Data wrangling in the tidyverse

Using dplyr and tidy data

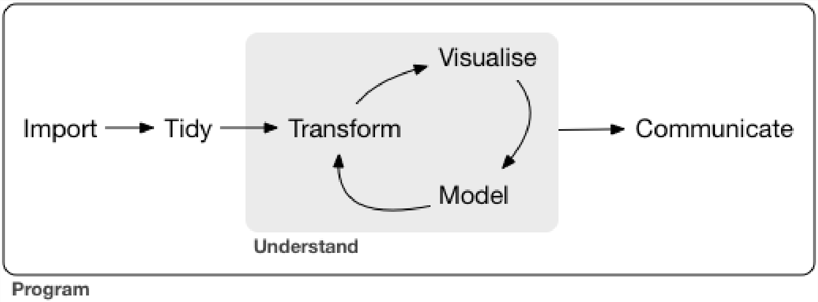
Thomas de Graaff

April 21, 2021

Department of Spatial Economics

Why bother?

- Because working with data
    1. For 20% of your time fancy techniques
    2. For the other 80%
    - working with real and messy data
    - that has to be read in, cleaned, restructured, changed, described, communicated, visualised, etc.
    - in a structural & consistent way



So why R and `tidyverse`?

R was written by statisticians for statisticians

- multiple ways in base R
- `dplyr` and `ggplot2`—later bundled in `tidyverse`
    - a more structured, encompassing, readable approach to data wrangling
- many, many, many offspring (`sf`, `dtplyr`, `ggraph`)

Invoking the tidyverse

```r
library("tidyverse")
```

```
## -- Attaching packages ----------------------------

## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.0.5     v dplyr   1.0.3
## v tidyr   1.1.2     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1

## -- Conflicts ------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

Read in data

```
library("nycflights13")
fdata <- flights
```

display the first six rows of the dataframe `fdata`

```
summary(fdata)
```

or view the dataset

```
glimpse(fdata)
```

or look at the structure of the dataset

```
str(fdata)
```

Dataframes and tibbles

- `fdata` is now a tibble (almost the same as a dataframe)
- dataframes and tibbles
    - observations in rows
    - variables in columns (can be of different type)
    - a list of equal length vectors

Example

```
grades <- tibble(
    name = c("Erik", "Eric", "Thomas"),
    quiz_1 = c(10, 9, 6),
    exam = c(7, 6, NA)
)
grades
```

```
## # A tibble: 3 x 3
##   name   quiz_1  exam
##   <chr>   <dbl> <dbl>
## 1 Erik       10     7
## 2 Eric        9     6
## 3 Thomas      6    NA
```

dplyr

Why a need for `dplyr`

- Most data manipulation is done in Excel
  - sorting
  - creating/transforming variables
  - renaming variables
  - selecting variables/filtering observations
- But this, and more, can be done in R as well:
  - `dplyr` package
  - big advantage: work can be reproduced!
  - faster with bigger datasets
  - computations with groupings of data
  - Cheatsheets to be found here

`dplyr` basic verbs

- `dplyr` revolves around 5 verbs
    - `arrange()`
    - `filter()`
    - `select()`
    - `mutate()`
    - `summarize()`
- Other important commands (there are more):
    - `group_by()`
    - `rename()`
    - `distinct()`
    - count (`n()`)
- and can as well merge & restructure datasets

You might want to sort variables

Sorting by arrival delay

```
arrange(fdata, arr_delay)
```

In descending order

```
arrange(fdata, -arr_delay)
```

Sorting by carrier arrival delay

```
arrange(fdata, carrier, -arr_delay)
```

Filtering out observations

Only keep for observation on the first of May

```
fdata_0501 <- filter(fdata, month == 5, day == 1)
```

Or remove those pesky missing values

```
filter(fdata, !is.na(arr_time))
```

Selecting variables

Only select arrival delay and carrier

