B01-1

R資料處理方法(I)

資料處理與篩選:
R內鍵指令及tidyverse系列

吳漢銘

國立政治大學 統計學系

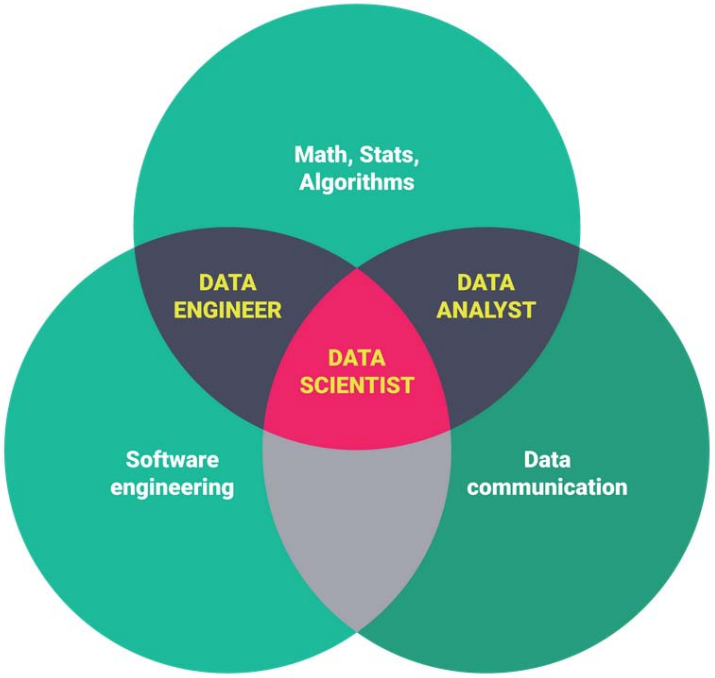
http://www.hmwu.idv.tw

本章大綱&學習目標

- 了解資料調處(處理) (Data Manipulation)的概念。
- 了解及運用表格處理函式: **rbind {base}, cbind {base}, table {base}, xtabs {stats}, expand.table {epitools}, tabulate {base}, ftable {stats}, xtable {xtable}, stack {utils}** .
- 了解及運用資料調處相關函式: **aggregate {stats}, by {base}, cut {base}, with {base}, merge {base}, split {base}.**
- 了解及運用**apply**系列於資料調處: **apply, tapply, lapply, sapply, mapply, rapply.**
- 了解及運用資料調處R套件: **plyr, dplyr, tidyr, reshape2, data.table.**

資料處理 (Data Manipulation)

- Some Terms:
  - Data Cleaning: https://en.wikipedia.org/wiki/Data_cleansing
  - Data Integration: https://en.wikipedia.org/wiki/Data_integration
  - Data Manipulation, Data Preprocessing
  - Data Munging (data wrangling): munging can mean manipulating raw data to achieve a final form. It can mean parsing or filtering data, or the many steps required for data recognition



資料調處(data manipulation)

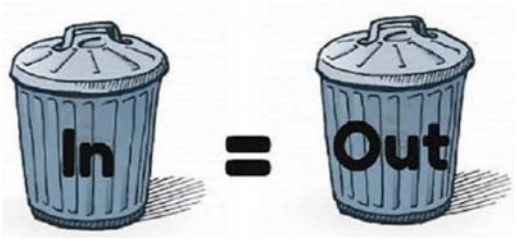
國家教育研究院雙語詞彙

Source: https://www.springboard.com/blog/data-science-career-paths-different-roles-industry/

資料清理 (Data Cleaning)

- Data cleaning is one part of data quality. Aim at:
  - **Accuracy** (data is recorded correctly)
  - **Completeness** (all relevant data is recorded)
  - **Uniqueness** (no duplicated data record)
  - **Timeliness** (the data is not old)
  - **Consistency** (the data is coherent)

GIGO



- Data cleaning attempts to fill in missing values, smooth out noise while identifying outliers, and correct inconsistencies in the data.

- Data cleaning is usually an iterative two-step process consisting of discrepancy detection and data transformation.

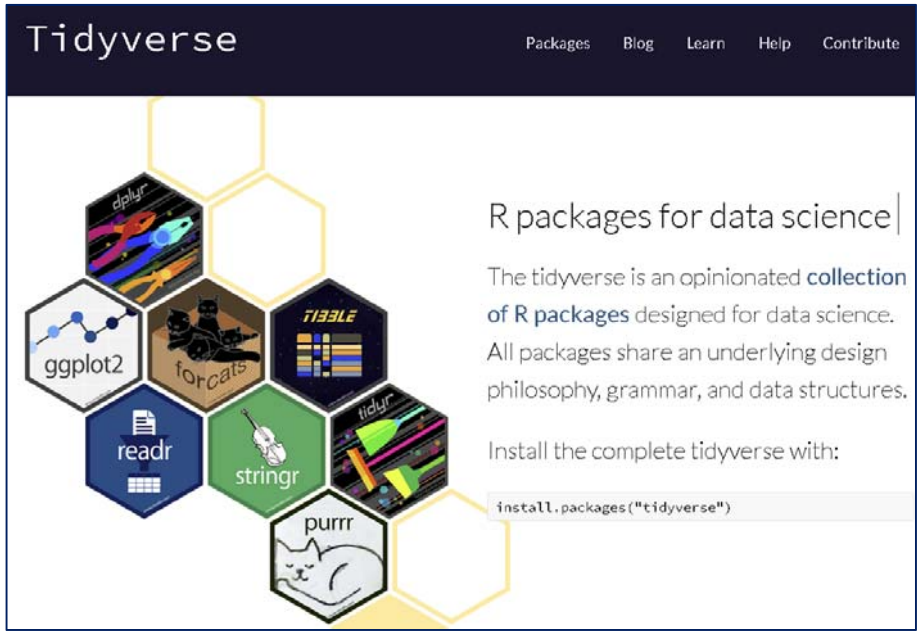
(1) Some R Functions for Data Manipulation

- R functions
  - **aggregate{stats}**: Compute Summary Statistics of Data Subsets
  - **by{base}**: Apply a Function to a Data Frame Split by Factors
  - **cut{base}**: Convert Numeric to Factor
  - **with{base}**: Evaluate an Expression in a Data Environment.
  - **merge{base}**: Merge Two Data Frames
  - **split{base}**: Divide into Groups and Reassemble

- 表格處理相關函式: **rbind{base}, cbind{base}, table{base}, xtabs{stats}, expand.table {epitools}, tabulate{base}, ftable{stats}, xtable{xtable}, stack{utils}.**

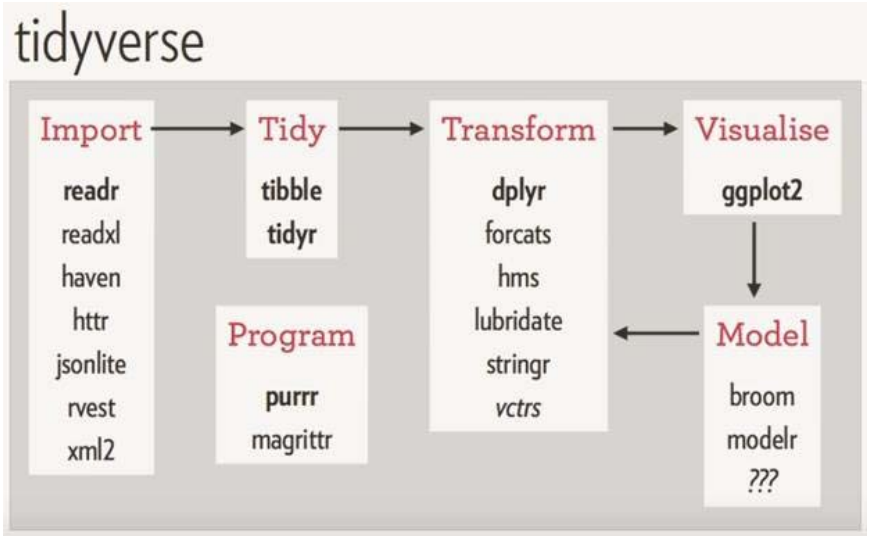
- **apply** 系列: **apply**, **tapply**, **sapply**, **lapply** , **rapply** , **mapply**.

(2) tidyverse: R packages for data science



https://www.tidyverse.org

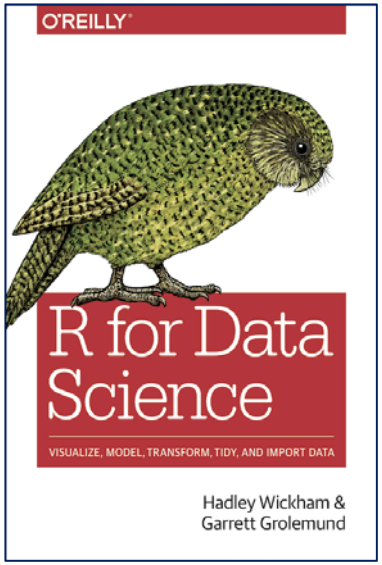
```
> install.packages("tidyverse")
> library(tidyverse)
```



- Core tidyverse: **ggplot2, dplyr, tidyr, readr, purrr, tibble, stringr, forcats**
- Import: **DBI, haven, httr, readxl, googlesheets4, googledrive, rvest, jsonlite, xml2**
- Wrangle: **lubridate, hms, blob, dbplyr, dtplyr**
- Program: **magrittr, glue**
- Model: **tidymodels**

Learn the tidyverse "R for Data Science"

Online Book (1st Edition): https://r4ds.had.co.nz/



Wrangle
- 9 Introduction
- 10 Tibbles
- 11 Data import
- 12 Tidy data
- 13 Relational data
- 14 Strings
- 15 Factors
- 16 Dates and times

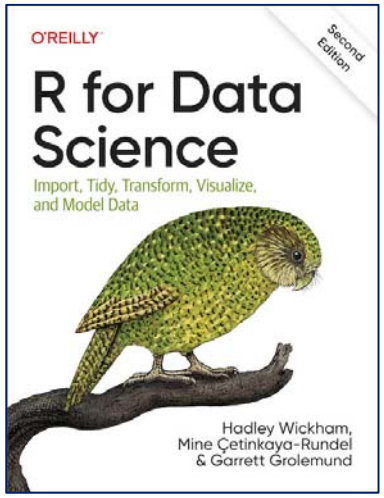
Program
- 17 Introduction
- 18 Pipes
- 19 Functions
- 20 Vectors
- 21 Iteration

Explore
- 2 Introduction
- 3 Data visualisation
- 4 Workflow: basics
- 5 Data transformation
- 6 Workflow: scripts
- 7 Exploratory Data Analysis
- 8 Workflow: projects

Model
- 22 Introduction
- 23 Model basics
- 24 Model building
- 25 Many models

Communicate
- 26 Introduction
- 27 R Markdown
- 28 Graphics for communication
- 29 R Markdown formats
- 30 R Markdown workflow

Online Book (2st Edition)(Not Finished): https://r4ds.hadley.nz/



Visualize
- 10 Layers
- 11 Exploratory data analysis
- 12 Communication

Transform
- 13 Logical vectors
- 14 Numbers
- 15 Strings
- 16 Regular expressions
- 17 Factors
- 18 Dates and times
- 19 Missing values
- 20 Joins

1 Introduction

Whole game
- 2 Data visualization
- 3 Workflow: basics
- 4 Data transformation
- 5 Workflow: code style
- 6 Data tidying
- 7 Workflow: scripts and projects
- 8 Data import
- 9 Workflow: getting help

Import
- 21 Spreadsheets
- 22 Databases
- 23 Arrow
- 24 Hierarchical data
- 25 Web scraping

Program
- 26 Functions
- 27 Iteration
- 28 A field guide to base R

Communicate
- 29 Quarto
- 30 Quarto formats

Tidy Data

- Tidy data is a standard way of mapping the meaning of a dataset to its structure.
- A dataset is messy or tidy depending on how rows, columns and tables are matched up with observations, variables and types. In tidy data:
  - Each variable forms a column.
  - Each observation forms a row.
  - Each type of observational unit forms a table. (Each value is placed in its own cell)

- Tidy data is particularly well suited for vectorised programming languages like R, because the layout ensures that values of different variables from the same observation are always paired.

messy

	treatmenta	treatmentb
John Smith	—	2
Jane Doe	16	11
Mary Johnson	3	1

	John Smith	Jane Doe	Mary Johnson
treatmenta	—	16	3
treatmentb	2	11	1

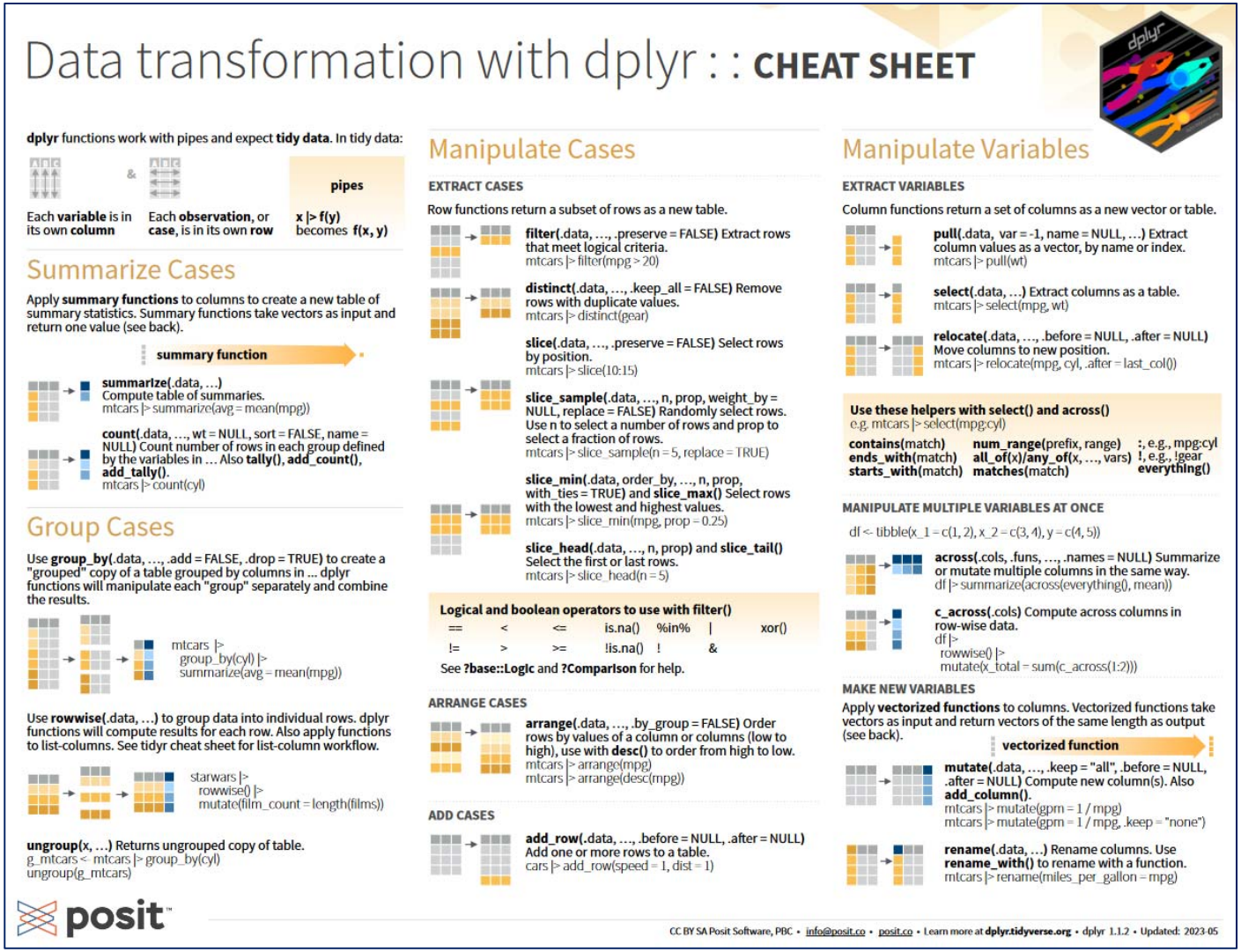
tidy

name	trt	result
John Smith	a	—
Jane Doe	a	16
Mary Johnson	a	3
John Smith	b	2
Jane Doe	b	11
Mary Johnson	b	1

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Data Transformation with dplyr Cheat Sheet by Posit (Former RStudio)

https://raw.githubusercontent.com/rstudio/cheatsheets/main/data-transformation.pdf



Data transformation with dplyr : : CHEAT SHEET

dplyr functions work with pipes and expect **tidy data**. In tidy data:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own **row**

pipes
x |> f(y) becomes **f(x, y)**

Summarize Cases

Apply **summary functions** to columns to create a new table of summary statistics. Summary functions take vectors as input and return one value (see back).

summary function

summarize(.data, …)
Compute table of summaries.
mtcars |> summarize(avg = mean(mpg))

count(.data, …, wt = NULL, sort = FALSE, name = NULL) Count number of rows in each group defined by the variables in … Also **tally()**, **add_count()**, **add_tally()**.
mtcars |> count(cyl)

Group Cases

Use **group_by**(.data, …, .add = FALSE, .drop = TRUE) to create a "grouped" copy of a table grouped by columns in … dplyr functions will manipulate each "group" separately and combine the results.

```
mtcars |>
  group_by(cyl) |>
  summarize(avg = mean(mpg))
```

Use **rowwise**(.data, …) to group data into individual rows. dplyr functions will compute results for each row. Also apply functions to list-columns. See tidyr cheat sheet for list-column workflow.

```
starwars |>
  rowwise() |>
  mutate(film_count = length(films))
```

ungroup(x, …) Returns ungrouped copy of table.

```
g_mtcars <- mtcars |> group_by(cyl)
ungroup(g_mtcars)
```

Manipulate Cases

EXTRACT CASES

Row functions return a subset of rows as a new table.

filter(.data, …, .preserve = FALSE) Extract rows that meet logical criteria.
mtcars |> filter(mpg > 20)

distinct(.data, …, .keep_all = FALSE) Remove rows with duplicate values.
mtcars |> distinct(gear)

slice(.data, …, .preserve = FALSE) Select rows by position.
mtcars |> slice(10:15)

slice_sample(.data, …, n, prop, weight_by = NULL, replace = FALSE) Randomly select rows. Use n to select a number of rows and prop to select a fraction of rows.
mtcars |> slice_sample(n = 5, replace = TRUE)

slice_min(.data, order_by, …, n, prop, with_ties = TRUE) and **slice_max()** Select rows with the lowest and highest values.
mtcars |> slice_min(mpg, prop = 0.25)

slice_head(.data, …, n, prop) and **slice_tail()** Select the first or last rows.
mtcars |> slice_head(n = 5)

Logical and boolean operators to use with filter()

==	<	<=	is.na()	%in%	|	xor()
!=	>	>=	!is.na()	!	&	

See **?base::Logic** and **?Comparison** for help.

ARRANGE CASES

arrange(.data, …, .by_group = FALSE) Order rows by values of a column or columns (low to high), use with **desc()** to order from high to low.
mtcars |> arrange(mpg)
mtcars |> arrange(desc(mpg))

ADD CASES

add_row(.data, …, .before = NULL, .after = NULL) Add one or more rows to a table.
cars |> add_row(speed = 1, dist = 1)

Manipulate Variables

EXTRACT VARIABLES

Column functions return a set of columns as a new vector or table.

pull(.data, var = -1, name = NULL, …) Extract column values as a vector, by name or index.
mtcars |> pull(wt)

select(.data, …) Extract columns as a table.
mtcars |> select(mpg, wt)

relocate(.data, …, .before = NULL, .after = NULL) Move columns to new position.
mtcars |> relocate(mpg, cyl, .after = last_col())

Use these helpers with select() and across()
e.g. mtcars |> select(mpg:cyl)

contains(match)
ends_with(match)
starts_with(match)
num_range(prefix, range)
all_of(x)/**any_of**(x, …, vars)
matches(match)
:, e.g., mpg:cyl
!, e.g., !gear
everything()

MANIPULATE MULTIPLE VARIABLES AT ONCE

```
df <- tibble(x_1 = c(1, 2), x_2 = c(3, 4), y = c(4, 5))
```

across(.cols, .funs, …, .names = NULL) Summarize or mutate multiple columns in the same way.
df |> summarize(across(everything(), mean))

c_across(.cols) Compute across columns in row-wise data.

```
df |>
  rowwise() |>
  mutate(x_total = sum(c_across(1:2)))
```

MAKE NEW VARIABLES

Apply **vectorized functions** to columns. Vectorized functions take vectors as input and return vectors of the same length as output (see back).

vectorized function

mutate(.data, …, .keep = "all", .before = NULL, .after = NULL) Compute new column(s). Also **add_column()**.
mtcars |> mutate(gpm = 1 / mpg)
mtcars |> mutate(gpm = 1 / mpg, .keep = "none")

rename(.data, …) Rename columns. Use **rename_with()** to rename with a function.
mtcars |> rename(miles_per_gallon = mpg)

posit

CC BY SA Posit Software, PBC • info@posit.co • posit.co • Learn more at **dplyr.tidyverse.org** • dplyr 1.1.2 • Updated: 2023-05

Data tidying with tidyr
Cheat Sheet by Posit (Former RStudio)

https://raw.githubusercontent.com/rstudio/cheatsheets/main/tidyr.pdf

Data tidying with tidyr : : CHEAT SHEET

Tidy data is a way to organize tabular data in a consistent data structure across packages.
A table is tidy if:

Each **variable** is in its own **column** & Each **observation**, or **case**, is in its own row

Access **variables** as **vectors** — Preserve **cases** in vectorized operations

Tibbles

AN ENHANCED DATA FRAME

Tibbles are a table format provided by the **tibble** package. They inherit the data frame class, but have improved behaviors:

- **Subset** a new tibble with], a vector with [[and $.
- **No partial matching** when subsetting columns.
- **Display** concise views of the data on one screen.

options(tibble.print_max = n, tibble.print_min = m, tibble.width = Inf) Control default display settings.

