E08

R語言
時間序列資料的視覺化
Visualization of Time Series Data

吳漢銘
國立政治大學 統計學系

https://hmwu.idv.tw

大綱

- CRAN Task View: Time Series Analysis
- 學習資源及參考書目
- R軟體中的時間序列物件
  - 資料輸入與輸出
  - 資料處理: `lubridate` 套件
- **時間序列資料視覺化: 線圖、熱圖、桑基圖**
- 象徵性資料分析 (Symbolic Data Analysis)

CRAN Task View: Time Series Analysis

CRAN Task View: Time Series Analysis

Maintainer:	Rob J Hyndman, Rebecca Killick
Contact:	Rob.Hyndman at monash.edu
Version:	2023-07-26
URL:	https://CRAN.R-project.org/view=TimeSeries
Source:	https://github.com/cran-task-views/TimeSeries/
Contributions:	Suggestions and improvements for this task view are very welcome and can be made through issues or pull requests on GitHub or via e-mail to the maintainer address. For further details see the Contributing guide.
Citation:	Rob J Hyndman, Rebecca Killick (2023). CRAN Task View: Time Series Analysis. Version 2023-07-26. URL https://CRAN.R-project.org/view=TimeSeries.
Installation:	The packages from this task view can be installed automatically using the ctv package. For example, `ctv::install.views("TimeSeries", coreOnly = TRUE)` installs all the core packages or `ctv::update.views("TimeSeries")` installs all packages that are not yet installed and up-to-date. See the CRAN Task View Initiative for more details.

https://cran.r-project.org/web/views/TimeSeries.html

Base R ships with a lot of functionality useful for time series, in particular in the stats package. This is complemented by many packages on CRAN, which are briefly summarized below. There is overlap between the tools for time series and those designed for specific domains including Econometrics, Finance and Environmetrics.

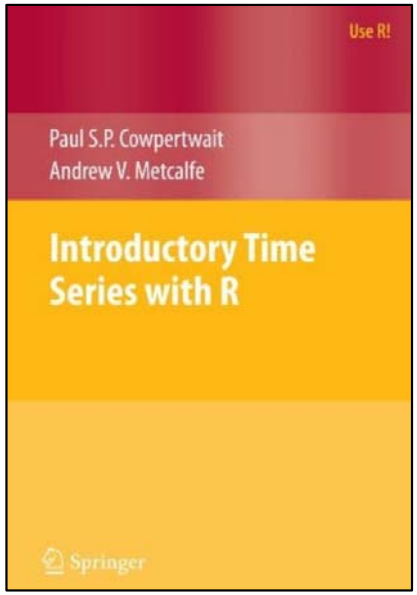
The packages in this view can be roughly structured into the following topics. If you think that some package is missing from the list, please let us know, either via e-mail to the maintainer or by submitting an issue or pull request in the GitHub repository linked above.

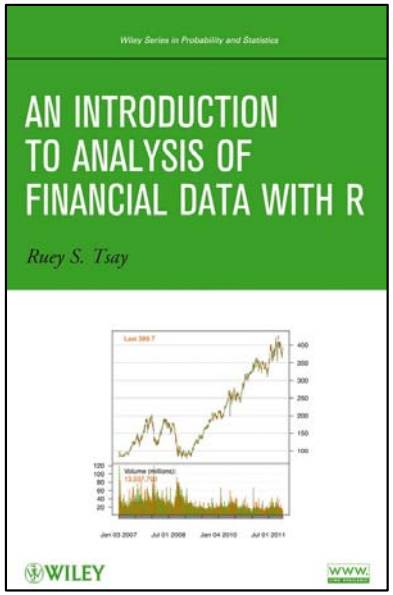
Basics

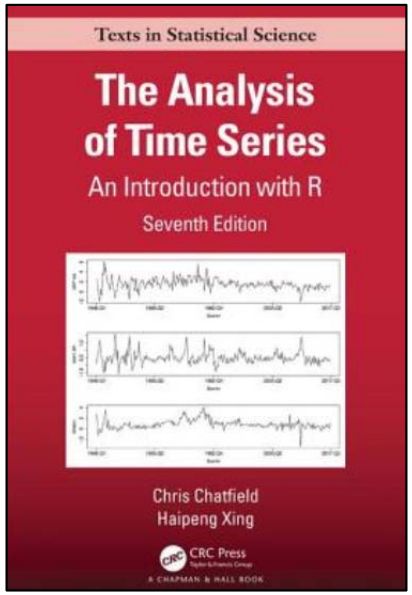
- *Infrastructure* : Base R contains substantial infrastructure for representing and analysing time series data. The fundamental class is `"ts"` that can represent regularly spaced time series (using numeric time stamps). Hence, it is particularly well-suited for annual,

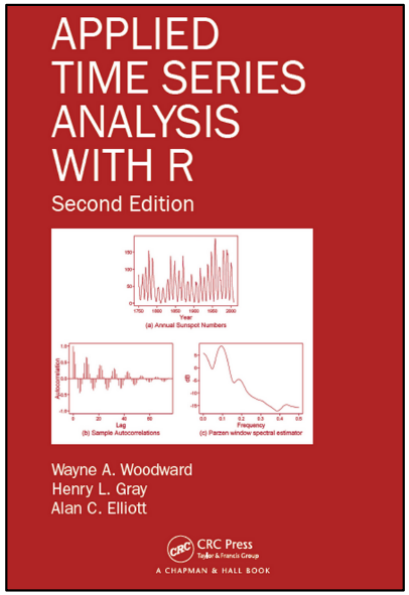
Basics、Times and Dates、Time Series Classes、Forecasting and Univariate Modeling、Frequency analysis、Decomposition and Filtering、Seasonality、Stationarity, Unit Roots, and Cointegration、Nonlinear Time Series Analysis、Entropy、Dynamic Regression Models、Multivariate Time Series Models、Analysis of large groups of time series、Functional time series、Matrix and tensor-valued time series、Continuous time models、Resampling、Time Series Data、Miscellaneous

