Data Wrangling Basics

AUTHOR
Brian Gural, Lorrie He, JP Flores

What is data wrangling?

- Data wrangling, manipulation, or cleaning is the process of transforming data into a format that is more suitable for analysis. This can include removing missing values, changing the format of data, or combining multiple datasets.
- There's rarely a single way to approach any given data-wrangling problem! Expanding your "toolkit" allows you to tackle problems from different angles.

Measure twice, cut once

Before you begin wrangling data, you should be able to:

- Define how you want the data to look and why
- Document it well so that others (and future you!) know what you did
- Know what tools you have and how to use them

Building a toolkit

Working with vectors

Pulling out specific parts of a data set is important when analyzing with R. **Indexing**, or accessing elements, subsets data based on numeric positions in a vector. Some things to be aware of when indexing:

- Indexing uses brackets. i.e. the 5th element in a vector will be returned if you run `vector[5]` .
- It's helpful for getting several elements at once, or reordering data.

Here are some examples:

```
# First, we'll make a vector to play with
names <- c("rosalind", "marie", "barbara")
```

```
# if we print the output, we'd get:
names
```

```
[1] "rosalind" "marie"    "barbara"
```

```
# If we want to access the first name, we can use brackets ar
names[1]
```

```
[1] "rosalind"
```

```
# This works with any position, for example the third name:
names[3]
```

```
[1] "barbara"
```

```
# You can index more than one position at a time too:
names[c(1,2)]
```

```
[1] "rosalind" "marie"
```

```
# Changing the order of numbers you supply changes the order
names[c(2,1)]
```

```
[1] "marie"    "rosalind"
```

Working with data frames

This works for two-dimensional structures too, like data frames and matrices. We'd just format it as: `dataframe[row,column]` . Let's try it out:

```
# Make a data frame
df <- data.frame(
  name = c("Rosalind Franklin", "Marie Curie", "Barbara McCl
           "Lise Meitner", "Grace Hopper", "Chien-Shiung Wu",
  field = c("DNA X-ray crystallography", "Radioactivity", "Ge
            "Nuclear Physics", "Computer Programming", "Exper
  school = c("Cambridge", "Sorbonne", "Cornell", "University
             "University of Berlin", "Yale", "Princeton", "Wa
  date_of_birth = c("1920-07-25", "1867-11-07", "1902-06-16",
                    "1878-11-07", "1906-12-09", "1912-05-31",
  working_region = c("Western Europe", "Western Europe", "Nor
)

# To get the first row:
df[1,]
```

```
               name                     field    school date_of_birth
1 Rosalind Franklin DNA X-ray crystallography Cambridge    1920-07-25
  working_region
1 Western Europe
```

```
# or the first column:
df[,1]
```

```
 [1] "Rosalind Franklin"  "Marie Curie"        "Barbara McClintock"
 [4] "Ada Lovelace"       "Dorothy Hodgkin"    "Lise Meitner"
 [7] "Grace Hopper"       "Chien-Shiung Wu"    "Gerty Cori"
[10] "Katherine Johnson"
```

```
# for specific cells:
df[2,3]
```

```
[1] "Sorbonne"
```

```
# We can use the column name instead of numbers:
df[2,"school"]
```

```
[1] "Sorbonne"
```

```
# We can do the same thing by using a dollar sign:
df$name
```

```
 [1] "Rosalind Franklin"  "Marie Curie"        "Barbara McClintock"
 [4] "Ada Lovelace"       "Dorothy Hodgkin"    "Lise Meitner"
 [7] "Grace Hopper"       "Chien-Shiung Wu"    "Gerty Cori"
[10] "Katherine Johnson"
```

