Data Wrangling with R: Day 1

Data manipulation with dplyr

Presented by Emi Tanaka
Department of Econometrics and Business Statistics

MONASH University

1st December 2020 @ Statistical Society of Australia | Zoom
library(dplyr) # or library(tidyverse)

- dplyr is a core package in tidyverse
- The earlier concept of dplyr (first on CRAN in 2014-01-29) was implemented in plyr (first on CRAN in 2008-10-08)
- The functions in dplyr has been evolving frequently but dplyr v1.0.0 was released on CRAN in 2020-05-29
- This new version contained new "verbs"
- The major release suggests that functions in dplyr are maturing and thus the user interface is unlikely to change
• Functions (and sometimes arguments of functions) in tidyverse packages often are labelled with a badge like above
• Find definitions of badges here
• Check out documentations below

help(mutate, package = "dplyr")
help(mutate_each, package = "dplyr")
The main functions of dplyr include:

- `arrange`
- `select`
- `mutate`
- `rename`
- `group_by`
- `summarise`

Notice that these functions are *verbs*.

Functions in dplyr generally have the form:

```
verb(data, args)
```

I.e., the first argument `data` is a `data.frame` object.

What do you think the following will do?

```
rename(mtcars, miles_per_gallon = mpg)
arrange(mtcars, wt)
```
Pipe operator %>%

- Almost all tidyverse packages import the `magrittr` package to use `%>%`
- \( x \ %>% f(y) \) is the same as \( f(x, y) \)
- \( x \ %>% f(y) \ %>% g(z) \) is the same as \( g(f(x, y), z) \)
- When you see the pipe operator `%>%`, read it as "and then"

```r
mtcars %>%
  rename(miles_per_gallon = mpg) %>%
  arrange(wt)
```

```
##                     miles_per_gallon cyl  disp  hp drat    wt  qsec vs am gea
## Lotus Europa                    30.4   4  95.1 113 3.77 1.513 16.90  1  1
## Honda Civic                     30.4   4  75.7  52 4.93 1.615 18.52  1  1
## Toyota Corolla                  33.9   4  71.1  65 4.22 1.835 19.90  1  1
## Fiat X1-9                       27.3   4 120.3  91 4.08 1.935 18.90  0  1
## Porsche 914-2                   26.0   4 120.3  91 4.43 2.140 16.70  0  1
```

Lazy and non-standard evaluation

- Remember in Base R:

```r
subset(mtcars, mpg > 31)

##                 mpg cyl disp hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1
```

- But the second argument cannot be evaluated:

```r
mpg > 31

## Error in eval(expr, envir, enclos): object 'mpg' not found
```

- R employs what is called **lazy evaluation** for function inputs.

- **Non-standard evaluation** uses this feature to capture the input expression within the function and evaluate only when requested.
• Tidy evaluation builds on the lazy and non-standard evaluation and is implemented in `rlang`

• All core tidyverse packages import `rlang`

• So what does it do?

• Let's consider `filter`, the Tidyverse version of subset

```r
filter(mtcars, mpg > 31)
```

```r
##                mpg cyl disp   hp drat    wt qsec vs am gear carb
## Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1
```

• ⚠ If you get an error using `filter`, replace it with `dplyr::filter`

for those interested, `dplyr::filter` is a conflict with `stats::filter` and it may be using `stats::filter` instead... I've fallen into this trap so many times!
Suppose we have a silly function that subsets `mtcars` for a given condition:

```r
myCarSubset <- function(cond) subset(mtcars, cond)
myCarFilter <- function(cond) filter(mtcars, cond)
```

This causes an issue because `cond` is evaluated before it is parsed into `subset` or `filter`:

```r
myCarSubset(mpg > 31)
## Error in eval(e, x, parent.frame()): object 'mpg' not found

myCarFilter(mpg > 31)
## Error: Problem with `filter()` input `..1`.  
## x object 'mpg' not found  
## i Input `..1` is `cond`.  
```
Functions that use non-standard evaluation is problematic

myCarSubsetNew <- function(cond) subset(mtcars, {{ cond }})
myCarFilterNew <- function(cond) filter(mtcars, {{ cond }})

myCarSubsetNew(mpg > 31)