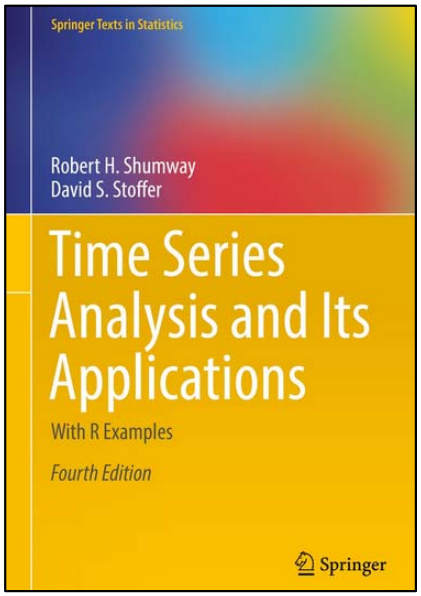
Time Series Analysis Uing R

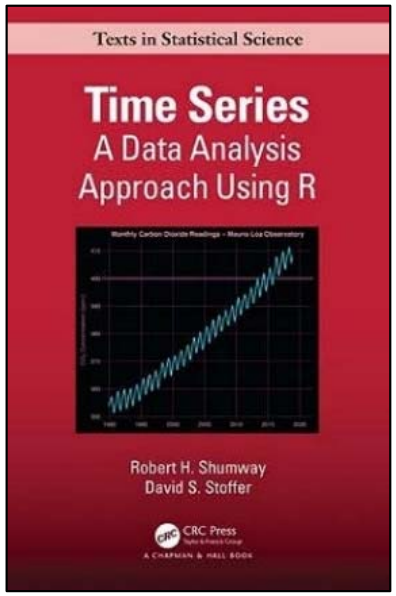


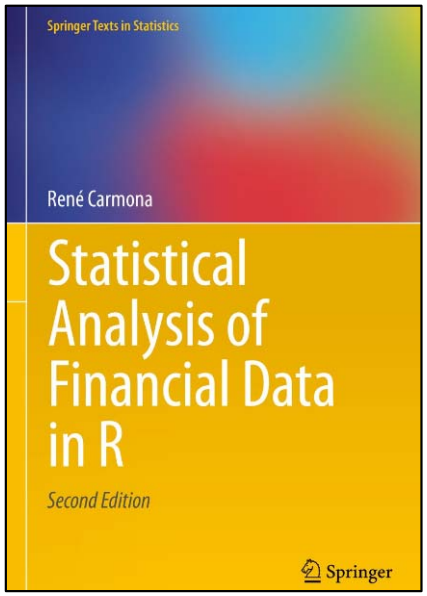


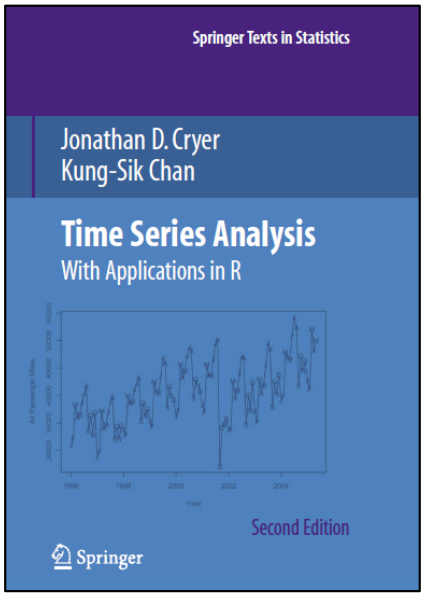




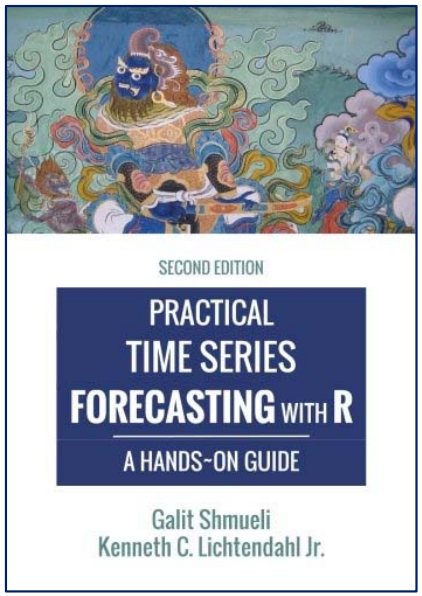


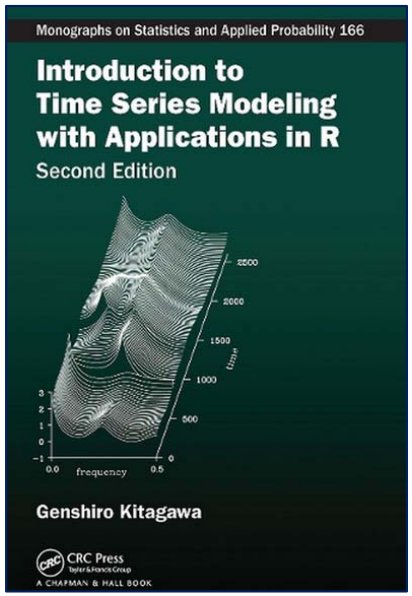


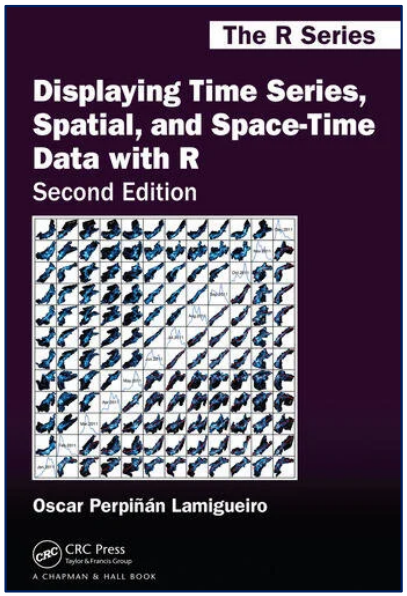


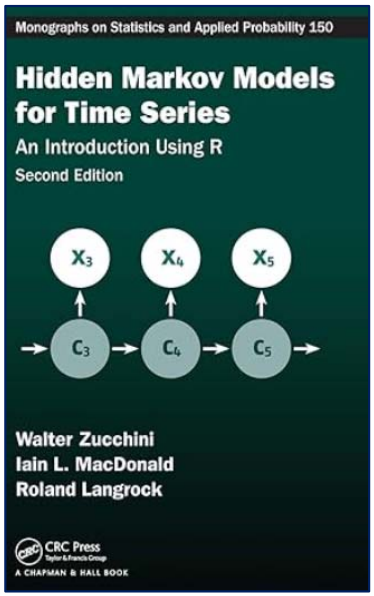


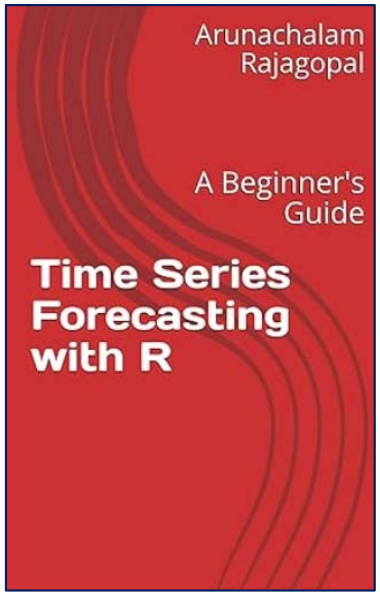
Time Series Analysis Using R

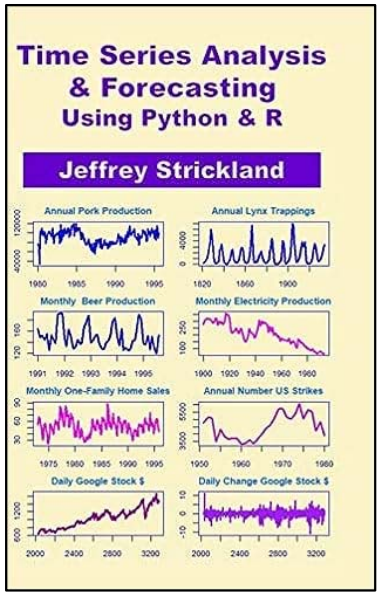


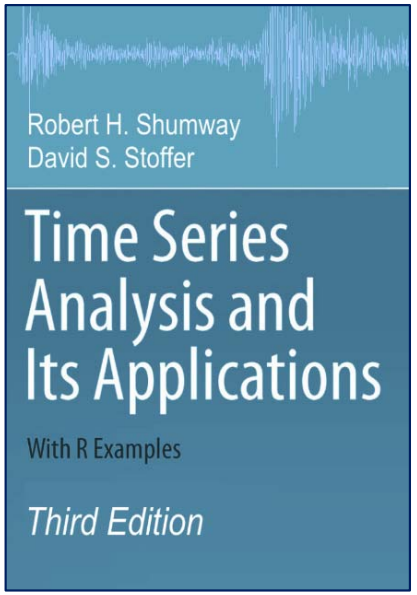


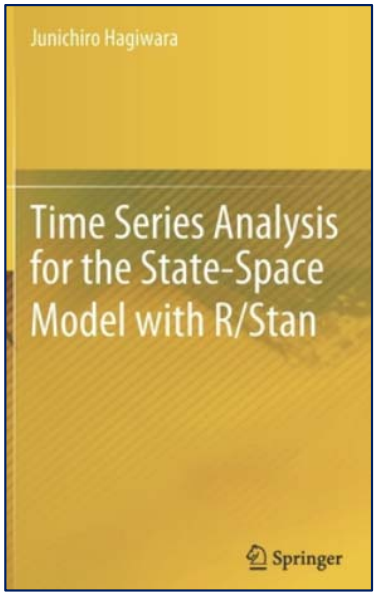




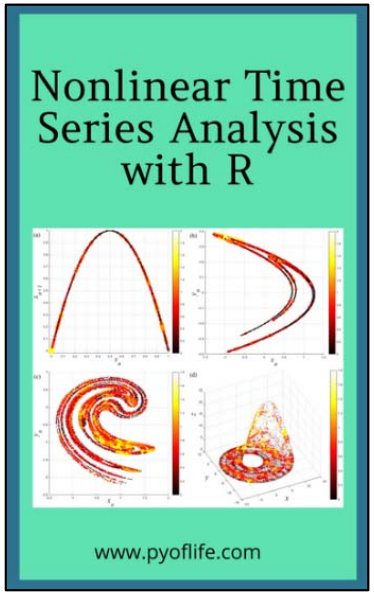


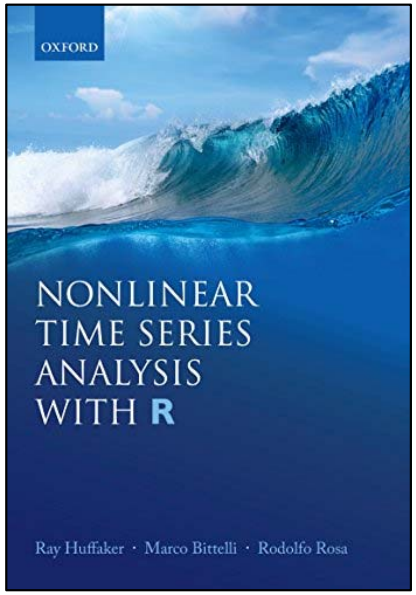


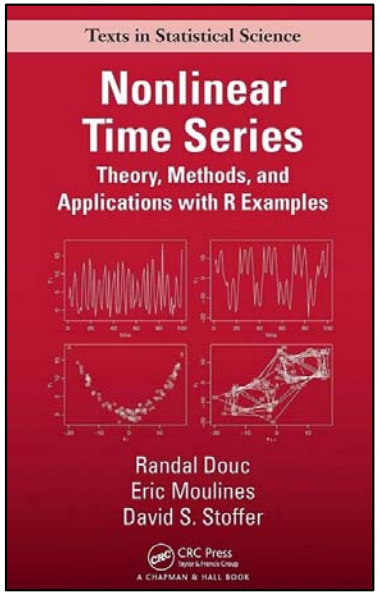


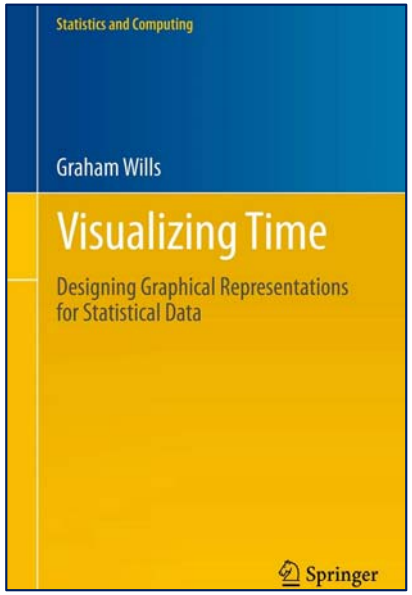


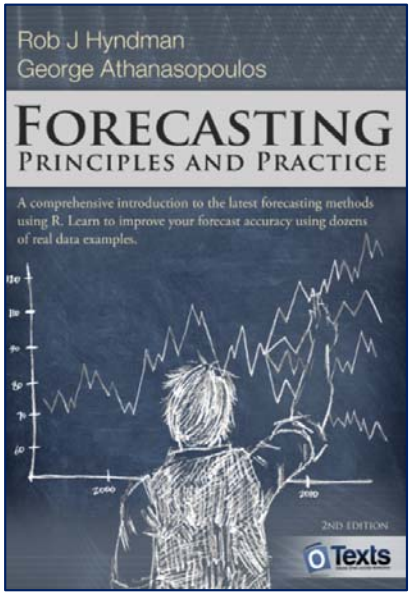
Time Series Analysis Using R

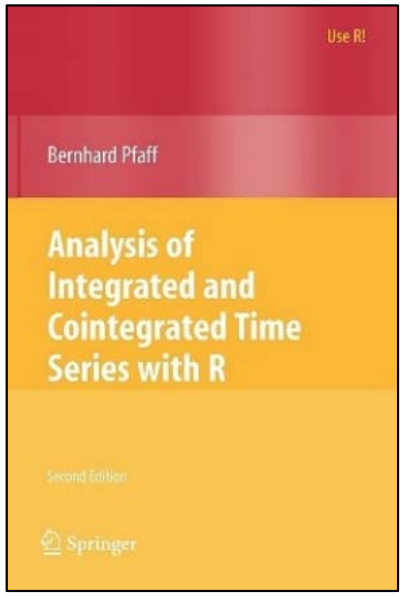


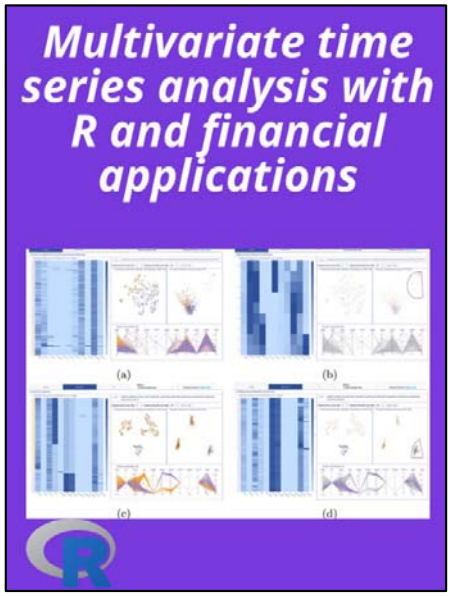


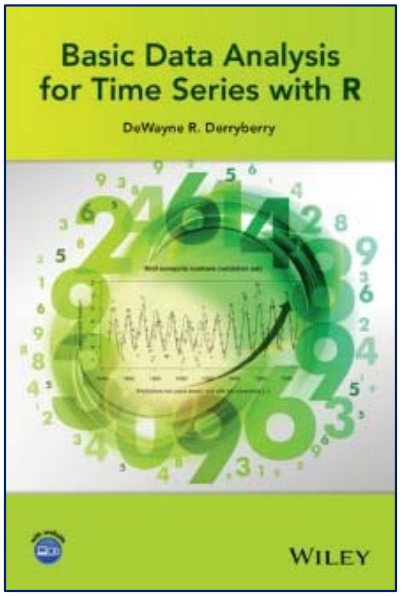












Overview of Time Series Objects in R

時間序列類別	套件	說明
`ts, mts`	**stats**: R statistical functions	This package contains functions for statistical calculations and random number generation.
`timeSeries`	**timeSeries**: Financial Time Series Objects (Rmetrics)	'S4' classes and various tools for financial time series: Basic functions such as scaling and sorting, subsetting, mathematical operations and statistical functions.
`ti`	**tis**: Time Indexes and Time Indexed Series	Functions and S3 classes for time indexes and time indexed series, which are compatible with FAME frequencies.
`zoo`	**zoo**: S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)	An S3 class with methods for totally ordered indexed observations. It is particularly aimed at irregular time series of numeric vectors/matrices and factors. zoo's key design goals are independence of a particular index/date/time class and consistency with ts and base R by providing methods to extend standard generics.
`xts`	**xts**: eXtensible Time Series	Provide for uniform handling of R's different time-based data classes by extending zoo, maximizing native format information preservation and allowing for user level customization and extension, while simplifying cross-class interoperability.

