Data Wrangling

Dr. Aline Coutinho

The principles of tidy data

The sun would shine brighter if all datasets were clean and ready to be analyzed. However, truth is, a huge portion of data analysis consists of cleaning data, or how we say in the R community, *tidying data*.

The raw data you collect often needs processing. It is highly advisable that any data reformatting is clearly recorded so to increase reproducibility (see tutorials on *reproducible research* and *data analysis*). The central principles of tidy data are that each column refers to a variable, each row to an observation, and each cell records one specific value or measurement (see Figure 1).

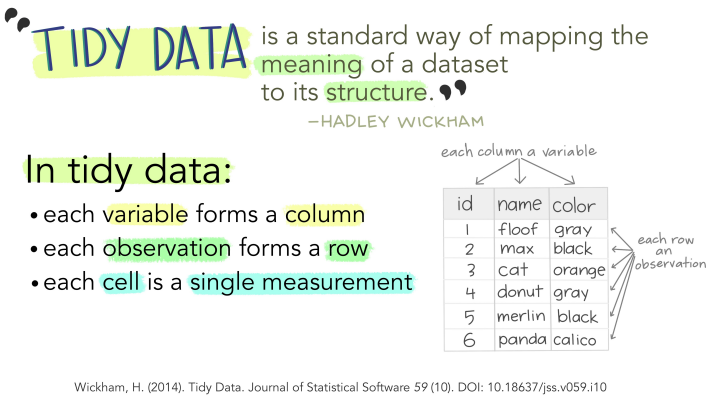


Figure 1: Tidy data, figure by Allison Horst

What is data wrangling?

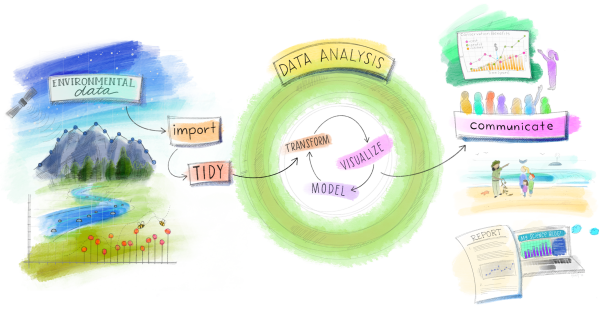


Figure 2: Data science, figure by Allison Horst

Data wrangling is the term that the R community uses to refer to the action of transforming data into tidy data. The packages `tidyr` and `dplyr` provide a useful grammar to wrangle data. Perhaps, the most important tool is the pipe operator `%>%` because it makes the code easier to read.

R is a functional language, which means that a function usually returns a value. The pipe operator `%>%` passes the value of a function to another function right after the pipe.

For example: f(x) returns A. That value A can be passed to g(f(x)), and so on. This is how the code would look like:

```
# Without the pipe operator:
function(dataframe, function argument)

# With the pipe operator:
dataframe %>% function(function argument)
```

The value of the function on the left always passes to the function on the right, and never vice-versa!

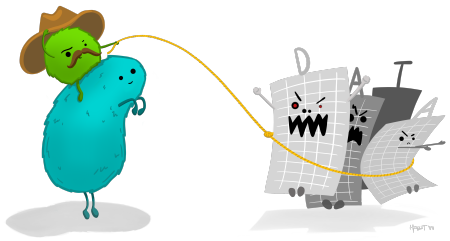


Figure 3: Data wrangling, figure by Allison Horst

```
# Suppose we want information about the unemployment rate in Canada.
# We know that we can get such information from the Statistics Canada.
# This is the table: 14-10-0020-01 (formerly CANSIM 282-0004).

# To do so, we upload the {tidyverse} and the {cansim} packages, and import the dataset into R.

# This is the code:

library(tidyverse) #So we can use the pipe operator
```

```
## Warning: package 'tibble' was built under R version 4.0.3

## Warning: package 'readr' was built under R version 4.0.3
```

```
library(cansim)

df <-get_cansim('14-10-0020-01')%>%
  normalize_cansim_values()
```