## Error in eval(e, x, parent.frame()): object 'mpg' not found

myCarFilterNew(mpg > 31)

##                 mpg cyl disp hp drat    wt  qsec vs am gear carb
## Fiat 128       32.4   4 78.7 66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90  1  1    4    1

{{{ }}} only works if the underlying function implements rlang
Data masking

```r
ind <- 1:nrow(cars) # nrow(cars) = 50
subset(cars, ind > 49)
##    speed dist
## 50    25   85

filter(cars, ind > 49)
##    speed dist
## 1    25   85
```

```r
speed <- c(40, 51)
subset(cars, speed > 24)
##    speed dist
## 50    25   85

filter(cars, speed > 24)
##    speed dist
## 1    25   85
```

- For any variables that don't exist in the data, R searches the parental environment for evaluation.

- The variables in data take priority for reference over those in parental environment.
Data masking

speed <- 1:nrow(cars)
filter(cars, .data$speed > 24)

---

• In Tidyverse, you can be explicit whether the variable is in the data or in the parental environment

• `.data` is a special pronoun referring to variables in the data parsed in the first argument

• `.env` is a special pronoun referring to variables in the environment (i.e. *not* in the data parsed in the first argument)
Tidyverse packages generally use syntax from the `tidyselect` package for variable/column selection.

```r
data(frog_signal, package = "dwexercise")
str(frog_signal)
```

```
## 'data.frame': 535 obs. of 22 variables:
## $ FrogID          : chr "13196" "13197" "13198" "13206" ...
## $ AlternativeCD  : num 28 27 31 33 26 31 33 28 29 34 ...
## $ AlternativeCR  : num 12 15 13 15 11 6 15 12 14 12 ...
## $ AlternativeDF  : num 2315 2304 2646 2281 2789 ...
## $ AlternativeRA  : num -10 -8 -12 -7 -8 -6 -12 -16 -12 -10 ...
## $ AlternativePR  : num 46 49 43 50 57 51 46 55 56 45 ...
## $ Standard1      : num 47 69 139 112 101 90 79 262 123 47 ...
## $ Standard2      : num 46 44 102 112 101 68 41 237 106 62 ...
## $ Standard3      : num 42 36 85 117 80 73 46 166 95 63 ...
```
The `tidyselect` syntax can be used to select contiguous columns in the data.
```r
frog_signal %>%
  select(starts_with("Standard"))
```

<table>
<thead>
<tr>
<th></th>
<th>Standard1</th>
<th>Standard2</th>
<th>Standard3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47</td>
<td>46</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>69</td>
<td>44</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>139</td>
<td>102</td>
<td>85</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>112</td>
<td>117</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>90</td>
<td>68</td>
<td>73</td>
</tr>
<tr>
<td>7</td>
<td>79</td>
<td>41</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>262</td>
<td>237</td>
<td>166</td>
</tr>
<tr>
<td>9</td>
<td>123</td>
<td>106</td>
<td>95</td>
</tr>
<tr>
<td>10</td>
<td>47</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>11</td>
<td>95</td>
<td>117</td>
<td>84</td>
</tr>
<tr>
<td>12</td>
<td>56</td>
<td>52</td>
<td>41</td>
</tr>
</tbody>
</table>

```r
frog_signal %>%
  select(num_range("Standard", 1:3))
```

