Data wrangling with dplyr and tidyr

R for Data Science workshop

2019-05-01 (updated: 2019-07-14)

Data wrangling with dplyr and tidyr

Outline

- Coding basics
    - Assignment
    - Naming
    - Calling functions
- Overview of dplyr
- More coding topics
    - Missing values (NA)
    - Pipe operator (%>%)
- Demo of tidyr

Data science workflow

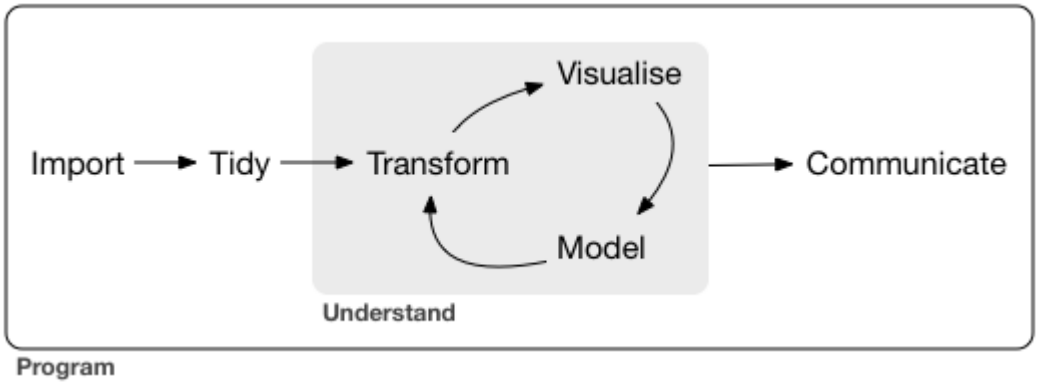

Image source: R for Data Science by Hadley Wickham & Garrett Grolemund.

Data science workflow

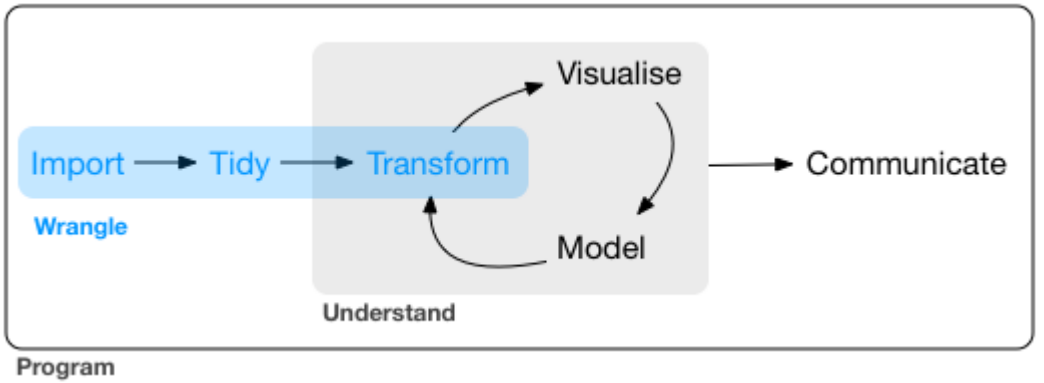

Image source: R for Data Science by Hadley Wickham & Garrett Grolemund.

Coding basics

Before we get to `dplyr`, let's talk about some basic R coding topics:

Assignment

```
x <- 3 * 4
x
```

```
## [1] 12
```

Naming

Choose names of variables and functions carefully.

```
modern_idiomatic_r_uses_snake_case <- TRUE
```

Coding basics, continued

Calling functions

R functions are called like this:

```
function_name(val1, arg2 = val2, ...)
```

- Positional arguments
- Named arguments
- Arguments may or may not have default values
- R functions can optionally take an arbitrary number of values (...)
- Arguments are lazily evaluated
- R functions always have a return value, possibly `NULL`

dplyr package

Like most Tidyverse packages, dplyr functions operate on **data frames**.

The Tidyverse uses enhanced data frames known as **tibbles**.

dplyr package

From the official dplyr website:

> 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.
>
> These all combine naturally with **group_by()** which allows you to perform any operation “by group”.

dplyr verbs

- mutate
- select
- filter
- summarize
- arrange
- group_by

Each function expects a **data frame** as its first argument.

Subsequent arguments describe **what to do**.

Result is always a **new data frame**. (Technically **tibble**).

Demo

More coding topics

Missing values NA

R has a special value `NA` that is used to represent missing values.

```
NA > 5
```

```
## [1] NA
```

```
10 == NA
```

```
## [1] NA
```

```
NA + 10
```

```
## [1] NA
```

```
NA == NA
```

```
## [1] NA
```

More coding topics, continued

Missing values, continued

Use `is.na()` to test for NA.

```
df <- tibble(x = c(1, NA, 3))
filter(df, x > 1)
```

```
## # A tibble: 1 x 1
##       x
##   <dbl>
## 1     3
```

```
filter(df, is.na(x) | x > 1)
```

```
## # A tibble: 2 x 1
##       x
##   <dbl>
## 1    NA
## 2     3
```

More coding topics, continued

Pipe operator %>%

- `x %>% f` is equivalent to `f(x)`
- `x %>% f(y)` is equivalent to `(fx, y)`
- `x %>% f %>% g %>% h` is equivalent to `h(g(f(x)))`

```
foo_foo <- little_bunny()

foo_foo_1 <- hop(foo_foo, through = forest)
foo_foo_2 <- scoop(foo_foo_1, up = field_mice)
foo_foo_3 <- bop(foo_foo_2, on = head)
```

compare

```
foo_foo %>%
  hop(through = forest) %>%
  scoop(up = field_mouse) %>%
  bop(on = head)
```

Demos

tidyr package

Definition of **tidy data**:

> 1. Each variable is in a column.
> 2. Each obervation is a row.
> 3. Each value is a cell.

Tidyverse packages generally require **tidy data frames**.

The `tidyr` packages provides some functions for transforming untidy data into tidy data.

The main functions for doing this are **gather()** and **spread()**.

These are in the process of being renamed to pivot_longer() and pivot_wider().

Demo

Your turn

Data wrangling with dplyr

your-turn/02-data-wrangling-with-dplyr.Rmd

15:00