We can get a look at the Table 14-10-0020-01, which we stored under the object *UnemploymentRate* by using the `glimpse()` function:

```
glimpse(df)
```

```
## Rows: 828,630
## Columns: 26
## $ REF_DATE                                                <chr> "1990", "199...
## $ GEO                                                     <chr> "Canada", "C...
## $ DGUID                                                   <chr> "2016A000011...
## $ `Labour force characteristics`                          <chr> "Population"...
## $ `Educational attainment`                                <chr> "Total, all ...
## $ Sex                                                     <chr> "Both sexes"...
## $ `Age group`                                             <chr> "15 years an...
## $ UOM                                                     <chr> "Persons", "...
## $ UOM_ID                                                  <chr> "249", "249"...
## $ VECTOR                                                  <chr> "v2582391", ...
## $ COORDINATE                                              <chr> "1.1.1.1.1",...
## $ VALUE                                                   <dbl> 21214700, 39...
## $ STATUS                                                  <chr> NA, NA, NA, ...
## $ SYMBOL                                                  <chr> NA, NA, NA, ...
## $ TERMINATED                                              <chr> NA, NA, NA, ...
## $ DECIMALS                                                <chr> "1", "1", "1...
## $ GeoUID                                                  <chr> "11124", "11...
## $ `Classification Code for Labour force characteristics`  <chr> NA, NA, NA, ...
## $ `Hierarchy for Labour force characteristics`            <chr> "1", "1", "1...
## $ `Classification Code for Educational attainment`        <chr> NA, NA, NA, ...
## $ `Hierarchy for Educational attainment`                  <chr> "1", "1", "1...
## $ `Classification Code for Sex`                           <chr> NA, NA, NA, ...
## $ `Hierarchy for Sex`                                     <chr> "1", "1", "1...
## $ `Classification Code for Age group`                     <chr> NA, NA, NA, ...
## $ `Hierarchy for Age group`                               <chr> "1", "2", "3...
## $ Date                                                    <date> 1990-01-01,...
```

Note how the dataset is enormous. It has 26 variables and 801,900 observations. Usually, we are only interested in some of the variables.

So, how to start manipulating this huge dataset? By making it more manageable.

Basic operations

The first thing you might want to do in your dataframe is to rename some variable. You can use the `rename()` verb for this task:

```
Renamed_DF <- rename(df, c("LFC" = "Labour force characteristics",
                           "EA" = "Educational attainment",
                           "Age" = "Age group"))

head(Renamed_DF)
```

```
## # A tibble: 6 x 26
##   REF_DATE GEO   DGUID LFC   EA    Sex   Age   UOM   UOM_ID VECTOR COORDINATE
##   <chr>    <chr> <chr> <chr> <chr> <chr> <chr> <chr> <chr>  <chr>  <chr>
## 1 1990     Cana~ 2016~ Popu~ Tota~ Both~ 15 y~ Pers~ 249    v2582~ 1.1.1.1.1
```

```
## 2 1990     Cana~ 2016~ Popu~ Tota~ Both~ 15 t~ Pers~ 249    v2582~ 1.1.1.1.2
## 3 1990     Cana~ 2016~ Popu~ Tota~ Both~ 25 y~ Pers~ 249    v2582~ 1.1.1.1.3
## 4 1990     Cana~ 2016~ Popu~ Tota~ Both~ 25 t~ Pers~ 249    v2582~ 1.1.1.1.4
## 5 1990     Cana~ 2016~ Popu~ Tota~ Both~ 45 y~ Pers~ 249    v2582~ 1.1.1.1.5
## 6 1990     Cana~ 2016~ Popu~ Tota~ Both~ 25 t~ Pers~ 249    v2582~ 1.1.1.1.6
## # ... with 15 more variables: VALUE <dbl>, STATUS <chr>, SYMBOL <chr>,
## #   TERMINATED <chr>, DECIMALS <chr>, GeoUID <chr>, `Classification Code for
## #   Labour force characteristics` <chr>, `Hierarchy for Labour force
## #   characteristics` <chr>, `Classification Code for Educational
## #   attainment` <chr>, `Hierarchy for Educational attainment` <chr>,
## #   `Classification Code for Sex` <chr>, `Hierarchy for Sex` <chr>,
## #   `Classification Code for Age group` <chr>, `Hierarchy for Age group` <chr>,
## #   Date <date>
```