View() or **glimpse()** View the entire data set.

CONSTRUCT A TIBBLE

tibble(…) Construct by columns.
tibble(x = 1:3, y = c("a", "b", "c"))

tribble(…) Construct by rows.
tribble(~x, ~y,
1, "a",
2, "b",
3, "c")

Both make this tibble

```
A tibble: 3 × 2
      x     y
  <int> <chr>
1     1     a
2     2     b
3     3     c
```

as_tibble(x, …) Convert a data frame to a tibble.

enframe(x, name = "name", value = "value") Convert a named vector to a tibble. Also **deframe**().

is_tibble(x) Test whether x is a tibble.

Reshape Data - Pivot data to reorganize values into a new layout.

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE)

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

pivot_longer(table4a, cols = 2:3, names_to = "year", values_to = "cases")

table2

country	year	type	count
A	1999	cases	0.7K
A	1999	pop	19M
A	2000	cases	2K
A	2000	pop	20M
B	1999	cases	37K
B	1999	pop	172M
B	2000	cases	80K
B	2000	pop	174M
C	1999	cases	212K
C	1999	pop	1T
C	2000	cases	213K
C	2000	pop	1T

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172M
B	2000	80K	174M
C	1999	212K	1T
C	2000	213K	1T

pivot_wider(data, names_from = "name", values_from = "value")

The inverse of pivot_longer(). "Widen" data by expanding two columns into several. One column provides the new column names, the other the values.

pivot_wider(table2, names_from = type, values_from = count)

Split Cells - Use these functions to split or combine cells into individual, isolated values.

table5

country	century	year
A	19	99
A	20	00
B	19	99
B	20	00

country	year
A	1999
A	2000
B	1999
B	2000

unite(data, col, …, sep = "_", remove = TRUE, na.rm = FALSE) Collapse cells across several columns into a single column.

unite(table5, century, year, col = "year", sep = "")

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	cases	pop
A	1999	0.7K	19M
A	2000	2K	20M
B	1999	37K	172
B	2000	80K	174

separate(data, col, into, sep = "[^[:alnum:]]+", remove = TRUE, convert = FALSE, extra = "warn", fill = "warn", …) Separate each cell in a column into several columns. Also **extract**().

separate(table3, rate, sep = "/", into = c("cases", "pop"))

table3

country	year	rate
A	1999	0.7K/19M
A	2000	2K/20M
B	1999	37K/172M
B	2000	80K/174M

country	year	rate
A	1999	0.7K
A	1999	19M
A	2000	2K
A	2000	20M
B	1999	37K
B	1999	172M
B	2000	80K
B	2000	174M

separate_rows(data, …, sep = "[^[:alnum:].]+", convert = FALSE) Separate each cell in a column into several rows.

separate_rows(table3, rate, sep = "/")

Expand Tables

Create new combinations of variables or identify implicit missing values (combinations of variables not present in the data).

expand(data, …) Create a new tibble with all possible combinations of the values of the variables listed in … Drop other variables.
expand(mtcars, cyl, gear, carb)

complete(data, …, fill = list()) Add missing possible combinations of values of variables listed in … Fill remaining variables with NA.
complete(mtcars, cyl, gear, carb)

Handle Missing Values

Drop or replace explicit missing values (NA).

drop_na(data, …) Drop rows containing NA's in … columns.
drop_na(x, x2)

fill(data, …, .direction = "down") Fill in NA's in … columns using the next or previous value.
fill(x, x2)

replace_na(data, replace) Specify a value to replace NA in selected columns.
replace_na(x, list(x2 = 2))

posit

CC BY SA Posit Software, PBC • Info@posit.co • posit.co • Learn more at tidyr.tidyverse.org • tibble 3.2.1 • tidyr 1.3.0 • Updated: 2023-05

(3) data.table: Extension of 'data.frame'

https://cran.r-project.org/web/packages/data.table/index.html

data.table: Extension of 'data.frame'

Fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete of columns by group using no copies at all, list columns, friendly and fast character-separated-value read/write. Offers a natural and flexible syntax, for faster development.

Version:	1.14.8
Depends:	R (≥ 3.1.0)
Imports:	methods
Suggests:	bit64 (≥ 4.0.0), bit (≥ 4.0.4), curl, R.utils, xts, nanotim zoo (≥ 1.8-1), yaml, knitr, rmarkdown
Published:	2023-02-17

Documentation:

Reference manual:	data.table.pdf
Vignettes:	Benchmarking data.table Frequently asked questions Importing data.table Introduction to data.table Keys and fast binary search based subset Reference semantics Efficient reshaping using data.tables Using .SD for Data Analysis Secondary indices and auto indexing

Introduction to data.table

2023-02-16

This vignette introduces the `data.table` syntax, its general form, how to *subset* rows, *select and compute* on columns, and perform aggregations *by group*. Familiarity with `data.frame` data structure from base R is useful, but not essential to follow this vignette.

Data analysis using `data.table`

Data manipulation operations such as *subset*, *group*, *update*, *join* etc., are all inherently related. Keeping these *related operations together* allows for:

- *concise* and *consistent* syntax irrespective of the set of operations you would like to perform to achieve your end goal.
- performing analysis *fluidly* without the cognitive burden of having to map each operation to a particular function from a potentially huge set of functions available before performing the analysis.
- *automatically* optimising operations internally, and very effectively, by knowing precisely the data required for each operation, leading to very fast and memory efficient code.

Briefly, if you are interested in reducing *programming* and *compute* time tremendously, then this package is for you. The philosophy that `data.table` adheres to makes this possible. Our goal is to illustrate it through this series of vignettes.

rbind and cbind

```
> begin.experiment <- data.frame(name=c("A", "B", "C", "D", "E", "F"),
+ weights=c(270, 263, 294, 218, 305, 261))
> middle.experiment <- data.frame(name=c("G", "H", "I"),
+ weights=c(169, 181, 201))
> end.experiment <- data.frame(name=c("C", "D", "A", "H", "I"),
+ weights=c(107, 104, 104, 102, 100))
> # merge the data for those who started and finished the experiment
> (common <- intersect(begin.experiment$name, end.experiment$name))
[1] "A" "C" "D"
> (b.at <- is.element(begin.experiment$name, common))
[1]  TRUE FALSE  TRUE  TRUE FALSE FALSE
> (e.at <- is.element(end.experiment$name, common))
[1]  TRUE  TRUE  TRUE FALSE FALSE
> experiment <- rbind(cbind(begin.experiment[b.at,], time="begin"),
+                     cbind(end.experiment[e.at,], time="end"))
> experiment
   name weights  time
1     A     270 begin
3     C     294 begin
4     D     218 begin
11    C     107   end
2     D     104   end
31    A     104   end
```

```
> begin.experiment
  name weights
1    A     270
2    B     263
3    C     294
4    D     218
5    E     305
6    F     261
> middle.experiment
  name weights
1    G     169
2    H     181
3    I     201
> end.experiment
  name weights
1    C     107
2    D     104
3    A     104
4    H     102
5    I     100
```

```
> tapply(experiment$weights, experiment$time, mean)
   begin      end
260.6667 105.0000
```

table {base}:
Cross Tabulation and Table Creation

Description: table uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

Usage:
```
table(...,
      exclude = if (useNA == "no") c(NA, NaN),
      useNA = c("no", "ifany", "always"),
      dnn = list.names(...), deparse.level = 1)
```

table {base}系列函數:
tabulate {base}、
ftable {stats}、
xtabs {stats}、
xtable {xtable}

```
> set.seed(12345)
> grade <- as.factor(sample(c("大一", "大二", "大三", "大四"), 50, replace=T))
> bloodtype <- as.factor(sample(c("A","AB","B","O"), 50, replace=T))
> record <- data.frame(grade, bloodtype)
> head(record)
  grade bloodtype
1  大三         O
2  大四         O
3  大四        AB
4  大四         A
5  大二         B
6  大一        AB
> record.t <- table(record)
> record.t
      bloodtype
grade  A AB B O
  大一 3  2 5 2
  大二 4  1 7 3
  大三 3  3 1 4
  大四 3  2 2 5
```

```
> as.data.frame(record.t)
   grade bloodtype Freq
1   大一         A    3
2   大二         A    4
3   大三         A    3
4   大四         A    3
5   大一        AB    2
6   大二        AB    1
7   大三        AB    3
8   大四        AB    2
9   大一         B    5
10  大二         B    7
11  大三         B    1
12  大四         B    2
13  大一         O    2
14  大二         O    3
15  大三         O    4
16  大四         O    5
```

Cross-tabulated data can be produced from raw data using **table**.

table {base}:
Cross Tabulation and Table Creation

```
> margin.table(record.t, 1)
grade
大一 大二 大三 大四
  12   15   11   12
> margin.table(record.t, 2)
bloodtype
 A AB  B  O
13  8 15 14
>
> colSums(record.t)
 A AB  B  O
13  8 15 14
> rowSums(record.t)
大一 大二 大三 大四
  12   15   11   12
>
> colMeans(record.t)
   A   AB    B    O
3.25 2.00 3.75 3.50
> rowMeans(record.t)
大一 大二 大三 大四
3.00 3.75 2.75 3.00
```

```
> prop.table(record.t)
      bloodtype
grade     A   AB    B    O
  大一 0.06 0.04 0.10 0.04
  大二 0.08 0.02 0.14 0.06
  大三 0.06 0.06 0.02 0.08
  大四 0.06 0.04 0.04 0.10
> prop.table(record.t, margin=1) # row margin
      bloodtype
grade          A         AB          B          O
  大一 0.25000000 0.16666667 0.41666667 0.16666667
  大二 0.26666667 0.06666667 0.46666667 0.20000000
  大三 0.27272727 0.27272727 0.09090909 0.36363636
  大四 0.25000000 0.16666667 0.16666667 0.41666667
> prop.table(record.t, margin=2) # column  margin
      bloodtype
grade          A         AB          B          O
  大一 0.23076923 0.25000000 0.33333333 0.14285714
  大二 0.30769231 0.12500000 0.46666667 0.21428571
  大三 0.23076923 0.37500000 0.06666667 0.28571429
  大四 0.23076923 0.25000000 0.13333333 0.35714286
```

tabulate {base}: Tabulation for Vectors

Description: tabulate takes the integer-valued vector bin and counts the number of times each integer occurs in it.

Usage: `tabulate(bin, nbins = max(1, bin, na.rm = TRUE))`

```
> set.seed(12345)
> (x <- sample(1:10, 5, replace=T))
[1] 8 9 8 9 5
> (y <- tabulate(x))
[1] 0 0 0 0 1 0 0 2 2
> names(y) <- as.character(1:max(x))
> y
1 2 3 4 5 6 7 8 9
0 0 0 0 1 0 0 2 2
```

xtabs {stats}: Cross Tabulation

Description: Create a contingency table (optionally a sparse matrix) from cross-classifying factors, usually contained in a data frame, using a formula interface.

Usage:
```
xtabs(formula = ~., data = parent.frame(), subset, sparse = FALSE,
      na.action, exclude = c(NA, NaN), drop.unused.levels = FALSE)
```

```
> Titanic
, , Age = Child, Survived = No

      Sex
Class  Male Female
  1st     0      0
  2nd     0      0
  3rd    35     17
  Crew    0      0

...
, , Age = Adult, Survived = Yes

      Sex
Class  Male Female
  1st    57    140
  2nd    14     80
  3rd    75     76
  Crew  192     20
```

Cross-tabulated data can be produced from aggregate data using **xtabs**.

```
> Titanic.df <- as.data.frame(Titanic)
> Titanic.df
   Class    Sex   Age Survived Freq
1    1st   Male Child       No    0
2    2nd   Male Child       No    0
3    3rd   Male Child       No   35
4   Crew   Male Child       No    0
5    1st Female Child       No    0
...
31   3rd Female Adult      Yes   76
32  Crew Female Adult      Yes   20
```

```
> xtabs(Freq ~ Sex + Age, data = Titanic.df)
        Age
Sex      Child Adult
  Male      64  1667
  Female    45   425
> xtabs(Freq ~ Sex + Age, data = Titanic.df,
+       subset = Class %in% c("1st", "2nd"))
        Age
Sex      Child Adult
  Male      16   343
  Female    14   237
```

樞紐分析

Item,	Date,	Count
A,	20170328,	2
A,	20170329,	6
A,	20170330,	4
A,	20170329,	9
B,	20170331,	6
C,	20170329,	1
C,	20170330,	7
C,	20170331,	0

```
> sale <- read.table("itemsale.csv", sep=",", header=T)
> sale
  Item     Date Count
1    A 20170328     2
2    A 20170329     6
3    A 20170330     4
4    A 20170329     9
5    B 20170331     6
6    C 20170329     1
7    C 20170330     7
8    C 20170331     0
> attach(sale)
> tb <- xtabs(Count ~ Item + Date)
> rbind(cbind(tb, row.total=margin.table(tb, 1)), col.total=c(margin.table(tb, 2), sum(tb)))
          20170328 20170329 20170330 20170331 row.total
A                2       15        4        0        21
B                0        0        0        6         6
C                0        1        7        0         8
col.total        2       16       11        6        35
> detach(sale)
```

See also: http://www.cookbook-r.com/Manipulating_data/Converting_between_data_frames_and_contingency_tables/

expand.table {epitools}

Expand contingency table into individual-level data set

```
> survey <- array(0, dim=c(3, 2, 1))
> survey[,1,1] <- c(2, 0, 1)
> survey[,2,1] <- c(3, 2, 4)
> Satisfactory <- c("Good", "Fair", "Bad")
> Sex <- c("Female", "Male")
> Times <- c("First")
> dimnames(survey) <- list(Satisfactory, Sex, Times)
> names(dimnames(survey)) <- c("Satisfactory", "Sex", "Times")
> survey
, , Times = First

             Sex
Satisfactory Female Male
        Good      2    3
        Fair      0    2
        Bad       1    4
```

```
> library(epitools)
> (survey.ex <- expand.table(survey))
   Satisfactory    Sex Times
1          Good Female First
2          Good Female First
3          Good   Male First
4          Good   Male First
5          Good   Male First
6          Fair   Male First
7          Fair   Male First
8           Bad Female First
9           Bad   Male First
10          Bad   Male First
11          Bad   Male First
12          Bad   Male First
```

expand.table {epitools}: Expand contingency table into individual-level data set

- **Usage**: `expand.table(x)`
- **Arguments**: `x`: table or array with `dimnames(x)` and `names(dimnames(x))`

課堂練習: expand.table

```
> data(HairEyeColor)
> HairEyeColor
, , Sex = Male

       Eye
Hair    Brown Blue Hazel Green
  Black    32   11    10     3
  Brown    53   50    25    15
  Red      10   10     7     7
  Blond     3   30     5     8

, , Sex = Female

       Eye
Hair    Brown Blue Hazel Green
  Black    36    9     5     2
  Brown    66   34    29    14
  Red      16    7     7     7
  Blond     4   64     5     8
```

```
> # Convert into individual-level data frame
> HairEyeColor.ex <- expand.table(HairEyeColor)
> HairEyeColor.ex
    Hair   Eye  Sex
1  Black Brown Male
2  Black Brown Male
3  Black Brown Male
...
589 Blond Green Female
590 Blond Green Female
591 Blond Green Female
592 Blond Green Female
```

32 items

```
> # Convert into group-level data frame
> as.data.frame(HairEyeColor)
    Hair   Eye    Sex Freq
1  Black Brown   Male   32
2  Brown Brown   Male   53
3    Red Brown   Male   10
...
30 Brown Green Female   14
31   Red Green Female    7
32 Blond Green Female    8
```

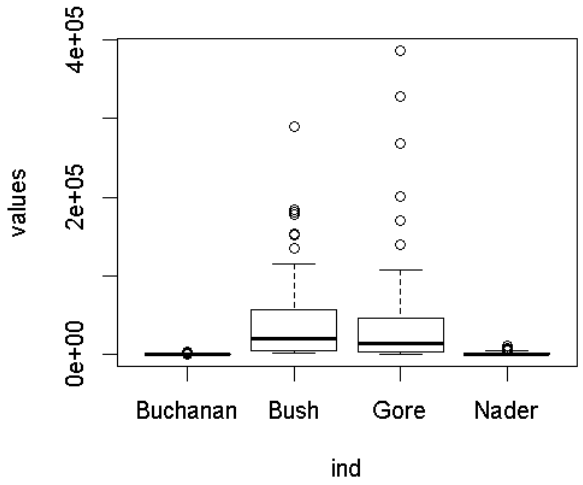
stack {utils}

Stack or Unstack Vectors from a Data Frame or List

- Stacking vectors concatenates multiple vectors into a single vector along with a factor indicating where each observation originated.
- Unstacking reverses this operation.

```
> elections <- read.csv('elections-2000.csv')
> elections
          County   Gore   Bush Buchanan Nader
1       ALACHUA  47365  34124      263  3226
2         BAKER   2392   5610       73    53
3           BAY  18850  38637      248   828
...
65      WAKULLA   3838   4512       46   149
66       WALTON   5642  12182      120   265
67   WASHINGTON   2798   4994       88    93
>
> elections.stacked <- cbind(stack(elections[,-1]),
+   county = elections$County)
> elections.stacked
    values      ind       county
1    47365     Gore      ALACHUA
2     2392     Gore        BAKER
3    18850     Gore          BAY
...
266    149    Nader      WAKULLA
267    265    Nader       WALTON
268     93    Nader   WASHINGTON
```

```
plot(elections.stacked[, c(2, 1)])
boxplot(elections[,-1])
```



