Applied Data Science with R: Data Transformation

Matthias Haber

27 February 2019

Last Weeks Homework

Quiz question: Symptoms of messy data

Which of the following is not a symptom of messy data?

- Multiple values are stored in one column

Tidy up dataset: Read data

```r
library(readxl)
library(tidyverse)
url <- paste0("http://s3.amazonaws.com/",
              "assets.datacamp.com/",
              "production/course_1294/datasets/mbta.xlsx")
download.file(url, "mbta.xlsx")
mbta <- read_excel("mbta.xlsx", skip = 1,
                   range = cell_cols(2:60))
```

Tidy up dataset: Look at data

```r
head(mbta)
```

```
## # A tibble: 6 x 59
##   mode  `2007-01` `2007-02` `2007-03` `2007-04` `2007-05` `2007-06`
##   <chr> <chr>     <chr>         <dbl> <chr>     <chr>         <dbl>
## 1 All ~ NA        NA           1188. NA        NA           1246.
## 2 Boat  4         3.6            40  4.3       4.9             5.8
## 3 Bus   335.819   338.675       340. 352.162   354.367       351.
## 4 Comm~ 142.2     138.5         138. 139.5     139           143
## 5 Heav~ 435.294   448.271       459. 472.201   474.579       477.
## 6 Ligh~ 227.231   240.262       241. 255.557   248.262       246.
## # ... with 52 more variables: `2007-07` <chr>, `2007-08` <chr>,
## #   `2007-09` <dbl>, `2007-10` <chr>, `2007-11` <chr>, `2007-12` <db
## #   `2008-01` <chr>, `2008-02` <chr>, `2008-03` <dbl>, `2008-04` <ch
## #   `2008-05` <chr>, `2008-06` <dbl>, `2008-07` <chr>, `2008-08` <ch
## #   `2008-09` <dbl>, `2008-10` <chr>, `2008-11` <chr>, `2008-12` <db
## #   `2009-01` <chr>, `2009-02` <chr>, `2009-03` <dbl>, `2009-04` <ch
## #   `2009-05` <chr>, `2009-06` <dbl>, `2009-07` <chr>, `2009-08` <ch
```

Tidy up dataset: Gather years

```r
mbta_tidy <- mbta %>%
  tidyr::gather(`2007-01`:`2011-10`,
                key = year, value = passengers,
                convert = TRUE)
mbta_tidy
```

```
## # A tibble: 638 x 3
##    mode                 year    passengers
##    <chr>                <chr>   <chr>
##  1 All Modes by Qtr     2007-01 NA
##  2 Boat                 2007-01 4
##  3 Bus                  2007-01 335.819
##  4 Commuter Rail        2007-01 142.2
##  5 Heavy Rail           2007-01 435.294
##  6 Light Rail           2007-01 227.231
##  7 Pct Chg / Yr         2007-01 0.02
##  8 Private Bus          2007-01 4.772
##  9 RIDE                 2007-01 4.9
```

Tidy up dataset: Separate year

```r
mbta_tidy <- mbta_tidy %>%
  tidyr::separate(year, into = c("year", "month"))
mbta_tidy
```

```
## # A tibble: 638 x 4
##    mode               year  month passengers
##    <chr>              <chr> <chr> <chr>
##  1 All Modes by Qtr   2007  01    NA
##  2 Boat               2007  01    4
##  3 Bus                2007  01    335.819
##  4 Commuter Rail      2007  01    142.2
##  5 Heavy Rail         2007  01    435.294
##  6 Light Rail         2007  01    227.231
##  7 Pct Chg / Yr       2007  01    0.02
##  8 Private Bus        2007  01    4.772
##  9 RIDE               2007  01    4.9
## 10 Trackless Trolley 2007  01    12.757
## # ... with 628 more rows
```

Tidy up dataset: Spread mode of transportation

```r
mbta_tidy <- mbta_tidy %>%
  tidyr::spread(mode, passengers)
mbta_tidy
```

```
## # A tibble: 58 x 13
##    year  month `All Modes by Q~ Boat  Bus   `Commuter Rail` `Heavy R
##    <chr> <chr> <chr>            <chr> <chr> <chr>           <chr>
##  1 2007  01    NA               4     335.~ 142.2           435.294
##  2 2007  02    NA               3.6   338.~ 138.5           448.271
##  3 2007  03    1187.653         40    339.~ 137.7           458.583
##  4 2007  04    NA               4.3   352.~ 139.5           472.201
##  5 2007  05    NA               4.9   354.~ 139             474.579
##  6 2007  06    1245.959         5.8   350.~ 143             477.032
##  7 2007  07    NA               6.521 357.~ 142.391         471.735
##  8 2007  08    NA               6.572 355.~ 142.364         461.605
##  9 2007  09    1256.571         5.469 372.~ 143.051         499.566
## 10 2007  10    NA               5.145 368.~ 146.542         457.741
## # ... with 48 more rows, and 6 more variables: `Light Rail` <chr>, `
```