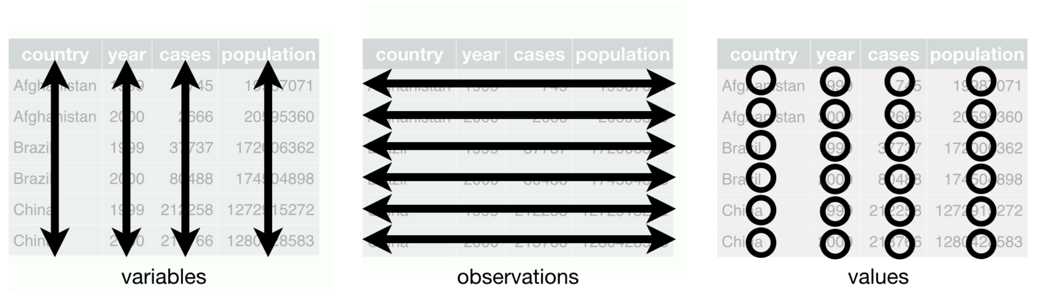
```
# We can also give a list of columns
# which return in the order provided
df[,c("school","name")]
```

```
                      school               name
1                  Cambridge  Rosalind Franklin
2                   Sorbonne        Marie Curie
3                    Cornell Barbara McClintock
4       University of London       Ada Lovelace
5                     Oxford    Dorothy Hodgkin
6       University of Berlin       Lise Meitner
7                       Yale       Grace Hopper
8                  Princeton    Chien-Shiung Wu
9      Washington University         Gerty Cori
10 West Virginia University  Katherine Johnson
```

Standard data formats and Tidy

That data, and most two-dimensional data sets (data frames, matrices, etc.) is often organized the similarly:

- Each variable is its own column
- Each observation is its own row
- Each value is a single cell.



Source: Hadley Wickham’s R for Data Science, 1st Edition

This follows the *tidy data* style, an approach to handling data in R that aims to be clear and readable.

::: callout-note title=“Tidiest Universe” The bundle of tidy-associated packages is called the `tidyverse`, and it’s a hot-topic in the R world. Most data wrangling problems can be solved with `tidy` or base (default) R functions. This can lead to some headaches for beginners! :::

`dplyr` verbs

One of the most popular `tidyverse` packages, `dplyr`, offers a suite of helpful and readable functions for data manipulation. Lets get started with how it can help you see your data:

```
dplyr::glimpse(df)
```

```
Rows: 10
Columns: 5
$ name          <chr> "Rosalind Franklin", "Marie Curie", "Barbara
McClintock…
$ field         <chr> "DNA X-ray crystallography", "Radioactivity",
"Genetics…
$ school        <chr> "Cambridge", "Sorbonne", "Cornell",
"University of Lond…
$ date_of_birth  <chr> "1920-07-25", "1867-11-07", "1902-06-16",
"1815-12-10",…
$ working_region <chr> "Western Europe", "Western Europe", "North
America", "W…
```

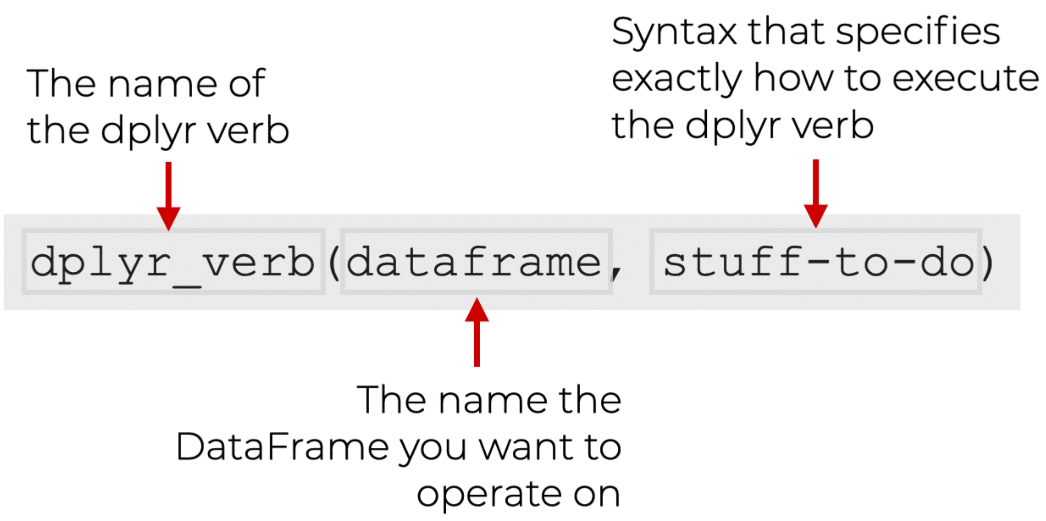
With the `glimpse` function we see that this is a data frame with 3 observations and 3 variables. We can also see the type of each variable and the first few values.