```
select(fdata, arr_delay, carrier)
```

Or select a range of variables

```
select(fdata, month : arr_time )
```

Creating new variables

Perhaps you would like to have arrival delay squared

```
mutate(fdata, arr_delay_2 = arr_delay^2 )
```

Or change from minutes to seconds

```
mutate(fdata, arr_delay_s = arr_delay/60)
```

And finally summarizing stuff (hey; statistics!)

Say, you would like to have the mean and standard deviation of arrival delay

```
summarize(fdata,
          ave_arr_delay = mean(arr_delay),
          sd_arr_delay = mean(arr_delay)
          )
```

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

Hey!

But not automatically corrected for missings!

```r
summarize(fdata,
          ave_arr_delay = mean(arr_delay, na.rm = TRUE),
          sd_arr_delay = sd(arr_delay, na.rm = TRUE)
)
```

```
## # A tibble: 1 x 2
##   ave_arr_delay sd_arr_delay
##           <dbl>        <dbl>
## 1          6.90         44.6
```

Other stuff

Rename stuff

```
rename(fdata, airline = carrier)
```

Remove duplicate rows:

```
distinct(fdata, carrier)
```

Count stuff

```
summarize(fdata, count = n() )
```

Making this work! Group data

```r
fdata_carrier <- group_by(fdata, carrier)
summarize(fdata_carrier,
          mean_arr_delay = mean(arr_delay, na.rm = TRUE),
          number = n())
```

```
## # A tibble: 16 x 3
##    carrier mean_arr_delay number
##    <chr>            <dbl>  <int>
##  1 9E               7.38   18460
##  2 AA               0.364  32729
##  3 AS              -9.93     714
##  4 B6               9.46   54635
##  5 DL               1.64   48110
##  6 EV              15.8    54173
```

Getting jiggy with it

```
fdata_carrier_month <- group_by(fdata, carrier, month)
summarize(fdata_carrier_month,
          mean_arr_delay = mean(arr_delay, na.rm = TRUE),
          number = n())
```

```
## # A tibble: 185 x 4
## # Groups:   carrier [16]
##    carrier month mean_arr_delay number
##    <chr>   <int>          <dbl>  <int>
##  1 9E          1          10.2    1573
##  2 9E          2           8.28   1459
##  3 9E          3           2.03   1627
##  4 9E          4           5.47   1511
##  5 9E          5          10.4    1462
```

Chaining your verbs: pipes

You do not need to create all kinds of new intermediate datasets. You can also link commands by using the pipe (`%>%`) operator:

```
fdata_carrier <- fdata %>%
            group_by(carrier) %>%
            summarize(
                mean_arr_delay = mean(arr_delay,
                                      na.rm = TRUE),
                number = n())
```

In combination with ggplot2

```r
ggplot(fdata_carrier,
       aes(x = reorder(carrier, -mean_arr_delay),
           y = mean_arr_delay)) +
    geom_bar(stat = "identity") +
    theme_bw() +
    labs(x = "Airline code", y = "Mean arrival delay")
```

Or just go wild!

We want to plot arrival delay by month and carrier:

```
fdata_carrier_month <- fdata %>%
    group_by(carrier, month) %>%
    summarize(
        mean_arr_delay = mean(arr_delay, na.rm = TRUE),
        number = n())
```

and plot it!