`select()` is a dplyr verb that allows the user to select columns (remember that in tidy data, a column corresponds to a variable). Note that `select()` and `pull()` are two similar functions. However, it is important to note that `pull` does not create a tibble.

```
Selected_DF <- df %>%
  rename(c("LFC" = "Labour force characteristics",
           "EA" = "Educational attainment",
           "Age" = "Age group"))%>%
  select(REF_DATE, GEO, LFC, EA, Sex, Age, VALUE, Date)

head(Selected_DF)
```

```
## # A tibble: 6 x 8
##   REF_DATE GEO    LFC      EA                Sex     Age         VALUE Date
##   <chr>    <chr>  <chr>    <chr>             <chr>   <chr>       <dbl> <date>
## 1 1990     Canada Populat~ Total, all educ~ Both s~ 15 years ~ 2.12e7 1990-01-01
## 2 1990     Canada Populat~ Total, all educ~ Both s~ 15 to 24 ~ 3.92e6 1990-01-01
## 3 1990     Canada Populat~ Total, all educ~ Both s~ 25 years ~ 1.73e7 1990-01-01
## 4 1990     Canada Populat~ Total, all educ~ Both s~ 25 to 44 ~ 9.18e6 1990-01-01
## 5 1990     Canada Populat~ Total, all educ~ Both s~ 45 years ~ 8.11e6 1990-01-01
## 6 1990     Canada Populat~ Total, all educ~ Both s~ 25 to 54 ~ 1.20e7 1990-01-01
```

The `recode()` function is a verb that allows you to recode some variables. This is a powerful tool to change characters to numbers and vice-versa. First identify the column, then describe the task (on the left is the original value, on the right is the new value). Your code will be similar to the following:

```
recode(dataframe$column, 'original value' = 'new value')
```

> **Important:**
>
> All values in the columns must be recoded otherwise the ones left out will be recorded as missing values!

Suppose you want to filter specific observations within a variable. For example, you might be solely interested in the unemployment rate. To do so, you have to use the funtion `filter()`.

Note that `filter()` is a dplyr verb that, as the name suggests, filter rows by an specified definition.

Suppose you want to explore solely the unemployment rate of Statistics Canada Labour Force Survey. You can use the verb `filter()` to pull out only the unemployment rate of the variable *Labour force characteristics*. This is the code:

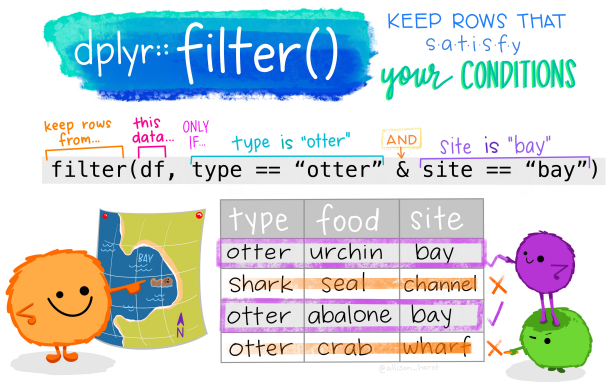


Figure 4: The function filter(), figure by Allison Horst

```
Filtered_DF1 <- df %>%
  rename(c("LFC" = "Labour force characteristics",
           "EA" = "Educational attainment",
           "Age" = "Age group")) %>%
  select(REF_DATE, GEO, LFC, EA, Sex, Age, VALUE, Date)%>%
  filter(LFC == "Unemployment rate")

head(Filtered_DF1)
```

```
## # A tibble: 6 x 8
##   REF_DATE GEO    LFC         EA              Sex     Age         VALUE Date
##   <chr>    <chr>  <chr>       <chr>           <chr>   <chr>       <dbl> <date>
## 1 1990     Canada Unemploym~ Total, all ed~ Both s~ 15 years ~ 0.081  1990-01-01
## 2 1990     Canada Unemploym~ Total, all ed~ Both s~ 15 to 24 ~ 0.123  1990-01-01
## 3 1990     Canada Unemploym~ Total, all ed~ Both s~ 25 years ~ 0.071  1990-01-01
## 4 1990     Canada Unemploym~ Total, all ed~ Both s~ 25 to 44 ~ 0.077  1990-01-01
## 5 1990     Canada Unemploym~ Total, all ed~ Both s~ 45 years ~ 0.0580 1990-01-01
## 6 1990     Canada Unemploym~ Total, all ed~ Both s~ 25 to 54 ~ 0.073  1990-01-01
```