<table>
<thead>
<tr>
<th></th>
<th>Standard1</th>
<th>Standard2</th>
<th>Standard3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>47</td>
<td>46</td>
<td>42</td>
</tr>
<tr>
<td>2</td>
<td>69</td>
<td>44</td>
<td>36</td>
</tr>
<tr>
<td>3</td>
<td>139</td>
<td>102</td>
<td>85</td>
</tr>
<tr>
<td>4</td>
<td>112</td>
<td>112</td>
<td>117</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>101</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>90</td>
<td>68</td>
<td>73</td>
</tr>
<tr>
<td>7</td>
<td>79</td>
<td>41</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>262</td>
<td>237</td>
<td>166</td>
</tr>
<tr>
<td>9</td>
<td>123</td>
<td>106</td>
<td>95</td>
</tr>
<tr>
<td>10</td>
<td>47</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>11</td>
<td>95</td>
<td>117</td>
<td>84</td>
</tr>
<tr>
<td>12</td>
<td>56</td>
<td>52</td>
<td>41</td>
</tr>
</tbody>
</table>
• : for selecting contiguous variables
• ! for taking complement set of variables
• & or | for selecting intersection or union of two sets of variables, e.g.

```r
frog_signal %>%
  select(starts_with("Alt") & ends_with("1")) %>%
  str()
```

```text
## 'data.frame': 535 obs. of 1 variable:
## $ Alternative1: num 28 33 227 101 126 143 50 123 76 53 ...
```

• c() for combining selections
• everything() to select all variables
• last_col() to select last variable, with option of an offset
Selection language Part 2

- `starts_with()` selects columns with the given prefix
- `ends_with()` selects columns with the given suffix
- `contains()` selects columns with a literal string
- `matches()` selects columns that match the regular expression
  we'll learn this next!
- `num_range()` selects columns with a numerical range
- `all_of()` for selecting columns based on a character vector
- `any_of()` is the same as `all_of()` but no error when variables do not exist
- `where()` selects based on where given function return TRUE

```r
help(language, package = "tidyselect")
```
Subsetting by column

Subsetting by column

Tidyverse

select(mtcars, c(mpg, cyl))
select(mtcars, c("mpg", "cyl"))
select(mtcars, mpg, cyl)
select(mtcars, "mpg", "cyl")

All the same result as below

mtcars %>% select(mpg, cyl)

##                      mpg cyl
## Mazda RX4           21.0   6
## Mazda RX4 Wag       21.0   6
## Datsun 710          22.8   4
## Hornet 4 Drive      21.4   6
## Hornet Sportabout   18.7   8
## Valiant             18.1   6
## Duster 360          14.3
## Merc 240D           24.4
## Merc 230            22.8
## Merc 280            19.2
## Merc 280C           17.8
## Merc 450SE          16.4
## Merc 450SL          17.3

mtcars %>% select(mpg)

##                      mpg
## Mazda RX4           21.0
## Mazda RX4 Wag       21.0
## Datsun 710          22.8
## Hornet 4 Drive      21.4
## Hornet Sportabout   18.7
## Valiant             18.1
## Duster 360          14.3
## Merc 240D           24.4
## Merc 230            22.8
## Merc 280            19.2
## Merc 280C           17.8
## Merc 450SE          16.4
## Merc 450SL          17.3

Selecting one column doesn't "drop" it to a vector.

If you really want the vector then use pull(mpg).
Subsetting by row

### mtcars %>% slice(3:1)

```
##                mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Datsun 710    22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Mazda RX4 Wag 21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Mazda RX4     21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
```

### mtcars %>% filter(rownames(.) %in% c("Datsun 710", "Mazda RX4"))

```
##             mpg cyl disp  hp drat   wt  qsec vs am gear carb
## Mazda RX4  21.0   6  160 110 3.90 2.62 16.46  0  1    4    4
## Datsun 710 22.8   4  108  93 3.85 2.32 18.61  1  1    4    1
```

- **Placeholder binding**
  - `x %>% f(y, g(.))` is the same as `f(x, y, g(x))`
  - `x %>% f(y, .)` is the same as `f(y, x)`

- **Note:** row names do not follow tidy data principles
  - Use `tibble::rownames_to_column()` to convert rownames to a column to make into a tidy data

- Use `slice` to subset by index and `filter` by logical vector
Adding or modifying a column

```r
mtcars %>%
mutate(gpm = 1 / mpg,
      wt = gpm^2,
      wt = if_else(cyl==6, 10, wt),
      hp = case_when(cyl==6 ~ 11,
                     cyl==4 ~ 10,
                     TRUE ~ 3))
```