Overview of Date and Date.Time Objects in R

類別	套件	說明
Date	base	Represent calendar dates as the number of days since 1970.01.01
POSIXct	base	Represent calendar dates and times within the day as the (signed) number of seconds since the beginning of 1970 as a numeric vector. Supports various time zone specifications (e.g. GMT, PST, EST etc.)
POSIXlt	base	Represents local dates and times within the day as named list of vectors with date.time components.
chron	chron: Chronological Objects which Can Handle Dates and Times	Provides chronological objects which can handle dates and times.
Yearmon (yearqtr)	Zoo: S3 Infrastructure for Regular and Irregular Time Series (Z's Ordered Observations)	Represent monthly (quarterly) data. Internally it holds the data as year plus 0 for January, 1/12 for February, 2/12 for March (Quarter 1, 1/4 for Quarter 2) and so on in order that its internal representation is the same as tsclass with frequency = 12 (4).
timeDate	timeDate: Rmetrics - Chronological and Calendar Objects	The 'timeDate' class fulfils the conventions of the ISO 8601 standard as well as of the ANSI C and POSIX standards. Beyond these standards it provides the "Financial Center" concept which allows to handle data records collected in different time zones and mix them up to have always the proper time stamps with respect to your personal financial center, or alternatively to the GMT reference time. It can thus also handle time stamps from historical data records from the same time zone, even if the financial centers changed day light saving times at different calendar dates.

日期 Dates
The Date Class (base R)

- R以"Date" 類別表示(不包括時間)日期: 年月日。
- Internally, Date objects are stored as the number of days since **January 1, 1970**, using negative numbers for earlier dates.
- The **as.numeric** function can be used to convert a Date object to its internal form.

Code	Value
%d	Day of the month (decimal number)
%m	Month (decimal number)
%b	Month (abbreviated)
%B	Month (full name)
%y	Year (2 digit)
%Y	Year (4 digit)

```
> as.numeric(as.Date("1970/1/1"))
[1] 0
```

```
> as.Date("1985-6-16")
[1] "1985-06-16"
> as.Date("2019/02/17")
[1] "2019-02-17"
> as.Date(1000, origin = "1900-01-01")
[1] "1902-09-28"
> as.Date("2/15/2011", format = "%m/%d/%Y")
[1] "2011-02-15"
>
> as.Date("April 26, 1993", format = "%B %d, %Y")
[1] "1993-04-26"
> as.Date("22JUN01", format = "%d%b%y")
[1] "2001-06-22"
> seq(as.Date('1976-7-4'), by = 'days', length = 10)
 [1] "1976-07-04" "1976-07-05" "1976-07-06" "1976-07-07" "1976-07-08" "1976-07-09" "1976-07-10"
 [8] "1976-07-11" "1976-07-12" "1976-07-13"
> seq(as.Date('2010-2-1'), to = as.Date('2010-4-1'), by = '2 weeks')
[1] "2010-02-01" "2010-02-15" "2010-03-01" "2010-03-15" "2010-03-29"
```

請先執行!!

```
> (lct <- Sys.getlocale("LC_TIME"))
[1] "C"
> Sys.setlocale("LC_TIME", "C")
[1] "C"
```

```
> Sys.getlocale("LC_TIME")
[1] "Chinese (Traditional)_Taiwan.950"
```

日期時間
The POSIXt classes (base R)

- R的"時間"用"POSIXct"或 "POSIXlt" 類別表示，
- 內部時間是以 "1970年1月1日" 起至今的秒數表示。
- (UTC, Universal Time, Coordinated, 世界協調時間)
- (GMT, Greenwich Mean Time, 格林威治標準時間)

```
> Sys.time()
[1] "2028-10-14 21:16:07 台北標準時間"
# extract date
> substr(as.character(Sys.time()), 1, 10)
[1] "2028-10-14"
# extract time
> substr(as.character(Sys.time()), 12, 19)
[1] "21:16:07"
> date()
[1] "Tue Oct 14 21:16:09 2028"
```

```
> now <- Sys.time()
> as.POSIXct(now)
[1] "2027-06-03 17:46:44 CST"
> as.POSIXlt(now)
[1] "2027-06-03 17:46:44 CST"
> class(now)
[1] "POSIXct" "POSIXt"
```

```
sec, min, hour,
mday (# day number within the month),
mon (#January=0),
year (#+1900),
wday (#day of the week starting at 0=sunday),
yday (#day of the year after 1 january=0)
```

```
> my.date <- as.POSIXlt(Sys.time())
> my.date
[1] "2028-10-14 21:18:31 台北標準時間"
> my.date$sec
[1] 31.304
> my.date$min
[1] 18
> my.date$hour
[1] 21
```

```
> my.date$mday
[1] 14
> my.date$mon
[1] 9
> my.date$year + 1900
[1] 2028
> my.date$wday
[1] 2
> my.date$yday
[1] 287
```

Two POSIXt sub-classes: POSIXlt / POSIXct 類別

- Functions to convert between character representations and objects of classes "POSIXlt" and "POSIXct" representing calendar dates and times.
  - Character input is first converted to class "POSIXlt" by **strptime**.
  - Numeric input is first converted to "POSIXct".
- Any conversion that needs to go between the two date-time classes requires a time zone: conversion from "POSIXlt" to "POSIXct" will validate times in the selected time zone.

Code	Meaning	Code	Meaning
%a	Abbreviated weekday	%A	Full weekday
%b	Abbreviated month	%B	Full month
%c	Locale-specific date and time	%d	Decimal date
%H	Decimal hours (24 hour)	%I	Decimal hours (12 hour)
%j	Decimal day of the year	%m	Decimal month
%M	Decimal minute	%p	Locale-specific AM/PM
%S	Decimal second	%U	Decimal week of the year (starting on Sunday)
%w	Decimal Weekday (0=Sunday)	%W	Decimal week of the year (starting on Monday)
%x	Locale-specific Date	%X	Locale-specific Time
%y	2-digit year	%Y	4-digit year
%z	Offset from GMT	%Z	Time zone (character)

```
> as.POSIXct("1969-12-31 23:59:59", format = "%Y-%m-%d %H:%M:%S", tz = "UTC")
[1] "1969-12-31 23:59:59 UTC"
> as.POSIXlt(Sys.time(), "GMT")
[1] "2017-08-27 13:17:45 GMT"
```

strptime {base}: Date-time Conversion Functions to and from Character

```
> x1 <- c("20040227", "20050412", "19930922")
> strptime(x1, format = "%Y%m%d")
[1] "2004-02-27 CST" "2005-04-12 CST" "1993-09-22 CST"
>
> x2 <- c("27/02/2004", "27/02/2005", "14/01/2003")
> strptime(x2, format = "%d/%m/%Y")
[1] "2004-02-27 CST" "2005-02-27 CST" "2003-01-14 CST"
>
> x3 <- c("1jan1960", "2jan1960", "31mar1960", "30jul1960")
> strptime(x3, "%d%b%Y")
[1] "1960-01-01 CST" "1960-01-02 CST" "1960-03-31 CST" "1960-07-30 CDT"
```

```
> dates <- c("02/27/92", "02/27/92", "01/14/92", "02/28/92", "02/01/92")
> times <- c("23:03:20", "22:29:56", "01:03:30", "18:21:03", "16:56:26")
> x <- paste(dates, times)
> strptime(x, "%m/%d/%y %H:%M:%S")
[1] "1992-02-27 23:03:20 CST" "1992-02-27 22:29:56 CST"
[3] "1992-01-14 01:03:30 CST" "1992-02-28 18:21:03 CST"
[5] "1992-02-01 16:56:26 CST"
```

時間序列資料輸入與輸出: 純文字檔

mydate.txt

```
1;73;2017/01/27 11:30:20
2;52;2017/03/05 12:01:40
3;57;2017/05/12 03:20:00
1;74;2017/08/27 14:00:00
2;51;2017/10/17 21:03:50
3;60;2017/12/08 08:40:30
```

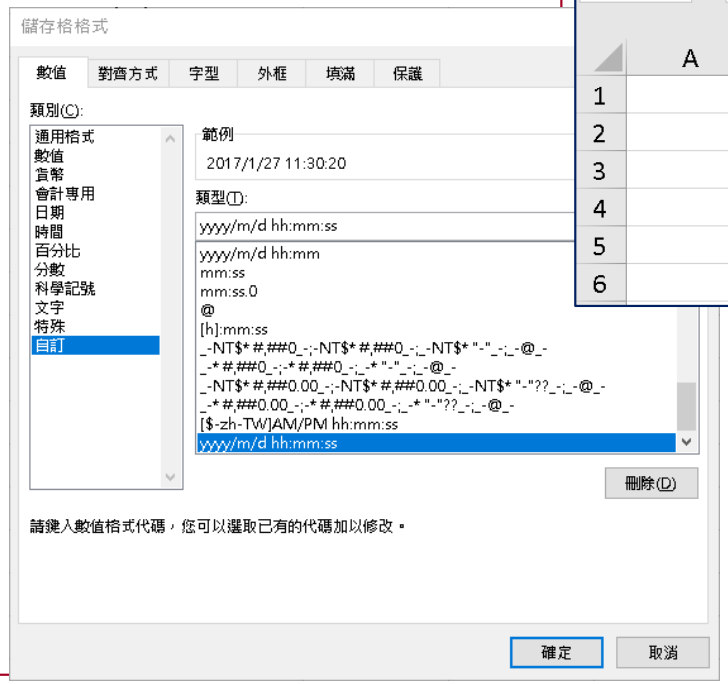
```
> mydate <- read.table("mydate.txt",
                       sep = ";")
> mydate
  V1 V2                   V3
1  1 73 2017/01/27 11:30:20
2  2 52 2017/03/05 12:01:40
3  3 57 2017/05/12 03:20:00
4  1 74 2017/08/27 14:00:00
5  2 51 2017/10/17 21:03:50
6  3 60 2017/12/08 08:40:30
> lapply(mydate, class)
$V1
[1] "integer"

$V2
[1] "integer"

$V3
[1] "factor"
```

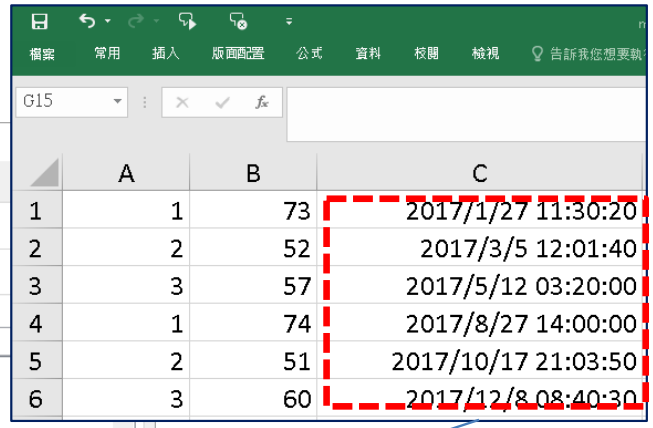
```
> # 方法一
> varNames <- c("ID", "Values", "DateTime")
> mydate <- read.table("mydate.txt", sep = ";",
                       col.names = varNames)
> mydate
  ID Values             DateTime
1  1     73 2017/01/27 11:30:20
2  2     52 2017/03/05 12:01:40
3  3     57 2017/05/12 03:20:00
4  1     74 2017/08/27 14:00:00
5  2     51 2017/10/17 21:03:50
6  3     60 2017/12/08 08:40:30
> lapply(mydate, class)
$ID
[1] "integer"
$Values
[1] "integer"
$DateTime
[1] "factor"

> mydate$DateTime <- strptime(mydate$DateTime,
                              "%Y/%m/%d %H:%M:%S")
> lapply(mydate, class)
$ID
[1] "integer"
$Values
[1] "integer"
$DateTime
[1] "POSIXlt" "POSIXt"
```