Tip

`dplyr` functions have a lot in common:

- The first argument is always a data frame
- The following arguments typically specify which columns to operate on, using the variable names (without quotes)

- The output is always a new data frame


The name of the dplyr verb
Syntax that specifies exactly how to execute the dplyr verb
dplyr_verb(dataframe, stuff-to-do)
The name the DataFrame you want to operate on

Source: Joshua Ebner's A Quick Introduction to Dplyr

The `dplyr` package has a set of functions called "verbs" that are used to manipulate data frames. These verbs can either act on rows (i.e. `filter` out specific row's by some condition) or columns (i.e. `select` columns XYZ). There are also has functions for working with groups (i.e. group rows by what values they have in a column with `group_by`)

Rows Columns groups

`filter` allows you to...

```
        df |>
          filter(school == "Cambridge")
```

```
               name                         field     school
date_of_birth
1 Rosalind Franklin DNA X-ray crystallography Cambridge    1920-
07-25
  working_region
1 Western Europe
```

`arrange` allows you to...

`distinct` allows you to...

Hopefully now, you feel a little more confident about working with vectors, data frames, and using `dplyr` verbs to clean and manipulate data. Happy Wrangling!

Functions on functions

An introduction to pipes

Data scientists often want to make many changes to their data at one time. Typically, this means using more than one function at once. However, the way we've been writing our scripts so far would make for some very confusing looking code.

For example, let's use `dplyr` functions to perform two operations on our data set of scientists: filter for those born after 1900 and then arrange them by date of birth.

Writing it as separate steps | Combining functions in one line

Using pipes to clean up the code

Here we first filter and then arrange. Note that we are creating an intermediate variable in between the steps.

```
# Filtering for scientists born after 1900
filtered_data <- filter(df, as.Date(date_of_birth) > as.I

# Arranging the filtered data by date of birth
arranged_data <- arrange(filtered_data, date_of_birth)
```

Tip

There are two pipe operators: `|>` and `%>%`. They work almost the exact same way. `%>%` is from the `magrittr` package and was the only way to pipe before version R 4.1.0. The only major difference is that `%>%` can specify which argument of the next function you want to pipe into.

`data |> function(argument_A, argument_B)` can only do `function(argument_A = data, argument_B)`

With `%>%` you can choose the argument with a period, `.`: `data %>% function(argument_A, argument_B = .)`

Case Study Introduction

A lot of people struggle to learn how to code without having an application. Since this course is geared towards biomedical sciences, we thought you might find it easier if we work through an actual research data set.

For this example, we have some data from an experiment that measured the proportions of different cell times within mouse cardiac tissue. These samples are from treatment vs. control and WT vs. mutant.

What are some things that we, as researchers, would want to know about our data?

- Did the experiment work?

> **Things to consider:**
>
> Check your controls or expected features!

- Do we see differences between our experimental groups?

To get at these questions, we need to be able to manipulate our data into the formats needed to check those features and for plotting. To start, lets take a look at how the results are structured before we start planning how to do the processing.

Getting familiar with the data

```
options(digits = 3)
# Load the data. The sample IDs were stored as the first row,
cell_props <- read.csv("wrangling-files/cellProportions.csv",
                       row.names = 1)

head(cell_props)
```

```
            Cardiomyocytes Fibroblast Endothelial.Cells Macrophage
whole_2              0.652     0.0886           0.06700     0.1761
fraction_13          0.824     0.0370           0.06387     0.0692
fraction_12          0.895     0.0213           0.04436     0.0390
fraction_19          0.000     0.9983           0.00167     0.0000
fraction_18          0.000     1.0000           0.00000     0.0000
whole_16             0.820     0.0208           0.08889     0.0501
            Pericytes.SMC
whole_2           0.01672
fraction_13       0.00558
fraction_12       0.00000
fraction_19       0.00000
fraction_18       0.00000
whole_16          0.02058
```

> **Whole vs. Fractions**
>
> `Fraction` samples are our controls. They are supposed to be almost completely one single cell type. They're just here to make sure we accurately measured cell type proportions.
>
> `Whole` samples are our test samples. They're from the treatment/control mice, which you'd expect to have a range of cell types.

Analysis Goals

For next class, you should brainstorm some ideas for how to approach the analysis. Try to consider these angles:

- What do we want to know about this data set?
- What kind of visuals would we want to make to check that?
- What would the data need to look like to get those visuals?
- How does the data look now?
- Which functions might we use to get the data from it's current state to what we need for plotting?

In the beginning of next class, we'll chat about what ideas you had!

(Ambitious students who want to try before then will also need the phenotype data located at `wrangling-files/cellPhenotypes.csv`)