Stack Character Vectors

```
> mydata <- data.frame(Area1=c("A", "B", "B", "C"), Area2=c("A", "D", "E", "B"))
> rownames(mydata) <- paste("rater", 1:4, sep="-")
> mydata
        Area1 Area2
rater-1     A     A
rater-2     B     D
rater-3     B     E
rater-4     C     B
>
> stack(mydata)
Error in stack.data.frame(mydata) : no vector columns were selected
> mydata.stack <- stack(lapply(mydata, as.character))
> colnames(mydata.stack) <-  c("Rate", "Area")
> mydata.stack
  Rate  Area
1    A Area1
2    B Area1
3    B Area1
4    C Area1
5    A Area2
6    D Area2
7    E Area2
8    B Area2
```

aggregate {stats}:
Compute Summary Statistics of Data Subsets

Usage: `aggregate(x, by, FUN, ..., simplify = TRUE)`

```
> head(state.x77)
           Population Income Illiteracy Life Exp Murder HS Grad Frost   Area
Alabama          3615   3624        2.1    69.05   15.1    41.3    20  50708
Alaska            365   6315        1.5    69.31   11.3    66.7   152 566432
Arizona          2212   4530        1.8    70.55    7.8    58.1    15 113417
Arkansas         2110   3378        1.9    70.66   10.1    39.9    65  51945
California      21198   5114        1.1    71.71   10.3    62.6    20 156361
Colorado         2541   4884        0.7    72.06    6.8    63.9   166 103766
> dim(state.x77)
[1] 50  8
> state.region
 [1] South         West          West          South         West
 [6] West          Northeast     South         South         South
...
[46] South         West          South         North Central West
Levels: Northeast South North Central West
> aggregate(state.x77, list(Region = state.region), mean)
         Region Population   Income Illiteracy Life Exp    Murder  HS Grad    Frost      Area
1     Northeast   5495.111 4570.222   1.000000 71.26444  4.722222 53.96667 132.7778  18141.00
2         South   4208.125 4011.938   1.737500 69.70625 10.581250 44.34375  64.6250  54605.12
3 North Central   4803.000 4611.083   0.700000 71.76667  5.275000 54.51667 138.8333  62652.00
4          West   2915.308 4702.615   1.023077 71.23462  7.215385 62.00000 102.1538 134463.00
```

tapply 可回傳list，aggregate僅能傳回向量、矩陣。

state{datasets}: US State Facts and Figures, Data sets related to the 50 states of the United States of America: **state.abb, state.area, state.center, state.division, state.name, state.region, state.x77**

http://127.0.0.1:11812/library/datasets/html/state.html

aggregate {stats},
Customized Statistics

```
> ## Compute the averages according to region and the occurrence of more
> ## than 130 days of frost.
> aggregate(state.x77,
+           by = list(Region = state.region,
+                Cold = state.x77[,"Frost"] > 130),
+           FUN = function(x){round(mean(x), 2)})
         Region  Cold Population  Income Illiteracy Life Exp Murder HS Grad  Frost      Area
1     Northeast FALSE    8802.80 4780.40       1.18    71.13   5.58   52.06 110.60  21838.60
2         South FALSE    4208.12 4011.94       1.74    69.71  10.58   44.34  64.62  54605.12
3 North Central FALSE    7233.83 4633.33       0.78    70.96   8.28   53.37 120.00  56736.50
4          West FALSE    4582.57 4550.14       1.26    71.70   6.83   60.11  51.00  91863.71
5     Northeast  TRUE    1360.50 4307.50       0.78    71.44   3.65   56.35 160.50  13519.00
6 North Central  TRUE    2372.17 4588.83       0.62    72.58   2.27   55.67 157.67  68567.50
7          West  TRUE     970.17 4880.50       0.75    70.69   7.67   64.20 161.83 184162.17
```

```
> aggregate(state.x77,
+           by = list(Region = state.region,
+           Cold = state.x77[,"Frost"] > 130),
+           FUN = function(x){ round(sqrt(sum(x^2)), 2)})
         Region  Cold Population   Income Illiteracy Life Exp Murder HS Grad  Frost      Area
1     Northeast FALSE   23576.34 10707.22       2.66   159.05  14.13  116.74 250.13  66548.16
2         South FALSE   19980.44 16218.16       7.27   278.85  43.53  178.76 285.52 313220.52
3 North Central FALSE   19487.79 11367.76       1.94   173.82  21.00  130.97 294.40 144108.91
4          West FALSE   21791.15 12108.37       3.74   189.72  19.07  159.20 184.01 269467.52
5     Northeast  TRUE    3407.56  8708.52       1.60   142.88   7.62  112.72 322.09  33868.12
6 North Central  TRUE    6918.96 11260.58       1.53   177.78   5.73  136.59 388.46 169705.71
7          West  TRUE    3013.82 12089.92       2.02   173.19  19.98  157.40 398.41 617310.24
```

aggregate {stats},
FUN with Arguments

```
> # Compute the average annual approval ratings for American presidents.
> presidents
     Qtr1 Qtr2 Qtr3 Qtr4
1945   NA   87   82   75
1946   63   50   43   32
1947   35   60   54   55
...
1974   28   25   24   24
> aggregate(presidents, nfrequency = 1, FUN = mean)
Time Series:
Start = 1945
End = 1974
Frequency = 1
 [1]    NA 47.00 51.00    NA 58.50 41.75 28.75    NA 67.00 65.00 72.75 72.25 65.25 52.25
[15] 61.50 62.75 76.00 71.50 64.75 72.75 66.50 52.25 45.00 41.00 61.25 58.00 50.50    NA
[29] 44.75 25.25
> # Give the summer less weight.
> aggregate(presidents, nfrequency = 1, FUN = weighted.mean, w = c(1, 1, 0.5, 1))
Time Series:
Start = 1945
End = 1974
Frequency = 1
 [1]       NA 47.57143 50.57143       NA 58.71429 41.14286 28.28571       NA 65.85714
[10] 64.14286 71.85714 73.00000 65.57143 52.85714 61.57143 63.00000 76.71429 72.85714
[19] 65.14286 73.28571 66.14286 51.71429 46.00000 41.85714 60.71429 57.57143 50.00000
[28]       NA 45.42857 25.42857
```

nfrequency: new number of observations per unit of time; must be a divisor of the frequency of x.

aggregate {stats},
Example with Character Variables and NAs

```
> testDF <- data.frame(v1 = c(1,3,5,7,8,3,5,NA,4,5,7,9),
+                      v2 = c(11,33,55,77,88,33,55,NA,44,55,77,99))
> by1 <- c("red", "blue", 1, 2, NA, "big", 1, 2, "red", 1, NA, 12)
> by2 <- c("wet", "dry", 99, 95, NA, "damp", 95, 99, "red", 99, NA, NA)
> aggregate(x = testDF, by = list(by1, by2), FUN = "mean")
  Group.1 Group.2 v1 v2
1       1      95  5 55
2       2      95  7 77
3       1      99  5 55
4       2      99 NA NA
5     big    damp  3 33
6    blue     dry  3 33
7     red     red  4 44
8     red     wet  1 11
> # Treat NAs as a group
> fby1 <- factor(by1, exclude = "")
> fby2 <- factor(by2, exclude = "")
> aggregate(x = testDF, by = list(fby1, fby2), FUN = "mean")
   Group.1 Group.2  v1   v2
1        1      95 5.0 55.0
2        2      95 7.0 77.0
3        1      99 5.0 55.0
4        2      99  NA   NA
5      big    damp 3.0 33.0
6     blue     dry 3.0 33.0
7      red     red 4.0 44.0
8      red     wet 1.0 11.0
9       12    <NA> 9.0 99.0
10    <NA>    <NA> 7.5 82.5
```

aggregate {stats},
Formulas, one ~ one, one ~ many

```
> aggregate(weight ~ feed, data = chickwts,
mean)
       feed   weight
1    casein 323.5833
2 horsebean 160.2000
3   linseed 218.7500
4  meatmeal 276.9091
5   soybean 246.4286
6 sunflower 328.9167
```

```
> summary(chickwts)
     weight               feed
 Min.   :108.0   casein   :12
 1st Qu.:204.5   horsebean:10
 Median :258.0   linseed  :12
 Mean   :261.3   meatmeal :11
 3rd Qu.:323.5   soybean  :14
 Max.   :423.0   sunflower:12
```

```
> aggregate(breaks ~ wool + tension, data = warpbreaks, mean)
  wool tension   breaks
1    A       L 44.55556
2    B       L 28.22222
3    A       M 24.00000
4    B       M 28.77778
5    A       H 24.55556
6    B       H 18.77778
```

```
> summary(warpbreaks)
     breaks       wool   tension
 Min.   :10.00   A:27   L:18
 1st Qu.:18.25   B:27   M:18
 Median :26.00          H:18
 Mean   :28.15
 3rd Qu.:34.00
 Max.   :70.00
```

aggregate {stats},

Formulas, many ~ one, and many ~ many

```
> summary(esoph)
   agegp          alcgp          tobgp        ncases          ncontrols
 25-34:15   0-39g/day:23   0-9g/day:24   Min.   : 0.000   Min.   : 1.00
 35-44:15   40-79    :23   10-19   :24   1st Qu.: 0.000   1st Qu.: 3.00
 45-54:16   80-119   :21   20-29   :20   Median : 1.000   Median : 6.00
 55-64:16   120+     :21   30+     :20   Mean   : 2.273   Mean   :11.08
 65-74:15                                3rd Qu.: 4.000   3rd Qu.:14.00
 75+  :11                                Max.   :17.000   Max.   :60.00
>
> aggregate(cbind(ncases, ncontrols) ~ alcgp + tobgp, data = esoph, sum)
       alcgp    tobgp ncases ncontrols
1  0-39g/day 0-9g/day      9       261
2      40-79 0-9g/day     34       179
3     80-119 0-9g/day     19        61
4       120+ 0-9g/day     16        24
5  0-39g/day    10-19     10        84
6      40-79    10-19     17        85
7     80-119    10-19     19        49
8       120+    10-19     12        18
9  0-39g/day    20-29      5        42
10     40-79    20-29     15        62
11    80-119    20-29      6        16
12      120+    20-29      7        12
13 0-39g/day      30+      5        28
14     40-79      30+      9        29
15    80-119      30+      7        12
16      120+      30+     10        13
```

by {base}:
Apply a Function to a Data Frame Split by Factors

```
by(data, INDICES, FUN, ..., simplify = TRUE)
```

```
> by(iris[,1:4] , iris$Species , summary)
iris$Species: setosa
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.300   Min.   :2.300   Min.   :1.000   Min.   :0.100
 1st Qu.:4.800   1st Qu.:3.200   1st Qu.:1.400   1st Qu.:0.200
 Median :5.000   Median :3.400   Median :1.500   Median :0.200
 Mean   :5.006   Mean   :3.428   Mean   :1.462   Mean   :0.246
 3rd Qu.:5.200   3rd Qu.:3.675   3rd Qu.:1.575   3rd Qu.:0.300
 Max.   :5.800   Max.   :4.400   Max.   :1.900   Max.   :0.600
------------------------------------------------------------------------
iris$Species: versicolor
  Sepal.Length    Sepal.Width     Petal.Length   Petal.Width
 Min.   :4.900   Min.   :2.000   Min.   :3.00   Min.   :1.000
 1st Qu.:5.600   1st Qu.:2.525   1st Qu.:4.00   1st Qu.:1.200
 Median :5.900   Median :2.800   Median :4.35   Median :1.300
 Mean   :5.936   Mean   :2.770   Mean   :4.26   Mean   :1.326
 3rd Qu.:6.300   3rd Qu.:3.000   3rd Qu.:4.60   3rd Qu.:1.500
 Max.   :7.000   Max.   :3.400   Max.   :5.10   Max.   :1.800
------------------------------------------------------------------------
iris$Species: virginica
  Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
 Min.   :4.900   Min.   :2.200   Min.   :4.500   Min.   :1.400
 1st Qu.:6.225   1st Qu.:2.800   1st Qu.:5.100   1st Qu.:1.800
 Median :6.500   Median :3.000   Median :5.550   Median :2.000
 Mean   :6.588   Mean   :2.974   Mean   :5.552   Mean   :2.026
 3rd Qu.:6.900   3rd Qu.:3.175   3rd Qu.:5.875   3rd Qu.:2.300
 Max.   :7.900   Max.   :3.800   Max.   :6.900   Max.   :2.500
```

by {base}, Example

```
> by(iris[,1:4] , iris$Species , mean)
iris$Species: setosa
[1] NA
------------------------------------------------------------
iris$Species: versicolor
[1] NA
------------------------------------------------------------
iris$Species: virginica
[1] NA
Warning messages:
1: In mean.default(data[x, , drop = FALSE], ...) :
  argument is not numeric or logical: returning NA
2: In mean.default(data[x, , drop = FALSE], ...) :
  argument is not numeric or logical: returning NA
3: In mean.default(data[x, , drop = FALSE], ...) :
  argument is not numeric or logical: returning NA
```

The problem here is that the function you are applying doesn't work on a data frame. In effect you are calling something like this i.e. you are passing a data frame of 4 columns, containing the rows of the original where **`Species == "setosa"`**.

```
> mean(iris[iris$Species == "setosa", 1:4])
[1] NA
Warning message:
In mean.default(iris[iris$Species == "setosa", 1:4]) :
  argument is not numeric or logical: returning NA
```

by {base}, Example

For **by()** you need to do this variable by variable.

```
> by(iris[,1] , iris$Species , mean)
iris$Species: setosa
[1] 5.006
------------------------------------------------
iris$Species: versicolor
[1] 5.936
------------------------------------------------
iris$Species: virginica
[1] 6.588
```

Use **colMeans()** instead of **mean()** as the FUN applied.

```
> by(iris[,1:4] , iris$Species , colMeans)
iris$Species: setosa
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.006        3.428        1.462        0.246
------------------------------------------------------
iris$Species: versicolor
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.936        2.770        4.260        1.326
------------------------------------------------------
iris$Species: virginica
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       6.588        2.974        5.552        2.026
```

by {base}, Example

Write a wrapper, to **sapply()**

```
> varMean <- function(x, ...) sapply(x, mean, ...)
> by(iris[, 1:4], iris$Species, varMean)
iris$Species: setosa
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.006        3.428        1.462        0.246
------------------------------------------------------------
iris$Species: versicolor
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       5.936        2.770        4.260        1.326
------------------------------------------------------------
iris$Species: virginica
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       6.588        2.974        5.552        2.026
```

Use **aggregate()**:

```
> with(iris, aggregate(iris[,1:4], list(Species = iris$Species), FUN = mean))
     Species Sepal.Length Sepal.Width Petal.Length Petal.Width
1     setosa        5.006       3.428        1.462       0.246
2 versicolor        5.936       2.770        4.260       1.326
3  virginica        6.588       2.974        5.552       2.026
```

cut {base}: Convert Numeric to Factor

- **cut{base}** divides the range of **x** into intervals and codes the values in **x** according to which interval they fall. The leftmost interval corresponds to level one, the next leftmost to level two and so on.

```
cut(x, breaks, labels = NULL,
    include.lowest = FALSE, right = TRUE, dig.lab = 3, ordered_result = FALSE, ...)
```

```
> x <- rnorm(50)
> (x.cut1 <- cut(x, breaks = -5:5))
 [1] (-1,0]  (-2,-1] (-2,-1] (-1,0]  (-1,0]  (-2,-1] (0,1]   (0,1]   (-1,0]  (1,2]   (0,1]
...
[45] (1,2]   (0,1]   (-1,0]  (-2,-1] (0,1]   (0,1]
Levels: (-5,-4] (-4,-3] (-3,-2] (-2,-1] (-1,0] (0,1] (1,2] (2,3] (3,4] (4,5]
> table(x.cut1)
x.cut1
(-5,-4] (-4,-3] (-3,-2] (-2,-1]  (-1,0]   (0,1]   (1,2]   (2,3]   (3,4]   (4,5]
      0       0       1      10      18      13       8       0       0       0
> (x.cut2 <- cut(x, breaks = -5:5, labels = FALSE))
 [1] 5 4 4 5 5 4 6 6 5 7 6 5 7 5 7 6 4 7 7 4 5 6 5 5 5 6 5 6 5 4 7 ...
[47] 5 4 6 6
> table(x.cut2)
x.cut2
 3  4  5  6  7
 1 10 18 13  8
> hist(x, breaks = -5:5, plot = FALSE)$counts
 [1]  0  0  1 10 18 13  8  0  0  0
```

cut {base} Examples

```
> #the outer limits are moved away by 0.1% of the range
> cut(0:10, 5)
 [1] (-0.01,2] (-0.01,2] (-0.01,2] (2,4]     (2,4]     (4,6]     (4,6]     (6,8]
 [9] (6,8]     (8,10]    (8,10]
Levels: (-0.01,2] (2,4] (4,6] (6,8] (8,10]
>
> age <- sample(0:80, 50, replace=T)
> summary(age)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   1.00   21.00   35.00   38.16   52.75   80.00
> cut(age, 5)
 [1] (48.4,64.2]  (16.8,32.6]  (16.8,32.6]  (48.4,64.2]  (16.8,32.6]  (32.6,48.4]
...
[49] (16.8,32.6]  (48.4,64.2]
Levels: (0.921,16.8] (16.8,32.6] (32.6,48.4] (48.4,64.2] (64.2,80.1]
> mygroup <- c(0, 15, 20, 50, 60, 80)
> (x.cut <- cut(age, mygroup))
 [1] (50,60] (20,50] (15,20] (20,50] (20,50] (20,50] (15,20] (0,15]  (0,15]  (60,80]
...
Levels: (0,15] (15,20] (20,50] (50,60] (60,80]
> table(x.cut)
x.cut
 (0,15] (15,20] (20,50] (50,60] (60,80]
      7       5      22       8       8
```

Note: Instead of `table(cut(x, br))`, `hist(x, br, plot = FALSE)` is more efficient and less memory hungry. Instead of `cut(*, labels = FALSE)`, `findInterval()` is more efficient.

with {base}:

Evaluate an Expression in a Data Environment

```
with(data, expr, ...)
```

```
> with(iris, {
+     iris.lm <- lm(Sepal.Length ~ Petal.Length)
+     summary(iris.lm)})

Call:
lm(formula = Sepal.Length ~ Petal.Length)

Residuals:
     Min       1Q   Median       3Q      Max
-1.24675 -0.29657 -0.01515  0.27676  1.00269

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   4.30660    0.07839   54.94   <2e-16 ***
Petal.Length  0.40892    0.01889   21.65   <2e-16 ***
---
Signif. codes:  0  ‘***’  0.001  ‘**’  0.01  ‘*’  0.05  ‘.’  0.1  ‘ ’  1

Residual standard error: 0.4071 on 148 degrees of freedom
Multiple R-squared:   0.76,     Adjusted R-squared:  0.7583
F-statistic: 468.6 on 1 and 148 DF,  p-value: < 2.2e-16
```

People often use the **attach()** and **detach()** functions to set up "search paths" for variable names in R, but because this alters global state that's hard to keep track of, people recommend using **with()** instead, which sets up a temporary alteration to the search path for the duration of a single expression.

merge {base}: Merge Two Data Frames

- Merge (adds variables to a dataset) two data frames horizontally by common columns or row names (key variables, either string or numeric). , or do other versions of database join operations.

```
merge(x, y, by = intersect(names(x), names(y)),
      by.x = by, by.y = by, all = FALSE, all.x = all, all.y = all,
      sort = TRUE, suffixes = c(".x",".y"),
      incomparables = NULL, ...)
```

```
# merge two data frames by ID
total <- merge(data.frame.A, data.frame.B, by="ID")

# merge two data frames by ID and Country
total <- merge(data.frame.A, data.frame.B, by=c("ID","Country"))
```

https://stat.ethz.ch/R-manual/R-devel/library/base/html/merge.html

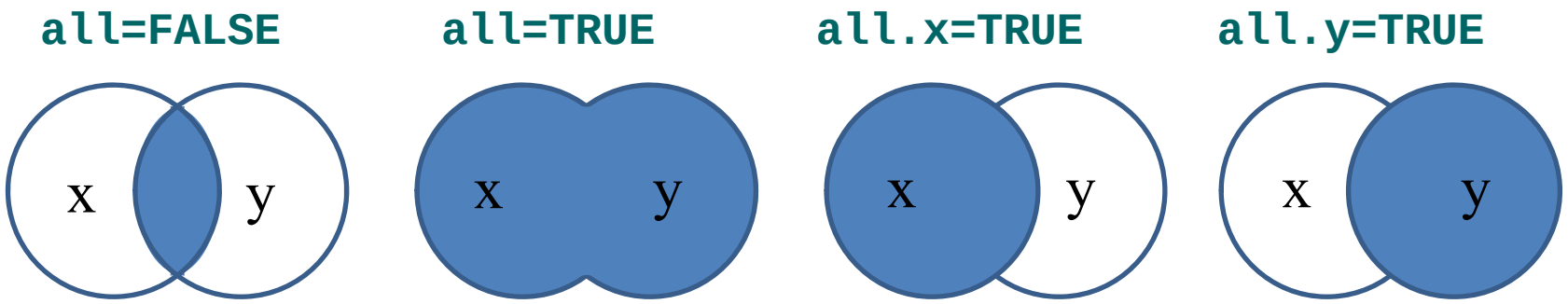
merge {base}, 合併的準則

- **merge{base}**合併的準則:
  - (default) the data frames are merged on the columns with names they both have.
  - The rows in the two data frames that match on the specified columns are extracted, and joined together.
  - If there is more than one match, all possible matches contribute one row each.

- **merge{base}重要Arguments**:
  - **by, by.x, by.y** : The names of the columns that are common to both x and y. The default is to use the columns with common names between the two data frames.
  - **all.x, (all.y)**: logical;
    - if **TRUE**, then extra rows will be added to the output, one for each row in **x** that has no matching row in **y**.
    - These rows will have NAs in those columns that are usually filled with values from **y**.
    - The default is **FALSE**, so that only rows with data from both **x** and **y** are included in the output.
  - **all = TRUE (FALSE)** is shorthand for **all.x = TRUE (FALSE)** and **all.y = TRUE (FALSE)**. Logical values that specify the type of merge.
    - The default value is **all=FALSE** (meaning that only the matching rows are returned).

Different Types of Merge

- **Natural join**: To keep only rows that match from the data frames, specify the argument `all=FALSE` **(by default)**.
- **Full outer join**: To keep all rows from both data frames, specify `all=TRUE`. Note that this performs the complete merge and fills the columns with NA values where there is no matching data.
- **Left outer join**: To include all the rows of your data frame `x` and only those from `y` that match, specify `all.x=TRUE`.
- **Right outer join**: To include all the rows of your data frame `y` and only those from `x` that match, specify `all.y=TRUE`.