> # 方法二
自定讀取類別

時間序列資料輸入與輸出: EXCEL檔

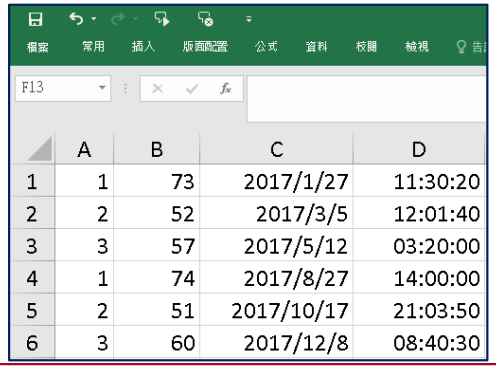
```
> library(readxl)
> mydate <- read_excel("data/mydate.xlsx", col_names = FALSE)
> mydate
# A tibble: 6 × 3
   ...1  ...2 ...3
  <dbl> <dbl> <dttm>
1     1    73 2017-01-27 11:30:20
...
6     3    60 2017-12-08 08:40:30
> lapply(mydate, class)
$...1
[1] "numeric"

$...2
[1] "numeric"

$...3
[1] "POSIXct" "POSIXt"
```





	A	B	C
1	1	73	2017/1/27 11:30:20
2	2	52	2017/3/5 12:01:40
3	3	57	2017/5/12 03:20:00
4	1	74	2017/8/27 14:00:00
5	2	51	2017/10/17 21:03:50
6	3	60	2017/12/8 08:40:30

滑鼠右鍵



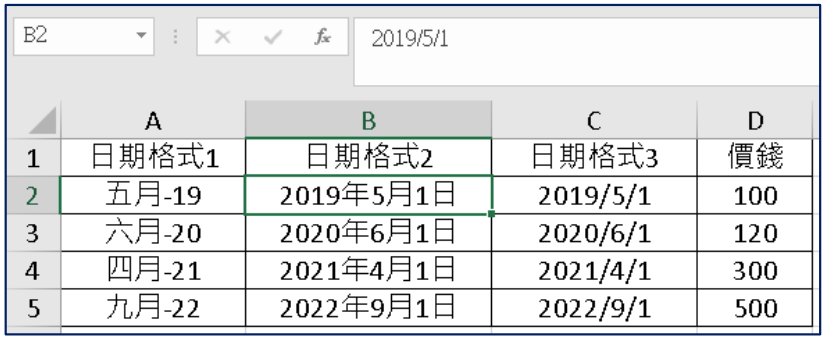
	A	B	C	D
1	1	73	2017/1/27	11:30:20
2	2	52	2017/3/5	12:01:40
3	3	57	2017/5/12	03:20:00
4	1	74	2017/8/27	14:00:00
5	2	51	2017/10/17	21:03:50
6	3	60	2017/12/8	08:40:30

```
> mydate2 <- read_excel("mydate2.xlsx", col_names = FALSE)
> mydate2
# A tibble: 6 × 4
   ...1  ...2 ...3                ...4
  <dbl> <dbl> <dttm>              <dttm>
1     1    73 2017-01-27 00:00:00 1899-12-31 11:30:20
...
6     3    60 2017-12-08 00:00:00 1899-12-31 08:40:30
```

Further modification is needed

```
library(writexl)
write_xlsx(iris, path = "iris.xlsx")
write_xlsx(list(sheet1 = iris[, 1:2], sheet2 = iris[, 3:4]), path = "iris.xlsx")
```

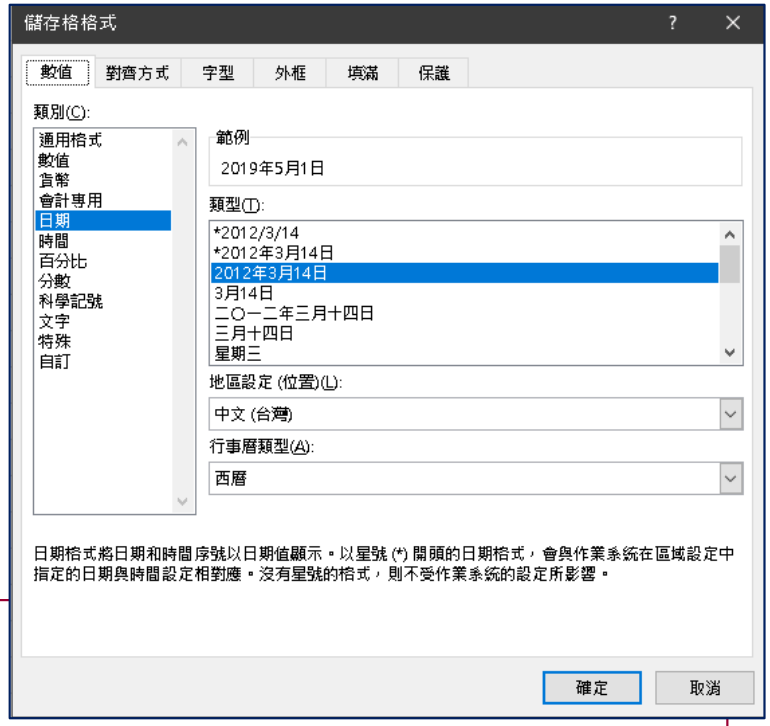
時間序列資料輸入與輸出: 中文格式



B2 | 2019/5/1

	A	B	C	D
1	日期格式1	日期格式2	日期格式3	價錢
2	五月-19	2019年5月1日	2019/5/1	100
3	六月-20	2020年6月1日	2020/6/1	120
4	四月-21	2021年4月1日	2021/4/1	300
5	九月-22	2022年9月1日	2022/9/1	500

TimeData-cht.xlsx




```
> library(readxl)
> TimeData_xlsx <- read_excel("data/TimeData-cht.xlsx")
> TimeData_xlsx
# A tibble: 4 × 4
  日期格式1           日期格式2           日期格式3           價錢
  <dttm>              <dttm>              <dttm>              <dbl>
1 2019-05-01 00:00:00 2019-05-01 00:00:00 2019-05-01 00:00:00   100
2 2020-06-01 00:00:00 2020-06-01 00:00:00 2020-06-01 00:00:00   120
3 2021-04-01 00:00:00 2021-04-01 00:00:00 2021-04-01 00:00:00   300
4 2022-09-01 00:00:00 2022-09-01 00:00:00 2022-09-01 00:00:00   500
> str(TimeData_xlsx)
tibble [4 × 4] (S3: tbl_df/tbl/data.frame)
 $ 日期格式1: POSIXct[1:4], format: "2019-05-01" "2020-06-01" "2021-04-01" ...
 $ 日期格式2: POSIXct[1:4], format: "2019-05-01" "2020-06-01" "2021-04-01" ...
 $ 日期格式3: POSIXct[1:4], format: "2019-05-01" "2020-06-01" "2021-04-01" ...
 $ 價錢     : num [1:4] 100 120 300 500
```

時間序列資料輸入與輸出: 中文格式

TimeData-cht.csv

日期格式1,	日期格式2,	日期格式3,	價錢
五月-19,	2019年5月1日,	2019/5/1,	100
六月-20,	2020年6月1日,	2020/6/1,	120
四月-21,	2021年4月1日,	2021/4/1,	300
九月-22,	2022年9月1日,	2022/9/1,	500

TimeData-cht.csv

```
> TimeData_csv <- read.csv("data/TimeData-cht.csv",
+                          stringsAsFactors = FALSE,
+                          fileEncoding = "big5")
> TimeData_csv
  日期格式1     日期格式2 日期格式3 價錢
1   五月-19 2019年5月1日   2019/5/1  100
2   六月-20 2020年6月1日   2020/6/1  120
3   四月-21 2021年4月1日   2021/4/1  300
4   九月-22 2022年9月1日   2022/9/1  500
> str(TimeData_csv)
'data.frame':       4 obs. of  4 variables:
 $ 日期格式1: chr  "五月-19" "六月-20" "四月-21" "九月-22"
 $ 日期格式2: chr  "2019年5月1日" "2020年6月1日" ...
 $ 日期格式3: chr  "2019/5/1" "2020/6/1" "2021/4/1" ...
 $ 價錢     : int  100 120 300 500
```

```
> TimeData_csv$日期格式2 <- as.POSIXct(strptime(TimeData_csv$日期格式2, format="%Y年%m月%d日"))
> TimeData_csv$日期格式2
[1] "2019-05-01 CST" "2020-06-01 CST" "2021-04-01 CST" "2022-09-01 CST"
>
> TimeData_csv$日期格式3 <- as.POSIXct(TimeData_csv$日期格式3, format="%Y/%m/%d")
> TimeData_csv$日期格式3
[1] "2019-05-01 CST" "2020-06-01 CST" "2021-04-01 CST" "2022-09-01 CST"
```

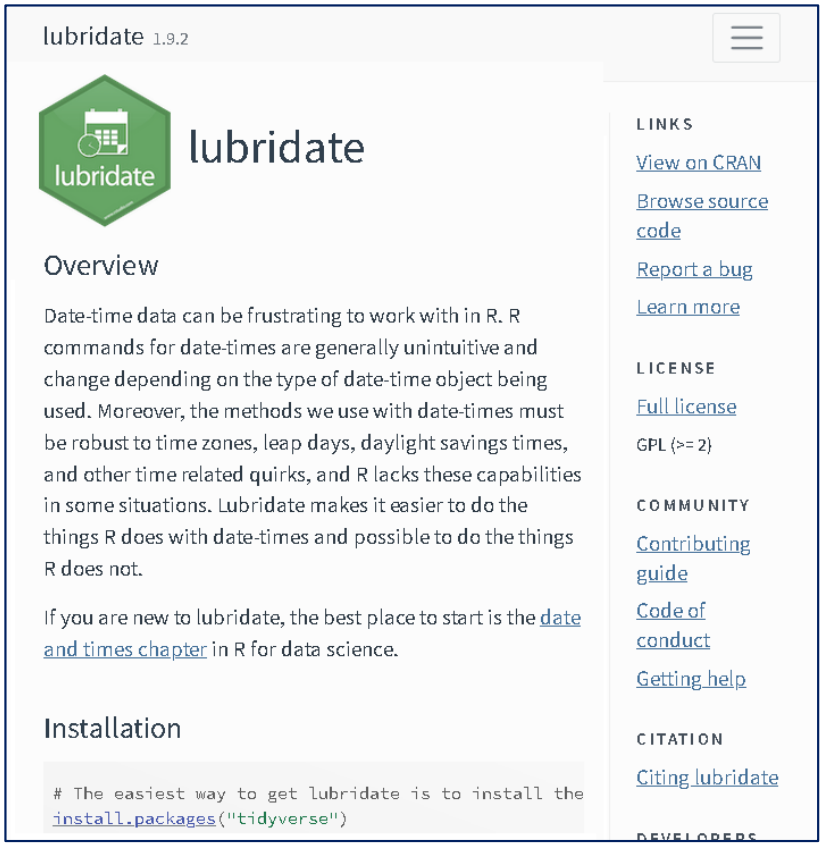
時間序列資料輸入與輸出: 中文格式

```
> months_map <- c("一月" = 1, "二月" = 2, "三月" = 3, "四月" = 4,
+                 "五月" = 5, "六月" = 6, "七月" = 7, "八月" = 8,
+                 "九月" = 9, "十月" = 10, "十一月" = 11, "十二月" = 12)
> date_str <- TimeData_csv$日期格式1
> parts <- strsplit(date_str, "-"); parts
> month <- months_map[sapply(parts, function(x) x[1])]; month
> year <- as.numeric(sapply(parts, function(x) x[2])) + 2000; year
> date_str <- paste(year, month, "1", sep = "-")
> TimeData_csv$日期格式1 <- as.POSIXct(date_str, format="%Y-%m-%d")
> TimeData_csv
   日期格式1  日期格式2  日期格式3 價錢
1 2019-05-01 2019-05-01 2019-05-01  100
2 2020-06-01 2020-06-01 2020-06-01  120
3 2021-04-01 2021-04-01 2021-04-01  300
4 2022-09-01 2022-09-01 2022-09-01  500
> str(TimeData_csv)
'data.frame':       4 obs. of  4 variables:
 $ 日期格式1: POSIXct, format: "2019-05-01" "2020-06-01" "2
 $ 日期格式2: POSIXct, format: "2019-05-01" "2020-06-01" "2
 $ 日期格式3: POSIXct, format: "2019-05-01" "2020-06-01" "2
 $ 價錢     : int  100 120 300 500
```

```
> months_map
  一月   二月   三月   四月   五月   六月   七月   八
     1      2      3      4      5      6      7
> parts
[[1]]
[1] "五月" "19"

[[2]]
[1] "六月" "20"

[[3]]
[1] "四月" "21"

[[4]]
[1] "九月" "22"

> month
五月 六月 四月 九月
   5    6    4    9
> year
[1] 2019 2020 2021 2022
> date_str
[1] "2019-5-1" "2020-6-1" "2021-4-1" "2022-9-1"
```

時間序列資料的處理

lubridate: Make Dealing with Dates a Little Easier



Garrett Grolemund, Hadley Wickham (2011). Dates and Times Made Easy with **lubridate**. *Journal of Statistical Software*, 40(3), 1-25.

