Data Management

R Programming - HAX815X

Jean-Michel Marin

March 2025

Faculty of Sciences, University of Montpellier

Slides based on the work R for statistics and data science

Introduction

- The origins of R date back to the creation of the language S in the mid-1970s
- At that time, the volume and organisation of data were radically different from what we know today
- Data manipulation is shared between two major packages:
  - data.table, which aims above all for efficiency by being highly optimised
  - dplyr, which strikes a balance between efficiency and ease of use

data.table

- The data.table package is designed to efficiently process very large data frames, both in terms of execution time and memory footprint
- The package defines a new syntax for data manipulation

data.table ; Import with fread

The creation of a data.table is similar to that of a data.frame

```
library(data.table)
set.seed(1234)
dt <- data.table(group1 = c("A", "B"),
group2 = rep(c("C", "D"), each = 5000),
value = rnorm(10000),
weight = sample(1:10, 10000, replace = TRUE))
```

data.table ; Import with fread

```
dt
```

```
##         group1 group2       value weight
##         <char> <char>       <num>  <int>
##      1:      A      C -1.20706575      5
##      2:      B      C  0.27742924      7
##      3:      A      C  1.08444118      1
##      4:      B      C -2.34569770      4
##      5:      A      C  0.42912469      9
##     ---
##   9996:      B      D  0.01973902      7
##   9997:      A      D -2.12674529      6
##   9998:      B      D -0.05022201      9
##   9999:      A      D -0.23817408      7
##  10000:      B      D  0.77640531      7
```

data.table ; Import with fread

```
system.time(df <- read.table("dt.csv", sep=",", header=TRUE,
stringsAsFactors=FALSE))
```

```
##    user  system elapsed
##   0.004   0.000   0.005
```

```
system.time(dt <- fread("dt.csv"))
```

```
##    user  system elapsed
##   0.001   0.001   0.002
```

```
all.equal(as.data.table(df),dt)
```

```
## [1] TRUE
```

data.table ; Import with fread

```
set.seed(1234)
n <- 1e6
dt2 <- data.table(a=sample(1:1000,n,replace=TRUE),
b=runif(n),
c=rnorm(n),
d=sample(c("A","B","C","D"),n,replace=TRUE))
write.table(dt2,"dt2.csv",sep=",",row.names=FALSE,quote=FALSE)
cat("Taille en (MB):", round(file.info("dt2.csv")$size/1024^2),"\n")
```

```
## Taille en (MB): 40
```

data.table ; Import with fread

```
system.time(df2 <- read.table("dt2.csv", sep=",", header=TRUE,
stringsAsFactors=FALSE))
```

```
##    user  system elapsed
##   2.319   0.044   2.367
```

```
system.time(dt2 <- fread("dt2.csv"))
```

```
##    user  system elapsed
##   0.072   0.006   0.015
```

data.table ; Import with fread

- There is a factor of 20 in time savings
- If we repeat this procedure with 10 million lines, which gives a file of about 400 Mb
  - reading with read.table leaves time for a coffee
  - with fread, the time saving this time is a factor of 30

Syntax

- At first glance, the syntax for manipulating a data.table may seem similar to that used for data.frame because it uses square brackets [...]
- However, it is very different in practice

In general, the manipulation of a data.table takes the following form

```
data[i, j, by]
```

where i defines a sub-selection of the rows, j a calculation (operation, modification, selection) on the columns and by a grouping of the rows

Syntax

```
dt[group1 == "A", mean(value), by = group2]
```

```
##    group2           V1
##    <char>        <num>
## 1:      C 0.0008501418
## 2:      D 0.0539426361
```

Here we can see the call to the different columns without having to prefix them with `dt$` as is the case in base R

Selection

- On rows `i`
- A `data.table` has no `rownames`; the selection can be made in different ways
- For example, using the row indices

```
dt[1:2, ]
```

```
##    group1 group2      value weight
##    <char> <char>      <num>  <int>
## 1:      A      C -1.2070657      5
## 2:      B      C  0.2774292      7
```

Selection

- On rows `i`

```
dt[c(1,5)]
```

```
##    group1 group2      value weight
##    <char> <char>      <num>  <int>
## 1:      A      C -1.2070657      5
## 2:      A      C  0.4291247      9
```

Selection

- On rows `i`

```
dt[order(value), ]
```

```
##         group1 group2     value weight
##         <char> <char>     <num>  <int>
##      1:      B      C -3.396064      3
##      2:      B      D -3.335108      9
##      3:      A      C -3.233152      2
##      4:      A      D -3.200732      3
##      5:      B      D -3.185662      3
##     ---
##   9996:      B      D  3.357021     10
##   9997:      B      D  3.456972      9
##   9998:      A      D  3.560775      4
##   9999:      B      D  3.606510     10
##  10000:      B      D  3.618107      2
```