`dplyr`: `inner_join` `full_join` `left_join` `right_join`

merge {base},
Example (1)

```
> authors <- data.frame(
+     surname = I(c("Tukey", "Venables", "Tierney", "Ripley", "McNeil")),
+     nationality = c("US", "Australia", "US", "UK", "Australia"),
+     deceased = c("yes", rep("no", 4)))
> books <- data.frame(
+     name = I(c("Tukey", "Venables", "Tierney",
+                "Ripley", "Ripley", "McNeil", "R Core")),
+     title = c("Exploratory Data Analysis",
+               "Modern Applied Statistics ...",
+               "LISP-STAT",
+               "Spatial Statistics", "Stochastic Simulation",
+               "Interactive Data Analysis",
+               "An Introduction to R"),
+     other.author = c(NA, "Ripley", NA, NA, NA, NA, "Venables & Smith"))
>  authors
   surname nationality deceased
1    Tukey          US      yes
2 Venables   Australia       no
3  Tierney          US       no
4   Ripley          UK       no
5   McNeil   Australia       no
> books
      name                          title     other.author
1    Tukey     Exploratory Data Analysis             <NA>
2 Venables Modern Applied Statistics ...           Ripley
3  Tierney                     LISP-STAT             <NA>
4   Ripley            Spatial Statistics             <NA>
5   Ripley         Stochastic Simulation             <NA>
6   McNeil     Interactive Data Analysis             <NA>
7   R Core           An Introduction to R Venables & Smith
```

https://stat.ethz.ch/R-manual/R-devel/library/base/html/merge.html

merge {base},
Example (1)

```
> (m1 <- merge(authors, books, by.x = "surname", by.y = "name"))
   surname nationality deceased                          title other.author
1   McNeil   Australia       no     Interactive Data Analysis         <NA>
2   Ripley          UK       no            Spatial Statistics         <NA>
3   Ripley          UK       no         Stochastic Simulation         <NA>
4  Tierney          US       no                     LISP-STAT         <NA>
5    Tukey          US      yes     Exploratory Data Analysis         <NA>
6 Venables   Australia       no Modern Applied Statistics ...       Ripley
> (m2 <- merge(books, authors, by.x = "name", by.y = "surname"))
      name                          title other.author nationality deceased
1   McNeil     Interactive Data Analysis         <NA>   Australia       no
2   Ripley            Spatial Statistics         <NA>          UK       no
3   Ripley         Stochastic Simulation         <NA>          UK       no
4  Tierney                     LISP-STAT         <NA>          US       no
5    Tukey     Exploratory Data Analysis         <NA>          US      yes
6 Venables Modern Applied Statistics ...       Ripley   Australia       no
```

```
> merge(authors, books, by.x = "surname", by.y = "name", all = TRUE)
   surname nationality deceased                          title     other.author
1   McNeil   Australia       no     Interactive Data Analysis             <NA>
2   R Core        <NA>     <NA>          An Introduction to R Venables & Smith
3   Ripley          UK       no            Spatial Statistics             <NA>
4   Ripley          UK       no         Stochastic Simulation             <NA>
5  Tierney          US       no                     LISP-STAT             <NA>
6    Tukey          US      yes     Exploratory Data Analysis             <NA>
7 Venables   Australia       no Modern Applied Statistics ...           Ripley
```

https://stat.ethz.ch/R-manual/R-devel/library/base/html/merge.html

merge {base},
Example (2)

```
> (x <- data.frame(k1 = c(NA,NA,3,4,5), k2 = c(1,NA,NA,4,5), data = 1:5))
  k1 k2 data
1 NA  1    1
2 NA NA    2
3  3 NA    3
4  4  4    4
5  5  5    5
> (y <- data.frame(k1 = c(NA,2,NA,4,5), k2 = c(NA,NA,3,4,5), data = 1:5))
  k1 k2 data
1 NA NA    1
2  2 NA    2
3 NA  3    3
4  4  4    4
5  5  5    5
> merge(x, y, by = c("k1","k2")) # NA's match
  k1 k2 data.x data.y
1  4  4      4      4
2  5  5      5      5
3 NA NA      2      1
> merge(x, y, by = "k1") # NA's match, so 6 rows
  k1 k2.x data.x k2.y data.y
1  4    4      4    4      4
2  5    5      5    5      5
3 NA    1      1   NA      1
4 NA    1      1    3      3
5 NA   NA      2   NA      1
6 NA   NA      2    3      3
> merge(x, y, by = "k2", incomparables = NA) # 2 rows
  k2 k1.x data.x k1.y data.y
1  4    4      4    4      4
2  5    5      5    5
```

merge {base},
Example (3)

```
> stories <- read.table(header=TRUE, text='
+    storyid  title
+     1       lions
+     2      tigers
+     3       bears
+ ')
> data <- read.table(header=TRUE, text='
+     subject storyid rating
+           1       1    6.7
+           1       2    4.5
+           1       3    3.7
+           2       2    3.3
+           2       3    4.1
+           2       1    5.2
+ ')
>
> merge(stories, data, by="storyid")
  storyid  title subject rating
1       1  lions       1    6.7
2       1  lions       2    5.2
3       2 tigers       1    4.5
4       2 tigers       2    3.3
5       3  bears       1    3.7
6       3  bears       2    4.1
```

```
>  stories2 <- read.table(header=TRUE, text='
+    id       title
+     1       lions
+     2      tigers
+     3       bears
+ ')
>
> merge(stories2, data, by.x="id", by.y="storyid")
  id  title subject rating
1  1  lions       1    6.7
2  1  lions       2    5.2
3  2 tigers       1    4.5
4  2 tigers       2    3.3
5  3  bears       1    3.7
6  3  bears       2    4.1
```

http://www.cookbook-r.com/Manipulating_data/Merging_data_frames/

Merge on Multiple Columns

```
> animals <- read.table(header=T, text='
+    size type         name
+   small  cat         lynx
+     big  cat        tiger
+   small  dog    chihuahua
+     big  dog "great dane"
+ ')
>
> observations <- read.table(header=T, text='
+    number  size type
+         1   big  cat
+         2 small  dog
+         3 small  dog
+         4   big  dog
+ ')
>
> merge(observations, animals, c("size","type"))
   size type number       name
1   big  cat      1      tiger
2   big  dog      4 great dane
3 small  dog      2  chihuahua
4 small  dog      3  chihuahua
```

split {base}:
Divide into Groups and Reassemble

- **split{base}** divides the data in the vector **x** into the groups defined by **f**. The replacement forms replace values corresponding to such a division. **unsplit{base}** reverses the effect of split.

```
split(x, f, drop = FALSE, ...)
split(x, f, drop = FALSE, ...) <- value
unsplit(value, f, drop = FALSE)
```

```
> n <- 10
> edu <- factor(sample(1:4, n, replace=T))
> score <- sample(0:100, n)
> cbind(edu, score)
      edu score
 [1,]   1    54
 [2,]   2    50
 [3,]   1     8
 [4,]   3    14
 [5,]   4    43
 [6,]   3     7
 [7,]   4    92
 [8,]   4    16
 [9,]   3    49
[10,]   4    51
```

```
> score.edu <- split(score, edu)
> score.edu
$`1`
[1] 54  8

$`2`
[1] 50

$`3`
[1] 14  7 49

$`4`
[1] 43 92 16 51
```

```
> unsplit(score.edu, edu)
 [1] 54 50  8 14 43  7 92 16 49 51
> sort(edu)
 [1] 1 1 2 3 3 3 4 4 4 4
Levels: 1 2 3 4
> unsplit(score.edu, sort(edu))
 [1] 54  8 50 14  7 49 43 92 16 51
```

split {base}, Example (1)

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
> month <- airquality$Month
> airquality.month <- split(airquality, month)
>
> mydata <- lapply(airquality.month, transform,
+  Oz.Z = scale(Ozone))
> airquality2 <- unsplit(mydata, month)
> head(airquality2)
  Ozone Solar.R Wind Temp Month Day       Oz.Z
1    41     190  7.4   67     5   1  0.7822293
2    36     118  8.0   72     5   2  0.5572518
3    12     149 12.6   74     5   3 -0.5226399
4    18     313 11.5   62     5   4 -0.2526670
5    NA      NA 14.3   56     5   5         NA
6    28      NA 14.9   66     5   6  0.1972879
```

```
> airquality.month
$`5`
   Ozone Solar.R Wind Temp Month Day
1     41     190  7.4   67     5   1
...
31    37     279  7.4   76     5  31

$`6`
   Ozone Solar.R Wind Temp Month Day
32    NA     286  8.6   78     6   1
...
61    NA     138  8.0   83     6  30
...

$`9`
    Ozone Solar.R Wind Temp Month Day
124    96     167  6.9   91     9   1
...
153    20     223 11.5   68     9  30
```

See also:

```
transform(airquality, Ozone = -Ozone)
transform(airquality, new = -Ozone, Temp = (Temp-32)/1.8)

attach(airquality)
transform(Ozone, logOzone = log(Ozone))
```

split{base}, Example (2)

```
> split(1:10, 1:2)
$`1`
[1] 1 3 5 7 9

$`2`
[1]  2  4  6  8 10
```

```
> # Split a matrix into a list by columns
> mat <- cbind(x = 1:10, y = (-4:5)^2)
> cbind(mat, col(mat))
       x  y
 [1,]  1 16 1 2
 [2,]  2  9 1 2
 [3,]  3  4 1 2
 [4,]  4  1 1 2
 [5,]  5  0 1 2
 [6,]  6  1 1 2
 [7,]  7  4 1 2
 [8,]  8  9 1 2
 [9,]  9 16 1 2
[10,] 10 25 1 2
> split(mat, col(mat))
$`1`
 [1]  1  2  3  4  5  6  7  8  9 10

$`2`
 [1] 16  9  4  1  0  1  4  9 16 25
```

apply {base}:
Apply Functions Over Array Margins

```
> (x <- matrix(1:24, nrow=4))
     [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    5    9   13   17   21
[2,]    2    6   10   14   18   22
[3,]    3    7   11   15   19   23
[4,]    4    8   12   16   20   24
>
> #1: rows, 2:columns
> apply(x, 1, sum)
[1] 66 72 78 84
> apply(x, 2, sum)
[1] 10 26 42 58 74 90
>
> #apply function to the individual elements
>
> apply(x, 1, sqrt)
         [,1]     [,2]     [,3]     [,4]
[1,] 1.000000 1.414214 1.732051 2.000000
[2,] 2.236068 2.449490 2.645751 2.828427
[3,] 3.000000 3.162278 3.316625 3.464102
[4,] 3.605551 3.741657 3.872983 4.000000
[5,] 4.123106 4.242641 4.358899 4.472136
[6,] 4.582576 4.690416 4.795832 4.898979
> apply(x, 2, sqrt)
         [,1]     [,2]     [,3]     [,4]     [,5]     [,6]
[1,] 1.000000 2.236068 3.000000 3.605551 4.123106 4.582576
[2,] 1.414214 2.449490 3.162278 3.741657 4.242641 4.690416
[3,] 1.732051 2.645751 3.316625 3.872983 4.358899 4.795832
[4,] 2.000000 2.828427 3.464102 4.000000 4.472136 4.898979
```

Description: Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.
Usage: `apply(X, MARGIN, FUN, ...)`

apply系列: tapply、lapply、sapply、mapply、rapply

apply {base}, 自定函式

將某班三科成績，皆以開根號乘以10重新計分。

```
> # generate score data
> math <- sample(1:100, 50, replace=T)
> english <- sample(1:100, 50, replace=T)
> algebra <- sample(1:100, 50, replace=T)
> ScoreData <- cbind(math, english, algebra)
> head(ScoreData, 5)
     math english algebra
[1,]    7      52      93
[2,]    7      17       9
[3,]   57      89      69
[4,]   69      21      97
[5,]   20      64      64
>
> myfun <- function(x){
+     sqrt(x)*10
+ }
> sdata1 <- apply(ScoreData, 2, myfun)
> head(sdata1, 5)
         math  english  algebra
[1,] 26.45751 72.11103 96.43651
[2,] 26.45751 41.23106 30.00000
[3,] 75.49834 94.33981 83.06624
[4,] 83.06624 45.82576 98.48858
[5,] 44.72136 80.00000 80.00000
```

```
> head(apply(ScoreData, 2, function(x) sqrt(x)*10), 5)
         math  english  algebra
[1,] 26.45751 72.11103 96.43651
[2,] 26.45751 41.23106 30.00000
[3,] 75.49834 94.33981 83.06624
[4,] 83.06624 45.82576 98.48858
[5,] 44.72136 80.00000 80.00000
>
> myfun2 <- function(x, attend){
+     y <- sqrt(x)*10 + attend
+     ifelse(y > 100, 100, y)
+ }
> sdata2 <- apply(ScoreData, 2, myfun2, attend=5)
> head(sdata2, 5)
         math  english   algebra
[1,] 31.45751 77.11103 100.00000
[2,] 31.45751 46.23106  35.00000
[3,] 80.49834 99.33981  88.06624
[4,] 88.06624 50.82576 100.00000
[5,] 49.72136 85.00000  85.00000
```

tapply {base}:
Apply a Function Over a "Ragged" Array

```
> tapply(iris$Sepal.Length, iris$Species, sum)
    setosa versicolor  virginica
     250.3      296.8      329.4
> tapply(iris$Sepal.Width, iris$Species, mean)
    setosa versicolor  virginica
     3.428      2.770      2.974
```

Description: Apply a function to each cell of a ragged array, that is to each (non-empty) group of values given by a unique combination of the levels of certain factors.

Usage: `tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)`

```
> set.seed(12345)
> scores <- sample(0:100, 50, replace=T)
> grade <- as.factor(sample(c("大一", "大二", "大三", "大四"), 50, replace=T))
> bloodtype <- as.factor(sample(c("A","AB","B","O"), 50, replace=T))
> tapply(scores, grade, mean)
    大一     大二     大三     大四
51.69231 55.87500 35.06667 59.42857
> tapply(scores, bloodtype, mean)
       A       AB        B        O
68.88889 43.12500 54.18750 37.94118
> tapply(scores, list(grade, bloodtype), mean)
         A       AB    B        O
大一 96.00       NA 65.5 31.14286
大二 97.00 50.33333 71.0 42.66667
大三 47.25 13.00000 39.0 25.66667
大四 71.00 56.00000 60.0 55.50000
```

```
> head(warpbreaks)
  breaks wool tension
1     26    A       L
2     30    A       L
3     54    A       L
4     25    A       L
5     70    A       L
6     52    A       L
```

```
> summary(warpbreaks[,-1])
 wool   tension
 A:27   L:18
 B:27   M:18
        H:18
> tapply(warpbreaks$breaks, warpbreaks[,-1], sum)
    tension
wool   L   M   H
   A 401 216 221
   B 254 259 169
```

tapply {base}:

Apply a Function Over a "Ragged" Array

```
> n <- 20
> (my.factor <- factor(rep(1:3, length = n), levels = 1:5))
 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
Levels: 1 2 3 4 5
> table(my.factor)
my.factor
1 2 3 4 5
7 7 6 0 0
> tapply(1:n, my.factor, sum)
 1  2  3  4  5
70 77 63 NA NA
```

```
> tapply(1:n, my.factor, range)
$`1`
[1]  1 19
$`2`
[1]  2 20
$`3`
[1]  3 18
$`4`
NULL
$`5`
NULL
```

```
> presidents
     Qtr1 Qtr2 Qtr3 Qtr4
1945   NA   87   82   75
1946   63   50   43   32
...
1974   28   25   24   24
> class(presidents)
[1] "ts"
> # gives the positions in the cycle of each observation.
> cycle(presidents)
Qtr1 Qtr2 Qtr3 Qtr4
1945    1    2    3    4
1946    1    2    3    4
...
1974    1    2    3    4
> tapply(presidents, cycle(presidents), mean, na.rm=T)
       1        2        3        4
58.44828 56.43333 57.22222 53.07143
```

lapply {base}:

Apply a Function over a List or Vector

lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.

```
> a <- c("a", "b", "c", "d")
> b <- c(1, 2, 3, 4, 4, 3, 2, 1)
> c <- c(T, T, F)
> list.object <- list(a,b,c)
> my.la1 <- lapply(list.object, length)
> my.la1
[[1]]
[1] 4

[[2]]
[1] 8

[[3]]
[1] 3
> my.la2 <- lapply(list.object, class)
> my.la2
[[1]]
[1] "character"

[[2]]
[1] "numeric"

[[3]]
[1] "logical"
```

```
> x <- list(a = 1:10, beta = exp(-3:3), logic =
c(TRUE,FALSE,FALSE,TRUE))
> lapply(x, mean) # return list
$a
[1] 5.5

$beta
[1] 4.535125

$logic
[1] 0.5
> # median and quartiles for each list element
> lapply(x, quantile, probs = 1:3/4)
$a
 25%  50%  75%
3.25 5.50 7.75

$beta
      25%       50%       75%
0.2516074 1.0000000 5.0536690

$logic
25% 50% 75%
0.0 0.5 1.0
```

sapply {base}:
Apply a Function over a List or Vector

```
sapply(X, FUN, ..., simplify = TRUE, USE.NAMES = TRUE)
```

- **sapply** is a user-friendly version and wrapper of **lapply** by default returning a vector, matrix or, if **simplify = "array"**, an array if appropriate, by applying **simplify2array()**.
- **sapply(x, f, simplify = FALSE, USE.NAMES = FALSE)** is the same as **lapply(x, f)**.
- **sapply** can be used to apply to each column of a data frame.

```
> x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
> sapply(x, mean)  # return vector
       a     beta    logic
5.500000 4.535125 0.500000
> sapply(x, quantile)
         a        beta logic
0%    1.00  0.04978707   0.0
25%   3.25  0.25160736   0.0
50%   5.50  1.00000000   0.5
75%   7.75  5.05366896   1.0
100% 10.00 20.08553692   1.0
```

```
> lapply(x, quantile)
$a
   0%   25%   50%   75%  100%
 1.00  3.25  5.50  7.75 10.00

$beta
         0%          25%          50%          75%
100%
 0.04978707  0.25160736  1.00000000  5.05366896
20.08553692

$logic
  0%  25%  50%  75% 100%
 0.0  0.0  0.5  1.0  1.0
```

sapply {base},

Example

```
> (i37 <- sapply(3:7, seq))
[[1]]
[1] 1 2 3

[[2]]
[1] 1 2 3 4

[[3]]
[1] 1 2 3 4 5

[[4]]
[1] 1 2 3 4 5 6

[[5]]
[1] 1 2 3 4 5 6 7

>  sapply(i37, fivenum)
     [,1] [,2] [,3] [,4] [,5]
[1,]  1.0  1.0    1  1.0  1.0
[2,]  1.5  1.5    2  2.0  2.5
[3,]  2.0  2.5    3  3.5  4.0
[4,]  2.5  3.5    4  5.0  5.5
[5,]  3.0  4.0    5  6.0  7.0
```

```
> (v <- structure(10*(5:7), names = LETTERS[1:3]))
 A  B  C
50 60 70
> f2 <- function(x, y) outer(rep(x, length.out = 3), y)
> sapply(v, f2, y = 1:4, simplify = "array")
, , A

     [,1] [,2] [,3] [,4]
[1,]   50  100  150  200
[2,]   50  100  150  200
[3,]   50  100  150  200

, , B

     [,1] [,2] [,3] [,4]
[1,]   60  120  180  240
[2,]   60  120  180  240
[3,]   60  120  180  240

, , C

     [,1] [,2] [,3] [,4]
[1,]   70  140  210  280
[2,]   70  140  210  280
[3,]   70  140  210  280
```

```
> sapply(v, f2, y = 1:4)
        A   B   C
 [1,]  50  60  70
 [2,]  50  60  70
 [3,]  50  60  70
 [4,] 100 120 140
 [5,] 100 120 140
 [6,] 100 120 140
 [7,] 150 180 210
 [8,] 150 180 210
 [9,] 150 180 210
[10,] 200 240 280
[11,] 200 240 280
[12,] 200 240 280
```

sapply {base},[[運算子

- "[[" 跟 "[" 是 operator (運算子)

```
> my.list <- list(name=c("George", "John", "Tom"),
+                 wife=c("Mary", "Sue", "Nico"),
+                 no.children=c(3, 2, 0),
+                 child.ages=list(c(4,7,9), c(2, 5), NA))
>
> # 取出某一家庭的資訊
> my.list[[1]][1]
[1] "George"
> my.list[[2]][1]
[1] "Mary"
> my.list[[3]][1]
[1] 3
> my.list[[4]][1]
[[1]]
[1] 4 7 9

> my.list[[1:4]][1] # Error
Error in my.list[[1:4]] : 遞迴索引在 2 層失敗
```

```
> George.family <- sapply(my.list,"[[", 1)
> George.family
$name
[1] "George"

$wife
[1] "Mary"

$no.children
[1] 3

$child.ages
[1] 4 7 9
```

mapply {base}:

Apply a Function to Multiple List or Vector Arguments

```
mapply(FUN, ..., MoreArgs = NULL, SIMPLIFY = TRUE, USE.NAMES = TRUE)
```

```
> mapply(rep, 1:4, 4:1)
[[1]]
[1] 1 1 1 1

[[2]]
[1] 2 2 2

[[3]]
[1] 3 3

[[4]]
[1] 4

> mapply(rep, times = 1:4, x = 4:1)
[[1]]
[1] 4

[[2]]
[1] 3 3

[[3]]
[1] 2 2 2

[[4]]
[1] 1 1 1 1
```

rep(x, ...)

...: further arguments to be passed to or from other methods: **times**, **each**, **length.out**.