Base R method	**lubridate** method
`date <- as.POSIXct("01-01-2010", format = "%d-%m-%Y", tz = "UTC")`	`date <- dmy("01-01-2010")`
`as.numeric(format(date, "%m"))` or `as.POSIXlt(date)$month + 1`	`month(date)`
`date <- as.POSIXct(format(date, "%Y-2-%d"), tz = "UTC")`	`month(date) <- 2`

Table 1: **lubridate** provides a simple way to parse a date into R, extract the month value and change it to February.

Base R method	**lubridate** method
`date <- seq(date, length = 2, by = "-1 day")[2]`	`date <- date - days(1)`
`as.POSIXct(format(as.POSIXct(date), tz = "UTC"), tz = "GMT")`	`with_tz(date, "GMT")`

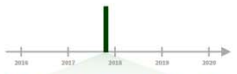
Table 2: **lubridate** easily displays a date one day earlier and in the GMT time zone.

- https://lubridate.tidyverse.org/
- https://rawgit.com/rstudio/cheatsheets/main/lubridate.pdf
- https://cran.r-project.org/web/packages/lubridate/index.html
- Do more with dates and times in R: https://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html

Cheatsheet, lubridate

Dates and times with lubridate : : CHEATSHEET

Date-times



`2017-11-28 12:00:00`

`2017-11-28 12:00:00`
A **date-time** is a point on the timeline, stored as the number of seconds since 1970-01-01 00:00:00 UTC

*dt <- **as_datetime**(1511870400)*
"2017-11-28 12:00:00 UTC"

`2017-11-28`
A **date** is a day stored as the number of days since 1970-01-01

*d <- **as_date**(17498)*
"2017-11-28"

`12:00:00`
An hms is a **time** stored as the number of seconds since 00:00:00

*t <- hms::**as_hms**(85)*
00:01:25

PARSE DATE-TIMES (Convert strings or numbers to date-times)

1. Identify the order of the year (**y**), month (**m**), day (**d**), hour (**h**), minute (**m**) and second (**s**) elements in your data.
2. Use the function below whose name replicates the order. Each accepts a tz argument to set the time zone, e.g. ymd(x, tz = "UTC").

`2017-11-28T14:02:00` **ymd_hms(), ymd_hm(), ymd_h().** ymd_hms("2017-11-28T14:02:00")

`2017-22-12 10:00:00` **ydm_hms(), ydm_hm(), ydm_h().** ydm_hms("2017-22-12 10:00:00")

`11/28/2017 1:02:03` **mdy_hms(), mdy_hm(), mdy_h().** mdy_hms("11/28/2017 1:02:03")

`1 Jan 2017 23:59:59` **dmy_hms(), dmy_hm(), dmy_h().** dmy_hms("1 Jan 2017 23:59:59")

`20170131` **ymd(), ydm().** ymd(20170131)

`July 4th, 2000` **mdy(), myd().** mdy("July 4th, 2000")

`4th of July '99` **dmy(), dym().** dmy("4th of July '99")

`2001: Q3` **yq()** Q for quarter. yq("2001: Q3")

`07-2020` **my(), ym().** my("07-2020")

`2:01` hms::**hms()** Also lubridate::**hms(), hm()** and **ms()**, which return periods.* hms::hms(seconds = 0, minutes = 1, hours = 2)

`2017.5` **date_decimal**(decimal, tz = "UTC") date_decimal(2017.5)

now(tzone = "") Current time in tz (defaults to system tz). now()

today(tzone = "") Current date in a tz (defaults to system tz). today()

fast_strptime() Faster strptime. fast_strptime("9/1/01", "%y/%m/%d")

parse_date_time() Easier strptime. parse_date_time("09-01-01", "ymd")

GET AND SET COMPONENTS

Use an accessor function to get a component. `d ## "2017-11-28"` `day(d) ## 28`

Assign into an accessor function to change a component in place. `day(d) <- 1` `d ## "2017-11-01"`

`2018-01-31 11:59:59` **date(x)** Date component. date(dt)

`2018-01-31 11:59:59` **year(x)** Year. year(dt) **isoyear(x)** The ISO 8601 year. **epiyear(x)** Epidemiological year.

`2018-01-31 11:59:59` **month**(x, label, abbr) Month. month(dt)

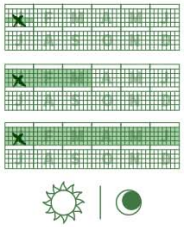
`2018-01-31 11:59:59` **day(x)** Day of month. day(dt) **wday**(x, label, abbr) Day of week. **qday(x)** Day of quarter.

`2018-01-31 11:59:59` **hour(x)** Hour. hour(dt)

`2018-01-31 11:59:59` **minute(x)** Minutes. minute(dt)

`2018-01-31 11:59:59` **second(x)** Seconds. second(dt)

`2018-01-31 11:59:59 UTC` **tz(x)** Time zone. tz(dt)



week(x) Week of the year. week(dt) **isoweek()** ISO 8601 week. **epiweek()** Epidemiological week.

quarter(x) Quarter. quarter(dt)

semester(x, with_year = FALSE) Semester. semester(dt)

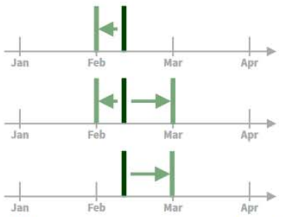
am(x) Is it in the am? am(dt) **pm(x)** Is it in the pm? pm(dt)

dst(x) Is it daylight savings? dst(d)

leap_year(x) Is it a leap year? leap_year(d)

update(object, ..., simple = FALSE) update(dt, mday = 2, hour = 1)

Round Date-times



floor_date(x, unit = "second") Round down to nearest unit. floor_date(dt, unit = "month")

round_date(x, unit = "second") Round to nearest unit. round_date(dt, unit = "month")

ceiling_date(x, unit = "second", change_on_boundary = NULL) Round up to nearest unit. ceiling_date(dt, unit = "month")

Valid units are second, minute, hour, day, week, month, bimonth, quarter, season, halfyear and year.

rollback(dates, roll_to_first = FALSE, preserve_hms = TRUE) Roll back to last day of previous month. Also **rollforward()**. rollback(dt)

Stamp Date-times

stamp() Derive a template from an example string and return a new function that will apply the template to date-times. Also **stamp_date()** and **stamp_time()**.

1. Derive a template, create a function
   sf <- stamp("Created Sunday, Jan 17, 1999 3:34")
2. Apply the template to dates
   sf(ymd("2010-04-05"))
   ## [1] "Created Monday, Apr 05, 2010 00:00"

Tip: use a date with day > 12

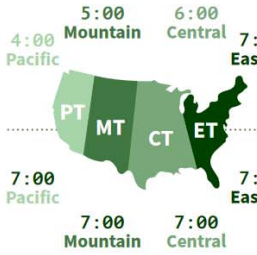
Time Zones

R recognizes ~600 time zones. Each encodes the time zone, Daylight Savings Time, and historical calendar variations for an area. R assigns *one* time zone per vector.

Use the **UTC** time zone to avoid Daylight Savings.

OlsonNames() Returns a list of valid time zone names. OlsonNames()

Sys.timezone() Gets current time zone.



with_tz(time, tzone = "") Get the **same date-time** in a new time zone (a new clock time). Also **local_time**(dt, tz, units). with_tz(dt, "US/Pacific")

force_tz(time, tzone = "") Get the **same clock time** in a new time zone (a new date-time). Also **force_tzs()**. force_tz(dt, "US/Pacific")

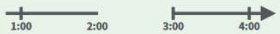
Cheatsheet, lubridate

Math with Date-times — Lubridate provides three classes of timespans to facilitate math with dates and date-times.

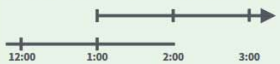
Math with date-times relies on the **timeline**, which behaves inconsistently. Consider how the timeline behaves during:

A normal day
nor <- ymd_hms("2018-01-01 01:30:00",tz="US/Eastern")

The start of daylight savings (spring forward)
gap <- ymd_hms("2018-03-11 01:30:00",tz="US/Eastern")



The end of daylight savings (fall back)
lap <- ymd_hms("2018-11-04 00:30:00",tz="US/Eastern")

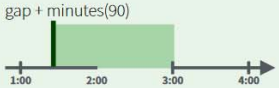


Leap years and leap seconds
leap <- ymd("2019-03-01")

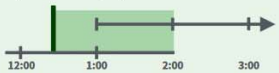
Periods track changes in clock times, which ignore time line irregularities.

nor + minutes(90)

gap + minutes(90)



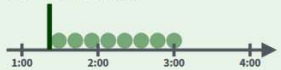
lap + minutes(90)



leap + years(1)

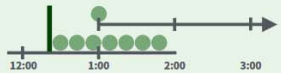
Durations track the passage of physical time, which deviates from clock time when irregularities occur.

nor + dminutes(90)



gap + dminutes(90)

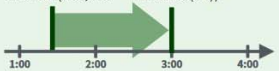
lap + dminutes(90)



leap + dyears(1)

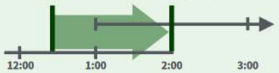
Intervals represent specific intervals of the timeline, bounded by start and end date-times.

interval(nor, nor + minutes(90))



interval(gap, gap + minutes(90))

interval(lap, lap + minutes(90))



interval(leap, leap + years(1))

Not all years are 365 days due to **leap days**.

Not all minutes are 60 seconds due to **leap seconds**.

It is possible to create an imaginary date by adding **months**, e.g. February 31st

```
jan31 <- ymd(20180131)
jan31 + months(1)
## NA
```

%m+% and **%m-%** will roll imaginary dates to the last day of the previous month.

```
jan31 %m+% months(1)
## "2018-02-28"
```

add_with_rollback(e1, e2, roll_to_first = TRUE) will roll imaginary dates to the first day of the new month.

```
add_with_rollback(jan31, months(1),
roll_to_first = TRUE)
## "2018-03-01"
```

PERIODS

Add or subtract periods to model events that happen at specific clock times, like the NYSE opening bell.

Make a period with the name of a time unit ***pluralized***, e.g.

```
p <- months(3) + days(12)
p
"3m 12d 0H 0M 0S"
```

Number of months | Number of days | etc.

years(x = 1) x years.
months(x) x months.
weeks(x = 1) x weeks.
days(x = 1) x days.
hours(x = 1) x hours.
minutes(x = 1) x minutes.
seconds(x = 1) x seconds.
milliseconds(x = 1) x milliseconds.
microseconds(x = 1) x microseconds
nanoseconds(x = 1) x nanoseconds.
picoseconds(x = 1) x picoseconds.

period(num = NULL, units = "second", ...) An automation friendly period constructor. period(5, unit = "years")

as.period(x, unit) Coerce a timespan to a period, optionally in the specified units. Also **is.period()**. as.period(p)

period_to_seconds(x) Convert a period to the "standard" number of seconds implied by the period. Also **seconds_to_period()**. period_to_seconds(p)

DURATIONS

Add or subtract durations to model physical processes, like battery life. Durations are stored as seconds, the only time unit with a consistent length. **Difftimes** are a class of durations found in base R.