Selection

- On rows `i`

```
dt[weight > 9, ]
```

```
##      group1 group2      value weight
##      <char> <char>      <num>  <int>
##   1:      A      C -0.7762539     10
##   2:      B      C -1.4482049     10
##   3:      A      C  0.5747557     10
##   4:      A      C -0.9943401     10
##   5:      B      C -1.2519859     10
##  ---
## 992:      B      D -1.5507432     10
## 993:      A      D -0.1802639     10
## 994:      A      D  1.0293445     10
## 995:      B      D  0.7022495     10
## 996:      A      D  1.0235271     10
```

Selection

- On rows `i`

```
dt[weight > 9 & group2 == "C", ]
```

```
##      group1 group2      value weight
##      <char> <char>      <num>  <int>
##   1:      A      C -0.7762539     10
##   2:      B      C -1.4482049     10
##   3:      A      C  0.5747557     10
##   4:      A      C -0.9943401     10
##   5:      B      C -1.2519859     10
##  ---
## 491:      B      C -0.9043558     10
## 492:      A      C -0.1230837     10
## 493:      B      C -1.1952174     10
## 494:      B      C -0.2176025     10
## 495:      A      C -1.2130983     10
```

Selection

- On columns `j`

```
dt[, 1]
```

```
##        group1
##        <char>
##     1:      A
##     2:      B
##     3:      A
##     4:      B
##     5:      A
##    ---
##  9996:      B
##  9997:      A
##  9998:      B
##  9999:      A
## 10000:      B
```

Selection

- On columns `j`

```
dt[, c(1,3)]
```

```
##        group1       value
##        <char>       <num>
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Selection

- On columns `j`

```
dt[, "group1"]
```

```
##        group1
##        <char>
##     1:      A
##     2:      B
##     3:      A
##     4:      B
##     5:      A
##    ---
##  9996:      B
##  9997:      A
##  9998:      B
##  9999:      A
## 10000:      B
```

Selection

- On columns `j`

```
dt[, c("group1", "value")]
```

```
##        group1       value
##        <char>       <num>
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Selection

- On columns `j`

```
dt[, list(group1, value)]
```

```
##        group1       value
##        <char>       <num>
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Selection

- On columns `j`

```
dt[, .(group1, value)]
```

```
##        group1       value
##        <char>       <num>
##     1:      A -1.20706575
##     2:      B  0.27742924
##     3:      A  1.08444118
##     4:      B -2.34569770
##     5:      A  0.42912469
##    ---
##  9996:      B  0.01973902
##  9997:      A -2.12674529
##  9998:      B -0.05022201
##  9999:      A -0.23817408
## 10000:      B  0.77640531
```

Changing columns

```
dt[, tval := trunc(value)][1:2]
```

```
##    group1 group2      value weight  tval
##    <char> <char>      <num>  <int> <num>
## 1:      A      C -1.2070657      5    -1
## 2:      B      C  0.2774292      7     0
```

- Unlike `data.frame`, it is not necessary to create a new object corresponding to the old dataset concatenated with the new column
- the initial object is modified. Here, the new column `tval` is now part of the `data.table` `dt`

Changing columns

You can create several columns in two ways

```
dt[, c("tvalue", "rvalue") := list(trunc(value), round(value, 2))]
dt[, ':=' (tvalue = trunc(value), rvalue = round(value ,2))]
```

Changing columns

Modifying an existing column is very easy, and deleting it is done by assigning the value NULL to the column

```
dt[, tvalue := tvalue + 10]
dt[, rvalue := NULL]
```

Changing columns

By combining the two parameters i and j, it is therefore possible to modify columns directly for a sub-population

```
dt[ group1 %in% "A", weight := weight * 10L][1:2]
```

```
##    group1 group2      value weight  tval tvalue
##    <char> <char>      <num>  <int> <num>  <num>
## 1:      A      C -1.2070657     50    -1      9
## 2:      B      C  0.2774292      7     0     10
```

In the first line of code, we use L to indicate that this is an integer and not a real (double), otherwise we get a warning

Changing columns

Using this syntax, for example, you can easily replace all the missing values in a column with its mean value

```
dt[is.na(value), value := mean(value, na.rm = TRUE)]
```

Calculations and aggregations with by

Calculating aggregates is made easier with the third part **by**

Aggregates on a single variable

```
dt[, sum(value), by = group1]
```

```
##    group1        V1
##    <char>     <num>
## 1:      A 136.98194
## 2:      B -75.82302
```

```
dt[, sum(value), by = "group1"]
```

```
##    group1        V1
##    <char>     <num>
## 1:      A 136.98194
## 2:      B -75.82302
```

Calculations and aggregations with by

Aggregates on several variables