##     mpg cyl  disp hp drat           wt  qsec vs am gear carb        gpm
## 1  21.0   6 160.0 11 3.90 1.000000e+01 16.46  0  1    4    4 0.04761905
## 2  21.0   6 160.0 11 3.90 1.000000e+01 17.02  0  1    4    4 0.04761905
## 3  22.8   4 108.0 10 3.85 1.923669e-03 18.61  1  1    4    1 0.04385965
## 4  21.4   6 258.0 11 3.08 1.000000e+01 19.44  1  0    3    1 0.04672897
## 5  18.7   8 360.0  3 3.15 2.859676e-03 17.02  0  0    3    2 0.05347594
## 6  18.1   6 225.0 11 2.76 1.000000e+01 20.22  1  0    3    1 0.05524862
## 7  14.3   8 360.0  3 3.21 4.890215e-03 15.84  0  0    3    4 0.06993007

Evaluation in `mutate` is done sequentially based on input order.
So you refer to the newly created variable in later input.
You can call multiple `mutate` but computational performance is usually better if done within the same `mutate` call.

```r
mtcars %>%
mutate(gpm = 1 / mpg) %>%
mutate(wt = gpm^2)
```
Sorting columns

```r
mtcars %>%
  select(sort(names(.)))
```

```
##                     am carb cyl  disp
## Mazda RX4            1    4   6 160.0
## Mazda RX4 Wag        1    4   6 160.0
## Datsun 710           1    1   4 108.0
## Hornet 4 Drive       0    1   6 258.0
```

```r
mtcars %>%
  relocate(am, carb, .before = cyl)
```

```
##                      mpg am carb cyl
## Mazda RX4           21.0  1    4   6 1
## Mazda RX4 Wag       21.0  1    4   6 1
## Datsun 710          22.8  1    1   4 1
## Hornet 4 Drive      21.4  0    1   6 2
```

```r
mtcars %>%
  select(wt, gear, everything())
```

```
##                        wt gear  mpg
## Mazda RX4           2.620    4 21.0
## Mazda RX4 Wag       2.875    4 21.0
## Datsun 710          2.320    4 22.8
## Hornet 4 Drive      3.215    3 21.4
```

```r
mtcars %>%
  relocate(wt, gear, .after = mpg)
```

```
##                      mpg    wt gear
## Mazda RX4           21.0 2.620    4
## Mazda RX4 Wag       21.0 2.875    4
## Datsun 710          22.8 2.320    4
## Hornet 4 Drive      21.4 3.215    3
```
Calculate the *average* weight \((wt)\) of a car for each gear type in \((gear)\)

```r
tidyverse
mtcars
mtcars %>%
  group_by(gear) %>%
  summarise(avg_wt = mean(wt))
```

```
# A tibble: 3 x 2
## Groups:   gear [3]
  gear avg_wt
  <dbl>  <dbl>
1   1   3.89
2   2   2.62
3   3   2.63
```

Calculate the *median* weight \((wt)\) of \((vs)\) type in \(mtcars\)

```r
tidyverse
mtcars %>%
  group_by(gear, vs) %>%
  summarise(avg_wt = mean(wt),
            med_wt = median(wt))
```

```
# A tibble: 6 x 4
## Groups:   gear [3]
  gear    vs avg_wt med_wt
  <dbl> <dbl>  <dbl>  <dbl>
1   3     0   4.10   3.81
2   3     1   3.05   3.22
3   4     0   2.75   2.75
```

```r
## `summarise()` regrouping output by