```
> mapply(rep, times = 1:4, MoreArgs = list(x = 42))
[[1]]
[1] 42

[[2]]
[1] 42 42

[[3]]
[1] 42 42 42

[[4]]
[1] 42 42 42 42

> mapply(function(x, y) seq_len(x) + y,
+        c(a =  1, b = 2, c = 3),
+        c(A = 10, B = 0, C = -10))
$a
[1] 11

$b
[1] 1 2

$c
[1] -9 -8 -7
```

mapply {base}

Example

```
> word <- function(C, k) paste(rep.int(C, k), collapse = "")
> mapply(word, LETTERS[1:6], 6:1, SIMPLIFY = FALSE)
$A
[1] "AAAAAA"

$B
[1] "BBBBB"

$C
[1] "CCCC"

$D
[1] "DDD"

$E
[1] "EE"

$F
[1] "F"
> str(mapply(word, LETTERS[1:6], 6:1, SIMPLIFY = FALSE))
List of 6
 $ A: chr "AAAAAA"
 $ B: chr "BBBBB"
 $ C: chr "CCCC"
 $ D: chr "DDD"
 $ E: chr "EE"
 $ F: chr "F"
```

```
> mapply(cor, iris[,1:2], iris[,3:4])
Sepal.Length  Sepal.Width
   0.8717538   -0.3661259
> cor(iris[,1:4])
             Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

rapply {base}: Recursively Apply a Function to a List

```
rapply(object, f, classes = "ANY", deflt = NULL,
       how = c("unlist", "replace", "list"), ...)
```

- **rapply** is a recursive version of **lapply**.
- Arguments:
  - **object**: A list.
  - **f**: A function of a single argument.
  - **classes**: A character vector of class names, or "ANY" to match any class.
  - **deflt**: The default result (not used if **how = "replace"**).
  - **how**: A character string partially matching the three possibilities given.
    - If **how = "replace"**, each element of the list which is not itself a list and has a class included in classes is replaced by the result of applying f to the element.
    - If **how = "list"** or **how = "unlist"**, the list is copied, all non-list elements which have a class included in classes are replaced by the result of applying f to the element and all others are replaced by deflt.
    - if **how = "unlist"**, **unlist(recursive = TRUE)** is called on the result.

rapply {base}

Example

```
> mydata <- list(list(a = pi, b = list(c = 1:1)), d = "a test")
> mydata
[[1]]
[[1]]$a
[1] 3.141593

[[1]]$b
[[1]]$b$c
[1] 1

$d
[1] "a test"
```

```
> rapply(mydata, sqrt, classes = "numeric", how = "replace")
[[1]]
[[1]]$a
[1] 1.772454

[[1]]$b
[[1]]$b$c
[1] 1

$d
[1] "a test"
```

```
> rapply(mydata, nchar, classes = "character",
+        deflt = as.integer(NA), how = "list")
[[1]]
[[1]]$a
[1] NA

[[1]]$b
[[1]]$b$c
[1] NA

$d
[1] 6
```

```
> rapply(mydata, log, classes =
"numeric", how = "replace", base = 2)
[[1]]
[[1]]$a
[1] 1.651496

[[1]]$b
[[1]]$b$c
[1] 1

$d
[1] "a test"
```

```
> rapply(mydata, nchar, classes = "character",
+        deflt = as.integer(NA), how = "unlist")
  a b.c   d
 NA  NA   6
> rapply(mydata, nchar, classes = "character",
how = "unlist")
d
6
```

R Package dplyr: A Grammar of Data Manipulation

- dplyr provides a flexible grammar of data manipulation. It's the next iteration of plyr, focused on tools for working with data frames (hence the d in the name).
- dplyr works with data frames, data tables, databases and multidimensional arrays.:
- dplyr aims to provide a function for each basic verb of data manipulation.
  - filter() : Return rows with matching conditions.
  - slice() : Select rows by position.
  - arrange() : Arrange rows by variables.
  - select() : Select/rename variables by name.
  - rename() : Select/rename variables by name.
  - distinct() : Select distinct/unique rows.
  - mutate() : Add new variables.
  - transmute(): Add new variables.
  - summarise() : Summarise multiple values to a single value.
  - sample_n() : Sample n rows from a table.
  - sample_frac() : Sample n rows from a table.
  - join(): Join two tbls together.

R Package dplyr

```
> library(dplyr)
> browseVignettes(package = "dplyr")
```

Vignettes in package dplyr

- Column-wise operations - HTML source R code
- dplyr <-> base R - HTML source R code
- Grouped data - HTML source R code
- Introduction to dplyr - HTML source R code
- Programming with dplyr - HTML source R code
- Row-wise operations - HTML source R code
- Two-table verbs - HTML source R code
- Using dplyr in packages - HTML source R code
- Window functions - HTML source R code

Code demonstrations:

dplyr::bench-merge	Benchmark merging between R and python
dplyr::bench-rbind	Benchmark various flavours of rbind
dplyr::bench-set	Benchmark set operations on data frames

中文介紹: CELESTIAL, Introduction to dplyr
http://chingchuan-chen.github.io/r/2015/07/03/dplyr

Pipe operator: %>%

dplyr provides the %>% operator. x %>% f(y) turns into f(x, y) so you can use it to rewrite multiple operations that you can read left-to-right, top-to-bottom.

```
> x <- rnorm(10)
> x %>% max
> # is the same thing as:
> max(x)
```

Flights Data

- **flights {nycflights13}:** On-time data for all flights that departed NYC in 2013.
- Variables:
  - `year,month,day`: Date of departure
  - `dep_time,arr_time`: Departure and arrival times, local tz.
  - `dep_delay,arr_delay`: Departure and arrival delays, in minutes. Negative times represent early departures/arrivals.
  - `hour,minute`: Time of departure broken in to hour and minutes
  - `carrier`: Two letter carrier abbreviation. See airlines to get name
  - `tailnum`: Plane tail number
  - `flight`: Flight number
  - `origin,dest`: Origin and destination. See airports for additional metadata.
  - `air_time`: Amount of time spent in the air
  - `distance`: Distance flown

NOTE: dealing with large data, it's worthwhile to convert them to a `tbl_df`: this is a wrapper around a data frame that won't accidentally print a lot of data to the screen.

```
> library(nycflights13)
> dim(flights)
[1] 336776     16
> head(flights, 4)
  year month day dep_time dep_delay arr_time arr_delay carrier
1 2013     1   1      517         2      830        11      UA
2 2013     1   1      533         4      850        20      UA
3 2013     1   1      542         2      923        33      AA
4 2013     1   1      544        -1     1004       -18      B6
  tailnum flight origin dest air_time distance hour minute
1  N14228   1545    EWR  IAH      227     1400    5     17
2  N24211   1714    LGA  IAH      227     1416    5     33
3  N619AA   1141    JFK  MIA      160     1089    5     42
4  N804JB    725    JFK  BQN      183     1576    5     44
```

https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html

filter {dplyr}:
Return Rows with Matching Conditions

```
> library(dplyr)
> filter(flights, month == 1, day == 1)   #same as filter(flights, month == 1 & day == 1)
Source: local data frame [842 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time distance  hour minute
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl) (dbl)  (dbl)
1   2013     1     1      517         2      830        11      UA  N14228   1545    EWR   IAH      227     1400     5     17
2   2013     1     1      533         4      850        20      UA  N24211   1714    LGA   IAH      227     1416     5     33
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...      ...   ...    ...
> flights[flights$month == 1 & flights$day == 1, ]
Source: local data frame [842 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time distance  hour minute
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl) (dbl)  (dbl)
1   2013     1     1      517         2      830        11      UA  N14228   1545    EWR   IAH      227     1400     5     17
2   2013     1     1      533         4      850        20      UA  N24211   1714    LGA   IAH      227     1416     5     33
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...      ...   ...    ...
> table(flights$carrier)
   9E    AA    AS    B6    DL    EV    F9    FL    HA    MQ    OO    UA    US    VX    WN
YV
18460 32729   714 54635 48110 54173   685  3260   342 26397    32 58665 20536  5162 12275
601
> filter(flights, carrier %in% c("OO", "YV"))
Source: local data frame [633 x 16]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time ...
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl) ...
1   2013     1     3     1428        -7     1539       -20      YV  N509MJ   3750    LGA   IAD       47 ...
2   2013     1     3     1551       -11     1659       -23      YV  N508MJ   3771    LGA   IAD       47 ...
3   2013     1     4     1430        -5     1546       -13      YV  N511MJ   3750    LGA   IAD       55 ...
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...    ...   ...      ... ...
```

subset {base}:

Subsetting Vectors, Matrices and Data Frames

- `filter()` works similarly to `subset()` except that you can give it any number of filtering conditions.

```
> subset(flights, dep_delay < 0, select = c(carrier, distance))
Source: local data frame [183,575 x 2]
   carrier distance
     (chr)    (dbl)
1       B6     1576
2       DL      762
..     ...      ...
> subset(flights, origin == "JFK", select = -year)
Source: local data frame [111,279 x 15]
   month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time distance  hour minute
   (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl) (dbl)  (dbl)
1      1     1      542         2      923        33      AA  N619AA   1141    JFK   MIA      160     1089     5     42
2      1     1      544        -1     1004       -18      B6  N804JB    725    JFK   BQN      183     1576     5     44
..   ...   ...      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...      ...   ...    ...
```

```
> airquality.sub1 <- subset(airquality, Day == 1, select = -Temp)
> head(airquality.sub1, 3)
   Ozone Solar.R Wind Month Day
1     41     190  7.4     5   1
32    NA     286  8.6     6   1
62   135     269  4.1     7   1
```

```
> head(airquality, 3)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
```

```
> airquality.sub2 <- subset(airquality, Temp > 80, select = c(Ozone, Temp))
> head(airquality.sub2, 3)
   Ozone Temp
29    45   81
35    NA   84
36    NA   85
```

```
> airquality.sub3 <- subset(airquality, select = Ozone:Wind)
> head(airquality.sub3, 3)
  Ozone Solar.R Wind
1    41     190  7.4
2    36     118  8.0
3    12     149 12.6
```

arrange{dplyr}:
Arrange Rows by Variables

```
> #  same as flights[order(flights$month, flights$day, flights$distance), ]
> arrange(flights, month, day, distance)
Source: local data frame [336,776 x 16]

 year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time distance  hour minute
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl) (dbl)  (dbl)
1   2013     1     1     1600       -10     1712       -17      9E  N8968E   4088    JFK   PHL       35       94    16      0
2   2013     1     1     2000         0     2054       -16      9E  N836AY   3664    JFK   PHL       30       94    20      0
3   2013     1     1      908        -7     1004       -29      US  N959UW   1467    LGA   PHL       32       96     9      8
4   2013     1     1     1318        -4     1358       -18      EV  N19554   4106    EWR   BDL       25      116    13     18
5   2013     1     1     2302        62     2342        49      EV  N13903   4276    EWR   BDL       24      116    23      2
6   2013     1     1     1315        -2     1413       -10      EV  N13538   4112    EWR   ALB       33      143    13     15
7   2013     1     1     1655        34     1804        40      EV  N19554   3260    EWR   ALB       36      143    16     55
8   2013     1     1     2056        52     2156        44      EV  N12540   4170    EWR   ALB       31      143    20     56
9   2013     1     1     2116         6     2202       -10      EV  N15912   4404    EWR   PVD       28      160    21     16
10  2013     1     1     1158        -2     1256        -4      WN  N783SW   1568    EWR   BWI       38      169    11     58
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...      ...   ...    ...
>
> # same as flights[order(flights$carrier, desc(flights$arr_delay)), ]
> arrange(flights, carrier, desc(arr_delay))
Source: local data frame [336,776 x 16]

year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time distance  hour minute
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl) (dbl)  (dbl)
1   2013     2    16      757       747     1013       744      9E  N8940E   3798    JFK   CLT       85      541     7     57
2   2013     7    24     1525       430     1808       458      9E  N927XJ   3538    JFK   MSP      150     1029    15     25
3   2013     7    10     2054       355      102       421      9E  N937XJ   3325    JFK   DFW      191     1391    20     54
4   2013    11    27     1503       408     1628       396      9E  N930XJ   2906    JFK   BUF       56      301    15      3
5   2013    12    14     1425       360     1604       386      9E  N936XJ   2901    JFK   BOS       34      187    14     25
6   2013     2    27     1529       404     1639       384      9E  N922XJ   3405    JFK   DCA       49      213    15     29
7   2013     7    22     2216       356      116       383      9E  N903XJ   3341    JFK   DTW       88      509    22     16
8   2013     6    25     1421       376     1602       372      9E  N8458A   3611    JFK   PIT       64      340    14     21
9   2013     1    25       15       360      208       370      9E  N8646A   4019    JFK   RIC       56      288     0     15
10  2013     3     1     1449       354     1701       357      9E  N937XJ   3552    JFK   DTW       90      509    14     49
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...      ...   ...    ...
```

select{dplyr}: Select/Rename Variables by Name

- There are a number of special functions that only work inside select
  - `starts_with(x, ignore.case = TRUE)`: names starts with x
  - `ends_with(x, ignore.case = TRUE)`: names ends in x
  - `contains(x, ignore.case = TRUE)`: selects all variables whose name contains x
  - `matches(x, ignore.case = TRUE)`: selects all variables whose name matches the regular expression x
  - `num_range("x", 1:5, width = 2)`: selects all variables (numerically) from x01 to x05.
  - `one_of("x", "y", "z")`: selects variables provided in a character vector.
  - `everything()`: selects all variables.
  - To drop variables, use -. You can rename variables with named arguments.

- This function works similarly to the select argument in `subset{base}`.

select{dplyr}

Example (1)

```
> select(flights, origin, carrier, distance) #Select columns by name
Source: local data frame [336,776 x 3]

   origin carrier distance
    (chr)   (chr)    (dbl)
1     EWR      UA     1400
2     LGA      UA     1416
3     JFK      AA     1089
..    ...     ...      ...
> select(flights, dep_time:arr_delay) #Select all columns between variables (inclusive)
Source: local data frame [336,776 x 4]

   dep_time dep_delay arr_time arr_delay
      (int)     (dbl)    (int)     (dbl)
1       517         2      830        11
2       533         4      850        20
3       542         2      923        33
..      ...       ...      ...       ...
> select(flights, -(year:day)) # Select all columns except those from year to day (inclusive)
Source: local data frame [336,776 x 13]

   dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time distance  hour minute
      (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl) (dbl)  (dbl)
1       517         2      830        11      UA  N14228   1545    EWR   IAH      227     1400     5     17
2       533         4      850        20      UA  N24211   1714    LGA   IAH      227     1416     5     33
3       542         2      923        33      AA  N619AA   1141    JFK   MIA      160     1089     5     42
..      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...      ...   ...    ...
```

select{dplyr}, Example (1)

```
> select(flights, tail_num = tailnum) #drops all the variables not explicitly mentioned
Source: local data frame [336,776 x 1]

   tail_num
      (chr)
1    N14228
2    N24211
..      ...
>
> rename(flights, DepTime = dep_time, DepDelay = dep_delay)
Source: local data frame [336,776 x 16]

    year month   day DepTime DepDelay arr_time arr_delay carrier tailnum flight origin  dest air_time distance ..
   (int) (int) (int)   (int)    (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl) ..
1   2013     1     1     517        2      830        11      UA  N14228   1545    EWR   IAH      227     1400 ..
2   2013     1     1     533        4      850        20      UA  N24211   1714    LGA   IAH      227     1416 ..
..   ...   ...   ...     ...      ...      ...       ...     ...     ...    ...    ...   ...      ...      ... ..
```

select{dplyr}, Example (2)

```
> iris <- as_tibble(iris) # little nicer for printing
> head(iris, 3)
Source: local data frame [6 x 5]

  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
         (dbl)       (dbl)        (dbl)       (dbl)  (fctr)
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
> select(iris, matches(".t."))
Source: local data frame [150 x 4]

   Sepal.Length Sepal.Width Petal.Length Petal.Width
          (dbl)       (dbl)        (dbl)       (dbl)
1           5.1         3.5          1.4         0.2
2           4.9         3.0          1.4         0.2
..          ...         ...          ...         ...
> select(iris, starts_with("Petal"))
Source: local data frame [150 x 2]

   Petal.Length Petal.Width
          (dbl)       (dbl)
1           1.4         0.2
2           1.4         0.2
..          ...         ...
> select(iris, ends_with("Width"))
Source: local data frame [150 x 2]

   Sepal.Width Petal.Width
         (dbl)       (dbl)
1          3.5         0.2
2          3.0         0.2
..         ...         ...
```

```
> select(iris, contains("etal"))
Source: local data frame [150 x 2]

   Petal.Length Petal.Width
          (dbl)       (dbl)
1           1.4         0.2
2           1.4         0.2
..          ...         ...
> select(iris, Petal.Length, Petal.Width)
Source: local data frame [150 x 2]

   Petal.Length Petal.Width
          (dbl)       (dbl)
1           1.4         0.2
2           1.4         0.2
..          ...         ...
> vars <- c("Petal.Length", "Petal.Width")
> select(iris, one_of(vars))
Source: local data frame [150 x 2]

   Petal.Length Petal.Width
          (dbl)       (dbl)
1           1.4         0.2
2           1.4         0.2
..          ...         ...
```

select{dplyr}, Example (2)

```
> select(iris, -starts_with("Petal"))  # Drop variables
Source: local data frame [150 x 3]

   Sepal.Length Sepal.Width Species
          (dbl)       (dbl)  (fctr)
1           5.1         3.5  setosa
2           4.9         3.0  setosa
..          ...         ...     ...
> select(iris, -ends_with("Width"))
Source: local data frame [150 x 3]

   Sepal.Length Petal.Length Species
          (dbl)        (dbl)  (fctr)
1           5.1          1.4  setosa
2           4.9          1.4  setosa
..          ...          ...     ...
> select(iris, -contains("etal"))
Source: local data frame [150 x 3]

   Sepal.Length Sepal.Width Species
          (dbl)       (dbl)  (fctr)
1           5.1         3.5  setosa
2           4.9         3.0  setosa
..          ...         ...     ...
```

```
> select(iris, -matches(".t."))
Source: local data frame [150 x 1]

   Species
    (fctr)
1   setosa
2   setosa
..     ...
> select(iris, -Petal.Length, -Petal.Width)
Source: local data frame [150 x 3]

   Sepal.Length Sepal.Width Species
          (dbl)       (dbl)  (fctr)
1           5.1         3.5  setosa
2           4.9         3.0  setosa
..          ...         ...     ...
```

distinct{dplyr}:
Extract Distinct (unique) Rows

- `select()` is particularly useful in conjunction with the `distinct()` verb which only returns the unique values in a table.
- This is very similar to `base::unique()` but should be much faster.

```
> distinct(select(flights, tailnum))
Source: local data frame [4,044 x 1]

   tailnum
     (chr)
1   N14228
2   N24211
3   N619AA
4   N804JB
5   N668DN
6   N39463
7   N516JB
8   N829AS
9   N593JB
10  N3ALAA
..     ...
```

```
> distinct(select(flights, origin, dest))
Source: local data frame [224 x 2]

   origin  dest
    (chr) (chr)
1     EWR   IAH
2     LGA   IAH
3     JFK   MIA
4     JFK   BQN
5     LGA   ATL
6     EWR   ORD
7     EWR   FLL
8     LGA   IAD
9     JFK   MCO
10    LGA   ORD
..    ...   ...
```

mutate {dplyr}:
Add New Variables

- Mutate adds new variables and preserves existing; transmute drops existing variables.
- dplyr::mutate() works the same way as plyr::mutate() and similarly to base::transform(). The key difference between mutate() and transform() is that mutate allows you to refer to columns that you’ ve just created:

```
> mutate(flights, gain = arr_delay - dep_delay, speed = distance/air_time * 60)
Source: local data frame [336,776 x 18]

    year month   day dep_time dep_delay arr_time arr_delay ...  hour minute  gain    speed
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl) ... (dbl)  (dbl) (dbl)    (dbl)
1   2013     1     1      517         2      830        11 ...     5     17     9 370.0441
2   2013     1     1      533         4      850        20 ...     5     33    16 374.2731
..   ...   ...   ...      ...       ...      ...       ... ...   ...    ...   ...      ...
> mutate(flights, gain = arr_delay - dep_delay, gain_per_hour = gain/(air_time/60))
Source: local data frame [336,776 x 18]

    year month   day dep_time dep_delay arr_time arr_delay ...  hour minute  gain gain_per_hour
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl) ...  (dbl)  (dbl) (dbl)         (dbl)
1   2013     1     1      517         2      830        11 ...     5     17     9      2.378855
2   2013     1     1      533         4      850        20 ...     5     33    16      4.229075
..   ...   ...   ...      ...       ...      ...       ... ...   ...    ...   ...           ...
> transform(flights, gain = arr_delay - dep_delay, gain_per_hour = gain/(air_time/60))
Error in eval(expr, envir, enclos) : 找不到物件 'gain'
> transmute(flights, gain = arr_delay - dep_delay, gain_per_hour = gain/(air_time/60))
Source: local data frame [336,776 x 2]

    gain gain_per_hour
   (dbl)         (dbl)
1     9      2.378855
2    16      4.229075
..   ...           ...
```

summarise{dplyr}:

Summarise Multiple Values to A Single Value

- `summarise()` collapses a data frame to a single row (this is exactly equivalent to `plyr::summarise()`).

```
> head(mtcars, 3)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
> group_by(mtcars, cyl)
Source: local data frame [32 x 11]
Groups: cyl [3]

     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
   (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
1   21.0     6 160.0   110  3.90 2.620 16.46     0     1     4     4
2   21.0     6 160.0   110  3.90 2.875 17.02     0     1     4     4
..   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
> summarise(group_by(mtcars, cyl), m = mean(disp), sd = sd(disp))
Source: local data frame [3 x 3]

    cyl        m       sd
  (dbl)    (dbl)    (dbl)
1     4 105.1364 26.87159
2     6 183.3143 41.56246
3     8 353.1000 67.77132
```

```
> summarise(flights, delay = mean(dep_delay, na.rm = TRUE))
Source: local data frame [1 x 1]

     delay
     (dbl)
1 12.63907

> mean(flights$dep_delay, na.rm=T)
[1] 12.63907
```