```
dt[, list(somme = sum(value)), by = list(group1, group2)]
```

```
##    group1 group2      somme
##    <char> <char>      <num>
## 1:      A      C   2.125355
## 2:      B      C -33.898836
## 3:      A      D 134.856590
## 4:      B      D -41.924180
```

```
dt[, list(somme = sum(value)), by = c("group1", "group2")]
```

```
##    group1 group2      somme
##    <char> <char>      <num>
## 1:      A      C   2.125355
## 2:      B      C -33.898836
## 3:      A      D 134.856590
## 4:      B      D -41.924180
```

Calculations and aggregations with by

We can rename the variables used and use R

```
dt[, list(somme = sum(value)), by = list(pop = group1, poids = weight>5)]
```

```
##       pop  poids     somme
##    <char> <lgcl>     <num>
## 1:      A   TRUE 136.98194
## 2:      B   TRUE -38.23590
## 3:      B  FALSE -37.58711
```

dplyr - Introduction

dplyr is one of the best-known packages in the tidyverse

It facilitates the processing and manipulation of data arrays (whether data frame or tibble)

It offers a clear and consistent syntax, in the form of verbs corresponding to functions

dplyr assumes that data is tidy. dplyr verbs take an array of data (data frame or tibble) as input and systematically return a tibble

```
library(dplyr)
```

In order to create a tibble from a csv file, there is the `read_csv` function from the `readr` library

It is also possible to use the `tibble` function with a data frame as an argument

dplyr - Introduction

In what follows, we will use the nycflights13 dataset, contained in the extension of the same name

This corresponds to the data for all flights departing from one of the three New York airports in 2013. It has the particularity of being divided into three tables

- nycflights13::flights contains information on flights: date, departure, destination, times, delays, etc.
- nycflights13::airports contains information on airports
- nycflights13::airlines contains data on airlines

We are going to load the three tables

dplyr - Introduction

```
library(dplyr)
library(nycflights13)
data(flights)
data(airports)
data(airlines)
```

These three datasets are `tibble` tables

Operations on lines

`dplyr::filter()` selects rows from a data table according to a condition

A test is passed to it as a parameter, and only the rows for which this test returns TRUE are kept

Operations on lines

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

```
## # A tibble: 27,004 x 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 26,994 more rows
## # i 11 more variables: 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>
```

Operations on lines

```
flights |> filter(month == 1)
```

```
## # A tibble: 27,004 x 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 26,994 more rows
## # i 11 more variables: 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>
```

Operations on lines

If we only want flights with a departure delay (variable dep_delay) of between 10 and 15 minutes

```
flights |>
  filter(dep_delay >= 10 & dep_delay <= 15)
```

```
## # A tibble: 14,919 x 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      611            600        11      945            931
##  2  2013     1     1      623            610        13      920            915
##  3  2013     1     1      743            730        13     1107           1100
##  4  2013     1     1      743            730        13     1059           1056
##  5  2013     1     1      851            840        11     1215           1206
##  6  2013     1     1      912            900        12     1241           1220
##  7  2013     1     1      914            900        14     1058           1043
##  8  2013     1     1      920            905        15     1039           1025
##  9  2013     1     1     1011           1001        10     1133           1128
## 10  2013     1     1     1112           1100        12     1440           1438
## # i 14,909 more rows
## # i 11 more variables: 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>
```

Operations on lines

If several arguments are passed to `dplyr::filter()`, it automatically adds an **AND** condition

```
flights |>
  filter(dep_delay >= 10, dep_delay <= 15)
```

```
## # A tibble: 14,919 x 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      611            600        11      945            931
##  2  2013     1     1      623            610        13      920            915
##  3  2013     1     1      743            730        13     1107           1100
##  4  2013     1     1      743            730        13     1059           1056
##  5  2013     1     1      851            840        11     1215           1206
##  6  2013     1     1      912            900        12     1241           1220
##  7  2013     1     1      914            900        14     1058           1043
##  8  2013     1     1      920            905        15     1039           1025
##  9  2013     1     1     1011           1001        10     1133           1128
## 10  2013     1     1     1112           1100        12     1440           1438
## # i 14,909 more rows
## # i 11 more variables: 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>
```

Operations on lines

Finally, we can also include functions in the tests, which enable us, for example, to select the flights with the greatest distance

