Statistical programming

Data manipulation and tidying data

Marc Comas-Cufí

- `dplyr`: a grammar of data manipulation
- `tidyr`: helpers to create **tidy data**
- More data wrangling:
  - Categorical variables (`forcats`)
  - Dates and time (`lubridate`)
  - Strings (`stringr`)

data manipulation

```r
library(nycflights13)
data(package = 'nycflights13')
```

```
Data sets in package ‘nycflights13’:

airlines                    Airline names.
airports                    Airport metadata
flights                     Flights data
planes                      Plane metadata.
weather                     Hourly weather data
```

```
  1  flights
```

```
# A tibble: 336,776 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
5  2013     1     1      554            600        -6      812
# … with 336,771 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

By loading `dplyr` package,

```
1  library(dplyr)
```

we get access to a very useful set of functions:

- `filter()` picks cases based on their values.
- `arrange()` changes the ordering of the rows.
- `select()` picks variables based on their names.
- `mutate()` adds new variables that are functions of existing variables.
- `summarise()` reduces multiple values down to a single summary.
- `group_by()` allows performing any operation “by group”.

Cheat sheet pdf

- Selecting all flights on January 1st (`month == 1` and `day == 1`)

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

```
# A tibble: 842 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
5  2013     1     1      554            600        -6      812
# … with 837 more rows, and 12 more variables: sched_arr_time <int>,
#   arr_delay <dbl>, carrier <chr>, flight <int>, tailnum <chr>,
#   origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#   hour <dbl>, minute <dbl>, time_hour <dttm>
```

- Selecting all flights on January 1st
  (`month == 1` and `day == 1`)
- Selecting all flights from six first years of the year
  (`month <= 6`)
- Selecting all flights departing with no more than 5 minutes from scheduled departure time
  (`-5 <= dep_delay` and `dep_delay <= 5`) or (`abs(dep_delay) <= 5`) or (`between(dep_delay, -5, 5)`)

Activity

- Select flights flying to “IAH” or “HOU”
- Departed in summer (July, August and September)

- `slice()`. Select rows by position (helpers `slice_head()`, `slice_tail()`, `slice_min()`, `slice_max()`, `slice_sample()`)
- `distinct()`. Select distinct observations given certain variables
- `sample_n()`, `sample_frac()`. Random selection of rows

- Order flights by year, month, day and delay

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

```
# A tibble: 336,776 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      940            955       -15     1226
2  2013     1     1     2030           2045       -15     2150
3  2013     1     1     1716           1730       -14     1947
4  2013     1     1      946            959       -13     1146
5  2013     1     1     2217           2229       -12      249
# … with 336,771 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

- Order flights by year, month, day and **decreasing** delay

```
  1  arrange(flights, year, month, day, desc(dep_delay))
```

```
# A tibble: 336,776 × 19
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      848           1835       853     1001
2  2013     1     1     2343           1724       379      314
3  2013     1     1     1815           1325       290     2120
4  2013     1     1     2205           1720       285       46
5  2013     1     1     1842           1422       260     1958
# … with 336,771 more rows, and 12 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>
```

- How could you use `arrange()` to sort all missing values to the start?
- Sort flights to find the fastest (highest speed) flights.
- Which flights travelled the farthest? Which travelled the shortest?

- Select year, month, day and flight number

```
  1  select(flights, year, month, day, flight)
```

```
# A tibble: 336,776 × 4
   year month   day flight
  <int> <int> <int>  <int>
1  2013     1     1   1545
2  2013     1     1   1714
3  2013     1     1   1141
4  2013     1     1    725
5  2013     1     1    461
# … with 336,771 more rows
```

Special functions can be used to facilitate variable selection.

- Matching names: `starts_with()`, `ends_with()`, `contains()`, `matches()`, `num_range()`.
- From vector of names: `all_of()`, `any_of()`.
- Using a function: `where()`

```
  1  select(flights, starts_with('dep_'), contains('arr_'))
```

```
# A tibble: 336,776 × 5
  dep_time dep_delay arr_time sched_arr_time arr_delay
     <int>     <dbl>    <int>          <int>     <dbl>
1      517         2      830            819        11
2      533         4      850            830        20
3      542         2      923            850        33
4      544        -1     1004           1022       -18
5      554        -6      812            837       -25
# … with 336,771 more rows
```

- Brainstorm as many ways as possible to **only** select `dep_time`, `dep_delay`, `arr_time`, and `arr_delay` from flights
- What does the `any_of()` function do? Why might it be helpful with this vector?