The same can be done for other variables. Suppose you want to filter the rows for variables such as *Labour force characteristics* and *Geography* and *Sex*:

```
Filtered_DF2 <- df %>%
  rename(c("LFC" = "Labour force characteristics",
           "EA" = "Educational attainment",
           "Age" = "Age group")) %>%
  select(REF_DATE, GEO, LFC, EA, Sex, Age, VALUE, Date)%>%
  filter(LFC == "Unemployment rate") %>%
  filter(Age == "25 years and over")%>%
  filter(Sex == "Both sexes") %>%
  filter(EA == "Total, all education levels")%>%
  filter(GEO == "Canada" | GEO == "Ontario")

head(Filtered_DF2)
```

```
## # A tibble: 6 x 8
```

```
##   REF_DATE GEO    LFC         EA              Sex     Age        VALUE Date
##   <chr>    <chr>  <chr>       <chr>           <chr>   <chr>      <dbl> <date>
## 1 1990     Canada Unemployme~ Total, all ed~ Both s~ 25 years ~ 0.071 1990-01-01
## 2 1990     Ontar~ Unemployme~ Total, all ed~ Both s~ 25 years ~ 0.052 1990-01-01
## 3 1991     Canada Unemployme~ Total, all ed~ Both s~ 25 years ~ 0.091 1991-01-01
## 4 1991     Ontar~ Unemployme~ Total, all ed~ Both s~ 25 years ~ 0.083 1991-01-01
## 5 1992     Canada Unemployme~ Total, all ed~ Both s~ 25 years ~ 0.099 1992-01-01
## 6 1992     Ontar~ Unemployme~ Total, all ed~ Both s~ 25 years ~ 0.093 1992-01-01
```

A bit more about the pipe operator

Figure 5: The pipe operator

The pipe operator is from the `magittr` package and it facilitates the flow among the dplyr verbs. It allows you to combine multiple commands.

Remember how the pipe operator passes the value of a function to the next function? Suppose you want to modify or create a new column. You can use the function `mutate()` to calculate the unemployment gap between men and women in Canada:

```
gap <- df %>%
  rename(c("LFC" = "Labour force characteristics",
           "EA" = "Educational attainment",
           "Age" = "Age group")) %>%
  select(REF_DATE, GEO, LFC, EA, Sex, Age, VALUE, REF_DATE)%>%
  filter(LFC == "Unemployment rate") %>%
  filter(Age == "25 years and over")%>%
  filter(EA == "Total, all education levels")%>%
  filter(GEO == "Canada")%>%
  filter(Sex == "Males" | Sex == "Females")%>%
  spread(Sex, VALUE)%>%
  mutate(gap = `Males` - `Females`)

head(gap)
```

```
## # A tibble: 6 x 8
##   REF_DATE GEO    LFC         EA              Age        Females Males      gap
##   <chr>    <chr>  <chr>       <chr>           <chr>        <dbl> <dbl>    <dbl>
## 1 1990     Canada Unemployme~ Total, all educ~ 25 years ~  0.073  0.07  -0.00300
```

Figure 6: The function mutate(), figure by Allison Horst

```
## 2 1991     Canada Unemployme~ Total, all educ~ 25 years ~  0.089  0.092  0.00300
## 3 1992     Canada Unemployme~ Total, all educ~ 25 years ~  0.092  0.104  0.012
## 4 1993     Canada Unemployme~ Total, all educ~ 25 years ~  0.098  0.104  0.006
## 5 1994     Canada Unemployme~ Total, all educ~ 25 years ~  0.089  0.095  0.00600
## 6 1995     Canada Unemployme~ Total, all educ~ 25 years ~  0.0820 0.085  0.003
```

Another important verb is `group_by` which allows you to group by certain specifications. This function is extremely helpful if you combine it with the `summarise()` function. This latter verb creates a new dataframe and generates summary statistics such as means, medians, standard deviations, and so on.

`summarise()` does not need to be used in conjunction with `group_by()`, but it allows to get summaries by groups you've specified.

Suppose you want to get the average unemployment rate for both men and women aged 25 years or older in Canada. This is the code. Note how men have a slighter unemployement rate than women.