```
flights |>
  filter(distance == max(distance))
```

```
## # A tibble: 342 x 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      857            900        -3     1516           1530
##  2  2013     1     2      909            900         9     1525           1530
##  3  2013     1     3      914            900        14     1504           1530
##  4  2013     1     4      900            900         0     1516           1530
##  5  2013     1     5      858            900        -2     1519           1530
##  6  2013     1     6     1019            900        79     1558           1530
##  7  2013     1     7     1042            900       102     1620           1530
##  8  2013     1     8      901            900         1     1504           1530
##  9  2013     1     9      641            900      1301     1242           1530
## 10  2013     1    10      859            900        -1     1449           1530
## # i 332 more rows
## # i 11 more variables: 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>
```

Operations on lines

Contextual evaluation

It is important to note that dplyr performs a contextual evaluation of the expressions passed to it

Thus, one can directly indicate the name of a variable and `dplyr` will interpret it in the context of the data table, i.e. it will look to see if there is a column with that name in the table

In the expression `flights |> filter(month == 1)`, `month` is interpreted as the month column of the flights table, i.e. `flights$month`

Operations on lines

It is also possible to indicate objects outside the table

```
m <- 2
flights |>
  filter(month == m)
```

```
## # A tibble: 24,951 x 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     2     1      456            500        -4      652            648
##  2  2013     2     1      520            525        -5      816            820
##  3  2013     2     1      527            530        -3      837            829
##  4  2013     2     1      532            540        -8     1007           1017
##  5  2013     2     1      540            540         0      859            850
##  6  2013     2     1      552            600        -8      714            715
##  7  2013     2     1      552            600        -8      919            910
##  8  2013     2     1      552            600        -8      655            709
##  9  2013     2     1      553            600        -7      833            815
## 10  2013     2     1      553            600        -7      821            825
## # i 24,941 more rows
## # i 11 more variables: 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>
```

Operations on lines

On the other hand, if a column exists in the table, it will take priority over objects of the same name in the environment

```
month <- 3
flights |>
  filter(month == month)
```

```
## # A tibble: 336,776 x 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
## # i 11 more variables: 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>
```

Operations on lines

In order to distinguish between what corresponds to a column in the table and an object in the environment, .data and .env can be used (see `help('.env', package = 'rlang')`)

Operations on lines

```
month <- 3
flights |>
  filter(.data$month == .env$month)
```

```
## # A tibble: 28,834 x 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     3     1        4           2159       125      318             56
##  2  2013     3     1       50           2358        52      526            438
##  3  2013     3     1      117           2245       152      223           2354
##  4  2013     3     1      454            500        -6      633            648
##  5  2013     3     1      505            515       -10      746            810
##  6  2013     3     1      521            530        -9      813            827
##  7  2013     3     1      537            540        -3      856            850
##  8  2013     3     1      541            545        -4     1014           1023
##  9  2013     3     1      549            600       -11      639            703
## 10  2013     3     1      550            600       -10      747            801
## # i 28,824 more rows
## # i 11 more variables: 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>
```

Operations on lines

The verb `dplyr::slice()` selects rows from the table according to their position. It is passed a number or a vector of numbers.

If you want to select the 345th row from the airports table

```
airports |>
  slice(345)
```

```
## # A tibble: 1 x 8
##   faa   name               lat   lon   alt    tz dst   tzone
##   <chr> <chr>            <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1 CYF   Chefornak Airport  60.1 -164.    40    -9 A     America/Anchorage
```

Operations on lines

If you want to select the first 5 lines

```
airports |>
  slice(1:5)
```

```
## # A tibble: 5 x 8
##   faa   name                            lat   lon   alt    tz dst   tzone
##   <chr> <chr>                         <dbl> <dbl> <dbl> <dbl> <chr> <chr>
## 1 04G   Lansdowne Airport              41.1 -80.6  1044    -5 A     America/New~
## 2 06A   Moton Field Municipal Airport  32.5 -85.7   264    -6 A     America/Chi~
## 3 06C   Schaumburg Regional            42.0 -88.1   801    -6 A     America/Chi~
## 4 06N   Randall Airport                41.4 -74.4   523    -5 A     America/New~
## 5 09J   Jekyll Island Airport          31.1 -81.4    11    -5 A     America/New~
```

Operations on lines

`dplyr::arrange()` reorders the rows of an array according to one or more columns. So, if we want to sort the flights array according to the delay at departure, in ascending order

```
flights |>
  arrange(dep_delay)
```

```
## # A tibble: 336,776 x 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     7     2040           2123       -43       40           2352
##  2  2013     2     3     2022           2055       -33     2240           2338
##  3  2013    11    10     1408           1440       -32     1549           1559
##  4  2013     1    11     1900           1930       -30     2233           2243
##  5  2013     1    29     1703           1730       -27     1947           1957
##  6  2013     8     9      729            755       -26     1002            955
##  7  2013    10    23     1907           1932       -25     2143           2143
##  8  2013     3    30     2030           2055       -25     2213           2250
##  9  2013     3     2     1431           1455       -24     1601           1631
## 10  2013     5     5      934            958       -24     1225           1309
## # i 336,766 more rows
## # i 11 more variables: 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>
```

Operations on lines

You can sort by several columns. For example, by month, then by delay at departure