Tidy up dataset: Keep wanted columns

```r
mbta_tidy <- mbta_tidy %>%
  .[,c(1:2,6:8)]
mbta_tidy
```

```
## # A tibble: 58 x 5
##    year  month `Commuter Rail` `Heavy Rail` `Light Rail`
##    <chr> <chr> <chr>           <chr>        <chr>
##  1 2007  01    142.2           435.294      227.231
##  2 2007  02    138.5           448.271      240.262
##  3 2007  03    137.7           458.583      241.444
##  4 2007  04    139.5           472.201      255.557
##  5 2007  05    139             474.579      248.262
##  6 2007  06    143             477.032      246.108
##  7 2007  07    142.391         471.735      243.286
##  8 2007  08    142.364         461.605      234.907
##  9 2007  09    143.051         499.566      265.748
## 10 2007  10    146.542         457.741      241.434
## # ... with 48 more rows
```

Tidy up dataset: Gather rail modes

```r
mbta_tidy <- mbta_tidy %>%
  tidyr::gather(`Commuter Rail`:`Light Rail`,
                key="rail_type", value = passengers)
mbta_tidy
```

```
## # A tibble: 174 x 4
##    year  month rail_type     passengers
##    <chr> <chr> <chr>         <chr>
##  1 2007  01    Commuter Rail 142.2
##  2 2007  02    Commuter Rail 138.5
##  3 2007  03    Commuter Rail 137.7
##  4 2007  04    Commuter Rail 139.5
##  5 2007  05    Commuter Rail 139
##  6 2007  06    Commuter Rail 143
##  7 2007  07    Commuter Rail 142.391
##  8 2007  08    Commuter Rail 142.364
##  9 2007  09    Commuter Rail 143.051
## 10 2007  10    Commuter Rail 146.542
```

Tidy up dataset: Compute sum

```r
mbta_tidy <- mbta_tidy %>%
  dplyr::mutate(passengers = as.numeric(passengers)) %>%
  dplyr::summarise(sum(passengers))
mbta_tidy
```

```
## # A tibble: 1 x 1
##   `sum(passengers)`
##               <dbl>
## 1            49859.
```

Week 4: Data transformation

Prerequisites

Packages

```r
library(tidyverse)
```

Data

336,776 flights that departed from New York City in 2013

```r
# install.packages("nycflights13")
library(nycflights13)
```

year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	1	517	515	2
2013	1	1	533	529	4
2013	1	1	542	540	2
2013	1	1	544	545	-1

Data transformation with dplyr

Variable types

- int: integers
- dbl: doubles, or real numbers
- chr: character vectors, or strings
- dttm: date-times (a date + a time)
- lgl: logical, vectors that contain only `TRUE` or `FALSE`
- fctr: factors
- date: dates

`dplyr` core functions

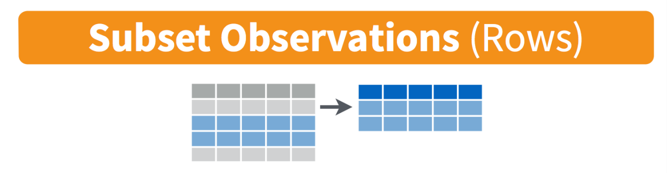
- `filter()`: select rows by their values
- `arrange()`: order rows
- `select()`: select columns by their names
- `mutate()`: create new variables
- `summarize()`: collapse many values down to a single summary
- `group_by()`: operate on it group-by-group
- `rename()`: rename columns
- `distinct()`: find distinct rows

Command structure (for all dplyr verbs):

- first argument is a data frame
- return value is a data frame
- nothing is modified in place

`filter()`

`filter()` allows to subset observations based on their values. The function takes logical expressions and returns the rows for which all are `TRUE`.



filter()

Let's select all flights on January 1st:

```
filter(flights, month == 1, day == 1)
```

year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	1	517	515	2
2013	1	1	533	529	4
2013	1	1	542	540	2
2013	1	1	544	545	-1
2013	1	1	554	600	-6
2013	1	1	554	558	-4

`filter()`

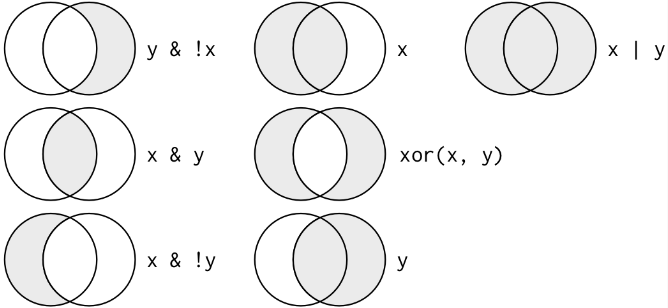
`filter()` revolves around using comparison operators: >, >=, <, <=, `!=` (not equal), and == (equal).