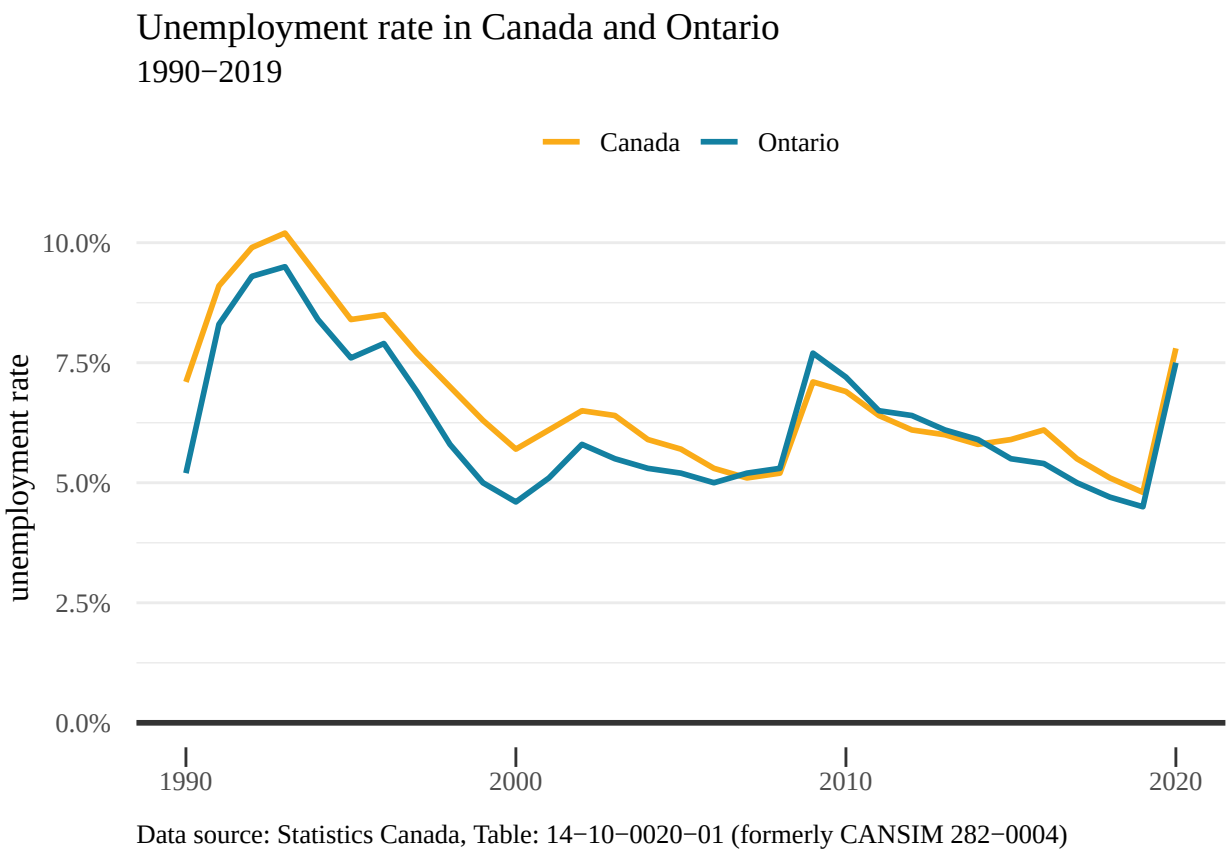
```
summary_DF <- df %>%
  rename(c("LFC" = "Labour force characteristics",
           "EA" = "Educational attainment",
           "Age" = "Age group")) %>%
  filter(LFC == "Unemployment rate") %>%
  filter(Age == "25 years and over")%>%
  filter(EA == "Total, all education levels")%>%
  filter(GEO == "Canada")%>%
  group_by(Sex)%>%
  summarise(mean(VALUE))

head(summary_DF)
```

```
## # A tibble: 3 x 2
```

```
##   Sex         `mean(VALUE)`
##   <chr>               <dbl>
## 1 Both sexes         0.0674
## 2 Females            0.0649
## 3 Males              0.0695
```

This is a simple plot produced with the dataset we wrangled.



But this is a topic for another time. For now, have fun wrangling data!

Links to additional resources

Hadley Wickham's famous article *Tidy Data* (2014)

An informal online summary of Wickham's article above

Chapter 5 of *R for Data Science*

Data Wrangling Cheat Sheet

Introduction to dplyr

Some dplyr verbs

More information on the mutate verb

More information on the summarise verb