github.com/joachim-gassen/tidycovid19
library(tidycovid19)
```

```
# get the data
covid_data <- download_merged_data(cached = TRUE)
```

Following we can see the definitions for each variable included in the merged country-day data frame:

Table 1: Dataset variables

var_name	var_def
iso3c	ISO3c country code as defined by ISO 3166-1 alpha-3
country	Country name
date	Calendar date
confirmed	Confirmed Covid-19 cases as reported by JHU CSSE (accumulated)
deaths	Covid-19-related deaths as reported by JHU CSSE (accumulated)
recovered	Covid-19 recoveries as reported by JHU CSSE (accumulated)
ecdc_cases	Covid-19 cases as reported by ECDC (accumulated, weekly post 2020-12-14)
ecdc_deaths	Covid-19-related deaths as reported by ECDC (accumulated, weekly post 2020-12-14)
total_tests	Accumulated test counts as reported by Our World in Data
tests_units	Definition of what constitutes a 'test'
positive_rate	The share of COVID-19 tests that are positive, given as a rolling 7-day average
hosp_patients	Number of COVID-19 patients in hospital on a given day
icu_patients	Number of COVID-19 patients in intensive care units (ICUs) on a given day
total_vaccinations	Total number of COVID-19 vaccination doses administered
soc_dist	Number of social distancing measures reported up to date by ACAPS, net of lifted restrictions
mov_rest	Number of movement restrictions reported up to date by ACAPS, net of lifted restrictions

var_name	var_def
pub_health	Number of public health measures reported up to date by ACAPS, net of lifted restrictions
gov_soc_econ	Number of social and economic measures reported up to date by ACAPS, net of lifted restrictions
lockdown	Number of lockdown measures reported up to date by ACAPS, net of lifted restrictions
apple_mtr_driving	Apple Maps usage for driving directions, as percentage*100 relative to the baseline of Jan 13, 2020
apple_mtr_walking	Apple Maps usage for walking directions, as percentage*100 relative to the baseline of Jan 13, 2020
apple_mtr_transit	Apple Maps usage for public transit directions, as percentage*100 relative to the baseline of Jan 13, 2020
gcmr_retail_recreation	Google Community Mobility Reports data for the frequency that people visit retail and recreation places expressed as a percentage*100 change relative to the baseline period Jan 3 - Feb 6, 2020
gcmr_grocery_pharmacy	Google Community Mobility Reports data for the frequency that people visit grocery stores and pharmacies expressed as a percentage*100 change relative to the baseline period Jan 3 - Feb 6, 2020
gcmr_parks	Google Community Mobility Reports data for the frequency that people visit parks expressed as a percentage*100 change relative to the baseline period Jan 3 - Feb 6, 2020
gcmr_transit_stations	Google Community Mobility Reports data for the frequency that people visit transit stations expressed as a percentage*100 change relative to the baseline period Jan 3 - Feb 6, 2020
gcmr_workplaces	Google Community Mobility Reports data for the frequency that people visit workplaces expressed as a percentage*100 change relative to the baseline period Jan 3 - Feb 6, 2020
gcmr_residential	Google Community Mobility Reports data for the frequency that people visit residential places expressed as a percentage*100 change relative to the baseline period Jan 3 - Feb 6, 2020

var_name	var_def
gtrends_score	Google search volume for the term 'coronavirus', relative across time with the country maximum scaled to 100
gtrends_country_score	Country-level Google search volume for the term 'coronavirus' over a period starting Jan 1, 2020, relative across countries with the country having the highest search volume scaled to 100 (time-stable)
region	Country region as classified by the World Bank (time-stable)
income	Country income group as classified by the World Bank (time-stable)
population	Country population as reported by the World Bank (original identifier 'SP.POP.TOTL', time-stable)
land_area_skm	Country land mass in square kilometers as reported by the World Bank (original identifier 'AG.LND.TOTL.K2', time-stable)
pop_density	Country population density as reported by the World Bank (original identifier 'EN.POP.DNST', time-stable)
pop_largest_city	Population in the largest metropolian area of the country as reported by the World Bank (original identifier 'EN.URB.LCTY', time-stable)
life_expectancy	Average life expectancy at birth of country citizens in years as reported by the World Bank (original identifier 'SP.DYN.LE00.IN', time-stable)
gdp_capita	Country gross domestic product per capita, measured in 2010 US-$ as reported by the World Bank (original identifier 'NY.GDP.PCAP.KD', time-stable)
timestamp	Date and time where data has been collected from authoritative sources

Let's have a look at the types of variables:

```r
glimpse(covid_data)
```

```
## Rows: 131,681
## Columns: 40
## $ iso3c                  <chr> "ABW", "ABW"~
## $ country                <chr> "Aruba", "Ar~
## $ date                   <date> 2020-03-13,~
## $ confirmed              <dbl> NA, NA, NA, ~
## $ deaths                 <dbl> NA, NA, NA, ~
## $ recovered              <dbl> NA, NA, NA, ~
## $ ecdc_cases             <dbl> 2, 2, 2, 2, ~
## $ ecdc_deaths            <dbl> 0, 0, 0, 0, ~
## $ total_tests            <dbl> NA, NA, NA, ~
## $ tests_units            <chr> NA, NA, NA, ~
## $ positive_rate          <dbl> NA, NA, NA, ~
## $ hosp_patients          <dbl> NA, NA, NA, ~
## $ icu_patients           <dbl> NA, NA, NA, ~
## $ total_vaccinations     <dbl> NA, NA, NA, ~
## $ soc_dist               <dbl> NA, NA, NA, ~
## $ mov_rest               <dbl> NA, NA, NA, ~
## $ pub_health             <dbl> NA, NA, NA, ~
## $ gov_soc_econ           <dbl> NA, NA, NA, ~
## $ lockdown               <dbl> NA, NA, NA, ~
## $ apple_mtr_driving      <dbl> NA, NA, NA, ~
## $ apple_mtr_walking      <dbl> NA, NA, NA, ~
## $ apple_mtr_transit      <dbl> NA, NA, NA, ~
## $ gcmr_place_id          <chr> "ChIJ23da4s8~
## $ gcmr_retail_recreation <dbl> -10, -23, -2~
## $ gcmr_grocery_pharmacy  <dbl> 40, 15, -13,~
## $ gcmr_parks             <dbl> -4, -7, -6, ~
## $ gcmr_transit_stations  <dbl> -5, -19, -18~
## $ gcmr_workplaces        <dbl> 3, -3, -5, -~
## $ gcmr_residential       <dbl> 1, 7, 6, 12,~
```

```
## $ gtrends_score          <dbl> NA, NA, NA, ~
## $ gtrends_country_score  <int> NA, NA, NA, ~
## $ region                 <chr> "Latin Ameri~
## $ income                 <chr> "High income~
## $ population             <dbl> 106766, 1067~
## $ land_area_skm          <dbl> 180, 180, 18~
## $ pop_density            <dbl> 593.1444, 59~
## $ pop_largest_city       <dbl> NA, NA, NA, ~
## $ life_expectancy        <dbl> 76.293, 76.2~
## $ gdp_capita             <dbl> 26631.47, 26~
## $ timestamp              <dttm> 2021-10-07 ~
```

The data frame contains more than 131681 rows and 40 variables. There are 32 numeric variables, 6 variables of character type, and two variables with dates (one of Date type and the other of POSIXct type).

1.3 More exploration

The `skim()` is an alternative to `glipmse()`, quickly providing a broad overview of a data frame:

```
skimr::skim(covid_data)
```

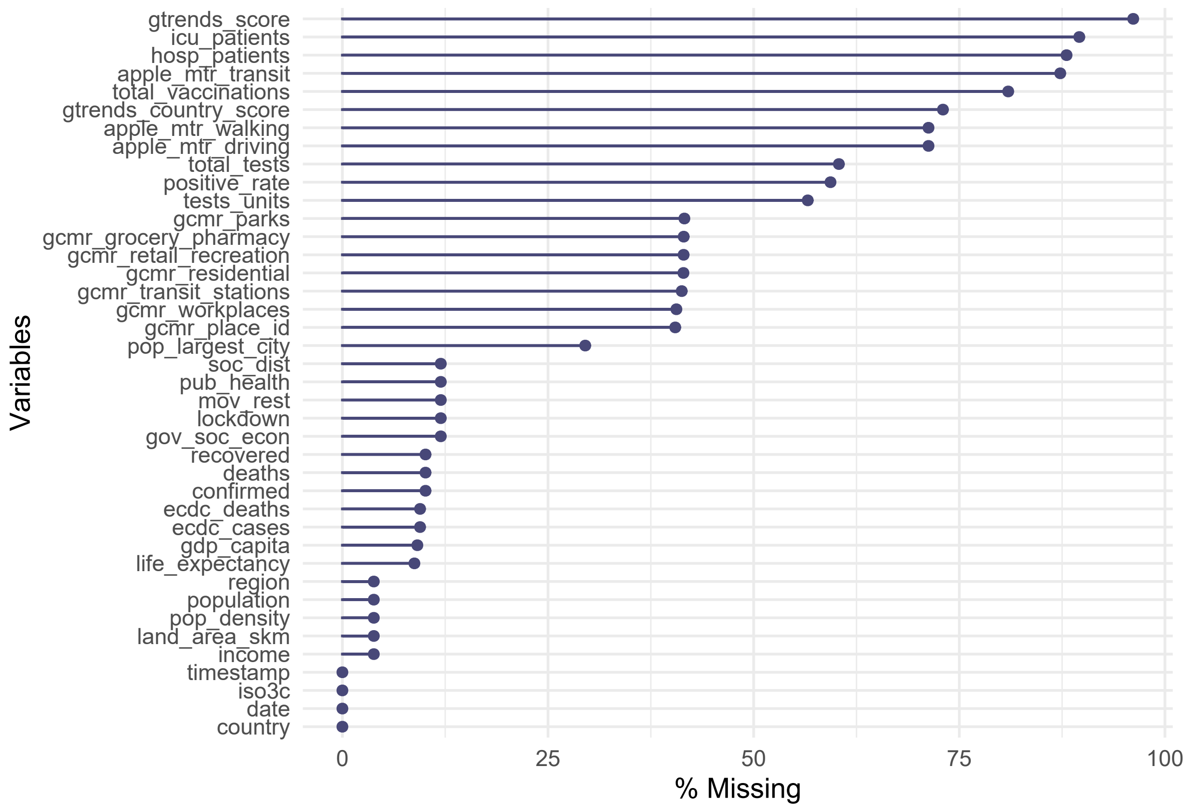
Another similar function to inspect descriptive characteristics of our dataset is the `describe()` from the `dlookr` package. Let's say we want to select specific variables (e.g., confirmed, deaths, recovered) from the dataset:

```
dlookr::describe(covid_data, confirmed, deaths, recovered)
```

If we wish, we can plot the % of missing values in each variable in our data:

```
naniar::gg_miss_var(covid_data, show_pct = TRUE)
```

```
## Warning: It is deprecated to specify `guide =
## FALSE` to remove a guide. Please use `guide =
## "none"` instead.
```



2 Data Wrangling

Data wrangling is the process of transforming and mapping data from one "raw" data form into another format with the intent of making it more appropriate and valuable for a variety of downstream purposes such as analytics.

2.1 Subseting observations (rows) using `filter()`

The `filter()` function from dplyr package allows us to subset observations (rows) based on their values.

Figure 1: Diagram of filter() rows

The first argument to this function is the name of the data frame (covid_data), and the second and subsequent arguments are the expressions that filter the data frame. For example, we can select all the data for the date `2021-06-11` with:

```
filter(covid_data, date == "2021-06-11")
```

```
## # A tibble: 215 x 40
##    iso3c country  date       confirmed deaths
##    <chr> <chr>    <date>         <dbl>  <dbl>
##  1 ABW   Aruba    2021-06-11        NA     NA
##  2 AFG   Afghani~ 2021-06-11     87716   3412
##  3 AGO   Angola   2021-06-11     36455    819
##  4 AIA   Anguilla 2021-06-11        NA     NA
##  5 ALB   Albania  2021-06-11    132437   2453
##  6 AND   Andorra  2021-06-11     13813    127
```

```
## 7 ARE   United ~ 2021-06-11    593894   1720
## 8 ARG   Argenti~ 2021-06-11   4093090  84628
## 9 ARM   Armenia  2021-06-11    223555   4482
## 10 ATG  Antigua~ 2021-06-11      1263     42
## # ... with 205 more rows, and 35 more
## #   variables: recovered <dbl>,
## #   ecdc_cases <dbl>, ecdc_deaths <dbl>,
## #   total_tests <dbl>, tests_units <chr>,
## #   positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>, ...
```

When we run that line of code, dplyr executes the filtering operation and returns a new data frame. If we want to save the result, we'll need to use the assignment operator, <-:

```
june11 <- filter(covid_data, date == "2021-06-11")
```

R either prints out the results, or saves them to a variable. If we want to do both, we can wrap the assignment in parentheses:

```
(june11 <- filter(covid_data, date == "2021-06-11"))
```

```
## # A tibble: 215 x 40
##    iso3c country  date       confirmed deaths
##    <chr> <chr>    <date>         <dbl>  <dbl>
##  1 ABW   Aruba    2021-06-11        NA     NA
##  2 AFG   Afghani~ 2021-06-11     87716   3412
##  3 AGO   Angola   2021-06-11     36455    819
##  4 AIA   Anguilla 2021-06-11        NA     NA
##  5 ALB   Albania  2021-06-11    132437   2453
##  6 AND   Andorra  2021-06-11     13813    127
##  7 ARE   United ~ 2021-06-11    593894   1720
##  8 ARG   Argenti~ 2021-06-11   4093090  84628
```

```
##  9 ARM   Armenia  2021-06-11    223555   4482
## 10 ATG   Antigua~ 2021-06-11      1263     42
## # ... with 205 more rows, and 35 more
## #   variables: recovered <dbl>,
## #   ecdc_cases <dbl>, ecdc_deaths <dbl>,
## #   total_tests <dbl>, tests_units <chr>,
## #   positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>, ...
```

Multiple arguments to `filter()` are combined with "and": every expression must be true in order for a row to be included in the output. For example, we can select all the data in which the date is "2021-06-11" and the confirmed cases are larger than 5,000,000.