```
flights |>
  arrange(month, dep_delay)
```

```
## # A tibble: 336,776 x 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    11     1900           1930       -30     2233           2243
##  2  2013     1    29     1703           1730       -27     1947           1957
##  3  2013     1    12     1354           1416       -22     1606           1650
##  4  2013     1    21     2137           2159       -22     2232           2316
##  5  2013     1    20      704            725       -21     1025           1035
##  6  2013     1    12     2050           2110       -20     2310           2355
##  7  2013     1    12     2134           2154       -20        4             50
##  8  2013     1    14     2050           2110       -20     2329           2355
##  9  2013     1     4     2140           2159       -19     2241           2316
## 10  2013     1    11     1947           2005       -18     2209           2230
## # i 336,766 more rows
## # i 11 more variables: 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>
```

Operations on lines

If you want to sort by a column in descending order, apply the `dplyr::desc()` function

```
flights |>
  arrange(desc(dep_delay))
```

```
## # A tibble: 336,776 x 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     9      641            900      1301     1242           1530
##  2  2013     6    15     1432           1935      1137     1607           2120
##  3  2013     1    10     1121           1635      1126     1239           1810
##  4  2013     9    20     1139           1845      1014     1457           2210
##  5  2013     7    22      845           1600      1005     1044           1815
##  6  2013     4    10     1100           1900       960     1342           2211
##  7  2013     3    17     2321            810       911      135           1020
##  8  2013     6    27      959           1900       899     1236           2226
##  9  2013     7    22     2257            759       898      121           1026
## 10  2013    12     5      756           1700       896     1058           2020
## # i 336,766 more rows
## # i 11 more variables: 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>
```

Operations on lines

Combined with `dplyr::slice()`, `dplyr::arrange()` allows you to select the three flights with the most delays, for example

```
flights |>
  arrange(desc(dep_delay)) |>
  slice(1:3)
```

```
## # A tibble: 3 x 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     9      641            900      1301     1242           1530
## 2  2013     6    15     1432           1935      1137     1607           2120
## 3  2013     1    10     1121           1635      1126     1239           1810
## # i 11 more variables: 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>
```

Operations on lines

`dplyr::slice_sample()` allows you to randomly select a number of rows or a fraction of the rows in an array. So if you want to choose 5 random rows from the airports array

```
airports |>
  slice_sample(n = 5)
```

```
## # A tibble: 5 x 8
##   faa   name                          lat    lon   alt    tz dst   tzone
##   <chr> <chr>                       <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
## 1 AKN   King Salmon                  58.7 -157.     68    -9 A     America/Anchor~
## 2 FXE   Fort Lauderdale Executive   26.2   -80.2    13    -5 A     America/New_Yo~
## 3 FAF   Felker Aaf                  37.1   -76.6    12    -5 A     America/New_Yo~
## 4 AOH   Lima Allen County Airport   40.7   -84.0   975    -5 A     America/New_Yo~
## 5 DAW   Skyhaven Airport            43.3   -70.9   321    -5 A     America/New_Yo~
```

Operations on lines

If we want to randomly select 10% of the flight lines

```
flights |>
  slice_sample(prop = .1)
```

```
## # A tibble: 33,677 x 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     2    11       47           2150       177      137           2256
##  2  2013     3     9      844            830        14     1152           1150
##  3  2013     5     9     1226           1205        21     1448           1425
##  4  2013     2    21     2120           2015        65     2236           2130
##  5  2013    12     1     1927           1910        17     2050           2054
##  6  2013     9     6     1829           1829         0     2140           2143
##  7  2013     2     1      705            700         5     1030           1019
##  8  2013     9    15      556            604        -8      850            906
##  9  2013     1     5      755            800        -5     1106           1106
## 10  2013    12    29     1444           1450        -6     1702           1649
## # i 33,667 more rows
## # i 11 more variables: 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>
```

Operations on lines

These functions are particularly useful for sampling by randomly drawing a certain number of observations from the table

Operations on lines

`dplyr::distinct()` filters the rows of the table to keep only the distinct rows, removing all duplicate rows

```
flights |>
  select(day, month) |>
  distinct()
```

```
## # A tibble: 365 x 2
##      day month
##    <int> <int>
##  1     1     1
##  2     2     1
##  3     3     1
##  4     4     1
##  5     5     1
##  6     6     1
##  7     7     1
##  8     8     1
##  9     9     1
## 10    10     1
## # i 355 more rows
```

Operations on lines

A list of variables can be specified: in this case, for all observations with identical values for the variables in question, `dplyr::distinct()` will only keep the first one

