Introduction to R and the tidyverse

– data wrangling part 1 –

Paolo Crosetto

Todays topics

> today we will deal with **four** topics:

1. data import and export
2. manipulating data
3. joining data from different tables
4. *tidying* data

Before we start: nycflights

```
1 #install.packages("nycflights13")
```

- some data about **all** flights from New York airports in 2013
- we get to know arrival & departure times, delays, carrier…
- tail number, origin, destination…
- not particularly interesting *per se* but big (336K observations)

1. Importing data

getting data into R: packages

> up to now we have worked with data from *packages* (`mpg`; `nycflights`)

- easy to do: install a package, then call a dataset
- all the hard work has been made for you
- if you wish you can import the data into your workspace

```
1 library(nycflights13)
2 df <- flights
```

getting data into R: other sources

> life is not always that easy. You might have…

- …data in the form of (aaarg!) Excel files
- …comma separated (`.csv`) data
- …data coming from SPSS, SAS, STATA, …
- or text data from ASCII sources

getting data into R: `readr`

- when you load the tidyverse you automatically load `readr`
- this is a package that gives you functions to load data

```
read_csv("filename")
```

- of the form `read_...` with different file types

A simple example

> you find some data here: https://goo.gl/kPycfH

- this is the `human development index`, by country
- highest numbers (nearest to 1) are better
- save the file to disk in your own folder
- save it as `HDI.csv`
- open it up with a text editor: what do you see?

A simple example

> now that your data is saved, how do you import it to R?

- you use `read_csv("path_to_file")`
- in your case, it should be "`/Student_Area/SURNAME/HDIdata.csv`"
- in my case:

```
1 df <- read_csv("/home/paolo/Dropbox/Public/HDIdata.csv")
```

There is more than .csv

> `read_csv` just made many things for you

- automatically detect names
- automatically detect column types
- ...

- other useful functions:
  - if the separator is `;` rather than `,` use `read_csv2`
  - if the separator is a `TAB` rather than `,` use `read_tsv`

Some hints

- you can always export to `.csv` in all programs
- *even in Excel!*
- so once you have exported to `.csv`, all is fine
- other binary formats (`.dta`, `.xlsx`) force you to have the right tool

> keep a copy of your data in a **text-based format**

2. Data inspection

Getting to know the data: inspection

```r
library(nycflights13)

flights
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
```

Getting to know the data: View

```
1 View(flights)
```

- `View()` opens an Rstudio data window
- in that window you can
- sort
- arrange
- inspect variables

Inspecting data and summary statistics

```
summary(flights)
```

```
      year          month             day          dep_time    sched_dep_time
 Min.   :2013   Min.   : 1.000   Min.   : 1.00   Min.   :   1   Min.   : 106
 1st Qu.:2013   1st Qu.: 4.000   1st Qu.: 8.00   1st Qu.: 907   1st Qu.: 906
 Median :2013   Median : 7.000   Median :16.00   Median :1401   Median :1359
 Mean   :2013   Mean   : 6.549   Mean   :15.71   Mean   :1349   Mean   :1344
 3rd Qu.:2013   3rd Qu.:10.000   3rd Qu.:23.00   3rd Qu.:1744   3rd Qu.:1729
 Max.   :2013   Max.   :12.000   Max.   :31.00   Max.   :2400   Max.   :2359
                                                 NA's   :8255
   dep_delay          arr_time    sched_arr_time   arr_delay
 Min.   : -43.00   Min.   :   1   Min.   :   1   Min.   : -86.000
 1st Qu.:  -5.00   1st Qu.:1104   1st Qu.:1124   1st Qu.: -17.000
 Median :  -2.00   Median :1535   Median :1556   Median :  -5.000
 Mean   :  12.64   Mean   :1502   Mean   :1536   Mean   :   6.895
 3rd Qu.:  11.00   3rd Qu.:1940   3rd Qu.:1945   3rd Qu.:  14.000
```

Importing data in your workspace

- `flights` is not yet in your workspace
- to import it, assign to an object using `<-`
- we will use the shorthand `df` (you can use any other)

```
1 df <- flights
2 df
```

```
# A tibble: 336,776 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
```

```
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# i 336,766 more rows
```

Inspecting with `skimr`

> There are other packages to inspect data

```
1 # install.packages(skimr)
2 library(skimr)
3 skim(df)
```

Data summary

Name	df
Number of rows	336776
Number of columns	19
_____	
Column type frequency:	

character	4
numeric	14
POSIXct	1
_____	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	e
carrier	0	1.00	2	2	
tailnum	2512	0.99	5	6	
origin	0	1.00	3	3	
dest	0	1.00	3	3	