```
june11b <- filter(covid_data, date == "2021-06-11", confirmed > 5000000)
june11b
```

```
## # A tibble: 6 x 40
##   iso3c country  date       confirmed deaths
##   <chr> <chr>    <date>         <dbl>  <dbl>
## 1 BRA   Brazil   2021-06-11  17296118 484235
## 2 FRA   France   2021-06-11   5795593 110516
## 3 IND   India    2021-06-11  29359155 367081
## 4 RUS   Russia   2021-06-11   5120578 123568
## 5 TUR   Turkey   2021-06-11   5319359  48593
## 6 USA   United S~ 2021-06-11  33502472 599401
## # ... with 35 more variables:
## #   recovered <dbl>, ecdc_cases <dbl>,
## #   ecdc_deaths <dbl>, total_tests <dbl>,
## #   tests_units <chr>, positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>,
## #   soc_dist <dbl>, mov_rest <dbl>, ...
```

For other types of combinations, we'll need to use Boolean operators ourself. For example, we can select all the data in which the date is "2021-06-11" or "2021-06-12":

```
june11_12 <- filter(covid_data, date == "2021-06-11" | date == "2021-06-12")
```

The order of operations doesn't work like English. We can't write `filter(covid_data, date == ("2021-06-11" | "2021-06-12"))`.

A useful short-hand for this problem is `x %in% y`. This will select every row where `x` is one of the values in `y`. We could use it to rewrite the code above:

```
june11_12 <- filter(covid_data, date %in% ymd("2021-06-11", "2021-06-12"))
june11_12
```

```
## # A tibble: 430 x 40
##    iso3c country  date       confirmed deaths
##    <chr> <chr>    <date>         <dbl>  <dbl>
##  1 ABW   Aruba    2021-06-11        NA     NA
##  2 ABW   Aruba    2021-06-12        NA     NA
##  3 AFG   Afghani~ 2021-06-11     87716   3412
##  4 AFG   Afghani~ 2021-06-12     88740   3449
##  5 AGO   Angola   2021-06-11     36455    819
##  6 AGO   Angola   2021-06-12     36600    825
##  7 AIA   Anguilla 2021-06-11        NA     NA
##  8 AIA   Anguilla 2021-06-12        NA     NA
##  9 ALB   Albania  2021-06-11    132437   2453
## 10 ALB   Albania  2021-06-12    132449   2453
## # ... with 420 more rows, and 35 more
## #   variables: recovered <dbl>,
## #   ecdc_cases <dbl>, ecdc_deaths <dbl>,
## #   total_tests <dbl>, tests_units <chr>,
## #   positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>, ...
```

Note that we used the `ymd` to define the stings "2021-06-11", "2021-06-12" as dates.

2.2 Reorder rows using `arrange()`

We can also arrange a table based on one or more variables. The `arrange()` function works similarly to `filter()` except that instead of selecting rows, it changes their order. It takes a data frame and a set of column names (or more complicated expressions) to order by. For example, we can arrange the rows of the `june11b` table by the number of confirmed cases (in ascending order which is the default):

```
arrange(june11b, confirmed)
```

```
## # A tibble: 6 x 40
##   iso3c country   date       confirmed deaths
##   <chr> <chr>     <date>         <dbl>  <dbl>
## 1 RUS   Russia    2021-06-11   5120578 123568
## 2 TUR   Turkey    2021-06-11   5319359  48593
## 3 FRA   France    2021-06-11   5795593 110516
## 4 BRA   Brazil    2021-06-11  17296118 484235
## 5 IND   India     2021-06-11  29359155 367081
## 6 USA   United S~ 2021-06-11  33502472 599401
## # ... with 35 more variables:
## #   recovered <dbl>, ecdc_cases <dbl>,
## #   ecdc_deaths <dbl>, total_tests <dbl>,
## #   tests_units <chr>, positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>,
## #   soc_dist <dbl>, mov_rest <dbl>, ...
```

We can use `desc()` to re-arrange in descending order. For example:

```
arrange(june11b, desc(confirmed))
```

```
## # A tibble: 6 x 40
##   iso3c country   date       confirmed deaths
##   <chr> <chr>     <date>         <dbl>  <dbl>
```