```
flights |>
  distinct(month, day)
```

```
## # A tibble: 365 x 2
##    month   day
##    <int> <int>
##  1     1     1
##  2     1     2
##  3     1     3
##  4     1     4
##  5     1     5
##  6     1     6
##  7     1     7
##  8     1     8
##  9     1     9
## 10     1    10
## # i 355 more rows
```

Operations on columns

`dplyr::select()` allows you to select columns from a data table. So, if you want to extract the lat and lon columns from the airports table

```
airports |>
  select(lat, lon)
```

```
## # A tibble: 1,458 x 2
##      lat    lon
##    <dbl>  <dbl>
##  1  41.1  -80.6
##  2  32.5  -85.7
##  3  42.0  -88.1
##  4  41.4  -74.4
##  5  31.1  -81.4
##  6  36.4  -82.2
##  7  41.5  -84.5
##  8  42.9  -76.8
##  9  39.8  -76.6
## 10  48.1 -123.
## # i 1,448 more rows
```

Operations on columns

If a `-` is placed before the name, the column is eliminated rather than selected

```
airports |>
  select(-lat, -lon)
```

```
## # A tibble: 1,458 x 6
##    faa   name                            alt    tz dst   tzone
##    <chr> <chr>                         <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport               1044    -5 A     America/New_York
##  2 06A   Moton Field Municipal Airport    264    -6 A     America/Chicago
##  3 06C   Schaumburg Regional              801    -6 A     America/Chicago
##  4 06N   Randall Airport                  523    -5 A     America/New_York
##  5 09J   Jekyll Island Airport             11    -5 A     America/New_York
##  6 0A9   Elizabethton Municipal Airport  1593    -5 A     America/New_York
##  7 0G6   Williams County Airport          730    -5 A     America/New_York
##  8 0G7   Finger Lakes Regional Airport    492    -5 A     America/New_York
##  9 0P2   Shoestring Aviation Airfield    1000    -5 U     America/New_York
## 10 0S9   Jefferson County Intl            108    -8 A     America/Los_Angeles
## # i 1,448 more rows
```

Operations on columns

`dplyr::select()` includes a whole series of functions to facilitate the selection of multiple columns ; for example, `dplyr::starts_with()`, `dplyr::ends_width()`, `dplyr::contains()` or `dplyr::matches()` allow you to express conditions on variable names

```
flights |>
  select(starts_with("dep_"))
```

```
## # A tibble: 336,776 x 2
##    dep_time dep_delay
##       <int>     <dbl>
##  1      517         2
##  2      533         4
##  3      542         2
##  4      544        -1
##  5      554        -6
##  6      554        -4
##  7      555        -5
##  8      557        -3
##  9      557        -3
## 10      558        -2
## # i 336,766 more rows
```

Operations on columns

The syntax `column1:column2` allows you to select all the columns between and including `column1` and `column2`

```
flights |>
  select(year:day)
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # i 336,766 more rows
```

Operations on columns

`dplyr::all_of()` and `dplyr::any_of()` allow you to provide a list of variables to be extracted as a text vector ; while `dplyr::all_of()` will return an error if a variable is not found in the starting table, `dplyr::any_of()` will be less strict

```
flights |>
  select(all_of(c("year", "month", "day")))
```

```
## # A tibble: 336,776 x 3
##     year month   day
##    <int> <int> <int>
##  1  2013     1     1
##  2  2013     1     1
##  3  2013     1     1
##  4  2013     1     1
##  5  2013     1     1
##  6  2013     1     1
##  7  2013     1     1
##  8  2013     1     1
##  9  2013     1     1
## 10  2013     1     1
## # i 336,766 more rows
```

Operations on columns

`dplyr::where()` allows you to select variables from a function that returns a logical value. For example, to select only text variables

```
flights |>
  select(where(is.character))
```

```
## # A tibble: 336,776 x 4
##    carrier tailnum origin dest
##    <chr>   <chr>   <chr>  <chr>
##  1 UA      N14228  EWR    IAH
##  2 UA      N24211  LGA    IAH
##  3 AA      N619AA  JFK    MIA
##  4 B6      N804JB  JFK    BQN
##  5 DL      N668DN  LGA    ATL
##  6 UA      N39463  EWR    ORD
##  7 B6      N516JB  EWR    FLL
##  8 EV      N829AS  LGA    IAD
##  9 B6      N593JB  JFK    MCO
## 10 AA      N3ALAA  LGA    ORD
## # i 336,766 more rows
```

Operations on columns

`dplyr::select()` can be used to reorder the columns of a table using the `dplyr::everything()` function, which selects all columns not yet selected