`dplyr` functions like `filter()` never modify inputs but instead return a new data frame that needs to be assigned to an object if you want to save the result.

```
jan1 <- filter(flights, month == 1, day == 1)
```

Boolean operators

`filter()` also supports the Boolean operators `&` ("and"), `|` ("or"), `!` (is "not"), and `xor` (exclusive "or".



De Morgan's law: `!(x & y)` = `!x | !y` & `!(x | y)` = `!x & !y`

Boolean operators

Why does this not work?

```
filter(flights, month == 11 | 12)
```

Generally a good idea to use `x %in% y`, which will select every row where `x` is part of the values of `y`.

```
filter(flights, month %in% c(11, 12))
```

`between` condition

Another useful dplyr filtering helper is `between()`. `between(x, left, right)` is equivalent to `x >= left & x <= right`.

To `filter()` all flights that departed between midnight and 6am (inclusive):

```
filter(flights, between(dep_time, 0, 600))
```

filter() exclusion

filter() by default excludes FALSE and NA values.

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

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

filter() exclusion

If you want to preserve missing values, you have to explicitly state it.

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

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

`filter()` exercises

1. Find all flights that
    1.1 Had an arrival delay of two or more hours.
    1.2 Arrived more than two hours late, but didn't leave late.
    1.3 Flew to Houston (`IAH` or `HOU`).
    1.4 Were operated by United, American, or Delta.
    1.5 Departed in summer (July, August, and September).

`arrange()`

`arrange()` takes a data frame and a set of column names to order the rows by. Multiple column names are evaluated subsequently.

```
arrange(flights, year, month, day)
```

year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	1	517	515	2
2013	1	1	533	529	4
2013	1	1	542	540	2
2013	1	1	544	545	-1
2013	1	1	554	600	-6
2013	1	1	554	558	-4

`arrange()` in descending order

By dafault `arrange()` sorts values in ascending order. Use `desc()` to re-order by a column in descending order.

```
arrange(flights, desc(arr_delay))
```

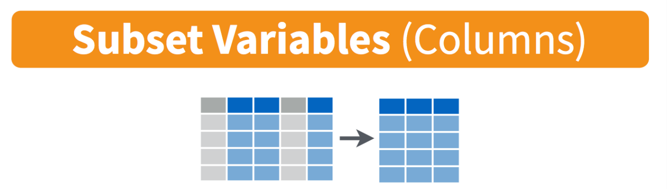
year	month	day	dep_time	sched_dep_time	dep_delay
2013	1	9	641	900	1301
2013	6	15	1432	1935	1137
2013	1	10	1121	1635	1126
2013	9	20	1139	1845	1014
2013	7	22	845	1600	1005
2013	4	10	1100	1900	960

`arrange()` exercises

2. Sort `flights` to
   2.1 find the flight that departed the earliest (earlier than scheduled)
   2.2 find the most delayed flight.
   2.3 find the flight that travelled the longest and that travelled the shortest distance.

`select()`

`select()` is used to select a subset of variables from a dataset.




```r
select(flights, year, month, day)
```

year	month	day
2013	1	1
2013	1	1
2013	1	1
2013	1	1

select()

`select()` has various helper functions:

- `everything()`: selects all variables.
- `starts_with("abc")`: matches names that begin with "abc".
- `ends_with("xyz")`: matches names that end with "xyz".
- `contains("ijk")`: matches names that contain "ijk".
- `matches("(.)\\1")`: selects variables that match a regular expression.
- `num_range("x", 1:3)` matches `x1`, `x2` and `x3`.

See `?select` for more details.

select()

You can use `select()` to rename variables

```r
select(flights, tail_num = tailnum)
```

which will drop all of the variables not explicitly mentioned. Therefore it's better to use `rename()` instead:

```r
rename(flights, tail_num = tailnum)
```

select() exercises

3.1 What are three distinct ways to select dep_time, dep_delay, arr_time, and arr_delay from flights.

3.2 What does the one_of() function do? Why might it be helpful in conjunction with this vector?

```
vars <- c("year", "month", "day", "dep_delay",
          "arr_delay")
```

`mutate()`

`mutate()` allows to add new columns to the end of your dataset that are functions of existing columns.

mutate()

```r
flights %>%
  select(ends_with("delay"), distance, air_time) %>%
  mutate(gain = arr_delay - dep_delay,
         speed = distance / air_time * 60
)
```

dep_delay	arr_delay	distance	air_time	gain	speed
2	11	1400	227	9	370.0441
4	20	1416	227	16	374.2731
2	33	1089	160	31	408.3750
-1	-18	1576	183	-17	516.7213
-6	-25	762	116	-19	394.1379
-4	12	719	150	16	287.6000

transmute()