```
## 1 USA   United S~ 2021-06-11  33502472 599401
## 2 IND   India     2021-06-11  29359155 367081
## 3 BRA   Brazil    2021-06-11  17296118 484235
## 4 FRA   France    2021-06-11   5795593 110516
## 5 TUR   Turkey    2021-06-11   5319359  48593
## 6 RUS   Russia    2021-06-11   5120578 123568
## # ... with 35 more variables:
## #   recovered <dbl>, ecdc_cases <dbl>,
## #   ecdc_deaths <dbl>, total_tests <dbl>,
## #   tests_units <chr>, positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>,
## #   soc_dist <dbl>, mov_rest <dbl>, ...
```

2.3 Subseting variables (columns) using `select()`

It's not uncommon to get datasets with hundreds or even thousands of variables. In this case, the first challenge is often narrowing in on the variables we're actually interested in.

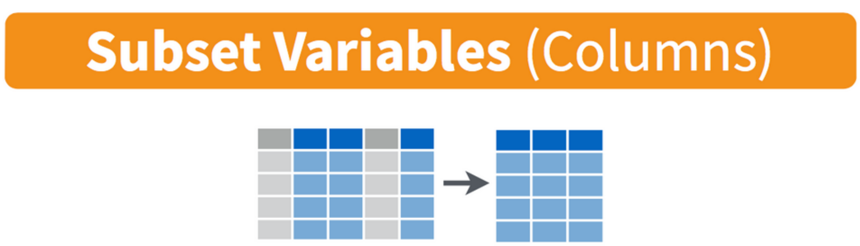


Figure 2: Diagram of select() columns

`select()` allows us to rapidly zoom in on a useful subset using operations based on the names of the variables. The first argument to this function is the data frame (covid_data), and the subsequent arguments are the columns to keep.

For example, we can select only the `country`, `confirmed`, `deaths`, `recovered` variables from the data frame:

```
select(covid_data, country, region, date, confirmed, deaths, recovered)
```

```
##    country                     region
## 1    Aruba Latin America & Caribbean
## 2    Aruba Latin America & Caribbean
## 3    Aruba Latin America & Caribbean
## 4    Aruba Latin America & Caribbean
## 5     <NA>                       <NA>
## 6 Zimbabwe         Sub-Saharan Africa
## 7 Zimbabwe         Sub-Saharan Africa
## 8 Zimbabwe         Sub-Saharan Africa
## 9 Zimbabwe         Sub-Saharan Africa
##         date confirmed deaths recovered
## 1 2020-03-13      <NA>   <NA>      <NA>
## 2 2020-03-14      <NA>   <NA>      <NA>
## 3 2020-03-15      <NA>   <NA>      <NA>
## 4 2020-03-16      <NA>   <NA>      <NA>
## 5       <NA>       ...    ...       ...
## 6 2021-10-02    131094   4625         0
## 7 2021-10-03    131129   4627         0
## 8 2021-10-04    131129   4627         0
## 9 2021-10-05    131205   4627         0
```

Alternatively, we can select variables by index (it is not suggested):

```
select(covid_data, 2, 32, 3:6)
```

Moreover, we can excluding variables by name. For example, we can exclude the first variable iso3c:

```
select(covid_data, -iso3c)
```

We can also select all, for example, character columns by using `select_if()`:

```
select_if(covid_data, is.character)
```

```
## # A tibble: 131,681 x 6
##    iso3c country tests_units gcmr_place_id
##    <chr> <chr>   <chr>       <chr>
##  1 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  2 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  3 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  4 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  5 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  6 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  7 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  8 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
##  9 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
## 10 ABW   Aruba   <NA>        ChIJ23da4s84hY4~
## # ... with 131,671 more rows, and 2 more
## #   variables: region <chr>, income <chr>
```

2.4 Subsetting columns and rows using pipe operator %>% and `dplyr`.

Now, let's introduce a very nifty tool that gets loaded along with the `dplyr` package: the pipe operator %>%. The pipe operator allows us to combine multiple operations on a computer into a single sequential *chain* of actions.The pipe, %>%, comes from the `magrittr` package. Packages in the tidyverse load %>% for us automatically, so we don't usually load magrittr explicitly.

Let's start with an example. Say we would like to perform a sequence of operations on a data frame `x` using the functions `f()`, and `g()`:

1. Take `x` *then*
2. Use `x` as an input to a function `f()` *then*
3. Use the output of `f(x)` as an input to a function `g()`

One way to achieve this sequence of operations is by using nesting parentheses as follows:

```
g(f(x))
```

This code isn't so hard to read since we are applying only two functions: `f()`, then `g()`. However, we can imagine that this will get progressively harder to read as the number of functions applied in our sequence increases. This is where the pipe operator `%>%` comes in handy. `%>%` takes the output of one function and then "pipes" it to be the input of the next function. Furthermore, a helpful trick is to read `%>%` as "then" or "and then." For example, we can obtain the same output as the hypothetical sequence of functions as follows:

```
x %>%
  f() %>%
  g()
```

We would read this sequence as:

1. Take `x` *then*
2. Use this output as the input to the function `f()` *then*
3. Use this output as the input to the next function `g()`

So while both approaches achieve the same goal, the latter is much more human-readable because we can clearly read the sequence of operations line-by-line. But what are the hypothetical `x`, `f()`, and `g()`? Throughout this chapter on data wrangling:

1. The starting value `x` will be a data frame. For example, the `covid_data`.
2. The sequence of functions, here `f()`, and `g()`, will mostly be a sequence of any number of the data wrangling verb-named functions included in the `dplyr` package. For example, the `filter()` and `select()` functions we previewed earlier.

3. The result will be the transformed/modified data frame that we want. In our example, we'll save the result in a new data frame by using the <- assignment operator with the name `covid_data2`.