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	
year	0	1.00	2013.00	
month	0	1.00	6.55	
day	0	1.00	15.71	
dep_time	8255	0.98	1349.11	4
sched_dep_time	0	1.00	1344.25	4
dep_delay	8255	0.98	12.64	
arr_time	8713	0.97	1502.05	5
sched_arr_time	0	1.00	1536.38	4
arr_delay	9430	0.97	6.90	

skim_variable	n_missing	complete_rate	mean	
flight	0	1.00	1971.92	16
air_time	9430	0.97	150.69	
distance	0	1.00	1039.91	7
hour	0	1.00	13.18	
minute	0	1.00	26.23	

Variable type: POSIXct

skim_variable	n_missing	complete_rate	min	max
time_hour	0	1	2013-01-01 05:00:00	201 12-3 23:0

3. Data manipulation

`dplyr`

> we will use the package `dplyr`, from the tidyverse

- don’t worry about the strange name, it has a reason.
- it is called when running `library(tidyverse)`

Structure of dplyr

> simple, direct *verbs* to do the main jobs for you

- they do **not** alter the data saved on memory
- which is **good**: `safe` manipulation!
- if you want to save the altered dataset, use assign `<-`

Data manipulation: `filter()`

> `filter()` allows to extract **rows** from the data frame

- `filter(data, logic)`
- `data` is your data
- `logic` is a logical statement that tells `filter()` what must be included or not
- R understands `>, >=, <, <=, !=, ==`
- R understand also `&, |, !`

Filtering

all December flights

```
 1 filter(df, month == 12)
```

```
# A tibble: 28,135 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013    12     1       13           2359        14      446            445
 2  2013    12     1       17           2359        18      443            437
 3  2013    12     1      453            500        -7      636            651
 4  2013    12     1      520            515         5      749            808
 5  2013    12     1      536            540        -4      845            850
 6  2013    12     1      540            550       -10     1005           1027
 7  2013    12     1      541            545        -4      734            755
 8  2013    12     1      546            545         1      826            835
 9  2013    12     1      549            600       -11      648            659
10  2013    12     1      550            600       -10      825            854
# i 28,125 more rows
```

Filtering over multiple criteria

> all Christmas flights departed around midday

```
 1 filter(df, month == 12 & day == 25 & dep_time>1100 & dep_time <1300)
```

```
# A tibble: 86 × 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013    12    25     1103           1105        -2     1209           1241
 2  2013    12    25     1105           1100         5     1359           1404
 3  2013    12    25     1105           1000        65     1332           1252
 4  2013    12    25     1107           1059         8     1341           1423
 5  2013    12    25     1109           1112        -3     1356           1400
 6  2013    12    25     1109           1112        -3     1258           1310
 7  2013    12    25     1110           1115        -5     1404           1425
 8  2013    12    25     1111           1116        -5     1339           1355
 9  2013    12    25     1114           1120        -6     1416           1432
10  2013    12    25     1118           1105        13     1323           1335
# i 76 more rows
```

Multiple *or* statements: `%in%`

> mutliple `or` statement can be tricky

```
1 # you want all summer flights (June, July, August)
2 filter(df, month == 6 | month == 7 | month == 8)
```

> can become cumbersome!

```
1 # you can use the alternative %in%
2 filter(df, month %in% c(6,7,8))
```

Sort data: `arrange()`

> `arrange()` lets you sort data

- it is like the sorting you do in `View()`, but:
  1. it is done on the console, and
  2. it lets you save the new order to a data frame
- this can be useful when looking for special observations

Selecting columns: `select()`

> What `filter()` is to rows, `select()` is to columns

- more than one variable? list all variables by bare name: `select(data, var1, var2, var3, ...)`
- exclude (drop) some variables? minus sign: `select(data, -var1, -var2, ...)`
- exploit naming patterns? `starts_with("string")`, `ends_with("string")`, `contains("string")`
- select all variables? `everything()`

Renaming variables: `rename()`

- `rename()` = `select()` but keeps all variables
- you use it as `rename(data, newname = oldname)`
- e.g. translating var names to French:

```
 1  rename(df, mois = month, jour = day)
```

```
# A tibble: 336,776 × 19
    year  mois  jour dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
```

```
10  2013     1     1      558           600        -2      753           745
# i 336,766 more rows
```