```
Using `across`, you can more easily apply a function to multiple columns

```r
mtcars %>%
group_by(gear, vs) %>%
  summarise(across(everything(), mean))
```
You can combine across with the selection helper `where`

```r
mtcars %>%
  group_by(gear, vs) %>%
  summarise(across(where(function(x) n_distinct(x) > 10), mean))
```

```r
## # A tibble: 6 x 8
## # Groups:   gear 
##    gear    vs   mpg  disp    hp  drat    wt  qsec
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1     3     0  15.0 358.  194.   3.12  4.10  17.1
## 2     3     1  20.3 201.  104    3.18  3.05  19.9
## 3     4     0  21   160   110    3.9   2.75  16.7
## 4     4     1  25.2 116.   85.4  4.07  2.59  19.4
## 5     5     0  19.1 229.  216.   3.95  2.91  15.3
## 6     5     1  30.4  95.1 113    3.77  1.51  16.9
```
Remember tidy selection only works with functions that are compatible

```r
mtcars %>%
  rowwise() %>%
  summarise(disp = disp, hp = hp, drat = drat, wt = wt,
             score = sum(c_across(disp:wt)))
```

```
# A tibble: 32 x 5
#  disp hp drat wt score
#1  160 110 3.9  2.62 277.
#2  160 110 3.9  2.88 277.
#3  108  93 3.85 2.32  207.
#4  258 110 3.08 3.22  374.
#5  360 175 3.15 3.44  542.
#6  225 105 2.76 3.46  336.
#7  360 245 3.21 3.57  612.
```
What happens if you omit `rowwise`?

```r
mtcars %>%
  #rowwise() %>%
  summarise(disp = disp, hp = hp, drat = drat, wt = wt,
             score = sum(c_across(disp:wt)))
```

<table>
<thead>
<tr>
<th></th>
<th>disp</th>
<th>hp</th>
<th>drat</th>
<th>wt</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>160.0</td>
<td>110</td>
<td>3.90</td>
<td>2.620</td>
<td>12295.14</td>
</tr>
<tr>
<td>#2</td>
<td>160.0</td>
<td>110</td>
<td>3.90</td>
<td>2.875</td>
<td>12295.14</td>
</tr>
<tr>
<td>#3</td>
<td>108.0</td>
<td>93</td>
<td>3.85</td>
<td>2.320</td>
<td>12295.14</td>
</tr>
<tr>
<td>#4</td>
<td>258.0</td>
<td>110</td>
<td>3.08</td>
<td>3.215</td>
<td>12295.14</td>
</tr>
<tr>
<td>#5</td>
<td>360.0</td>
<td>175</td>
<td>3.15</td>
<td>3.440</td>
<td>12295.14</td>
</tr>
<tr>
<td>#6</td>
<td>225.0</td>
<td>105</td>
<td>2.76</td>
<td>3.460</td>
<td>12295.14</td>
</tr>
<tr>
<td>#7</td>
<td>360.0</td>
<td>245</td>
<td>3.21</td>
<td>3.570</td>
<td>12295.14</td>
</tr>
<tr>
<td>#8</td>
<td>146.7</td>
<td>62</td>
<td>3.69</td>
<td>3.190</td>
<td>12295.14</td>
</tr>
</tbody>
</table>
If you installed the `dwexercise` package, run below in your R console:

```r
learnr::run_tutorial("day1-exercise-02", package = "dwexercise")
```

If the above doesn't work for you, go [here](#).  
Questions or issues, let us know!
devtools::session_info()

## ─ Session info ───────────────────────────────────────────────────────────────
##  setting   value
##  version   R version 4.0.1 (2020-06-06)
##  os         macOS Catalina 10.15.7
##  system     x86_64, darwin17.0
##  ui         X11
##  language   (EN)
##  collate    en_AU.UTF-8
##  ctype      en_AU.UTF-8
##  tz          Australia/Melbourne
##  date       2020-11-30

## ─ Packages ───────────────────────────────────────────────────────────────────
##  package   * version    date              lib source
##  anicon     0.1.0        2020-06-21 [1] Github (emitanaka/anicon@0b756df)
##  assertthat 0.2.1       2019-03-21 [2] CRAN (R 4.0.0)

These slides are licensed under [CC BY-SA 4.0](https://creativecommons.org/licenses/by-sa/4.0/).