Introduction to data wrangling

Emma Vestesson

05-11-2019

What we will cover today

Workshop is aimed at advanced beginners and I will assume some familiarity with R and the tidyverse.

We will spend a few minutes refamiliarising ourselves with R.

You will need to have `tidyverse`, `tidylog` and `here` packages installed and loaded. You also need to load the `lubridate` package.

Open the script called code.

- Dplyr main verbs and their friends
- Reshaping data
- Joining two data sets

Packages in R

The first time you use it you need to install the package.

```r
install.packages("tidyverse")
```

Load the package

```r
library(tidyverse)
```

The pipe

Simplifying R code with pipes (%>%)

- Easy way to pass data through functions without nesting
- First argument of each function is "piped" in to reduce redundancy
- f(x) is the same as x %>% f()
- f(x, y) is the same as x %>% f(y)
- Keyboard shortcut ctrl+shift +m

The pipe

```r
leave_house(get_dressed(get_out_of_bed(wake_up(me))))
```

VS

```r
me %>%
  wake_up() %>%
  get_out_of_bed() %>%
  get_dressed() %>%
  leave_house()
```

General R tips

Getting help

- `?functionname` will search all loaded packages
- `??functionname` will search all installed packages
- Most packages have vignettes
- Most tidyverse packages have cheatsheet (under help in the menu)

Debugging

- start a new R session (keyboard short ctrl+shift+F10)
- the :: operator is your friend eg dplyr::select()

Dlyr verbs

Grammar of data manipulation:

Main verbs

- select() picks variables (columns) based on their names
- filter() allows row selection based on given criteria
- mutate() creates new variables (columns) from existing ones
- summarise() reduces multiple values down to a single summary

helper verbs

- group_by() performs any of the above on a group-by-group basis
- arrange() changes the ordering of rows

dplyr syntax

- All calls to dplyr verbs follow the same format:

1. The first argument is a dataframe
2. The subsequent arguments describe what to do to that dataframe, using unquoted variable names.

Helper functions you can use with select():

- starts_with("e_") matches names that begin with "e_"
- ends_with("_end") matches names that ends with "_end"
- contains("_h12") matches names that contain "_h12"
- matches() allows you to do regex matching on names eg matches("abc|abd")

Example select()

```r
new_dat <- starwars %>%
  select( -height, -mass)

starwars %>%
  select(name, -ends_with("color"))
```

Which bit of code produces the output?

starwars %>% select(birth_year) (pink) **or** starwars %>% select(-birth_year) (green)

```
# A tibble: 87 x 1
   birth_year
        <dbl>
 1         19
 2        112
 3         33
 4         41.9
 5         19
 6         52
 7         47
 8         NA
 9         24
10         57
# ... with 77 more rows
```

filter()

- Allows pointed row selection based on given criteria
- First argument is the dataframe, subsequent arguments are logical expressions used to filter the dataframe

Useful functions when working with filter

- is.na() and ! to negate
- str_detect() to search for a string
- Logical comparisons ($<$, $<=$, $>$, $>=$, $!=$)

Example filter

```r
starwars %>%
  filter(mass>80 | hair_color=="white")

starwars %>%
  filter(is.na(hair_color))
```

Will this code run without an error?

Yes (pink) **or** No (green)

```
starwars %>%
  filter(height+5)
```

mutate() and transmute()

- Mutate creates new variables (columns) from existing ones and keeps existing ones
- can be used to overwrite and old variable eg starwars %>% mutate(mass=mass+5)
- Note: columns created with mutate() are always added to end of dataset
- transmute() creates new variables and drops existing ones

mutate() useful functions

- Arithmetic operators (+, -, *, /, ^)
- Log functions (like log10())
- Offsets like lead() and lag()
- ifelse statements (if this, then this, else this)
- Or when more than 1 logical split then use case_when
- Cumulative and rolling aggregates
- Ranking (like ntile())
- toupper, str_replace, str_to_title for character variables

Example mutate

```
starwars %>%
  mutate(height_m=height/100, bmi=mass/(height_m^2), bmi=ro
```

How many variables will the resulting dataset have

1 variable (pink) **or** 88 variables (green)

```
starwars %>%
  transmute(height_m=height/100)
```

Example summarise

```
starwars %>%
  summarise(height_mean = mean(height, na.rm=TRUE))

starwars %>%
  group_by(homeworld) %>%
  summarise(height_max = max(height, na.rm=TRUE))
```

How many rows will the resulting data set have?

1 row (pink) **or** 87 rows (green)

```r
starwars %>%
  summarise(mass_mean = mean(mass, na.rm=TRUE))
```

Time for you to try!

- Find the rows where height $>$ 90 and hair_color is not brown
- Find the rows where eye_color is brown
- Select name and mass
- Find the rows where hair_color is **NOT** missing
- Find the rows where hair_color contains the word white
- Select columns with the word `color` in them
- Select all columns BUT height and mass
- Create a new variable called `half_mass` that is half of mass
- Create variable `height_cat` with 4 categories "short", "medium", "tall" if `height` [0,70],]70,90],]90,inf[and 'Not recorded' if `height` is missing.
- Create variable `small` that is 1 if mass is less than the mean of `mass` and 0 otherwise.