```
airports |>
  select(name, everything())
```

```
## # A tibble: 1,458 x 8
##    name                           faa     lat    lon   alt    tz dst   tzone
##    <chr>                          <chr> <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
##  1 Lansdowne Airport              04G    41.1  -80.6  1044    -5 A     America/~
##  2 Moton Field Municipal Airport  06A    32.5  -85.7   264    -6 A     America/~
##  3 Schaumburg Regional            06C    42.0  -88.1   801    -6 A     America/~
##  4 Randall Airport                06N    41.4  -74.4   523    -5 A     America/~
##  5 Jekyll Island Airport          09J    31.1  -81.4    11    -5 A     America/~
##  6 Elizabethton Municipal Airport 0A9    36.4  -82.2  1593    -5 A     America/~
##  7 Williams County Airport        0G6    41.5  -84.5   730    -5 A     America/~
##  8 Finger Lakes Regional Airport  0G7    42.9  -76.8   492    -5 A     America/~
##  9 Shoestring Aviation Airfield   0P2    39.8  -76.6  1000    -5 U     America/~
## 10 Jefferson County Intl          0S9    48.1 -123.    108    -8 A     America/~
## # i 1,448 more rows
```

Operations on columns

To reorder columns, you can also use `dplyr::relocate()` by specifying the first variables, it is not necessary to add `everything()` because with `dplyr::relocate()` all variables are preserved

```
airports |>
  relocate(lon, lat, name)
```

```
## # A tibble: 1,458 x 8
##       lon   lat name                           faa     alt    tz dst   tzone
##     <dbl> <dbl> <chr>                          <chr> <dbl> <dbl> <chr> <chr>
##  1  -80.6  41.1 Lansdowne Airport               04G    1044    -5 A     America/~
##  2  -85.7  32.5 Moton Field Municipal Airport   06A     264    -6 A     America/~
##  3  -88.1  42.0 Schaumburg Regional             06C     801    -6 A     America/~
##  4  -74.4  41.4 Randall Airport                 06N     523    -5 A     America/~
##  5  -81.4  31.1 Jekyll Island Airport           09J      11    -5 A     America/~
##  6  -82.2  36.4 Elizabethton Municipal Airport  0A9    1593    -5 A     America/~
##  7  -84.5  41.5 Williams County Airport         0G6     730    -5 A     America/~
##  8  -76.8  42.9 Finger Lakes Regional Airport   0G7     492    -5 A     America/~
##  9  -76.6  39.8 Shoestring Aviation Airfield    0P2    1000    -5 U     America/~
## 10 -123.   48.1 Jefferson County Intl           0S9     108    -8 A     America/~
## # i 1,448 more rows
```

Operations on columns

A variant of `dplyr::select()` is `dplyr::rename()`, which allows you to easily rename columns, it is used by passing parameters of the form `new_name = old_name`

```
airports |>
  rename(longitude = lon, latitude = lat)
```

```
## # A tibble: 1,458 x 8
##    faa   name                           latitude longitude   alt    tz dst   tzone
##    <chr> <chr>                             <dbl>     <dbl> <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport                   41.1     -80.6  1044    -5 A     Amer~
##  2 06A   Moton Field Municipal Airpo~       32.5     -85.7   264    -6 A     Amer~
##  3 06C   Schaumburg Regional                42.0     -88.1   801    -6 A     Amer~
##  4 06N   Randall Airport                    41.4     -74.4   523    -5 A     Amer~
##  5 09J   Jekyll Island Airport              31.1     -81.4    11    -5 A     Amer~
##  6 0A9   Elizabethton Municipal Airp~       36.4     -82.2  1593    -5 A     Amer~
##  7 0G6   Williams County Airport            41.5     -84.5   730    -5 A     Amer~
##  8 0G7   Finger Lakes Regional Airpo~       42.9     -76.8   492    -5 A     Amer~
##  9 0P2   Shoestring Aviation Airfield       39.8     -76.6  1000    -5 U     Amer~
## 10 0S9   Jefferson County Intl              48.1    -123.    108    -8 A     Amer~
## # i 1,448 more rows
```

Operations on columns

If the names of columns contain spaces or special characters, they can be enclosed in quotation marks (“) or backticks (‘)

```
flights |>
  rename(
    "retard départ" = dep_delay,
    "retard arrivée" = arr_delay
  ) |>
  select(`retard départ`, `retard arrivée`)
```

```
## # A tibble: 336,776 x 2
##    `retard départ` `retard arrivée`
##              <dbl>            <dbl>
##  1               2               11
##  2               4               20
##  3               2               33
##  4              -1              -18
##  5              -6              -25
##  6              -4               12
##  7              -5               19
##  8              -3              -14
##  9              -3               -8
## 10              -2                8
```

Operations on columns

The `dplyr::rename_with()` function allows you to rename several columns at once by passing a function, for example `toupper()` which changes all the characters to uppercase