Use transmute() to only keep the new variables:

```
transmute(flights,
  gain = arr_delay - dep_delay,
  hours = air_time / 60,
  gain_per_hour = gain / hours
)
```

Functions to use with `mutate()`

There are many functions for creating new variables with `mutate()`:

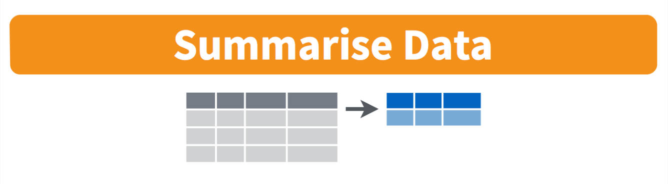
- Arithmetic operators: `+`, `-`, `*`, `/`, `^` (e.g. `air_time / 60`).
- Aggregate functions: `sum(x)` `mean(y)` (e.g. `mean(dep_delay)`).
- Modular arithmetic: `%/%` (integer division) and `%%` (remainder), where `x == y * (x %/% y) + (x %% y)`.
- Logs: `log()`, `log2()`, `log10()`.
- Offsets: `lead()` and `lag()` (e.g. `x - lag(x)`).
- Cumulative and rolling aggregates: `cumsum()`, `cumprod()`, `cummin()`, `cummax()`, `cummean()`.
- Logical comparisons, `<`, `<=`, `>`, `>=`, `!=`.
- Ranking: `min_rank()`, `row_number()`, `dense_rank()`, `percent_rank()`, `cume_dist()`, `ntile()`.

`mutate()` exercises

4. Use `mutate()` to
    4.1 Create new variables for `dep_time` and `sched_dep_time` that measure time in the number of minutes after midnight.
    4.2 Compare `air_time` with `arr_time - dep_time`. What do you see? What do you need to do to fix it?
    4.3 Find the 10 most delayed flights using a ranking function. How do you want to handle ties?

summarize()

summarize() collapses a data frame to a single row.

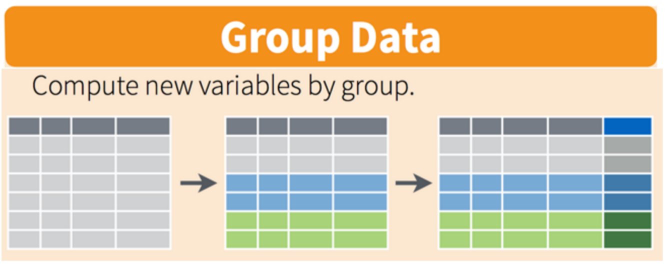



```r
summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
```

```
## # A tibble: 1 x 1
##   delay
##   <dbl>
## 1  12.6
```

`summarize()` with `group_by()`

`summarize()` is most effectively used with `group_by()`, which changes the unit of analysis from the complete dataset to individual groups.



Grouping is most useful in conjunction with `summarise()`, but you can also do convenient operations with `mutate()` and `filter()`.

summarize() with group_by()

For example, to get the average delay per date

```
flights %>%
  group_by(year, month, day) %>%
  summarise(delay = mean(dep_delay, na.rm = TRUE))
```

summarize() count

For aggregations it is generally a good idea to include a count `n()`. For example, let's look at the (not cancelled) planes that have the highest average delays:

```
flights %>%
  filter(!is.na(dep_delay), !is.na(arr_delay))
  group_by(tailnum) %>%
  summarise(delay = mean(arr_delay)) %>%
  arrange(delay)
```

`summarize()` useful functions

There are a number of useful summary functions:

- Measures of location: `mean(x)`, `sum(x)`, `median(x)`.
- Measures of spread: `sd(x)`, `IQR(x)`, `mad(x)`.
- Measures of rank: `min(x)`, `quantile(x, 0.25)`, `max(x)`.
- Measures of position: `first(x)`, `nth(x, 2)`, `last(x)`.
- Counts: `n()`, `sum(!is.na(x))`, `n_distinct(x)`.
- Counts and proportions of logical values: `sum(x > 10)`, `mean(y == 0)`.

`summarize()` exercises

5. Use `summarize()` to
   5.1 Look at the number of cancelled flights per day. Is there a pattern? Is the proportion of cancelled flights related to the average delay?
   5.2 Find the carrier with the worst delays.

Homework Exercises

Homework Exercises

For this week's homework exersises go to Moodle and answer the Quiz posted in the Week 4: Data Transformation section.

Deadline: Tuesday, March 5.

That's it for today. Questions?