Make a duration with the name of a period prefixed with a ***d***, e.g.

```
dd <- ddays(14)
dd
"1209600s (~2 weeks)"
```

Exact length in seconds | Equivalent in common units

dyears(x = 1) 31536000x seconds.
dmonths(x = 1) 2629800x seconds.
dweeks(x = 1) 604800x seconds.
ddays(x = 1) 86400x seconds.
dhours(x = 1) 3600x seconds.
dminutes(x = 1) 60x seconds.
dseconds(x = 1) x seconds.
dmilliseconds(x = 1) x $\times 10^{-3}$ seconds.
dmicroseconds(x = 1) x $\times 10^{-6}$ seconds.
dnanoseconds(x = 1) x $\times 10^{-9}$ seconds.
dpicoseconds(x = 1) x $\times 10^{-12}$ seconds.

duration(num = NULL, units = "second", ...) An automation friendly duration constructor. duration(5, unit = "years")

as.duration(x, ...) Coerce a timespan to a duration. Also **is.duration()**, **is.difftime()**. as.duration(i)

make_difftime(x) Make difftime with the specified number of units. make_difftime(99999)

INTERVALS

Divide an interval by a duration to determine its physical length, divide an interval by a period to determine its implied length in clock time.

Make an interval with **interval()** or **%--%**, e.g.

Start Date | End Date

```
i <- interval(ymd("2017-01-01"), d)
j <- d %--% ymd("2017-12-31")
## 2017-01-01 UTC--2017-11-28 UTC
## 2017-11-28 UTC--2017-12-31 UTC
```

a **%within%** b Does interval or date-time *a* fall within interval *b*? now() %within% i

int_start(int) Access/set the start date-time of an interval. Also **int_end()**. int_start(i) <- now(); int_start(i)

int_aligns(int1, int2) Do two intervals share a boundary? Also **int_overlaps()**. int_aligns(i, j)

int_diff(times) Make the intervals that occur between the date-times in a vector. v <-c(dt, dt + 100, dt + 1000); int_diff(v)

int_flip(int) Reverse the direction of an interval. Also **int_standardize()**. int_flip(i)

int_length(int) Length in seconds. int_length(i)

int_shift(int, by) Shifts an interval up or down the timeline by a timespan. int_shift(i, days(-1))

as.interval(x, start, ...) Coerce a timespan to an interval with the start date-time. Also **is.interval()**. as.interval(days(1), start = now())

posit™

Parsing Dates and Times

```
> library(lubridate)
>
> today()
[1] "2023-09-10"
> current1 <- now()
> current1
[1] "2023-09-10 08:17:32 UTC"
> class(current1)
[1] "POSIXct" "POSIXt"
>
> current2 <- date()
> current2
[1] "Sun Sep 10 16:17:32 2023"
> class(current2)
[1] "character"
```

```
> my_date <- ymd("20230910")
> my_date
[1] "2023-09-10"
> class(my_date)
[1] "Date"
>
> mdy("09-10-2023")
[1] "2023-09-10"
> dmy("10/09/2023")
[1] "2023-09-10"
>
> ymd_hms("20131219101707")
[1] "2013-12-19 10:17:07 UTC"
> ymd_hms("2013 Dec 19 10:17:07")
[1] "2013-12-19 10:17:07 UTC"
>
> mdy("Dec 19, 2013")
[1] "2013-12-19"
>
> mdy_hms("December 19, 2013 10:17:07")
[1] "2013-12-19 10:17:07 UTC"
>
> dmy_hms("19-Dec, 2013 10:17:07")
[1] "2013-12-19 10:17:07 UTC"
```

```
> two_dates <- ymd_hms("2023-01-01 01:15:00", "2023-02-01 01:30:00")
> minute(two_dates)
[1] 15 30
> mean(minute(two_dates))
[1] 22.5
```

時區 (Time Zone)

- The default time zone for the lubridate functions is Greenwich Mean Time (**GMT**, formerly)/Universal Time, Coordinated (**UTC**)
- Taipei: CST (China Standard Time), Offset: UTC + 8 hours
- London: BST (British Summer Time) (Apr. ~ Oct.), Offset: UTC/GMT + 1 hour

```
> OlsonNames() ## typically around six hundred names
  [1] "Africa/Abidjan" "Africa/Accra" "Africa/Addis_Ababa" "Africa/Algiers"
  [5] "Africa/Asmara" "Africa/Asmera" "Africa/Bamako" "Africa/Bangui"
  [9] "Africa/Banjul" "Africa/Bissau" "Africa/Blantyre" "Africa/Brazzaville"
...
[589] "US/Mountain" "US/Pacific" "US/Samoa" "UTC"
[593] "W-SU" "WET""Zulu"
attr(,"Version")
[1] "2022e"
```

```
> Sys.timezone()
[1] "UTC"
> Sys.setenv(TZ = "UTC")
> ymd_hms("2023-08-17 08:50:00", tz = "Asia/Taipei")
[1] "2023-08-17 08:50:00 CST"
> ymd_hms("2023-08-17 19:25:00", tz = "Europe/London")
[1] "2023-08-17 19:25:00 BST"
>
> departure <- ymd_hms("2023-08-17 08:50:00", tz = "Asia/Taipei")
> departure
[1] "2023-08-17 08:50:00 CST"
> with_tz(departure, "Europe/London")
[1] "2023-08-17 01:50:00 BST"
```

```
> changed <- force_tz(departure, "America/Chicago")
> changed
[1] "2023-08-17 08:50:00 CDT"
> with_tz(changed, "Europe/London")
[1] "2023-08-17 14:50:00 BST"
```

Central Daylight Time
中部夏令時間

Setting and Extracting Information

```
> departure <- ymd_hms("2023-08-17 08:50:00", tz = "Asia/Taipei")
> year(departure)
[1] 2023
> month(departure)
[1] 8
> week(departure)
[1] 33
> yday(departure)
[1] 229
> mday(departure)
[1] 17
> wday(departure)
[1] 5
> hour(departure)
[1] 8
> minute(departure)
[1] 50
> second(departure)
[1] 0
```

Date Component	Extractor Function
Year	`year()`
Month	`month()`
Week	`week()`
Day of year	`yday()`
Day of month	`mday()`
Day of week	`wday()`
Hour	`hour()`
Minute	`minute()`
Second	`second()`
Time zone	`tz()`

```
> second(departure) <- 25
> departure
[1] "2023-08-17 08:50:25 CST"
```

```
> month(departure, label = TRUE)
[1] 八月
Levels: 一月 < 二月 < 三月 < 四月 < 五月 < 六月 < 七月 < 八月 < 九月 < 十月 < 十一月 < 十二月
> month(departure, label = TRUE, abbr = FALSE)
[1] 八月
Levels: 一月 < 二月 < 三月 < 四月 < 五月 < 六月 < 七月 < 八月 < 九月 < 十月 < 十一月 < 十二月
> wday(departure, label = TRUE)
[1] 週四
Levels: 週日 < 週一 < 週二 < 週三 < 週四 < 週五 < 週六
```

英文: **August**, **Aug**

```
> Sys.getenv()
> Sys.setenv(LANG = "en")
> Sys.setenv(LANG = "zh")
```

Time Intervals

```
> summer_camp_start <- ymd_hms("2023-07-20 08:00:00")
> summer_camp_end <- ymd_hms("2023-08-03 12:00:00")
> camp_period <- interval(summer_camp_start, summer_camp_end)
> camp_period
[1] 2023-07-20 08:00:00 UTC--2023-08-03 12:00:00 UTC
>
> # Toronto
> # Current:     EDT — Eastern Daylight Time
> # Next Change: EST — Eastern Standard Time
>
> JSM2023 <- interval(ymd(20230805, tz = "Canada/Eastern"),
+                     ymd(20230810, tz = "Canada/Eastern"))
> JSM2023
[1] 2023-08-05 EDT--2023-08-10 EDT

> int_overlaps(JSM2023, camp_period)
[1] FALSE
> union(JSM2023, camp_period)
[1] 2023-07-20 04:00:00 EDT--2023-08-10 EDT
>
> # Other functions that work with intervals:
> # int_start, int_end, int_flip, int_shift, int_aligns, union,
> # intersect, setdiff, %within%.
```

Arithmetic with Date Times

```
> # Periods record a time span in units larger
than seconds
> ymd(20230805) + 1
[1] "2023-08-06"
> ymd(20230805) + days(1)
[1] "2023-08-06"
> days(0:5)
[1] "0S" "1d 0H 0M 0S" "2d 0H 0M 0S" "3d 0H 0M
0S" "4d 0H 0M 0S" "5d 0H 0M 0S"
> weeks(1:5)
[1] "7d 0H 0M 0S"  "14d 0H 0M 0S" "21d 0H 0M
0S" "28d 0H 0M 0S" "35d 0H 0M 0S"
> months(1)
[1] "1m 0d 0H 0M 0S"
> ymd(20230805) + months(1)
[1] "2023-09-05"
> years(1:2)
[1] "1y 0m 0d 0H 0M 0S" "2y 0m 0d 0H 0M 0S"
> ymd(20230805) + years(1:2)
[1] "2024-08-05" "2025-08-05"
```

```
> # Durations: record the time span in
seconds
> dweeks(1)
[1] "604800s (~1 weeks)"
> ddays(6)
[1] "518400s (~6 days)"
> dhours(1)
[1] "3600s (~1 hours)"
> dminutes(1)
[1] "60s (~1 minutes)"
> dseconds(60)
[1] "60s (~1 minutes)"
```

Arithmetic with Date Times

- **Question**: January 31st + one month. Should the answer be
  (1) February 31st (which doesn't exist)
  (2) March 4th (31 days after January 31), or
  (3) February 28th (assuming its not a leap year)

```
> jan31 <- ymd("2013-01-31")
> jan31
[1] "2013-01-31"
```

```
> #(1)
> jan31 + months(1)
[1] NA
> jan31 + months(0:11)
 [1] "2013-01-31" NA "2013-03-31" NA "2013-05-31" NA
 [7] "2013-07-31" "2013-08-31" NA "2013-10-31" NA "2013-12-31"
>
> #(2)
> floor_date(jan31, "month")
[1] "2013-01-01"
> floor_date(jan31, "month") + months(0:11) + days(31)
 [1] "2013-02-01" "2013-03-04" "2013-04-01" "2013-05-02" "2013-06-01" "2013-07-02"
 [7] "2013-08-01" "2013-09-01" "2013-10-02" "2013-11-01" "2013-12-02" "2014-01-01"
>
> #(3)
> # %m+% and %m-%: roll dates back to the last day of the month
> jan31 %m+% months(0:11)
 [1] "2013-01-31" "2013-02-28" "2013-03-31" "2013-04-30" "2013-05-31" "2013-06-30"
 [7] "2013-07-31" "2013-08-31" "2013-09-30" "2013-10-31" "2013-11-30" "2013-12-31"
```

Creating ts Time Series Object

```
ts(data = NA, start = 1, end = numeric(), frequency = 1,
   deltat = 1, ts.eps = getOption("ts.eps"), class = , names = )
as.ts(x, ...)
is.ts(x)
```

Year	Observation
2012	123
2013	39
2014	78
2015	52
2016	110

ts(data, frequency, start)