Randomly Sample Rows

```
> sample_n(iris, 5)
Source: local data frame [5 x 5]

  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
         (dbl)       (dbl)        (dbl)       (dbl)     (fctr)
1          6.7         3.0          5.2         2.3  virginica
2          6.3         2.5          4.9         1.5 versicolor
3          6.9         3.1          5.1         2.3  virginica
4          6.2         3.4          5.4         2.3  virginica
5          5.0         3.5          1.6         0.6     setosa
> sample_frac(iris, 0.05)
Source: local data frame [8 x 5]

  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
         (dbl)       (dbl)        (dbl)       (dbl)     (fctr)
1          5.0         3.4          1.6         0.4     setosa
2          7.0         3.2          4.7         1.4 versicolor
3          5.6         3.0          4.5         1.5 versicolor
4          7.7         2.6          6.9         2.3  virginica
5          5.1         3.7          1.5         0.4     setosa
6          5.0         3.3          1.4         0.2     setosa
7          5.1         3.5          1.4         0.3     setosa
8          6.4         2.9          4.3         1.3 versicolor
```

group_by{dplyr}: Grouped Operations

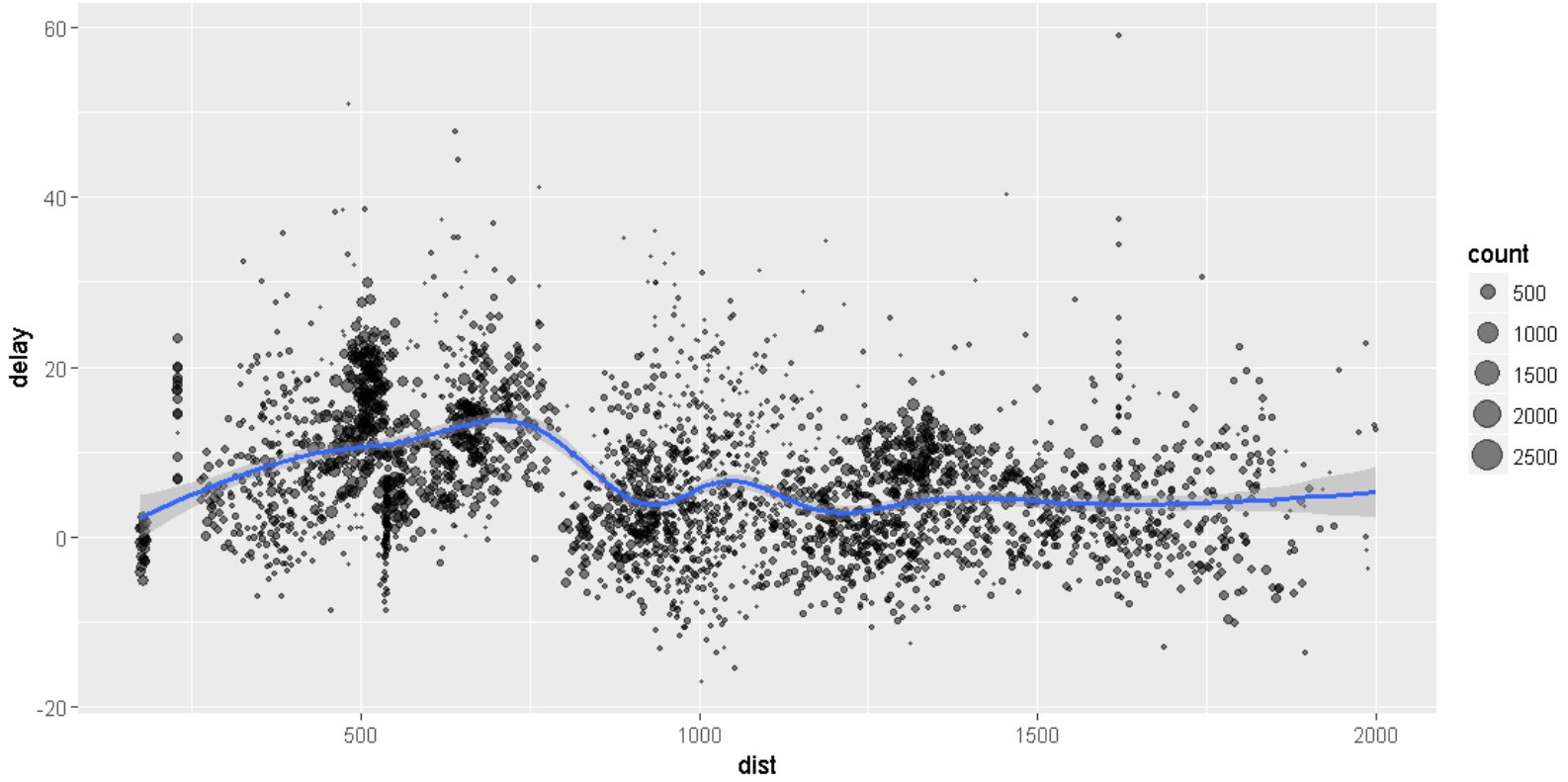
```
> by_tailnum <- group_by(flights, tailnum)
> by_tailnum
Source: local data frame [336,776 x 16]
Groups: tailnum [4044]

    year month   day dep_time dep_delay arr_time arr_delay carrier tailnum flight origin  dest air_time distance
   (int) (int) (int)    (int)     (dbl)    (int)     (dbl)   (chr)   (chr)  (int)  (chr) (chr)    (dbl)    (dbl)
1   2013     1     1      517         2      830        11      UA  N14228   1545    EWR   IAH      227     1400
2   2013     1     1      533         4      850        20      UA  N24211   1714    LGA   IAH      227     1416
..   ...   ...   ...      ...       ...      ...       ...     ...     ...    ...    ...   ...      ...      ...
Variables not shown: hour (dbl), minute (dbl)
> delay <- summarise(by_tailnum,
+                    count = n(),
+                    dist = mean(distance, na.rm = TRUE),
+                    delay = mean(arr_delay, na.rm = TRUE))
> delay
Source: local data frame [4,044 x 4]

   tailnum count     dist      delay
     (chr) (int)    (dbl)      (dbl)
1           2512 710.2576        NaN
2   D942DN     4 854.5000 31.5000000
3   N0EGMQ   371 676.1887  9.9829545
..     ...   ...      ...        ...
> delay <- filter(delay, count > 20, dist < 2000)
> delay
Source: local data frame [2,962 x 4]

   tailnum count     dist      delay
     (chr) (int)    (dbl)      (dbl)
1           2512 710.2576        NaN
2   N0EGMQ   371 676.1887  9.9829545
3   N10156   153 757.9477 12.7172414
..     ...   ...      ...        ...
```

Group the complete dataset by planes numbers (`tailnum`) and then summarise each plane by counting the number of flights (`count = n()`) and computing the average distance (`dist = mean(Distance, na.rm = TRUE)`) and arrival delay (`delay = mean(ArrDelay, na.rm = TRUE)`).

Visualize the Results

```
ggplot(delay, aes(dist, delay)) +
  geom_point(aes(size = count), alpha = 1/2) +
  geom_smooth() +
  scale_size_area()
```

The average delay is only slightly related to the average distance flown by a plane.



Some Useful Arguments

- `n()`: the number of observations in the current group
- `n_distinct(x)`: the number of unique values in x.
- `first(x)`, `last(x)` and `nth(x, n)` - these work similarly to `x[1]`, `x[length(x)]`, and `x[n]` but give you more control over the result if the value is missing.

```
> destinations <- group_by(flights, dest)
> summarise(destinations, planes = n_distinct(tailnum), flights = n())
Source: local data frame [105 x 3]

    dest planes flights
   (chr)  (int)   (int)
1    ABQ    108     254
2    ACK     58     265
3    ALB    172     439
4    ANC      6       8
5    ATL   1180   17215
6    AUS    993    2439
7    AVL    159     275
8    BDL    186     443
9    BGR     46     375
10   BHM     45     297
..   ...    ...     ...
```

Summarise{dplyr}

with Some Other Helpful Functions

```
> daily <- group_by(flights, year, month, day)
> (per_day   <- summarise(daily, flights = n()))
Source: local data frame [365 x 4]
Groups: year, month [?]

    year month   day flights
   (int) (int) (int)   (int)
1   2013     1     1     842
2   2013     1     2     943
3   2013     1     3     914
4   2013     1     4     915
5   2013     1     5     720
6   2013     1     6     832
7   2013     1     7     933
8   2013     1     8     899
9   2013     1     9     902
10  2013     1    10     932
..   ...   ...   ...     ...
```

```
> (per_month <- summarise(per_day, flights = sum(flights)))
Source: local data frame [12 x 3]
Groups: year [?]

    year month flights
   (int) (int)   (int)
1   2013     1   27004
2   2013     2   24951
3   2013     3   28834
4   2013     4   28330
5   2013     5   28796
6   2013     6   28243
7   2013     7   29425
8   2013     8   29327
9   2013     9   27574
10  2013    10   28889
11  2013    11   27268
12  2013    12   28135
```

```
> (per_year  <- summarise(per_month, flights = sum(flights)))
Source: local data frame [1 x 2]

   year flights
  (int)   (int)
1  2013  336776
```

Chaining (Pipe operator: %>%)

```
a1 <- group_by(flights, year, month, day)
a2 <- select(a1, arr_delay, dep_delay)
a3 <- summarise(a2,
  arr = mean(arr_delay, na.rm = TRUE),
  dep = mean(dep_delay, na.rm = TRUE))
a4 <- filter(a3, arr > 30 | dep > 30)
```

```
filter(
  summarise(
    select(
      group_by(flights, year, month, day),
      arr_delay, dep_delay
    ),
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ),
  arr > 30 | dep > 30
)
```

This is difficult to read because the order of the operations is from inside to out. Thus, the arguments are a long way away from the function.

```
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
```

dplyr provides the %>% operator. **x %>% f(y)** turns into **f(x, y)** so you can use it to rewrite multiple operations that you can read left-to-right, top-to-bottom.

https://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html

Join in dplyr

- **join{dplyr}**: **anti_join, full_join, inner_join, left_join, right_join, semi_join**.

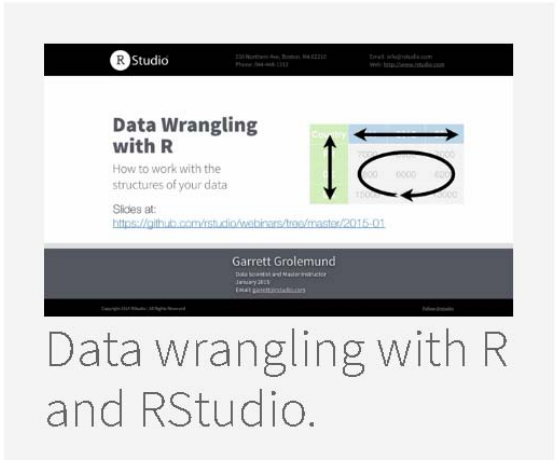
- **semi_join()**: return all rows from **x** where there are matching values in **y**, keeping just columns from **x**.
  - A semi join differs from an inner join because an inner join will return one row of **x** for each matching row of **y**, where a semi join will never duplicate rows of **x**.

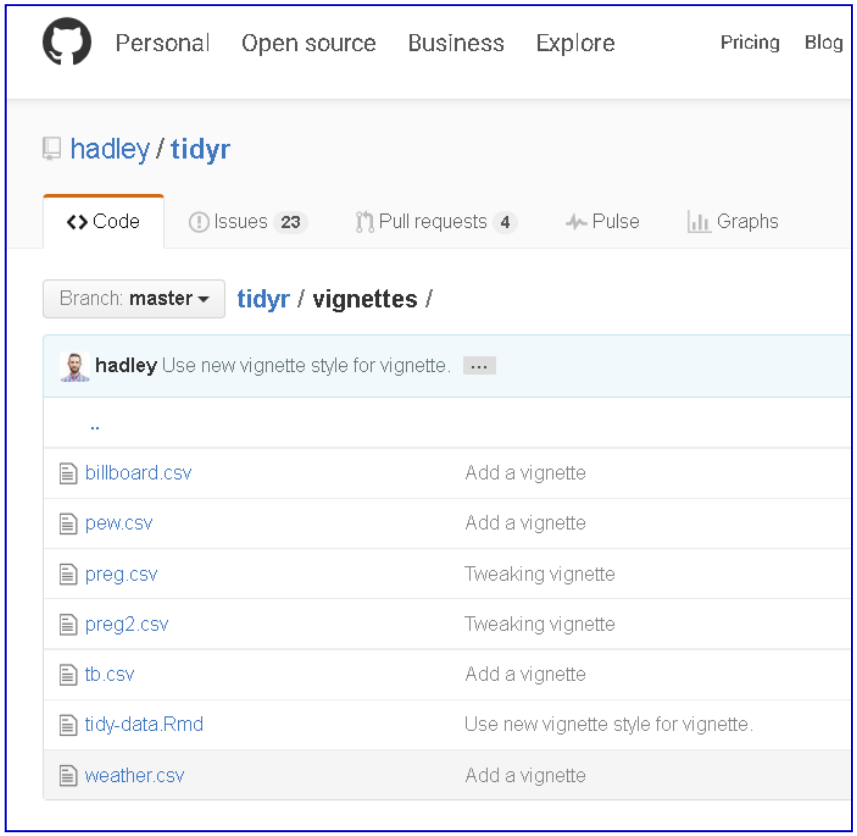
- **anti_join()**: return all rows from **x** where there are not matching values in **y**, keeping just columns from **x**.

- The difference between **merge** and **join{dplyr}** : dplyr的join不會去比對by variable都是NA的情況。

R Package tidyr: Easily Tidy Data

Hadley Wickham, Tidy Data, Journal of Statistical Software. August 2014, Volume 59, Issue 10.





- Some references
  - https://github.com/hadley/tidyr/tree/master/vignettes
  - https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html
  - blog.rstudio.org/2014/07/22/introducing-tidyr/
  - http://garrettgman.github.io/tidying/
  - https://rpubs.com/bradleyboehmke/data_wrangling

Data Semantics (語義學)

- In addition to appearance, we need a way to describe the underlying semantics, or meaning, of the values displayed in the table.
- Data tidying is to structure datasets to facilitate analysis.
- Tidy datasets provide a standardized way to link the structure of a dataset (its physical layout) with its semantics (its meaning).
- A dataset is a collection of values, usually either numbers (if quantitative) or strings (if qualitative). Values are organised in two ways.
  - Every value belongs to a variable and an observation.
  - A variable contains all values that measure the same underlying attribute (like height, temperature, duration) across units.
  - An observation contains all values measured on the same unit (like a person, or a day, or a race) across attributes.

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Data Semantics

- The **preg** data has the different layouts. We need a way to describe the underlying semantics, or meaning, of the values displayed in the table.

```
> preg <- read.table("preg.txt", sep="\t", header=T, stringsAsFactors = FALSE)
> preg
          name treatmenta treatmentb
1   John Smith         NA        184
2     Jane Doe          4          1
3 Mary Johnson          6          7
> preg2 <- read.table("preg2.txt", sep="\t", header=T, stringsAsFactors = FALSE)
> preg2
  treatment John.Smith Jane.Doe Mary.Johnson
1         a         NA        4            6
2         b         18        1            7
```

```
> library(tidyr)
> library(dplyr)
> preg.tidy <- preg %>%
+              gather(treatment, n, treatmenta:treatmentb) %>%
+              mutate(treatment = gsub("treatment", "", treatment)) %>%
+              arrange(name, treatment)
> preg.tidy
          name treatment   n
1     Jane Doe         a   4
2     Jane Doe         b   1
3   John Smith         a  NA
4   John Smith         b 184
5 Mary Johnson         a   6
6 Mary Johnson         b   7
```

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Tidying Messy Datasets

- Five most common problems with messy datasets:
  - (P1) Column headers are values, not variable names.
  - (P2) Multiple variables are stored in one column.
  - (P3) Variables are stored in both rows and columns.
  - (P4) Multiple types of observational units are stored in the same table.
  - (P5) A single observational unit is stored in multiple tables.

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

gather {tidyr}:
Gather Columns into Key-Value Pairs

```
gather(data, key, value, ..., na.rm = FALSE, convert = FALSE)
```

data: A data frame.
key,value: Names of key and value columns to create in output.
...: Specification of columns to gather. Use bare variable names. Select all variables between **x** and **z** with **x:z**, exclude **y** with **-y**.

```
> xdata <- data.frame(Group=letters[1:4], matrix(rnorm(12), ncol=3))
> xdata
  Group          X1         X2         X3
1     a  2.9077373 -0.6077071 -0.5020209
2     b  1.1250308  1.9919512 -0.5429116
3     c  1.0895884  1.1823842  0.6811148
4     d -0.3249144  1.5894412 -1.4850463
> gather(xdata, key = KEY, value = VALUE, -Group)
   Group KEY      VALUE
1      a  X1  2.9077373
2      b  X1  1.1250308
3      c  X1  1.0895884
4      d  X1 -0.3249144
5      a  X2 -0.6077071
6      b  X2  1.9919512
7      c  X2  1.1823842
8      d  X2  1.5894412
9      a  X3 -0.5020209
10     b  X3 -0.5429116
11     c  X3  0.6811148
12     d  X3 -1.4850463
```

gather {tidyr},
Example (1)

```
> gather(xdata, key = KEY, value = VALUE)
     KEY              VALUE
1  Group                  a
2  Group                  b
3  Group                  c
4  Group                  d
5     X1   2.90773725494251
6     X1   1.12503076222075
7     X1   1.08958842851567
8     X1 -0.324914417778845
9     X2 -0.607707100381958
10    X2   1.99195119965524
11    X2   1.18238423339043
12    X2   1.58944115738675
13    X3 -0.502020934726878
14    X3 -0.542911635761563
15    X3   0.68111483297255
16    X3  -1.48504628535113
Warning message:
attributes are not identical across measure variables;
they will be dropped
> gather(xdata, key = KEY, value = VALUE, Group)
         X1         X2         X3   KEY VALUE
1  2.9077373 -0.6077071 -0.5020209 Group     a
2  1.1250308  1.9919512 -0.5429116 Group     b
3  1.0895884  1.1823842  0.6811148 Group     c
4 -0.3249144  1.5894412 -1.4850463 Group     d
> gather(xdata, key = KEY, value = VALUE, X1)
  Group         X2         X3 KEY      VALUE
1     a -0.6077071 -0.5020209  X1  2.9077373
2     b  1.9919512 -0.5429116  X1  1.1250308
3     c  1.1823842  0.6811148  X1  1.0895884
4     d  1.5894412 -1.4850463  X1 -0.3249144
```

```
> xdata
  Group         X1         X2         X3
1     a  2.9077373 -0.6077071 -0.5020209
2     b  1.1250308  1.9919512 -0.5429116
3     c  1.0895884  1.1823842  0.6811148
4     d -0.3249144  1.5894412 -1.4850463
```

```
> gather(xdata, key = KEY, value = VALUE,
X1, X2)
  Group         X3 KEY      VALUE
1     a -0.5020209  X1  2.9077373
2     b -0.5429116  X1  1.1250308
3     c  0.6811148  X1  1.0895884
4     d -1.4850463  X1 -0.3249144
5     a -0.5020209  X2 -0.6077071
6     b -0.5429116  X2  1.9919512
7     c  0.6811148  X2  1.1823842
8     d -1.4850463  X2  1.5894412
> gather(xdata, key = KEY, value = VALUE,
X1:X3)
   Group KEY      VALUE
1      a  X1  2.9077373
2      b  X1  1.1250308
3      c  X1  1.0895884
4      d  X1 -0.3249144
5      a  X2 -0.6077071
6      b  X2  1.9919512
7      c  X2  1.1823842
8      d  X2  1.5894412
9      a  X3 -0.5020209
10     b  X3 -0.5429116
11     c  X3  0.6811148
12     d  X3 -1.4850463
```

gather{tidyr}, Example (2)

```
> (mini_iris <- iris[c(1, 51, 101), ])
Source: local data frame [3 x 5]

  Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
         (dbl)       (dbl)        (dbl)       (dbl)     (fctr)
1          5.1         3.5          1.4         0.2     setosa
2          7.0         3.2          4.7         1.4 versicolor
3          6.3         3.3          6.0         2.5  virginica
> gather(mini_iris, key = flower_att, value = measurement,
+        Sepal.Length:Petal.Width)
Source: local data frame [12 x 3]

      Species   flower_att measurement
       (fctr)       (fctr)       (dbl)
1      setosa Sepal.Length         5.1
2  versicolor Sepal.Length         7.0
3   virginica Sepal.Length         6.3
4      setosa  Sepal.Width         3.5
5  versicolor  Sepal.Width         3.2
6   virginica  Sepal.Width         3.3
7      setosa Petal.Length         1.4
8  versicolor Petal.Length         4.7
9   virginica Petal.Length         6.0
10     setosa  Petal.Width         0.2
11 versicolor  Petal.Width         1.4
12  virginica  Petal.Width         2.5
```

```
> gather(mini_iris, key = flower_att, value =
measurement, -Species)
Source: local data frame [12 x 3]

      Species   flower_att measurement
       (fctr)       (fctr)       (dbl)
1      setosa Sepal.Length         5.1
2  versicolor Sepal.Length         7.0
3   virginica Sepal.Length         6.3
4      setosa  Sepal.Width         3.5
5  versicolor  Sepal.Width         3.2
6   virginica  Sepal.Width         3.3
7      setosa Petal.Length         1.4
8  versicolor Petal.Length         4.7
9   virginica Petal.Length         6.0
10     setosa  Petal.Width         0.2
11 versicolor  Petal.Width         1.4
12  virginica  Petal.Width         2.5
```

(P1) Column Headers are Values

- `pew.csv` dataset explores the relationship between income and religion in the US. It comes from a report[1] produced by the Pew Research Center, an American think-tank that collects data on attitudes to topics ranging from religion to the internet, and produces many reports that contain datasets in this format.

```
> pew <- tbl_df(read.csv("pew.csv", stringsAsFactors = FALSE, check.names = FALSE))
> pew
Source: local data frame [18 x 11]

                    religion <$10k $10-20k $20-30k $30-40k $40-50k $50-75k $75-100k $100-150k
                       (chr) (int)   (int)   (int)   (int)   (int)   (int)    (int)     (int)
1                   Agnostic    27      34      60      81      76     137      122       109
2                    Atheist    12      27      37      52      35      70       73        59
3                   Buddhist    27      21      30      34      33      58       62        39
4                   Catholic   418     617     732     670     638    1116      949       792
5  Don\xe2\u0080   know/refused    15      14      15      11      10      35       21        17
6             Evangelical Prot   575     869    1064     982     881    1486      949       723
7                       Hindu     1       9       7       9      11      34       47        48
8      Historically Black Prot   228     244     236
9            Jehovah's Witness    20      27      24
10                     Jewish    19      19      25
11               Mainline Prot   289     495     619
12                     Mormon    29      40      48
13                     Muslim     6       7       9
14                   Orthodox    13      17      23
15              Other Christian     9       7      11
16                 Other Faiths    20      33      40
17       Other World Religions     5       2       3
18                 Unaffiliated   217     299     374
Variables not shown: >150k (int), Don't know/refused (int)
```

```
pew %>%
  gather(income, frequency, -religion)
#> Source: local data frame [180 x 3]
#>
#>                    religion income frequency
#>                       (chr)  (chr)     (int)
#> 1                  Agnostic  <$10k        27
#> 2                   Atheist  <$10k        12
```

This dataset has three variables, religion, income and frequency. To tidy it, we need to gather the non-variable columns into a two-column key-value pair.