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

- Does the result of running the following code surprise you? How do the select helpers deal with case by default? How can you change that default?

```
1  select(flights, contains("TIME"))
```

- Calculating average flight speed (km/h)

```r
mutate(flights,
       distance_km = distance * 1.60934,
       air_time_h = air_time * 60,
       speed_km_h = distance_km / air_time_h)
```

```
# A tibble: 336,776 × 22
   year month   day dep_time sched_dep_time dep_delay arr_time
  <int> <int> <int>    <int>          <int>     <dbl>    <int>
1  2013     1     1      517            515         2      830
2  2013     1     1      533            529         4      850
3  2013     1     1      542            540         2      923
4  2013     1     1      544            545        -1     1004
5  2013     1     1      554            600        -6      812
# … with 336,771 more rows, and 15 more variables:
#   sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#   flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>,
#   time_hour <dttm>, distance_km <dbl>, air_time_h <dbl>,
#   speed_km_h <dbl>
```

- Compare `dep_time`, `sched_dep_time`, and `dep_delay`. How would you expect those three numbers to be related?

(summarise())

- Calculate the average delays in departure and arrival

```
1  summarise(flights,
2            average_dep_delay = mean(dep_delay, na.rm=TRUE),
3            average_arr_delay = mean(arr_delay, na.rm=TRUE))
```

```
# A tibble: 1 × 2
  average_dep_delay average_arr_delay
              <dbl>             <dbl>
1              12.6              6.90
```

- Stratify `flight` table by day

```
1  flights_by_day = group_by(flights, year, month, day)
```

- Calculate the average delays in departure and arrival **by day**

```
1  summarise(flights_by_day,
2            average_dep_delay = mean(dep_delay, na.rm=TRUE),
3            average_arr_delay = mean(arr_delay, na.rm=TRUE))
```

```
# A tibble: 365 × 5
# Groups:   year, month [12]
   year month   day average_dep_delay average_arr_delay
  <int> <int> <int>             <dbl>             <dbl>
1  2013     1     1              11.5              12.7
2  2013     1     2              13.9              12.7
3  2013     1     3              11.0               5.73
4  2013     1     4               8.95             -1.93
5  2013     1     5               5.73             -1.53
# … with 360 more rows
```

- Considering a cancelled flight, a flight with missing either in `dep_delay` or `arr_delay`. Look at the number of cancelled flights per day. Is the proportion of cancelled flights related to the departure delay?

- Summarise the `arr_delay` variable for each month (giving the number of flights and the average delay), for those flights that had a positive `dep_delay`.
  - Nesting functions:

```
summarise(group_by(filter(flights, dep_delay > 0), month),
          average_arr_delay = mean(arr_delay, na.rm = TRUE))
```

  - Temporal tables:

```
flights_filtered = filter(flights, dep_delay > 0)
flights_grouped = group_by(flights_filtered, month)
summarise(flights_grouped,
          average_arr_delay = mean(arr_delay, na.rm = TRUE))
```

- The pipe (%>%) approach:

```
flights %>%
  filter(dep_delay > 0) %>%
  group_by(month) %>%
  summarise(
    average_arr_delay = mean(arr_delay, na.rm = TRUE))
```

- Mutating joins: `inner_join()`, `left_join()`, `right_join`, ...
- `expand()`, `expand_grid()`. Create a tibble from all combinations of inputs.
- `bind_rows()`, `bind_cols()`. Efficiently bind multiple data frames by row and column.

Tidy data is a standard way of mapping the meaning of a dataset to its structure.

1. Every column is a variable
2. Every row is an observation
3. Every cell is a single value

- “Pivotting”. `pivot_longer()` and `pivot_wider()`
- “Rectangling”. `unnest_longer()`, `unnest_wider()`
- “Nesting”. `nest()`, `unnest()`
- Splitting and combining: `separate()` and `unite()`.
- Missings: `complete()`, `drop_na()`, `fill()`.

Cheat Sheet pdf

Column headers are values, no variable names

```
  1  tidyr::relig_income
```

```
# A tibble: 18 × 11
  religion   `<$10k` `$10-20k` `$20-30k` `$30-40k` `$40-50k` `$50-75k`
  <chr>        <dbl>     <dbl>     <dbl>     <dbl>     <dbl>     <dbl>
1 Agnostic        27        34        60        81        76       137
2 Atheist         12        27        37        52        35        70
3 Buddhist        27        21        30        34        33        58
4 Catholic       418       617       732       670       638      1116
5 Don't kno…      15        14        15        11        10        35
# … with 13 more rows, and 4 more variables: `$75-100k` <dbl>,
#   `$100-150k` <dbl>, `>150k` <dbl>, `Don't know/refused` <dbl>
```