Solution

```r
starwars %>%
  filter(height>90, hair_color!="brown")

starwars %>%
  filter(eye_color=="brown")

starwars %>%
  select(name,mass)

starwars %>%
  filter(!is.na(hair_color))

starwars %>%
  filter(str_detect(hair_color, "white"))

starwars %>%
  select(contains("color"))
```

Scoped verbs

- Terminology: we have been using “single table verbs”
- Now we can affect multiple variables simultaneously with the scoped verbs
- Three extensions
  - _if pick variables based on a predicate function like is.numeric() or a user defined function function(x) do_this(x)
  - _at pick variables using the same syntax as select().
  - _all operates on all variables

mutate friends

```r
starwars %>%
  mutate_if(is.numeric, ~round(.) )

starwars %>%
  mutate_at(vars(contains('color')), toupper)
```

select friends

```r
colour_vars <- c("hair_color", "skin_color", "eye_color")

starwars %>%
  select_at(colour_vars)

starwars %>%
  select_at(vars(-colour_vars))

starwars %>%
  select_if(~n_distinct(.) < 10)

numeric_vars <- starwars %>%
  select_if(is.numeric) %>%
  names()
```

summarise friends

```
starwars %>%
  summarise_at(numeric_vars, ~mean(.x, na.rm=TRUE))

starwars %>%
  group_by(hair_color) %>%
  summarise_if(is.numeric,
  list(min = ~min(., na.rm = FALSE), max = ~max(., na.rm =
```

Time for you to try again!

- What variables are not characters?
- For all numeric variables, create new variables with the mean by homeworld. The new variables should have the suffix `mean` eg `height_mean`. (Bonus, move the new variables so they are after `name`. Try ?everything)
- For all character variables, replace missing values by ‘not recorded’

Solution

“‘{r, echo=FALSE} starwars %>% select_if(∼!is.character(.)) %>% names()

starwars %>% group_by(homeworld) %>% mutate_if(is.numeric, list(mean=∼mean(.x, na.rm=TRUE))) %>% select(name, ends_with(‘mean’), everything())

starwars %>% group_by(homeworld) %>% summarise_if(is.numeric, mean, na.rm=TRUE)

starwars %>% mutate_if(is.character, list(∼replace_na(., “not recorded”)))

“‘

Tidy data

Data comes in all kinds of shapes and forms.

1. Each variable forms a column.
2. Each observation forms a row.
3. Each type of observational unit forms a table.

Sitrep data

Downloaded data from NHSE website and saved in R data format.

```
sitrep <- readRDS(here::here('data', 'sitrep.rds')) # all d
sitrep_60sec <- readRDS(here::here('data', 'sitrep_60sec.rd
```

Look at the data. Is is tidy?

New tidyr package

The `tidyr` package was updated early September. `Spread` and `gather` have been replaced by `pivot_longer` and `pivot_wider`.

```
==================================
```

```
sitrep_long <- sitrep %>%
  pivot_longer(-c(NHS_111_area_name, year), names_to='day_n
```

Our data is long! But we can make it even tidier. Suggestions?

Sorting out the date

```
sitrep_long <- sitrep_long %>%
  mutate(day_month=str_replace(day_month, '_', '-'), date=p
```

Time for you to try

Reshape the `sitrep_60sec` data frame and create a date variable. Call your new data frame `sitrep_60sec_long`.

Solution

```
sitrep_60sec_long <- sitrep_60sec %>%
  pivot_longer(-c(NHS_111_area_name, year), names_to='day_r
    mutate(day_month=str_replace(day_month, '_', '-'), date
```

Join the two data set

Joining data in R is very similar to sql.

Mutating joins

A mutating join allows you to combine variables from two tables. It first matches observations by their keys, then copies across variables from one table to the other.

- inner_join(x,y) only keeps observations where in both
- full_join(x,y) keeps all observations even if not in both
- left_join(x,y) keeps all observations in x
- right_join(x,y) keeps all observations in y

Join our two data sets

```
sitrep_full <- full_join(sitrep_long, sitrep_60sec_long, by
```

Play with the data set

- Drop all the extra variables
- Create a new variable called `calls_60_p` with % of calls answered within 60 sec.
- Which area had the most calls in total looking at the full time period?
- Calculate the % of calls answered within 60 sec over the full time period by area and sort by your new variable.

Solution

```
sitrep_full <- sitrep_full %>%
  select(-contains('year'), -contains('day_month'))

sitrep_full %>%
  mutate(calls_60_p=calls_60sec/calls_all*100)

sitrep_full %>%
  group_by(NHS_111_area_name) %>%
  summarise(calls_60_p=sum(calls_60sec, na.rm=TRUE)/sum(ca
  arrange(calls_60_p)
```

Filtering joins

Filtering joins match observations in the same way as mutating joins, but affect the observations, not the variables.

- anti_join(x,y) drops all observations in x that have a match in y
- semi_join(x,y) keeps all observations in x that have a match in y

Messy data

There is another file with some more data but something has gone wrong.

Which regions are missing?

```
sitrep_clinical_input_corrupt_long <- readRDS(here::here('s
```

Solution

```
anti_join(sitrep_clinical_input_corrupt_long, sitrep_full,
```