```
covid_data2 <- covid_data %>%
  filter(date == "2021-06-12") %>%
  select(country, region, date, confirmed, deaths, recovered)
covid_data2
```

```
## # A tibble: 215 x 6
##    country region date       confirmed deaths
##    <chr>   <chr>  <date>         <dbl>  <dbl>
##  1 Aruba   "Lati~ 2021-06-12        NA     NA
##  2 Afghan~ "Sout~ 2021-06-12     88740   3449
##  3 Angola  "Sub-~ 2021-06-12     36600    825
##  4 Anguil~  <NA>  2021-06-12        NA     NA
##  5 Albania "Euro~ 2021-06-12    132449   2453
##  6 Andorra "Euro~ 2021-06-12     13813    127
##  7 United~ "Midd~ 2021-06-12    596017   1724
##  8 Argent~ "Lati~ 2021-06-12   4111147  85075
##  9 Armenia "Euro~ 2021-06-12    223643   4482
## 10 Antigu~ "Lati~ 2021-06-12      1263     42
## # ... with 205 more rows, and 1 more
## #   variable: recovered <dbl>
```

Mario analogy

2.5 Summaries of variables using `summarise()` and `across()`

The next common task when working with data frames is to compute summary statistics that are single numerical values that summarize a large number of values.

Figure 3: Diagram of summarise() rows

Commonly known examples of summary statistics for continuous variables include the mean (also called the average) and the median (the middle value). Other examples of summary statistics that might not immediately come to mind include the sum, the smallest value also called the minimum, the largest value also called the maximum, the 1st quartile and 3rd quartile, the standard deviation, the skewness, and the kurtosis. Summary statistics for categorical variables include frequencies and percentages.

For example, we can calculate the summary statistics for the confirmed cases until the date `2021-06-12`:

```
summary_confirmed <- covid_data %>%
  filter(date == "2021-06-12") %>%
  dplyr::summarise(mean_confirmed = mean(confirmed, na.rm = TRUE),
                   sd_confirmed = sd(confirmed, na.rm = TRUE))
summary_confirmed
```

```
## # A tibble: 1 x 2
##   mean_confirmed sd_confirmed
##            <dbl>        <dbl>
## 1        924261.     3550982.
```

Next, we can utilize the `across()` function in the `summarise()` function to apply statistical calculations to multiple columns. For example for the `confirmed` and `deaths` variables:

```r
summary_2variables <- covid_data %>%
  filter(date == "2021-06-12") %>%
  dplyr::summarise(across(
    .cols = c(confirmed, deaths),
    .fns = list(
      N = ~n(),
      Min = min,
      Q1 = ~quantile(., 0.25, na.rm = TRUE),
      median = median,
      Q3 = ~quantile(., 0.75, na.rm = TRUE),
      Max = max,
      Mean = mean,
      Sd = sd,
      Skewness = EnvStats::skewness,
      Kurtosis= EnvStats::kurtosis),
    na.rm = TRUE,
    .names = "{col}_{fn}")
    )

summary_2variables
```

Stats	Values
confirmed_N	215.0
confirmed_Min	0.0
confirmed_Q1	12794.0
confirmed_median	101567.0
confirmed_Q3	412313.8
confirmed_Max	33511160.0
confirmed_Mean	924260.6
confirmed_Sd	3550981.6
confirmed_Skewness	7.5
confirmed_Kurtosis	61.0
deaths_N	215.0
deaths_Min	0.0
deaths_Q1	167.0
deaths_median	1356.0
deaths_Q3	8307.5
deaths_Max	599695.0
deaths_Mean	19982.9
deaths_Sd	67896.0
deaths_Skewness	6.1
deaths_Kurtosis	42.9

2.6 Grouped summaries with `group_by()` and `summarise()`

To better understand the data, we can calculate summary statistics by geographic region. Let's say we are interested in the number of confirmed cases per 100,000 inhabitants up to `2021-06-12` and the number of countries for each geographic region.

```
cases_by_region <- covid_data %>%
  filter(date == "2021-06-12") %>%
  group_by(region) %>%
  dplyr::summarise(
    cases_per_100k = sum(confirmed, na.rm = TRUE) /
      sum(population, na.rm = TRUE)*100000,
    countries = n()
    ) %>%
  filter(region!= 'NA') %>%
  ungroup() # ungrouping variable is a good habit to prevent errors
```

Table 2: Covid19 by Region

region	cases_per_100k	countries
East Asia & Pacific	238.4853	29
Europe & Central Asia	5921.4061	55
Latin America & Caribbean	5352.4518	41
Middle East & North Africa	2230.2717	21
North America	9500.6897	3
South Asia	1733.7837	8
Sub-Saharan Africa	310.6703	48

Together `group_by()` and `summarise()` provide one of the tools that we'll use most commonly when working with dplyr: grouped summaries.

Grouping the filtered (up to `2021-06-12`) `covid_data` by `region` and then applying the `summarize()` function yields a data frame that displays the cases per 100,000 inhabitants split by region.

It is important to note that the `group_by()` function doesn't change data frame by itself. It is only after we apply the `summarize()` function that the data frame changes.

2.7 Add new variables with `mutate()`

Besides transforming of existing columns, it's often useful to add new columns that are functions of existing columns. That's the job of `mutate()` that adds new columns at the end of our dataset. For example we want to calculate the cases per 100,000 inhabitants and tests per capita up to `2021-06-12` for countries with more than 1,000,000:

```
dat <- covid_data %>%
  filter(date == "2021-06-12", population > 1000000) %>%
  mutate(cases_per_100k = confirmed /
           population * 100000,
         tests_per_capita = total_tests /
           population)
dat
```

```
## # A tibble: 155 x 42
##    iso3c country  date       confirmed deaths
##    <chr> <chr>    <date>         <dbl>  <dbl>
##  1 AFG   Afghani~ 2021-06-12     88740   3449
##  2 AGO   Angola   2021-06-12     36600    825
##  3 ALB   Albania  2021-06-12    132449   2453
##  4 ARE   United ~ 2021-06-12    596017   1724
##  5 ARG   Argenti~ 2021-06-12   4111147  85075
##  6 ARM   Armenia  2021-06-12    223643   4482
##  7 AUS   Austral~ 2021-06-12     30248    910
##  8 AUT   Austria  2021-06-12    648387  10652
```

```
##  9 AZE   Azerbai~ 2021-06-12    335126   4953
## 10 BDI   Burundi  2021-06-12      4995      8
## # ... with 145 more rows, and 37 more
## #   variables: recovered <dbl>,
## #   ecdc_cases <dbl>, ecdc_deaths <dbl>,
## #   total_tests <dbl>, tests_units <chr>,
## #   positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>, ...
```

Let's say that we also want to categorize the numeric variable `life_expectancy` to countries with life expectancy 65 years or less and countries with more than 65 years. We can use the `cut()` function inside the `mutate()`:

```
dat2 <- dat %>%
  mutate(life_expectancy_cat=cut(life_expectancy,
                                 breaks=c(-Inf, 65, Inf),
                                 labels=c("65 yrs or less","more than 65 yrs")))

dat2
```

```
## # A tibble: 155 x 43
##    iso3c country  date       confirmed deaths
##    <chr> <chr>    <date>         <dbl>  <dbl>
##  1 AFG   Afghani~ 2021-06-12     88740   3449
##  2 AGO   Angola   2021-06-12     36600    825
##  3 ALB   Albania  2021-06-12    132449   2453
##  4 ARE   United ~ 2021-06-12    596017   1724
##  5 ARG   Argenti~ 2021-06-12   4111147  85075
##  6 ARM   Armenia  2021-06-12    223643   4482
##  7 AUS   Austral~ 2021-06-12     30248    910
##  8 AUT   Austria  2021-06-12    648387  10652
##  9 AZE   Azerbai~ 2021-06-12    335126   4953
## 10 BDI   Burundi  2021-06-12      4995      8
```

```
## # ... with 145 more rows, and 38 more
## #   variables: recovered <dbl>,
## #   ecdc_cases <dbl>, ecdc_deaths <dbl>,
## #   total_tests <dbl>, tests_units <chr>,
## #   positive_rate <dbl>,
## #   hosp_patients <dbl>, icu_patients <dbl>,
## #   total_vaccinations <dbl>, ...
```

2.8 Count the unique values with `count()`

Using `count()` is a convenient way to get a sense of the distribution of values of one or more categorical variables in a dataset. For example, we can count the countries with life expectancy 65 years or less and countries with more than 65 years of our new `life_expectancy_cat` variable in the `dat2`:

```
count_dat <- dat2 %>%
  count(life_expectancy_cat)

count_dat
```

```
## # A tibble: 2 x 2
##   life_expectancy_cat     n
##   <fct>               <int>
## 1 65 yrs or less         36
## 2 more than 65 yrs      119
```

In this instance, the output is equivalent to:

```
table(dat2$life_expectancy_cat)
```

```
##
##   65 yrs or less more than 65 yrs
##               36              119
```

3 Reshaping data - long vs wide format

So far, all of the examples we've shown you have been using 'tidy' data. Data is 'tidy' when it is in long format: each variable is in its own column, and each observation is in its own row. This long format is efficient to use in data analysis and visualisation and can also be considered "computer readable".

But sometimes when presenting data in tables for humans to read, or when collecting data directly into a spreadsheet, it can be convenient to have data in a wide format. Data is 'wide' when some or all of the columns are levels of a factor. An example makes this easier to see.