4. Changing data

Creating new variables: `mutate()`

> you want to create a *new* variable with some manipulation

- you can use `mutate(data, newvar = f(oldvar))`
- where `f()` is some function or manipulation

```
1 df_new <- mutate(df, starting_year = year - 2012)
2 select(df_new, starting_year)
```

```
# A tibble: 336,776 × 1
   starting_year
           <dbl>
 1             1
 2             1
 3             1
 4             1
 5             1
 6             1
 7             1
```

```
 8             1
 9             1
10             1
# i 336,766 more rows
```

`mutate()` properties

- you can create more variables in one single `mutate()` call

```
df_new <- mutate(df, starting_year = year - 2013,
                     speed = distance /air_time * 60,
                     speed2 = speed**2)
select(df_new, starting_year, speed, speed2)
```

```
# A tibble: 336,776 × 3
   starting_year speed  speed2
           <dbl> <dbl>   <dbl>
 1             0  370. 136933.
 2             0  374. 140080.
 3             0  408. 166770.
 4             0  517. 267001.
 5             0  394. 155345.
 6             0  288.  82714.
 7             0  404. 163564.
 8             0  259.  67208.
 9             0  405. 163678.
10             0  319. 101567.
# i 336,766 more rows
```

`mutate()` properties

- you can *use* right away the newly created variables!

```
df_new <- mutate(df, speed = distance /air_time * 60,
                     speed_squared = speed ** 2)
select(df_new, speed, speed_squared)
```

```
# A tibble: 336,776 × 2
   speed speed_squared
   <dbl>         <dbl>
 1  370.       136933.
 2  374.       140080.
 3  408.       166770.
 4  517.       267001.
 5  394.       155345.
 6  288.        82714.
 7  404.       163564.
 8  259.        67208.
 9  405.       163678.
10  319.       101567.
# i 336,766 more rows
```

Functions that work with `mutate()`

- all *vector* functions: input a vector, output a vector
- arithmetic operations: `+ - * / ^`, `exp()`, `sqrt()`, `log()`
- offset: `lag()` to refer to the -1 period. *Useful for increments*
- cumulative functions: `cumsum()`, etc…
- logical (see `filter()`)

Summarise your data: `summarize()`

- create aggregations and summary of your data
- `summarize(data, newvar = f(oldvar))`
- similar to `mutate()` but returns a **scalar (one value)**

```
  1  summarize(df, meandelay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 1 × 1
  meandelay
      <dbl>
1      12.6
```

`summarize()` properties

- you can do more than one summary in one go

```
summarize(df, meandelay = mean(dep_delay, na.rm = TRUE),
          sddelay = sd(dep_delay, na.rm = TRUE),
          maxdeptime = max(dep_time, na.rm = TRUE))
```

```
# A tibble: 1 × 3
  meandelay sddelay maxdeptime
      <dbl>   <dbl>      <int>
1      12.6    40.2       2400
```

Empowering `summarize()`: `group_by()`

> `summarize()` is not so useful: returns **one** value…

- much more powerful if we can **group data**: 1 value/group
- `group_by()` groups the `df` by grouping variable levels
- nothing changes; but R knows

```
 1 group_by(df, month)
```

```
# A tibble: 336,776 × 19
# Groups:   month [12]
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
```

3	2013	1	1	542	540	2	923	850
4	2013	1	1	544	545	-1	1004	1022
5	2013	1	1	554	600	-6	812	837
6	2013	1	1	554	558	-4	740	728
7	2013	1	1	555	600	-5	913	854
8	2013	1	1	557	600	-3	709	723
9	2013	1	1	557	600	-3	838	846
10	2013	1	1	558	600	-2	753	745

example: delay by month

- output has 12 rows

```r
df_bymonth <- group_by(df, month)
summarize(df_bymonth, meandelay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 12 × 2
   month meandelay
   <int>     <dbl>
 1     1     10.0
 2     2     10.8
 3     3     13.2
 4     4     13.9
 5     5     13.0
 6     6     20.8
 7     7     21.7
 8     8     12.6
 9     9      6.72
10    10      6.24
11    11      5.44
```

example: delay by month and day

- output has 365 rows

```r
df_bymonth_byday <- group_by(df, month, day)
summarise(df_bymonth_byday, meandelay = mean(dep_delay, na.rm = TRUE))
```

```
# A tibble: 365 × 3
# Groups:   month [12]
   month   day meandelay
   <int> <int>     <dbl>
 1     1     1     11.5
 2     1     2     13.9
 3     1     3     11.0
 4     1     4      8.95
 5     1     5      5.73
 6     1     6      7.15
 7     1     7      5.42
 8     1     8      2.55
 9     1     9      2.28
10     1    10      2.84
```