Type of data	frequency	start example
Annual	1	1995
Quarterly	4	c(1995,2)
Monthly	12	c(1995,9)
Daily	7 *or* 365.25	1 *or* c(1995,234)
Weekly	52.18	c(1995,23)
Hourly	24 *or* 168 *or* 8,766	1
Half-hourly	48 *or* 336 *or* 17,532	1

```
Observation <- ts(c(123, 39, 78, 52, 110), start = 2012)
```

```
> ts_matrix
       [,1]  [,2]  [,3]
 [1,] -0.33  0.14  0.52
 [2,]  0.44  2.63  0.58
 [3,] -1.05 -0.37 -1.33
 [4,] -0.42 -2.10 -0.59
 [5,]  0.69  0.05 -0.59
 [6,]  0.85  0.74  0.64
 [7,] -0.74 -1.97  0.20
 [8,] -0.67 -0.13  0.28
 [9,]  0.42  1.69  1.18
[10,]  1.66 -2.16  0.90
[11,]  0.30 -1.83  0.90
[12,] -0.03 -1.21  0.60
[13,] -0.10 -0.49 -0.40
[14,] -0.35  0.19 -1.01
[15,] -0.68 -1.26  2.20
[16,]  1.34 -0.64 -0.05
[17,]  1.15  1.28  0.90
[18,]  0.55  0.34 -2.00
[19,] -0.03 -0.17  0.59
[20,]  0.53  0.42 -1.25
```

```
> ts_obj
         Series 1 Series 2 Series 3
Jan 1961    -0.33     0.14     0.52
Feb 1961     0.44     2.63     0.58
Mar 1961    -1.05    -0.37    -1.33
Apr 1961    -0.42    -2.10    -0.59
May 1961     0.69     0.05    -0.59
Jun 1961     0.85     0.74     0.64
Jul 1961    -0.74    -1.97     0.20
Aug 1961    -0.67    -0.13     0.28
Sep 1961     0.42     1.69     1.18
Oct 1961     1.66    -2.16     0.90
Nov 1961     0.30    -1.83     0.90
Dec 1961    -0.03    -1.21     0.60
Jan 1962    -0.10    -0.49    -0.40
Feb 1962    -0.35     0.19    -1.01
Mar 1962    -0.68    -1.26     2.20
Apr 1962     1.34    -0.64    -0.05
May 1962     1.15     1.28     0.90
Jun 1962     0.55     0.34    -2.00
Jul 1962    -0.03    -0.17     0.59
Aug 1962     0.53     0.42    -1.25
```

```
> ts_matrix <- matrix(round(rnorm(60), 2), 20, 3)
> ts_matrix
> ts_obj <- ts(ts_matrix, start = c(1961, 1), frequency = 12)
> ts_obj
> class(ts_obj)
[1] "mts"    "ts"     "matrix"
> is.mts(ts_obj)
[1] TRUE
```

The R Graph Gallery: Time Series

The R Graph Gallery https://r-graph-gallery.com/time-series.html

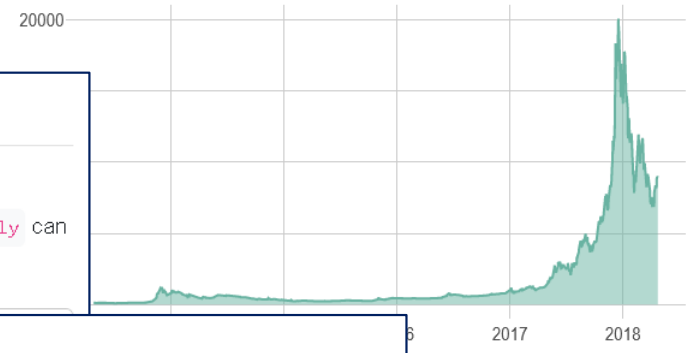
← Gallery CHART TYPES PKG BEST QUICK TOOLS ALL RELATED SUBSCRIBE

Time Series

INTERACTIVE VERSION: PLOTLY

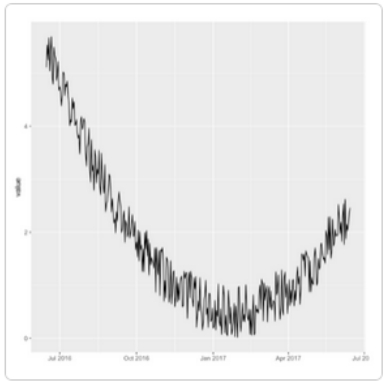
The `ggplotly()` function of the `plotly` library makes it a breeze to build an interactive version. Try to hover circles to get a tooltip, or select an area of interest for zooming. Double click to reinitialize.

GET CODE



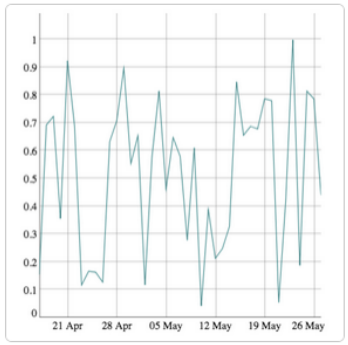
TIME SERIES WITH GGPLOT2

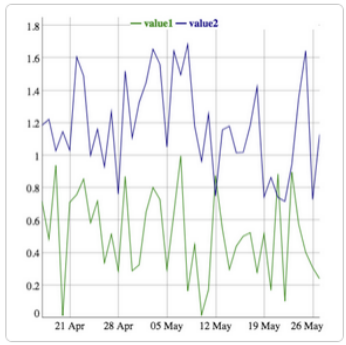
`ggplot2` offers great features when it comes to visualize time series. The `date` format will be recognized automatically, resulting in neat X axis labels. The `scale_x_data()` makes it a breeze to customize those labels. Last but not least, `plotly` can turn the resulting chart interactive in one more line of code.

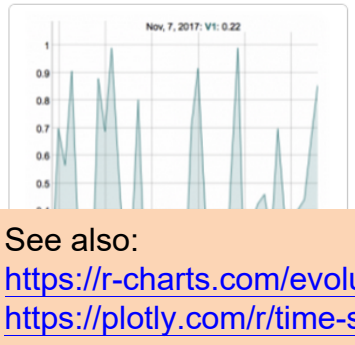


TIME SERIES WITH DYGRAPH

The dygraphs package is a html widget. It allows to make interactive time series chart: you can zoom and hover data points to get additional information. Start by reading the chart #316 for quick introduction and input description. Then, the graph #317 gives an overview of the different types of charts that are offered. To go further, check the graph #318 (interactive version below).







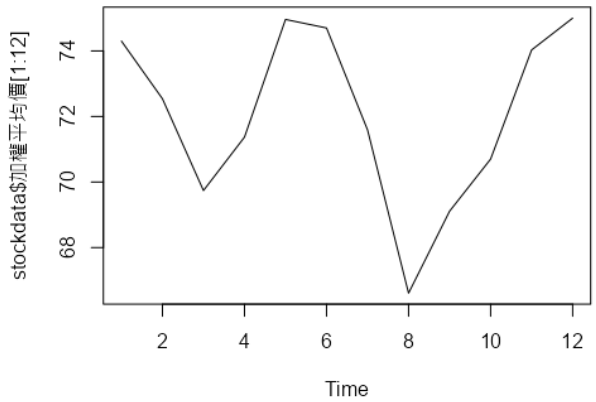
See also:
https://r-charts.com/evolution/time-series-ggplot2/
https://plotly.com/r/time-series/

matplot {graphics}, geom_line {ggplot2}

民國100年5家半導體公司股票月成交資訊(元,股)

半導體公司	年度	月份	最高價	最低價	加權平均價	成交筆數	成交金額	
台積電	100	1	78.3	69.6	74.3	263,999	100,578,274,926	1,353,616,348	5.22
台積電	100	2	77	69.9	72.54	235,159	74,985,055,548	1,033,654,452	3.98
台積電	100	3	72.2	65.7	69.74	276,434	88,459,924,495	1,268,289,393	4.89
台積電	100	4	73.9	68	71.37	211,611	70,177,023,098	983,177,475	3.79
台積電	100	5	76.9	73	74.96	213,185	74,005,599,560	987,256,484	3.8
台積電	100	6	78.2	70.4	74.7	260,507	96,761,306,205	1,295,262,736	4.99
台積電	100	7	73.9	68.5	71.59	238,386	73,569,965,426	1,027,567,656	3.96
台積電	100	8	72.8	62.2	66.61	305,409	84,617,942,159	1,270,302,342	4.9
台積電	100	9	72.1	65.9	69.11	266,720	74,225,030,814	1,073,997,108	4.14
台積電	100	10	74	68.1	70.7	181,361	59,947,670,693	847,821,278	3.27
台積電	100	11	76	71.3	74.03	197,579	65,432,526,407	883,753,804	3.41
台積電	100	12	76.8	72	75	179,107	53,687,756,290	715,808,271	2.76
威盛	100	1	33.4	29.3	30.97	55,107	4,580,913,795	147,912,893	21.54
威盛	100	2	32.65	28.35	30.54	26,901	2,060,809,696	67,459,942	9.82
威盛	100	3	35.45	28.5					
威盛	100	4	32.8	27.55					
威盛	100	5	32.6	25.95					
威盛	100	6	37.25	31.2					
威盛	100	7	38.15	32.45					
威盛	100	8	35.4	26.6					
威盛	100	9	29	23.1					
威盛	100	10	25.15	20.4					
威盛	100	11	25.7	18.7					
威盛	100	12	20.2	14.8					
聯發科	100	1	424	378	403.55	106,530	57,621,649,341	142,786,216	12.98
聯發科	100	2	380	325.5	348.98	97,339	46,409,931,806	132,985,689	12.08
聯發科	100	3	355	312.5	339.96	117,960	52,887,228,668	155,567,203	14.14

```
stockdata <- read.table("data/stock-data.txt", skip = 1, header = T)
head(stockdata, 5)
sapply(stockdata, class)
for(at in 7:9){
  stockdata[,at] <- as.numeric(gsub(",", "", stockdata[,at]))
}
```