```
ggplot(fdata_carrier_month,
       aes(x = carrier, y = mean_arr_delay)) +
    geom_bar(stat = "identity") +
    theme_bw() +
    labs(x = "Airline code",
         y = "Mean arrival delay") +
    theme(text = element_text(size = 12) ,
          axis.text.x = element_text(angle=90,
                                     hjust=1))  +
    facet_wrap(~month)
```

Tidy data

Tidy data: what's that?

Happy families are all alike; every unhappy family is unhappy in its own way — Leo Tolstoy

- A dataset is a collection of values, usually either numbers (if quantitative) or strings (if qualitative)
  - every column is a variable
  - every row is an observation
  - every cell is a single value
- Values are organised in two ways. Every value belongs to a variable (rows) and an observation (columns).

When do you need it?

- Requirements of programs or packages
  - for visualisation (`ggplot2` or `sf`)
  - for estimation (regression (?), conditional logits!, time-series, spatial statistics, `igraph`, etc.)
- Because you need to restructure data
  - for reading in data (remember sawtooth output!)
  - data from government websites (Statline!)
  - structure depends on unit of analysis

Violations of tidyness

- Column headers are values, not variable names
- Multiple variables are stored in one column
- Variables are stored in both rows and columns
- [Multiple types of observational units are stored in the same table]
- [A single observational unit is stored in multiple tables]

Example

This is fine if you are interested in the whole educational career

```
grades <- tibble(
    name = c("Erik", "Eric", "Thomas", "Jos"),
    quiz_1 = c(10, 9, 6, 8),
    quiz_2 = c(9, 9, 5, 7),
    exam = c(7, 6, NA, 7)
)
grades
```

Example (strikes back)

But if you are interested in grades, then invoke `pivot_longer`

```
grades2 <- grades %>%
    pivot_longer(quiz_1:exam,
                 names_to = "assessment",
                 values_to = "grade") %>%
    arrange(name, assessment)
grades2
```

Similar example

Time-series context

```
billboard
billboard2 <- billboard %>%
    pivot_longer(
        wk1:wk76,
        names_to = "week",
        values_to = "rank",
        values_drop_na = TRUE
    )
billboard2
```

Multiple values in one cell

```
who1 <- who %>%
    pivot_longer(
        cols = new_sp_m014:newrel_f65,
        names_to = "key",
        values_to = "cases",
        values_drop_na = TRUE
    )
who1
```

Multiple values in one cell (revisited)

```
who2 <- who1 %>%
    mutate(key = stringr::
             str_replace(key, ”newrel”, ”new_rel”)) >%
    separate(key, c(”new”, ”type”,
                    ”sexage”), sep = ”_”) >%
    select(-new, -iso2, -iso3) >%
    separate(sexage, c(”sex”, ”age”), sep = 1) >%
who2
```

Variables are stores in rows

```
weather <- tibble(
    station = c("Amsterdam", "Amsterdam", "Maastricht", "
    element = c("min", "max", "min", "max"),
    d1 = c(4, 11, 3, 15),
    d2 = c(7, 10, 2, 10),
    d3 = c(5, 14, 5, 16)
)
weather
```

Variables are stores in rows (the return of)

Now make wide dataset (`pivot_wider()`)

```
weather2 <- weather %>%
    pivot_longer(
        d1:d3,
        names_to = "day",
        values_to = "temperature",
        values_drop_na = TRUE
    ) %>%
    pivot_wider(names_from = element, values_from = tempe
```

Conditional logit example

```r
housing_prices <- tibble(
    prices = c(600000, 500000, 470000),
    city = c("Aadam", "Bedam", "Cedam"),
    historic_centre = c(0, 0, 1)
)
housing_prices
```

Conditional logit example (with a vengeance)

```
housing_prices <- housing_prices %>%
    mutate(obs = seq(1: nrow(housing_prices)))
    mutate(chosen = rep(1, nrow(housing_prices) ) ) %>%
    pivot_wider(names_from = city,
                values_from = chosen) %>%
    replace(is.na(.), 0) %>%
    pivot_longer(Aadam:Cedam,
                 names_to = "city",
                 values_to =  "chosen")
housing_prices
```

In conclusion

- Script everything you do with data
    - both ex-ante and ex-post analysis (during)
- `tidyverse` package provides excellent support
    - for small/medium sized datasets (until a couple of million obs.)
- tidying data (restructuring) is more important than you might think as you need to restructure some of your own data
    - avoid using Excel
    - first think about unit of analysis and what you want