[1] http://www.pewforum.org/religious-landscape-study/

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

gather{tidyr}, Example (3)

```
> stocks
         time           X          Y          Z
1  2009-01-01 -1.33362673  3.6589529  0.1509051
2  2009-01-02  0.21038002 -0.1723368  6.8247903
3  2009-01-03 -0.16678875 -1.9590724  2.5699189
4  2009-01-04 -0.15690945 -2.1929030 -1.8019189
5  2009-01-05  0.27238365  0.7867679 -2.6380988
6  2009-01-06  1.78031897  2.5329838  8.8152759
7  2009-01-07 -2.06912971  2.6226384  3.9167952
8  2009-01-08 -1.02148600 -2.5632745  2.8496395
9  2009-01-09  0.32494284 -3.4165507 -6.1016899
10 2009-01-10 -0.04459401  1.0952851 -3.2018426
> # stocks %>% gather(stock, price, -time)
> stocks.ga <- gather(stocks, key = stock,
                      value = price, -time)
> stocks.ga
          time stock       price
1   2009-01-01     X -1.33362673
2   2009-01-02     X  0.21038002
...
10  2009-01-10     X -0.04459401
11  2009-01-01     Y  3.65895291
12  2009-01-02     Y -0.17233676
...
19  2009-01-09     Y -3.41655065
20  2009-01-10     Y  1.09528513
...
29  2009-01-09     Z -6.10168986
30  2009-01-10     Z -3.20184256
```

```
stocks <- data.frame(
  time = as.Date('2009-01-01') + 0:9,
  X = rnorm(10, 0, 1),
  Y = rnorm(10, 0, 2),
  Z = rnorm(10, 0, 4)
)
```

This form is tidy because each column represents a variable and each row represents an observation, in this case a demographic unit corresponding to a combination of religion and income.

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Billboard Dataset

- Billboard dataset records the date a song first entered the billboard top 100. It has variables for artist, track, date.entered, rank and week. The rank in each week after it enters the top 100 is recorded in 75 columns, wk1 to wk75.

- This form of storage is useful for data entry. It reduces duplication since otherwise each song in each week would need its own row, and song metadata like title and artist would need to be repeated.

```
> billboard <- tbl_df(read.csv("billboard.csv", stringsAsFactors = FALSE))
> billboard
Source: local data frame [317 x 81]

    year         artist                  track  time date.entered   wk1   wk2
   (int)          (chr)                  (chr) (chr)        (chr) (int) (int)
1   2000          2 Pac Baby Don't Cry (Keep...  4:22   2000-02-26    87    82
2   2000        2Ge+her The Hardest Part Of ...  3:15   2000-09-02    91    87
...
10  2000 Adams, Yolanda           Open My Heart  5:30   2000-08-26    76    76
..   ...             ...                    ...   ...          ...   ...   ...
Variables not shown: wk3 (int), wk4 (int), wk5 (int), wk6 (int), wk7 (int), wk8
...
  (lgl), wk74 (lgl), wk75 (lgl), wk76 (lgl)
```

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Tidy Billboard Dataset by gather()

- Tidy Billboard dataset by gathering together all the **wk** columns. The column names give the week and the values are the ranks.
- Use **na.rm** to drop any missing values (weeks that the song wasnot in the charts) from the gather columns.

```
> billboard2 <- billboard %>%
+   gather(week, rank, wk1:wk76, na.rm = TRUE)
> billboard2
Source: local data frame [5,307 x 7]

    year         artist                    track  time date.entered   week  rank
   (int)          (chr)                    (chr) (chr)        (chr) (fctr) (int)
1   2000          2 Pac Baby Don't Cry (Keep...  4:22   2000-02-26    wk1    87
2   2000        2Ge+her The Hardest Part Of ...  3:15   2000-09-02    wk1    91
3   2000   3 Doors Down               Kryptonite  3:53   2000-04-08    wk1    81
4   2000   3 Doors Down                    Loser  4:24   2000-10-21    wk1    76
5   2000       504 Boyz            Wobble Wobble  3:35   2000-04-15    wk1    57
6   2000           98^0 Give Me Just One Nig...  3:24   2000-08-19    wk1    51
7   2000        A*Teens            Dancing Queen  3:44   2000-07-08    wk1    97
8   2000        Aaliyah            I Don't Wanna  4:15   2000-01-29    wk1    84
9   2000        Aaliyah                Try Again  4:03   2000-03-18    wk1    59
10  2000 Adams, Yolanda            Open My Heart  5:30   2000-08-26    wk1    76
..   ...            ...                      ...   ...          ...    ...   ...
```

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Cleaning Billboard Dataset

- cleaning: converting the week variable to a number, and figuring out the date corresponding to each week on the charts.

```
> billboard3 <- billboard2 %>%
+     mutate(
+         week = extract_numeric(week),
+         date = as.Date(date.entered) + 7 * (week - 1)) %>%
+         select(-date.entered)
> billboard3
Source: local data frame [5,307 x 7]

    year         artist                  track  time  week  rank       date
   (int)          (chr)                  (chr) (chr) (dbl) (int)     (date)
1   2000         2 Pac Baby Don't Cry (Keep...  4:22     1    87 2000-02-26
2   2000        2Ge+her The Hardest Part Of ...  3:15     1    91 2000-09-02
3   2000   3 Doors Down              Kryptonite  3:53     1    81 2000-04-08
4   2000   3 Doors Down                   Loser  4:24     1    76 2000-10-21
5   2000       504 Boyz           Wobble Wobble  3:35     1    57 2000-04-15
6   2000           98^0 Give Me Just One Nig...  3:24     1    51 2000-08-19
7   2000        A*Teens           Dancing Queen  3:44     1    97 2000-07-08
8   2000        Aaliyah           I Don't Wanna  4:15     1    84 2000-01-29
9   2000        Aaliyah               Try Again  4:03     1    59 2000-03-18
10  2000 Adams, Yolanda           Open My Heart  5:30     1    76 2000-08-26
..   ...            ...                     ...   ...   ...   ...        ...
```

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

arrange{dplyr}:
Arrange Rows by Variables

- Sort the data by artist, track and week.

```
> billboard3 %>% arrange(artist, track, week)
Source: local data frame [5,307 x 7]

    year  artist                    track  time  week  rank       date
   (int)   (chr)                    (chr) (chr) (dbl) (int)     (date)
1   2000   2 Pac Baby Don't Cry (Keep...  4:22     1    87 2000-02-26
2   2000   2 Pac Baby Don't Cry (Keep...  4:22     2    82 2000-03-04
3   2000   2 Pac Baby Don't Cry (Keep...  4:22     3    72 2000-03-11
...
9   2000 2Ge+her The Hardest Part Of ...  3:15     2    87 2000-09-09
10  2000 2Ge+her The Hardest Part Of ...  3:15     3    92 2000-09-16
..   ...     ...                     ...   ...   ...   ...        ...
```

- Or by date and rank.

```
> billboard3 %>% arrange(date, rank)
Source: local data frame [5,307 x 7]

    year   artist  track  time  week  rank       date
   (int)    (chr)  (chr) (chr) (dbl) (int)     (date)
1   2000 Lonestar Amazed  4:25     1    81 1999-06-05
2   2000 Lonestar Amazed  4:25     2    54 1999-06-12
3   2000 Lonestar Amazed  4:25     3    44 1999-06-19
...
9   2000 Lonestar Amazed  4:25     8    29 1999-07-24
10  2000    Amber Sexual  4:38     2    99 1999-07-24
..   ...      ...    ...   ...   ...   ...        ...
```

separate{tidyr}:
Separate One Column into Multiple Columns

```
separate(data, col, into, sep = "[^[:alnum:]]+", remove =
TRUE, convert = FALSE, extra = "warn", fill = "warn", ...)
```

- **data**: A data frame.
- **col**: Bare column name.
- **into**: Names of new variables to create as character vector.
- **sep**: Separator between columns.
  - If character, is interpreted as a regular expression. The default value is a regular expression that matches any sequence of non-alphanumeric values.
  - If numeric, interpreted as positions to split at. Positive values start at 1 at the far-left of the string; negative value start at -1 at the far-right of the string. The length of sep should be one less than into.
- **remove** : If TRUE, remove input column from output data frame.
- **convert**: If TRUE, will run type.convert with as.is = TRUE on new columns. This is useful if the component columns are integer, numeric or logical.
- **extra**: If sep is a character vector, this controls what happens when there are too many pieces. There are three valid options:
  - "warn" (the default): emit a waring and drop extra values.
  - "drop": drop any extra values without a warning.
  - "merge": only splits at most length(into) times
- **fill**: If sep is a character vector, this controls what happens when there are not enough pieces. There are three valid options:
  - "warn" (the default): emit a waring and fill from the right
  - "right": fill with missing values on the right
  - "left": fill with missing values on the left

separate {dplyr}

```
> (df <- data.frame(x = c(NA, "a.b", "a.d", "b.c")))
     x
1 <NA>
2  a.b
3  a.d
4  b.c
> df %>% separate(x, c("X1", "X2"))
    X1   X2
1 <NA> <NA>
2    a    b
3    a    d
4    b    c
> # separate(df, x, c("X1", "X2"))
>
> (df <- data.frame(x = c("a", "a b", "a b c", NA)))
      x
1     a
2   a b
3 a b c
4  <NA>
> df %>% separate(x, c("X1", "X2"))
    X1   X2
1    a <NA>
2    a    b
3    a    b
4 <NA> <NA>
Warning messages:
1: Too many values at 1 locations: 3
2: Too few values at 1 locations: 1
```

`separate()` makes it easy to split a compound variables into individual variables. You can either pass it a regular expression to split on (the default is to split on non-alphanumeric columns), or a vector of character positions.

```
> (df %>% separate(x, c("X1", "X2"), extra =
"drop", fill = "right"))
    X1   X2
1    a <NA>
2    a    b
3    a    b
4 <NA> <NA>
>
```

no warnings

separate{dplyr}

```
> df
      x
1     a
2   a b
3 a b c
4  <NA>
> df %>% separate(x, c("X1", "X2"), extra = "merge", fill = "left")
    X1   X2
1 <NA>    a
2    a    b
3    a  b c
4 <NA> <NA>
>
> (df <- data.frame(x = c("x: 123", "y: error: 7")))
            x
1      x: 123
2 y: error: 7
> df %>% separate(x, c("Key", "Value"), ": ", extra = "merge")
  Key    Value
1   x      123
2   y error: 7
```

use extra = "merge" to split specified number of times

(P2) Multiple Variables Stored in One Column

- Tuberculosis dataset comes from the World Health Organisation, and records the counts of confirmed tuberculosis cases by country, year, and demographic group. The demographic groups are broken down by sex (m, f) and age (0-14, 15-25, 25-34, 35-44, 45-54, 55-64, unknown).

```
> tb <- tbl_df(read.csv("tb.csv", stringsAsFactors = FALSE))
> tb
Source: local data frame [5,769 x 22]

    iso2  year   m04  m514  m014 m1524 m2534 m3544 m4554 m5564   m65    mu   f04
   (chr) (int) (int) (int) (int) (int) (int) (int) (int) (int) (int) (int) (int)
1     AD  1989    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
2     AD  1990    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
3     AD  1991    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
4     AD  1992    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
5     AD  1993    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
6     AD  1994    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA    NA
7     AD  1996    NA    NA     0     0     0     4     1     0     0    NA    NA
8     AD  1997    NA    NA     0     0     1     2     2     1     6    NA    NA
9     AD  1998    NA    NA     0     0     0     1     0     0     0    NA    NA
10    AD  1999    NA    NA     0     0     0     1     1     0     0    NA    NA
..   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...   ...
Variables not shown: f514 (int), f014 (int), f1524 (int), f2534 (int), f3544
  (int), f4554 (int), f5564 (int), f65 (int), fu (int)
```

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Tidy Tuberculosis Dataset

gather up the non-variable columns

```
> tb2 <- tb %>% gather(demo, n, -iso2, -year, na.rm = TRUE)
> tb2
Source: local data frame [35,750 x 4]

    iso2  year   demo     n
   (chr) (int) (fctr) (int)
1     AD  2005    m04     0
2     AD  2006    m04     0
3     AD  2008    m04     0
4     AE  2006    m04     0
5     AE  2007    m04     0
6     AE  2008    m04     0
7     AG  2007    m04     0
8     AL  2005    m04     0
9     AL  2006    m04     1
10    AL  2007    m04     0
..   ...   ...    ...   ...
```

Column headers in this format are often separated by a non-alphanumeric character (e.g. ., -, _, :), or have a fixed width format, like in this dataset. In this case we want to split after the first character:

```
> tb3 <- tb2 %>% separate(demo, c("sex",
"age"), 1)
> tb3
Source: local data frame [35,750 x 5]

    iso2  year   sex   age     n
   (chr) (int) (chr) (chr) (int)
1     AD  2005     m    04     0
2     AD  2006     m    04     0
3     AD  2008     m    04     0
4     AE  2006     m    04     0
5     AE  2007     m    04     0
6     AE  2008     m    04     0
7     AG  2007     m    04     0
8     AL  2005     m    04     0
9     AL  2006     m    04     1
10    AL  2007     m    04     0
..   ...   ...   ...   ...   ...
```

Storing the values in this form resolves a problem in the original data. We want to compare rates, not counts, which means we need to know the population. In the original format, there is no easy way to add a population variable. It has to be stored in a separate table, which makes it hard to correctly match populations to counts. In tidy form, adding variables for population and rate is easy because they're just additional columns.

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

spread {tidyr}:

Spread a Key-Value Pair Across Multiple Columns

```
spread(data, key, value, fill = NA, convert = FALSE, drop = TRUE)
```

- **data** : A data frame.
- **key, value**: Bare (unquoted) names of key and value columns.
- **fill**: If set, missing values (i.e. **NA**, aren't present), will be replaced with this value.
- **convert:** If **TRUE**, **type.convert** with **asis = TRUE** will be run on each of the new columns. This is useful if the value column was a mix of variables that was coerced to a string.
- **drop**: If **FALSE**, will keep factor levels that don't appear in the data, filling in missing combinations with fill.

```
> (df <- data.frame(x = c("a", "b"), y = c(3, 4), z = c(5, 6)))
  x y z
1 a 3 5
2 b 4 6
> df %>% spread(x, y)
  z  a  b
1 5  3 NA
2 6 NA  4
> df %>% spread(x, y) %>% gather(x, y, a:b, na.rm = TRUE)
  z x y
1 5 a 3
4 6 b 4
```

Spread and gather are complements

spread {tidyr},
Example

```
> stocks
         time           X          Y          Z
1  2009-01-01 -0.63120696  0.7047430 -4.9227997
2  2009-01-02  0.86850592  3.9330433  2.3410468
3  2009-01-03 -0.13134067  0.6719418 -3.4463902
...
10 2009-01-10 -0.86706770 -3.4354217  2.0495285
> stocks.ga <- stocks %>% gather(company, price, -time)
> stocks.ga
         time company      price
1  2009-01-01     X  0.2836269
2  2009-01-02     X  0.4151764
...
29 2009-01-09     Z  4.5319832
30 2009-01-10     Z  3.0196810
> stocks.ga %>% spread(company, price)
         time           X          Y          Z
1  2009-01-01 -0.63120696  0.7047430 -4.9227997
2  2009-01-02  0.86850592  3.9330433  2.3410468
...
10 2009-01-10 -0.86706770 -3.4354217  2.0495285
> stocks.ga %>% spread(time, price)
  company 2009-01-01 2009-01-02 2009-01-03 2009-01-04  2009-01-05 2009-01-06
1       X  -0.631207  0.8685059 -0.1313407  0.7155024  0.02419769  -1.826751
2       Y   0.704743  3.9330433  0.6719418 -0.4681197 -2.68079953  -1.076981
3       Z  -4.922800  2.3410468 -3.4463902 -2.9161013  0.75513726  -5.250819
  2009-01-07 2009-01-08 2009-01-09 2009-01-10
1 -0.7436164  0.3965774  2.0627961 -0.8670677
2  0.2790172  5.7711958 -0.3719509 -3.4354217
3 -2.4922130  3.6001196  6.6939517  2.0495285
```

```
stocks <- data.frame(
    time = as.Date('2009-01-01') + 0:9,
    X = rnorm(10, 0, 1),
    Y = rnorm(10, 0, 2),
    Z = rnorm(10, 0, 4)
)
```

(P3) Variables are Stored in Both Rows and Columns

- The daily weather data was from the Global Historical Climatology Network for one weather station (MX17004) in Mexico for five months in 2010.
- It has variables in individual columns (id, year, month), spread across columns (day, d1-d31) and across rows (tmin, tmax) (minimum and maximum temperature). Months with fewer than 31 days have structural missing values for the last day(s) of the month.

```
> weather <- tbl_df(read.csv("weather.csv", stringsAsFactors = FALSE))
> weather
Source: local data frame [22 x 35]

        id  year month element    d1    d2    d3    d4    d5    d6    d7    d8
     (chr) (int) (int)   (chr) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl) (dbl)
1  MX17004  2010     1    tmax    NA    NA    NA    NA    NA    NA    NA    NA
2  MX17004  2010     1    tmin    NA    NA    NA    NA    NA    NA    NA    NA
3  MX17004  2010     2    tmax    NA  27.3  24.1    NA    NA    NA    NA    NA
...
10 MX17004  2010     5    tmin    NA    NA    NA    NA    NA    NA    NA    NA
..     ...   ...   ...     ...   ...   ...   ...   ...   ...   ...   ...   ...
Variables not shown: 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)
```

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Tidy Weather Dataset

```
> # Tidy the dataset by gathering the day columns
> weather2 <- weather %>% gather(day, value, d1:d31, na.rm = TRUE)
> weather2
Source: local data frame [66 x 6]

        id  year month element    day value
     (chr) (int) (int)   (chr) (fctr) (dbl)
1  MX17004  2010    12    tmax     d1  29.9
2  MX17004  2010    12    tmin     d1  13.8
...
10 MX17004  2010     7    tmin     d3  17.5
..     ...   ...   ...     ...    ...   ...
```

For presentation, missing values were dropped, making them implicit rather than explicit. This is ok because we know how many days are in each month and can easily reconstruct the explicit missing values.

```
> # do a little cleaning
> weather3 <- weather2 %>%
+     mutate(day = extract_numeric(day)) %>%
+     select(id, year, month, day, element, value) %>%
+     arrange(id, year, month, day)
> weather3
Source: local data frame [66 x 6]

        id  year month   day element value
     (chr) (int) (int) (dbl)   (chr) (dbl)
1  MX17004  2010     1    30    tmax  27.8
2  MX17004  2010     1    30    tmin  14.5
...
10 MX17004  2010     2    23    tmin  10.7
..     ...   ...   ...   ...     ...   ...
```

The element column is not a variable; it stores the names of variables. (Not shown in this example are the other meteorological variables prcp (precipitation) and snow (snowfall)).