Multiple variables stored in a column

```
  1  tidyr::who
```

```
# A tibble: 7,240 × 60
  country     iso2  iso3   year new_sp_m014 new_sp_m1524 new_sp_m2534
  <chr>       <chr> <chr> <int>       <int>        <int>        <int>
1 Afghanistan AF    AFG    1980          NA           NA           NA
2 Afghanistan AF    AFG    1981          NA           NA           NA
3 Afghanistan AF    AFG    1982          NA           NA           NA
4 Afghanistan AF    AFG    1983          NA           NA           NA
5 Afghanistan AF    AFG    1984          NA           NA           NA
# … with 7,235 more rows, and 53 more variables: new_sp_m3544 <int>,
#   new_sp_m4554 <int>, new_sp_m5564 <int>, new_sp_m65 <int>,
#   new_sp_f014 <int>, new_sp_f1524 <int>, new_sp_f2534 <int>,
#   new_sp_f3544 <int>, new_sp_f4554 <int>, new_sp_f5564 <int>,
#   new_sp_f65 <int>, new_sn_m014 <int>, new_sn_m1524 <int>,
#   new_sn_m2534 <int>, new_sn_m3544 <int>, new_sn_m4554 <int>,
#   new_sn_m5564 <int>, new_sn_m65 <int>, new_sn_f014 <int>, …
```

Variables are stored in both rows and columns

```
  1  library(readr)
  2  read_csv("session-02-presentation/weather.csv")
```

```
# A tibble: 22 × 35
   id       year month element    d1    d2    d3    d4    d5    d6    d7
   <chr>   <dbl> <dbl> <chr>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
 1 MX170…  2010     1 tmax       NA  NA    NA      NA  NA      NA    NA
 2 MX170…  2010     1 tmin       NA  NA    NA      NA  NA      NA    NA
 3 MX170…  2010     2 tmax       NA  27.3  24.1    NA  NA      NA    NA
 4 MX170…  2010     2 tmin       NA  14.4  14.4    NA  NA      NA    NA
 5 MX170…  2010     3 tmax       NA  NA    NA      NA  32.1    NA    NA
# … with 17 more rows, and 24 more variables: d8 <dbl>, d9 <lgl>,
#   d10 <dbl>, d11 <dbl>, d12 <lgl>, d13 <dbl>, d14 <dbl>, d15 <dbl>,
#   d16 <dbl>, d17 <dbl>, d18 <lgl>, d19 <lgl>, d20 <lgl>, d21 <lgl>,
#   d22 <lgl>, d23 <dbl>, d24 <lgl>, d25 <dbl>, d26 <dbl>, d27 <dbl>,
#   d28 <dbl>, d29 <dbl>, d30 <dbl>, d31 <dbl>
```

- Parsing dates: `ymd()`, `ydm()`, `myd()`, `mdy()`, `dmy()`, `dym()`

```
library(lubridate)
x <- c(20090101, "2009-01-02", "2009 01 03", "2009-1-4",
       "2009-1, 5", "Created on 2009 1 6", "200901 !!! 07")
(ddates = ymd(x))
```

```
[1] "2009-01-01" "2009-01-02" "2009-01-03" "2009-01-04" "2009-01-05"
[6] "2009-01-06" "2009-01-07"
```

- Extracting `year`, `month` or `day`

```
year(ddates)
```

```
[1] 2009 2009 2009 2009 2009 2009 2009
```

- Calculating time differences

```
1  diff(ymd(20100101) + years(1:5))
```

```
Time differences in days
[1] 365 366 365 365
```

- How many weeks?

```
1  interval(ymd(20101001), ymd(20101101)) / weeks(1)
```

```
[1] 4.428571
```

Cheat Sheet pdf

A cohesive set of functions designed to make working with strings as easy as possible.

- Detect matches: `str_detect()`
- Subset strings: `str_sub()`
- Manage lengths: `str_length()`, `str_trim()`
- Mutate strings: `str_replace()`, `str_to_lower()`, `str_to_title()`
- Join and split: `str_c()`, `str_split()`

Cheat Sheet pdf

A suite of tools that solve common problems with factors:

- `fct_reoder()`: reorder the levels of a factor according to some function
- `fct_infreq()`: reorder the levels of a factor according to category frequencies
- `fct_relevel()`: reorder the levels of a factor manually
- `fct_lump_min()`, `fct_lump_n()`, `fct_lump_prop()`: collapse the least frequent values

Cheat Sheet pdf

- `ggplot2`: a system for declaratively creating graphics, based on “The Grammar of Graphics”
- `ggplot2` extensions.