5. Putting it all together

Exercise: a complex example

> **what is the speed of flights departing around midday (11 - 13), by month?**

- the input is the flights df
- use `filter()`, `select()`, `mutate()`, `group_by()` and `summarize()`
- the final output is a df with 2 variables (month and speed) and 12 observations (one for each month)

> *à vous de jouer!*

Exercise: solution

- let's combine `filter()`, `select()`, `mutate()`, etc...
- we want to know the average speed of flights departing around midday, by month

```
df_midday <- filter(df, dep_time > 1100 & dep_time < 1300)
df_midday_reduced <- select(df_midday, month, air_time, distance)
df_midday_speed <- mutate(df_midday_reduced,
                          speed = distance / air_time * 60)
df_midday_grouped <- group_by(df_midday_speed, month)
df_midday_final <- summarise(df_midday_grouped,
                             meanspeed = mean(speed, na.rm = TRUE))
```

```
df_midday_final
```

```
# A tibble: 12 × 2
   month meanspeed
   <int>     <dbl>
 1     1      365.
 2     2      369.
```

3	3	385.
4	4	385.
5	5	401.
6	6	397.
7	7	401.
8	8	397.
9	9	407.
10	10	393.
11	11	370.

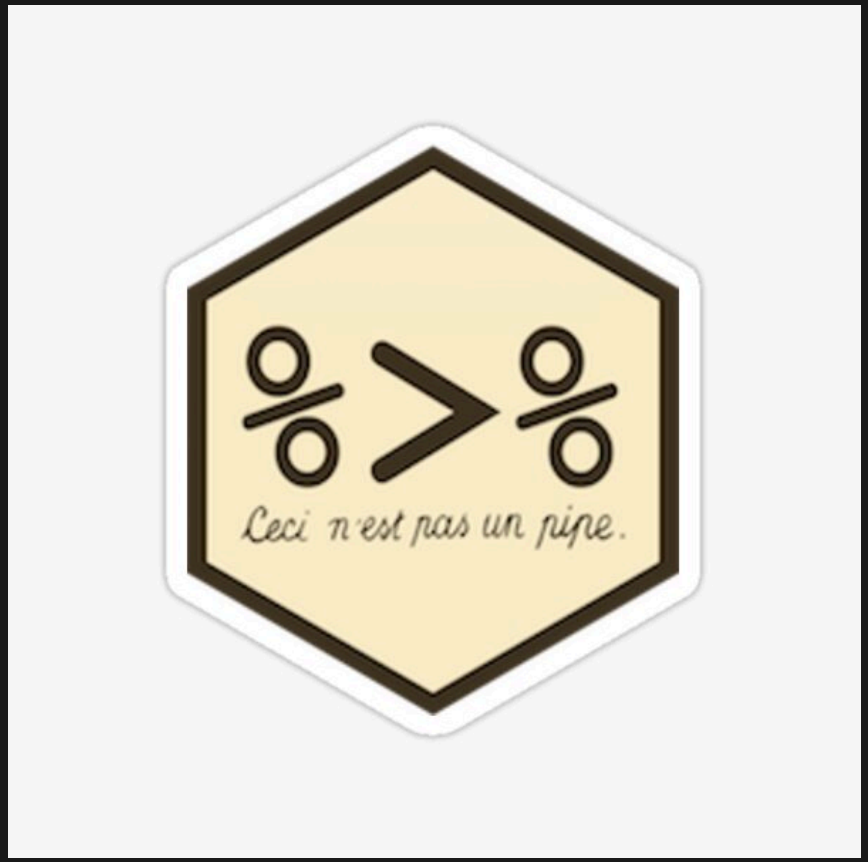
Multiple operations: problems

> the code in the previous slide has several problems

- it forces you to create several different `data frames`
- this clutters your environment
- hard to find proper names
- easy to make errors
- `what can we do?`