Tidy Weather Dataset

Use **`spread()`** to perform the inverse of gathering by spreading the element and value columns back out into the columns.

```
> weather3 %>% spread(element, value)
Source: local data frame [33 x 6]

         id  year month   day  tmax  tmin
      (chr) (int) (int) (dbl) (dbl) (dbl)
1  MX17004  2010     1    30  27.8  14.5
2  MX17004  2010     2     2  27.3  14.4
3  MX17004  2010     2     3  24.1  14.4
4  MX17004  2010     2    11  29.7  13.4
5  MX17004  2010     2    23  29.9  10.7
6  MX17004  2010     3     5  32.1  14.2
7  MX17004  2010     3    10  34.5  16.8
8  MX17004  2010     3    16  31.1  17.6
9  MX17004  2010     4    27  36.3  16.7
10 MX17004  2010     5    27  33.2  18.2
..     ...   ...   ...   ...   ...   ...
```

This form is tidy: there's one variable in each column, and each row represents one day.

(P4) Multiple Types in One Table

- The billboard dataset contains observations on two types of observational units: the song and its rank in each week.
- This manifests itself through the duplication of facts about the song: artist, year and time are repeated many times.

```
> billboard3
Source: local data frame [5,307 x 7]

    year         artist                   track  time  week  rank       date
   (int)          (chr)                   (chr) (chr) (dbl) (int)     (date)
1   2000          2 Pac Baby Don't Cry (Keep...  4:22     1    87 2000-02-26
2   2000        2Ge+her The Hardest Part Of ...  3:15     1    91 2000-09-02
3   2000   3 Doors Down               Kryptonite  3:53     1    81 2000-04-08
4   2000   3 Doors Down                    Loser  4:24     1    76 2000-10-21
5   2000       504 Boyz            Wobble Wobble  3:35     1    57 2000-04-15
6   2000           98^0 Give Me Just One Nig...  3:24     1    51 2000-08-19
7   2000        A*Teens           Dancing Queen  3:44     1    97 2000-07-08
8   2000        Aaliyah           I Don't Wanna  4:15     1    84 2000-01-29
9   2000        Aaliyah               Try Again  4:03     1    59 2000-03-18
10  2000 Adams, Yolanda           Open My Heart  5:30     1    76 2000-08-26
..   ...            ...                     ...   ...   ...   ...        ...
```

https://cran.r-project.org/web/packages/tidyr/vignettes/tidy-data.html

Tidy Billboard Dataset

- This dataset needs to be broken down into two pieces: a song dataset which stores artist, song name and time, and a ranking dataset which gives the rank of the song in each week.

```
> #First extract a song dataset
> song <- billboard3 %>%
+     select(artist, track, year, time) %>%
+     unique() %>%
+     mutate(song_id = row_number())
> song
Source: local data frame [317 x 5]

           artist                    track  year  time song_id
            (chr)                    (chr) (int) (chr)   (int)
1          2 Pac Baby Don't Cry (Keep...  2000  4:22       1
2         2Ge+her The Hardest Part Of ...  2000  3:15       2
3    3 Doors Down               Kryptonite  2000  3:53       3
4    3 Doors Down                    Loser  2000  4:24       4
5       504 Boyz             Wobble Wobble  2000  3:35       5
6           98^0 Give Me Just One Nig...  2000  3:24       6
7         A*Teens           Dancing Queen  2000  3:44       7
8         Aaliyah           I Don't Wanna  2000  4:15       8
9         Aaliyah               Try Again  2000  4:03       9
10 Adams, Yolanda           Open My Heart  2000  5:30      10
..            ...                     ...   ...   ...     ...
```

Tidy Billboard Dataset

Then use that to make a rank dataset by replacing repeated song facts with a pointer to song details (a unique song id).

```
> rank <- billboard3 %>%
+     left_join(song, c("artist", "track", "year", "time")) %>%
+     select(song_id, date, week, rank) %>%
+     arrange(song_id, date)
> rank
Source: local data frame [5,307 x 4]

   song_id       date  week  rank
     (int)     (date) (dbl) (int)
1        1 2000-02-26     1    87
2        1 2000-03-04     2    82
3        1 2000-03-11     3    72
4        1 2000-03-18     4    77
5        1 2000-03-25     5    87
6        1 2000-04-01     6    94
7        1 2000-04-08     7    99
8        2 2000-09-02     1    91
9        2 2000-09-09     2    87
10       2 2000-09-16     3    92
..     ...        ...   ...   ...
```

You could also imagine a week dataset which would record background information about the week, maybe the total number of songs sold or similar "demographic" information.

Normalisation is useful for tidying and eliminating inconsistencies. However, there are few data analysis tools that work directly with relational data, so analysis usually also requires denormalisation or the merging the datasets back into one table.

data.table: Extension of 'data.frame'

選讀

- **data.table**: Fast aggregation of large data (e.g. 100GB in RAM), fast ordered joins, fast add/modify/delete of columns by group using no copies at all, list columns, a fast friendly file reader and parallel file writer. Offers a natural and flexible syntax, for faster development.

https://cran.r-project.org/web/packages/data.table/

```
> library(data.table)
> # create a data.table object
> id <- letters[1:6]
> g <- sample(c("F", "M"), 6, replace=T)
> s <- sample(0:100, 6, replace=T)
> p <- sample(c("T", "F"), 6, replace=T)
> DT <- data.table(ID=id, gender=g, score=s, pass=p)
> str(DT)
Classes ‘data.table’ and 'data.frame':  6 obs. of  4 variables:
 $ ID    : chr  "a" "b" "c" "d" ...
 $ gender: chr  "F" "M" "M" "M" ...
 $ score : int  93 34 3 13 72 66
 $ pass  : chr  "T" "T" "F" "T" ...
 - attr(*, ".internal.selfref")=<externalptr>
```

tidytable: Tidy Interface to 'data.table'
https://cran.r-project.org/web/packages/tidytable/
A tidy interface to 'data.table', giving users the speed of 'data.table' while using tidyverse-like syntax.

```
> DT
   ID gender score pass
1:  a      F    93    T
2:  b      M    34    T
3:  c      M     3    F
4:  d      M    13    T
5:  e      F    72    T
6:  f      F    66    F
```

Unlike data.frames, columns of character type are never converted to factors by default.

DT[i, j, by]: Subset rows in i

選讀

NYC-flights14 data: on-time flights data from the Bureau of Transporation Statistics for all the flights that departed from New York City airports in 2014. The data is available only for Jan-Oct' 14.

https://github.com/arunsrinivasan/flights/wiki/NYCflights14/flights14.csv

General form of data.table syntax

DT[i, j, by]

Take DT, subset rows using "i", then calculate "j", grouped by "by".

```
> flights <- fread("flights14.csv")
> # Get all the flights with  "JFK"  origin airport in June
> ans <- flights[origin == "JFK" & month == 6L]
> head(ans)
>
>
>
>
> # flights[flights$origin == "JFK" & flights$month == 6L, ] OK
> # flights[origin == "JFK" & month == 6L, ] OK
>
> # Get the first two rows from flights
> ans <- flights[1:2]
> ans
>
>
> # Sort flights by origin in ascending order, and then by dest in descending order
> ans <- flights[order(origin, -dest)]
> head(ans)
>
```

```
> head(ans)
   year month day dep_time dep_delay arr_time arr_delay cancelled carrier tailnum flight origin dest air_time distance hour min
1: 2014     6   1      851        -9     1205        -5         0      AA  N787AA      1    JFK  LAX      324     2475    8  51
2: 2014     6   1     1220       -10     1522       -13         0      AA  N795AA      3    JFK  LAX      329     2475   12  20
3: 2014     6   1      718        18     1014        -1         0      AA  N784AA      9    JFK  LAX      326     2475    7  18
4: 2014     6   1     1024        -6     1314       -16         0      AA  N791AA     19    JFK  LAX      320     2475   10  24
5: 2014     6   1     1841        -4     2125       -45         0      AA  N790AA     21    JFK  LAX      326     2475   18  41
6: 2014     6   1     1454        -6     1757       -23         0      AA  N785AA    117    JFK  LAX      329     2475   14  54
```

```
> ans
   year month day dep_time dep_delay arr_time arr_delay cancelled carrier tailnum flight origin dest air_time distance hour min
1: 2014     1   1      914        14     1238        13         0      AA  N338AA      1    JFK  LAX      359     2475    9  14
2: 2014     1   1     1157        -3     1523        13         0      AA  N335AA      3    JFK  LAX      363     2475   11  57
```

```
> head(ans)
   year month day dep_time dep_delay arr_time arr_delay cancelled carrier tailnum flight origin dest air_time distance hour min
1: 2014     1   5      836         6     1151        49         0      EV  N12175   4419    EWR  XNA      195     1131    8  36
2: 2014     1   6      833         7     1111        13         0      EV  N24128   4419    EWR  XNA      190     1131    8  33
3: 2014     1   7      811        -6     1035       -13         0      EV  N12142   4419    EWR  XNA      179     1131    8  11
4: 2014     1   8      810        -7     1036       -12         0      EV  N11193   4419    EWR  XNA      184     1131    8  10
5: 2014     1   9      833        16     1055         7         0      EV  N14198   4419    EWR  XNA      181     1131    8  33
6: 2014     1  13      923        66     1154        66         0      EV  N12157   4419    EWR  XNA      188     1131    9  23
```

https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

DT[i, j, by]: Select column(s) in j

選讀

```
> # Select arr_delay column, return a "vector".
> ans <- flights[, arr_delay]
> head(ans)
[1]  13  13   9 -26   1   0
>
> # Select arr_delay column, return a "data.table"
> ans <- flights[, list(arr_delay)]
> head(ans)
   arr_delay
1:        13
2:        13
3:         9
4:       -26
5:         1
6:         0
>
> ans <- flights[, list(arr_delay, dep_delay)]
> # same as flights[, .(arr_delay, dep_delay)]
> head(ans)
   arr_delay dep_delay
1:        13        14
2:        13        -3
3:         9         2
4:       -26        -8
5:         1         2
6:         0         4
```

```
> # Select both arr_delay and dep_delay columns and rename
> ans <- flights[, .(X1 = arr_delay, X2 = dep_delay)]
> head(ans)
    X1 X2
1:  13 14
2:  13 -3
3:   9  2
4: -26 -8
5:   1  2
6:   0  4
```

https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

DT[i, j, by]: do in j

選讀

```
> # How many trips have had total delay < 0?
> ans <- flights[, sum((arr_delay + dep_delay) < 0)]
> ans
[1] 141814
>
> # Calculate the average arrival and departure delay for all flights with
> # "JFK" as the origin airport in the month of June.
> ans <- flights[origin == "JFK" & month == 6L,
+                .(m.arr = mean(arr_delay), m.dep = mean(dep_delay))]
> ans
      m.arr    m.dep
1: 5.839349 9.807884
>
> # How many trips have been made in 2014 from "JFK" airport in the month
of June?
> ans <- flights[origin == "JFK" & month == 6L, length(dest)]
> ans
[1] 8422
>
> # .N: the number of observations in the current group.
> ans <- flights[origin == "JFK" & month == 6L, .N]
> ans
[1] 8422
>
> # inefficient: nrow(flights[origin == "JFK" & month == 6L])
```

https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html

DT[i, j, by]:

select columns by names in j

選讀

```
> # Select both arr_delay and dep_delay columns (like in a data.frame)
> ans <- flights[, c("arr_delay", "dep_delay"), with = FALSE]
> head(ans)
   arr_delay dep_delay
1:        13        14
2:        13        -3
...
>
> # returns all columns except arr_delay and dep_delay
> ans <- flights[, !c("arr_delay", "dep_delay"), with = FALSE]
> # or
> ans <- flights[, -c("arr_delay", "dep_delay"), with = FALSE]
> head(ans)
   year month day dep_time arr_time cancelled carrier tailnum flight origin dest air_time distance hour min
1: 2014     1   1      914     1238         0      AA  N338AA      1    JFK  LAX      359     2475    9  14
2: 2014     1   1     1157     1523         0      AA  N335AA      3    JFK  LAX      363     2475   11  57
...
>
> # returns year, month and day
> ans <- flights[, year:day, with = FALSE]
> # returns day, month and year
> ans <- flights[, day:year, with = FALSE]
> head(ans)
   day month year
1:   1     1 2014
2:   1     1 2014
...
```

DT[i, j, by]: Grouping using by

選讀

```
> ans <- flights[, .N, by = origin]
> # flights[, .(.N), by = "origin"] OK
> ans
   origin     N
1:    JFK 81483
2:    LGA 84433
3:    EWR 87400
```

```
> # the number of trips for each origin airport for carrier code  "AA"
> ans <- flights[carrier == "AA", .N, by = origin]
> ans
   origin     N
1:    JFK 11923
2:    LGA 11730
3:    EWR  2649
```

```
> # the total number of trips for each origin, dest pair for carrier code  "AA" ?
> ans <- flights[carrier == "AA", .N, by = .(origin, dest)]
> # flights[carrier == "AA", .N, by = c("origin", "dest")] OK
> head(ans)
   origin dest    N
1:    JFK  LAX 3387
2:    LGA  PBI  245
3:    EWR  LAX   62
4:    JFK  MIA 1876
5:    JFK  SEA  298
6:    EWR  MIA  848
```

```
> # the average arrival and departure delay for each orig,
dest pair for each month for carrier code  "AA" ?
> ans <- flights[carrier == "AA",
+          .(mean(arr_delay), mean(dep_delay)),
+          by = .(origin, dest, month)]
> ans
     origin dest month         V1         V2
  1:    JFK  LAX     1   6.590361 14.2289157
  2:    LGA  PBI     1  -7.758621  0.3103448
...
  5:    JFK  SEA     1  14.357143 30.7500000
 ---
196:    LGA  MIA    10  -6.251799 -1.4208633
197:    JFK  MIA    10  -1.880184  6.6774194
...
200:    JFK  DCA    10  16.483871 15.5161290
```

DT[i, j, by]:
keyby、Chaining、Expression

選讀

```
> # increasingly order by all the grouping variables (keyby)
> ans <- flights[carrier == "AA",
+         .(mean(arr_delay), mean(dep_delay)),
+         keyby = .(origin, dest, month)]
> ans
     origin dest month         V1         V2
  1:    EWR  DFW     1   6.427673 10.0125786
  2:    EWR  DFW     2  10.536765 11.3455882
...
  5:    EWR  DFW     5  18.487805 18.6829268
 ---
196:    LGA  PBI     1  -7.758621  0.3103448
197:    LGA  PBI     2  -7.865385  2.4038462
...
200:    LGA  PBI     5 -10.357143 -6.8571429
>
> # Chaining: forming a chain of operations, DT[ ... ][ ... ][ ... ]
> # order ans using the columns origin in ascending order, and dest in descending order
> ans <- flights[carrier == "AA", .N, by = .(origin, dest)][order(origin, -dest)]
> # same as
> # ans <- flights[carrier == "AA", .N, by = .(origin, dest)]
> # ans[order(origin, -dest)]
> head(ans)
   origin dest    N
1:    EWR  PHX  121
2:    EWR  MIA  848
...
5:    JFK  STT  229
6:    JFK  SJU  690
```

```
> # Expressions in by
> # how many flights started late but
arrived early (or on time), started and
arrived late etc…
> ans <- flights[, .N, .(dep_delay > 0,
arr_delay > 0)]
> ans
   dep_delay arr_delay      N
1:      TRUE      TRUE  72836
2:     FALSE      TRUE  34583
3:     FALSE     FALSE 119304
4:      TRUE     FALSE  26593
```

- .SD: Subset of Data
  - It by itself is a data.table that holds the data for the current group defined using by.
  - .SD contains all the columns except the grouping columns by default.

```
> DT <- data.table(ID = c("b", "b", "b", "a", "a", "c"),
                   X1 = 1:6,
                   X2 = 7:12,
                   X3 = 13:18)
> DT
 ID X1 X2 X3
1:  b  1  7 13
2:  b  2  8 14
3:  b  3  9 15
4:  a  4 10 16
5:  a  5 11 17
6:  c  6 12 18
> DT[, print(.SD), by = ID]
 X1 X2 X3
1:  1  7 13
2:  2  8 14
3:  3  9 15
   X1 X2 X3
1:  4 10 16
2:  5 11 17
   X1 X2 X3
1:  6 12 18
Empty data.table (0 rows) of 1 col: ID
```

```
> # compute means on multiple columns for each groups
> DT[, lapply(.SD, mean), by = ID]
   ID  X1   X2   X3
1:  b 2.0  8.0 14.0
2:  a 4.5 10.5 16.5
3:  c 6.0 12.0 18.0
```

```
> ans <- flights[, head(.SD, 2), by = month]
> head(ans)
```

```
> head(ans)
   month year day dep_time dep_delay arr_time arr_delay cancelled carrier tailnum flight origin dest air_time distance hour min
1:     1 2014   1      914        14     1238        13         0      AA  N338AA      1    JFK  LAX      359     2475    9  14
2:     1 2014   1     1157        -3     1523        13         0      AA  N335AA      3    JFK  LAX      363     2475   11  57
3:     2 2014   1      859        -1     1226         1         0      AA  N783AA      1    JFK  LAX      358     2475    8  59
4:     2 2014   1     1155        -5     1528         3         0      AA  N784AA      3    JFK  LAX      358     2475   11  55
5:     3 2014   1      849       -11     1306        36         0      AA  N784AA      1    JFK  LAX      375     2475    8  49
6:     3 2014   1     1157        -3     1529        14         0      AA  N787AA      3    JFK  LAX      368     2475   11  57
```

DT[i, j, by]:
.SDcols: specify the columns

選讀

- .SDcols: specify the columns
  - e.g., .SDcols = c("arr_delay", "dep_delay") ensures that .SD contains only these two columns for each group.
  - - or !: remove columns
  - colA:colB: select consecutive columns
  - !(colA:colB) or -(colA:colB): deselect consecutive columns

```
> # compute the mean() of arr_delay and dep_delay columns grouped by origin, dest and
month.
> flights[carrier == "AA",                        # rows: Only on trips with carrier "AA"
+         lapply(.SD, mean),                      # do: compute the mean
+         by = .(origin, dest, month),            # by: for every 'origin, dest, month'
+         .SDcols = c("arr_delay", "dep_delay")] # for just those specified in .SDcols
     origin dest month  arr_delay  dep_delay
  1:    JFK  LAX     1   6.590361 14.2289157
  2:    LGA  PBI     1  -7.758621  0.3103448
  3:    EWR  LAX     1   1.366667  7.5000000
  4:    JFK  MIA     1  15.720670 18.7430168
  5:    JFK  SEA     1  14.357143 30.7500000
 ---
196:    LGA  MIA    10  -6.251799 -1.4208633
197:    JFK  MIA    10  -1.880184  6.6774194
198:    EWR  PHX    10  -3.032258 -4.2903226
199:    JFK  MCO    10 -10.048387 -1.6129032
200:    JFK  DCA    10  16.483871 15.5161290
```

DT[i, j, by]: concatenation

選讀

https://cran.r-project.org/web/packages/data.table/

Reference manual:	data.table.pdf
Vignettes:	Frequently asked questions
	Introduction to data.table
	Keys and fast binary search based subset
	Reference semantics
	Efficient reshaping using data.tables
	Secondary indices and auto indexing

https://s3.amazonaws.com/assets.datacamp.com/img/blog/data+table+cheat+sheet.pdf

```
?data.table
```

```
> DT
   ID X1 X2 X3
1:  b  1  7 13
2:  b  2  8 14
3:  b  3  9 15
4:  a  4 10 16
5:  a  5 11 17
6:  c  6 12 18
>
> # concatenate columns a and b for each group in ID
> DT[, .(NewX = c(X1, X2)), by = ID]
    ID NewX
 1:  b    1
 2:  b    2
 3:  b    3
 4:  b    7
 5:  b    8
 6:  b    9
 7:  a    4
 8:  a    5
 9:  a   10
10:  a   11
11:  c    6
12:  c   12
> DT[, .(NewX = list(c(X1, X2))), by = ID]
   ID        NewX
1:  b 1,2,3,7,8,9
2:  a  4, 5,10,11
3:  c        6,12
```

First concatenate the values with `c(a,b)` for each group, and wrap that with `list()` to return a list of all concatenated values for each group.

```
> DT[, print(c(X1, X2)), by = ID]
[1] 1 2 3 7 8 9
[1]  4  5 10 11
[1]  6 12
Empty data.table (0 rows) of 1 col: ID
>
> DT[, print(list(c(X1, X2))), by = ID]
[[1]]
[1] 1 2 3 7 8 9

[[1]]
[1]  4  5 10 11

[[1]]
[1]  6 12

Empty data.table (0 rows) of 1 col: ID
```