```
airports |>
  rename_with(toupper)
```

```
## # A tibble: 1,458 x 8
##    FAA   NAME                            LAT    LON   ALT    TZ DST   TZONE
##    <chr> <chr>                         <dbl>  <dbl> <dbl> <dbl> <chr> <chr>
##  1 04G   Lansdowne Airport              41.1  -80.6  1044    -5 A     America/~
##  2 06A   Moton Field Municipal Airport  32.5  -85.7   264    -6 A     America/~
##  3 06C   Schaumburg Regional            42.0  -88.1   801    -6 A     America/~
##  4 06N   Randall Airport                41.4  -74.4   523    -5 A     America/~
##  5 09J   Jekyll Island Airport          31.1  -81.4    11    -5 A     America/~
##  6 0A9   Elizabethton Municipal Airport 36.4  -82.2  1593    -5 A     America/~
##  7 0G6   Williams County Airport        41.5  -84.5   730    -5 A     America/~
##  8 0G7   Finger Lakes Regional Airport  42.9  -76.8   492    -5 A     America/~
##  9 0P2   Shoestring Aviation Airfield   39.8  -76.6  1000    -5 U     America/~
## 10 0S9   Jefferson County Intl          48.1 -123.    108    -8 A     America/~
## # i 1,448 more rows
```

Operations on columns

The `dplyr::pull()` function allows access to the content of a variable. It is equivalent to the `$` or `[]` operators. It can be passed a variable name or its position

```
airports |>
  pull(alt) |>
  mean()
```

```
## [1] 1001.416
```

Operations on columns

`dplyr::mutate()` allows you to create new columns in the data table, generally from existing variables

```
airports <-
  airports |>
  mutate(alt_m = alt / 3.2808)
```

Operations on columns

Several new columns can be created at once, and successive expressions can take into account the results of previous calculations

```
flights <-
  flights |>
  mutate(
    distance_km = distance / 0.62137,
    vitesse = distance_km / air_time * 60
)
```

Grouped operations

A very important element of `dplyr` is the `dplyr::group_by()` function, it allows you to define groups of lines based on the values of one or more columns

```
flights |>
  group_by(month)
```

Grouped operations

By default, this does not make anything visible, apart from the appearance of a mention of Groups in the display of the result ; but from the moment that groups have been defined, verbs such as `dplyr::slice()` or `dplyr::mutate()` will take them into account during their operations

```
flights |>
  group_by(month) |>
  slice(1)
```

Grouped operations

```
flights |>
  group_by(month) |>
  mutate(mean_delay_month = mean(dep_delay, na.rm = TRUE))
```

Grouped operations

`dplyr::group_by()` can also be useful with `dplyr::filter()`

```
flights |>
  group_by(month) |>
  filter(dep_delay == max(dep_delay, na.rm = TRUE))
```

summarise()

`dplyr::summarise()` allows you to aggregate the rows of the table by performing a summarised operation on one or more columns

```
flights |>
  summarise(
    retard_dep = mean(dep_delay, na.rm=TRUE),
    retard_arr = mean(arr_delay, na.rm=TRUE)
  )
```

summarise()

This function is generally used with `dplyr::group_by()`

```
flights |>
  group_by(month) |>
  summarise(
    max_delay = max(dep_delay, na.rm=TRUE),
    min_delay = min(dep_delay, na.rm=TRUE),
    mean_delay = mean(dep_delay, na.rm=TRUE)
  )
```

summarise()

`dplyr::summarise()` has a special function `dplyr::n()`, which returns the number of rows in the group

```
flights |>
  group_by(dest) |>
  summarise(n = n())
```

count()

Note that when you want to count the number of lines per group, you can directly use the `dplyr::count()` function

```
flights |>
  count(dest)
```

Group according to several variables

Groups can be categorised according to several variables at the same time

```
flights |>
  group_by(month, dest) |>
  summarise(nb = n()) |>
  arrange(desc(nb))
```

Group according to several variables

One can also count according to several variables

```
flights |>
  count(origin, dest) |>
  arrange(desc(n))
```

Group according to several variables

When you perform a `dplyr::group_by()` followed by a `dplyr::summarise()`, the resulting table is automatically ungrouped from the last grouping variable

```
flights |>
  group_by(month, origin, dest) |>
  summarise(nb = n())
```

Group according to several variables

A table can be ungrouped at any time using `dplyr::ungroup()`

```
flights |>
  group_by(month, dest) |>
  summarise(nb = n()) |>
  ungroup() |>
  mutate(pourcentage = nb / sum(nb) * 100)
```