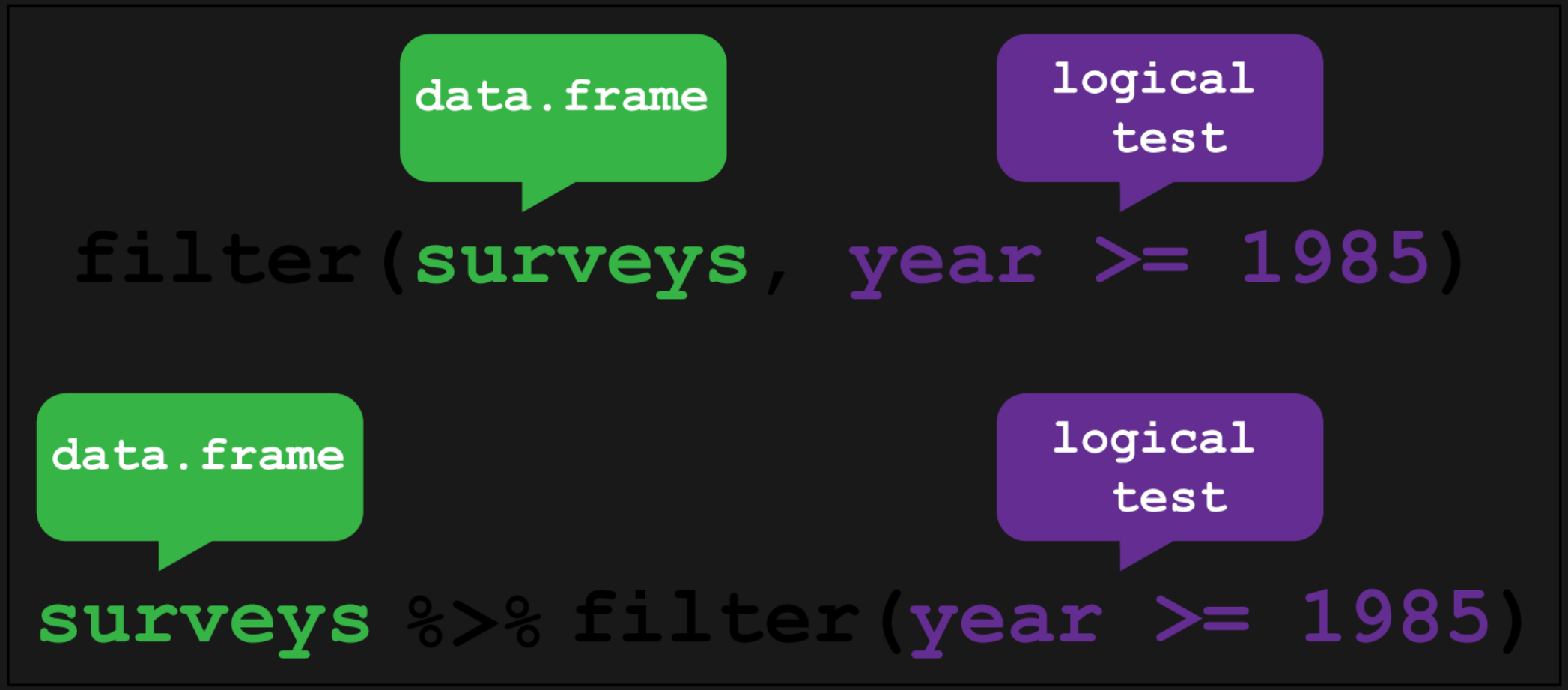
Enters the pipe! %>%

> the *pipe* operator (“*then*”) (“*et après*”)
>
> Insert with `ctrl+shift+M`



Enters the pipe! %>%

> the *pipe* operator ("*then*") ("*et après*")



Pipe as maths

> $f(a, x) \Rightarrow a \text{ \%>\% } f(x)$

```
 1 2*2
```

```
[1] 4
```

```
 1 exp(4)
```

```
[1] 54.59815
```

```
 1 (2*2) %>% exp()
```

```
[1] 54.59815
```

Pipe and `dplyr`

the same code with the pipe

```
df_midday  %>%  filter(dep_time > 1100 & dep_time < 1300) %>%
                select(month, air_time, distance) %>%
                mutate(speed = distance / air_time * 60) %>%
                group_by(month) %>%
                summarise(meanspeed = mean(speed, na.rm = TRUE))
```

```
# A tibble: 12 × 2
   month meanspeed
   <int>     <dbl>
 1     1      365.
 2     2      369.
 3     3      385.
 4     4      385.
 5     5      401.
 6     6      397.
 7     7      401.
 8     8      397.
 9     9      407.
10    10      393.
11    11      379.
```

Exercises: use the pipe

1. what is the `mean delay` of flights by `carrier`, for each `month`?
2. what is the `maximum departure delay` occurred for each of the three NYC `airports`, by each `day`?
3. what is the `mean air time` for each of these three `airports`? from which airport do the longer haul flights depart?