```
gbd_wide <- read_csv(here::here("data", "global_burden_disease_wide-format.csv"))
```

```
## Rows: 3 Columns: 5

## -- Column specification ---------------------
## Delimiter: ","
## chr (1): cause
## dbl (4): Female_1990, Female_2017, Male_1...

##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
gbd_wide
```

```
## # A tibble: 3 x 5
##   cause     Female_1990 Female_2017 Male_1990
##   <chr>           <dbl>       <dbl>     <dbl>
## 1 Communic~        7.3         4.91      8.06
## 2 Injuries         1.41        1.42      2.84
## 3 Non-comm~       12.8        19.2      13.9
## # ... with 1 more variable: Male_2017 <dbl>
```

3.1 Pivot values from columns to rows (longer)

We'll need to know how to wrangle the variables currently spread across different columns into the tidy format (where each column is a variable, each row is an observation). For example, here we want to collect all the columns that include the words Female or Male:

```
gbd_wide %>%
  pivot_longer(starts_with(c("Female", "Male")),
               names_to = "sex_year",
               values_to = "deaths_millions")
```

```
## # A tibble: 12 x 3
##    cause                     sex_year deaths_millions
##    <chr>                     <chr>              <dbl>
##  1 Communicable diseases     Female_~            7.3
##  2 Communicable diseases     Female_~            4.91
##  3 Communicable diseases     Male_19~            8.06
##  4 Communicable diseases     Male_20~            5.47
##  5 Injuries                  Female_~            1.41
##  6 Injuries                  Female_~            1.42
##  7 Injuries                  Male_19~            2.84
##  8 Injuries                  Male_20~            3.05
##  9 Non-communicable diseases Female_~           12.8
## 10 Non-communicable diseases Female_~           19.2
## 11 Non-communicable diseases Male_19~           13.9
## 12 Non-communicable diseases Male_20~           21.7
```

While `pivot_longer()` did a great job fetching the different observations that were spread across multiple columns into a single one, it's still a combination of two variables - sex and year. We can use the `separate()` function to deal with that.

```
gbd_long <- gbd_wide %>%
  pivot_longer(starts_with(c("Female", "Male")),
               names_to = "sex_year",
               values_to = "deaths_millions") %>%
  separate(sex_year, into = c("sex", "year"), sep = "_", convert = TRUE)
gbd_long
```

```
## # A tibble: 12 x 4
##    cause           sex    year deaths_millions
##    <chr>           <chr> <int>           <dbl>
##  1 Communicable ~ Fema~   1990            7.3
##  2 Communicable ~ Fema~   2017            4.91
##  3 Communicable ~ Male    1990            8.06
##  4 Communicable ~ Male    2017            5.47
##  5 Injuries       Fema~   1990            1.41
##  6 Injuries       Fema~   2017            1.42
##  7 Injuries       Male    1990            2.84
##  8 Injuries       Male    2017            3.05
##  9 Non-communica~ Fema~   1990           12.8
## 10 Non-communica~ Fema~   2017           19.2
## 11 Non-communica~ Male    1990           13.9
## 12 Non-communica~ Male    2017           21.7
```

We've also added `convert = TRUE` to `separate()` so year would get converted into a numeric variable. The combination of, e.g., "Female-1990" is a character variable, so after separating them both sex and year would still be classified as characters. But the `convert = TRUE` recognises that year is a number and will appropriately convert it into an integer.

3.2 Pivot values from rows into columns (wider)

The inverse of `pivot_longer()` is the `pivot_wider()`. First, we need to combine the `sex` and `year` vectors into a single character vector `sex_year` using the `str_c()`

function. Then we use `pivot_wider()` with two arguments: `names_from=` and `values_from=`:

```
gbd_wider <- gbd_long %>%
  mutate(sex_year = str_c(sex, year, sep = "_")) %>%
  select(-sex, -year) %>%
  pivot_wider(names_from = sex_year, values_from = deaths_millions)

gbd_wider
```

```
## # A tibble: 3 x 5
##   cause     Female_1990 Female_2017 Male_1990
##   <chr>           <dbl>       <dbl>     <dbl>
## 1 Communic~        7.3         4.91      8.06
## 2 Injuries         1.41        1.42      2.84
## 3 Non-comm~       12.8        19.2      13.9
## # ... with 1 more variable: Male_2017 <dbl>
```

4 Activities

Activity 1

Read the data from the `arrhythmia.csv` file and inspect the data frame.

- Convert the variable `sex` into a factor variable (0 for males and 1 for females) (hint: use `factor` or `recode_factor`).
- Calculate the Body mass index (BMI) for the participants from weight(kg) and height (cm). CDC
- Create BMI categories only for the adults (age >=18) based on the CDC

Activity 2

For the adults:

- filter only the overweight or obese patients
- filter those overweight or obese people who have heart rate greater than or equal to 85.
- calculate the mean and standard deviation of the QRS variable using the `summarise()` from `{dplyr}` package.
- calculate the mean and sd of QRS for the participants in each BMI category, separately.