ts.plot {stats}

Plot Multiple Time Series

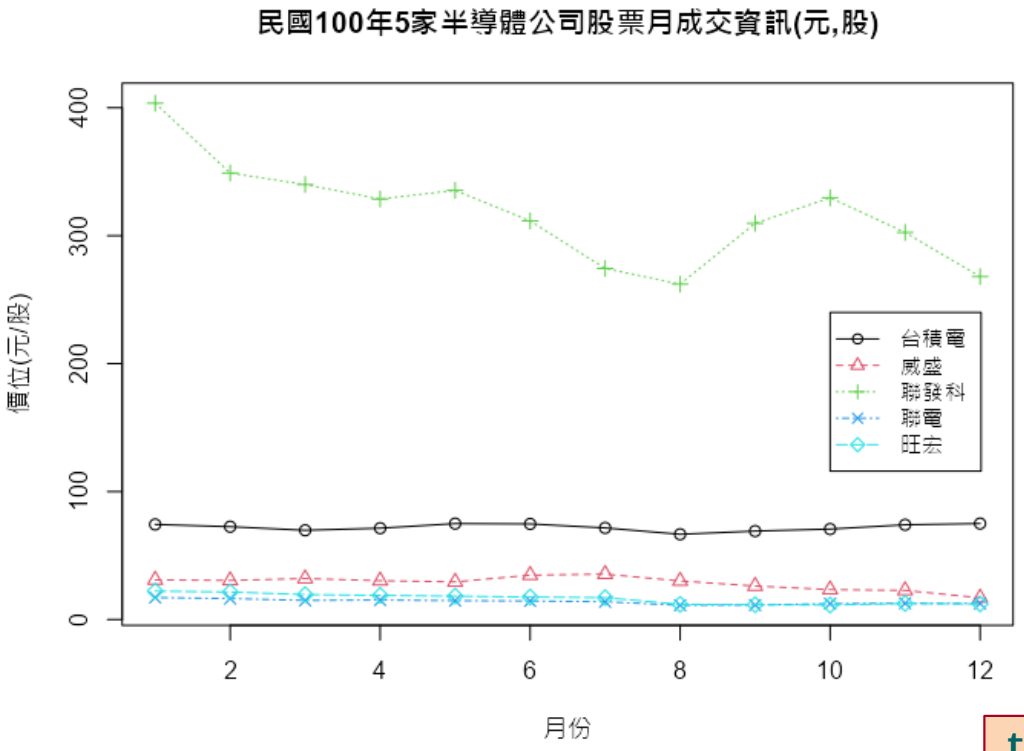
```
ts.plot(stockdata$加權平均價[1:12])
```

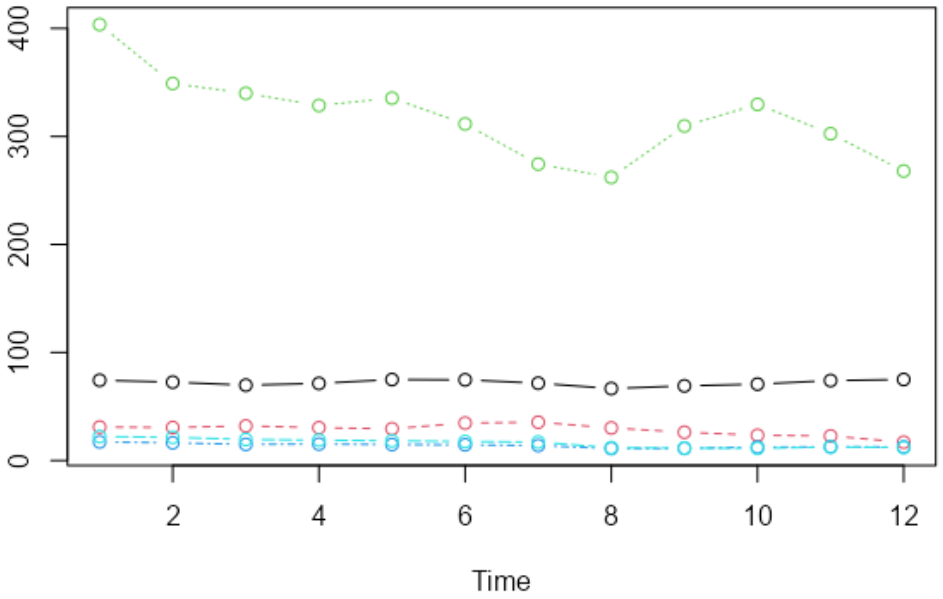
使用 matplot {graphics}

```
mat <- as.data.frame(matrix(stockdata$加權平均價, ncol = 5))
colnames(mat) <- unique(stockdata$半導體公司)
mat

matplot(1:12, mat, type = "o", lty = 1:5, col = 1:5, pch = 1:5,
        main = "民國100年5家半導體公司股票月成交資訊(元,股)",
        xlab = "月份", ylab = "價位(元/股)")
legend(10, 240, legend = colnames(mat), lty = 1:5, col = 1:5,
       pch = 1:5, cex = 0.9)
```

type = "l"

```
> mat
   台積電  威盛 聯發科  聯電  旺宏
1   74.30 30.97 403.55 17.19 22.19
2   72.54 30.54 348.98 16.38 21.49
3   69.74 32.01 339.96 14.92 19.48
4   71.37 30.35 328.65 15.21 18.88
5   74.96 29.40 335.42 14.76 18.25
6   74.70 34.68 311.57 14.51 17.60
7   71.59 35.47 274.39 13.89 17.09
8   66.61 30.13 262.09 11.13 11.84
9   69.11 26.17 309.66 11.25 11.55
10  70.70 23.39 329.66 12.39 11.31
11  74.03 22.74 302.52 12.68 12.54
12  75.00 16.96 268.01 12.51 12.17
```

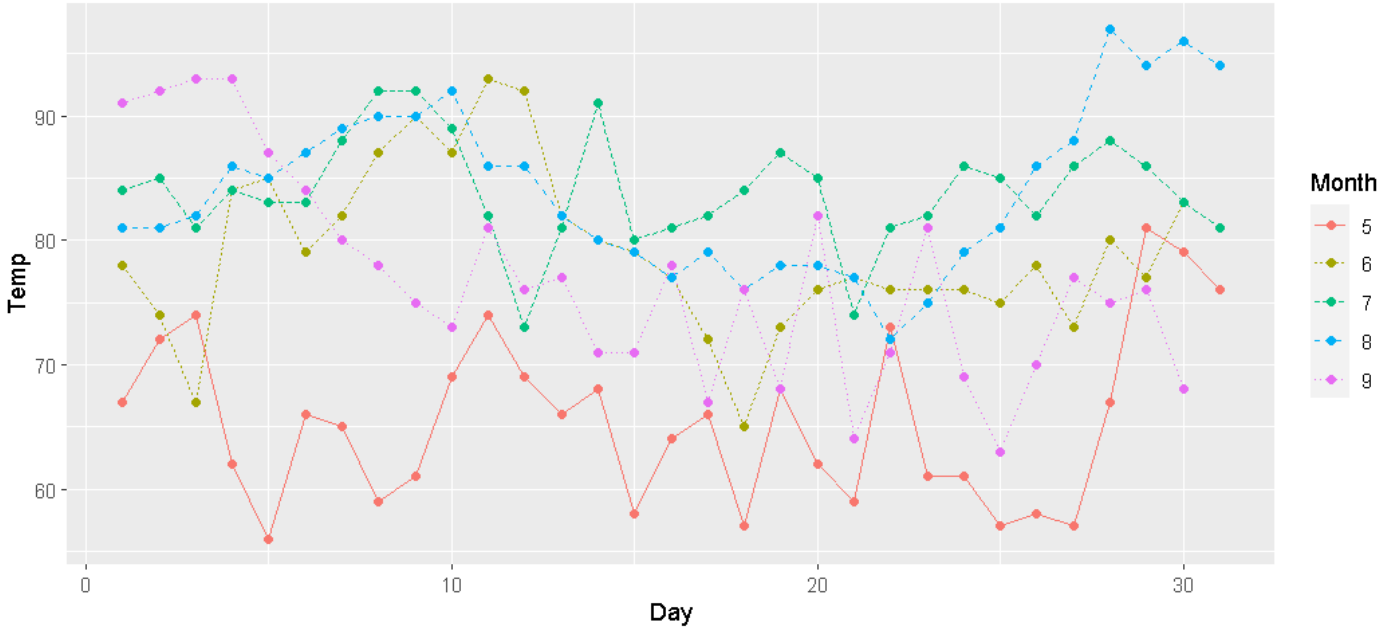





```
ts.plot(mat, lty = 1:5, col = 1:5, type = "b")
```

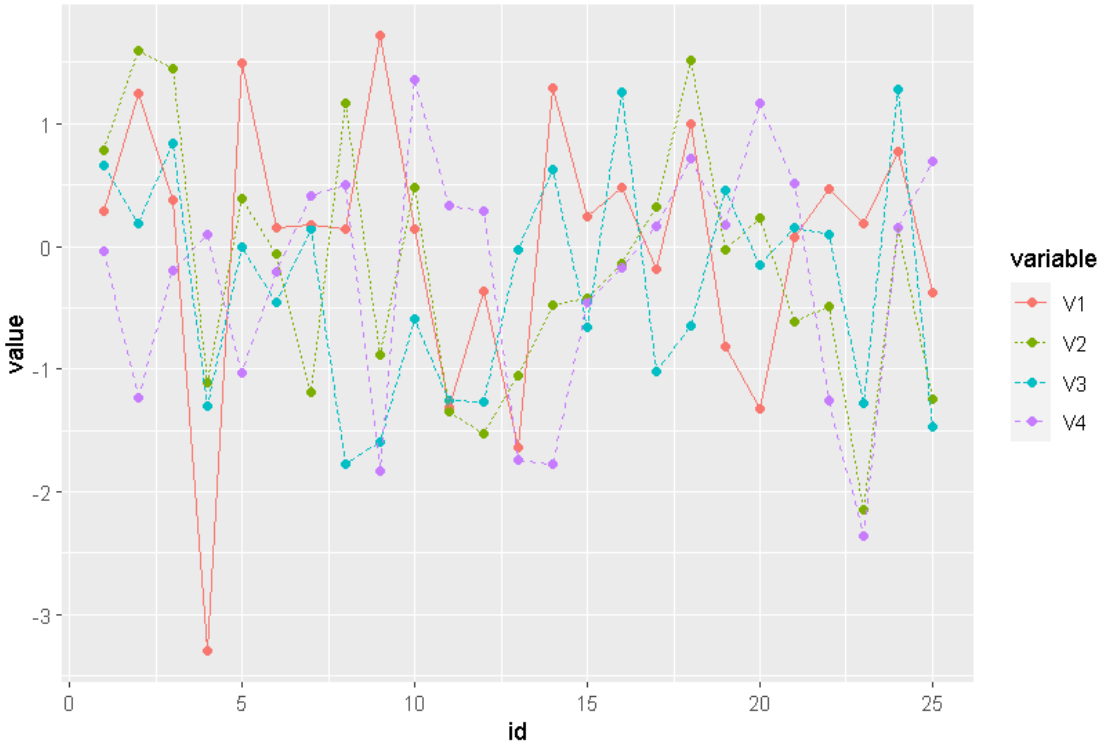
線圖 (Line plot)
使用 geom_line {ggplot2}

```
> 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
> airquality$Month <- factor(airquality$Month)
> ggplot(airquality, aes(x = Day, y = Temp, group = Month, color = Month)) +
+   geom_line(aes(linetype = Month)) +
+   geom_point()
```



線圖 (Line plot)
使用 geom_line {ggplot2}

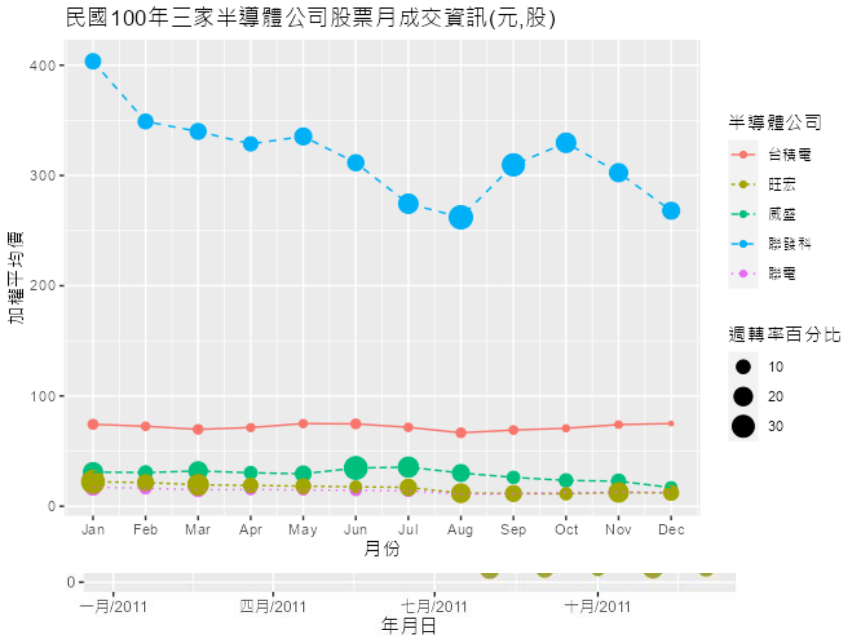
```
> mydata <- as.data.frame(matrix(rnorm(100), ncol = 4))
> library(reshape2)
> head(mydata, 3)
          V1          V2          V3          V4
1  0.2846997  0.78283129  0.659318307 -0.04318195
2  1.2510186  1.59782230  0.187074422 -1.23161275
3  0.3827782  1.44676700  0.840148794 -0.20081868
> #id variable for position in matrix
> mydata$id <- 1:nrow(mydata)
> #reshape to long format
> mydata.lf <- melt(mydata, id.var = "id")
> head(mydata.lf)
  id variable      value
1  1       V1  0.2846997
2  2       V1  1.2510186
3  3       V1  0.3827782
4  4       V1 -3.2994010
5  5       V1  1.4943630
6  6       V1  0.1557203
> tail(mydata.lf)
    id variable      value
95  20       V4  1.1655219
96  21       V4  0.5081844
97  22       V4 -1.2523577
98  23       V4 -2.3553732
99  24       V4  0.1542803
100 25       V4  0.6899416
> ggplot(mydata.lf, aes(x = id, y = value, group = variable, colour = variable)) +
+   geom_point()+
+   geom_line(aes(lty = variable))
```

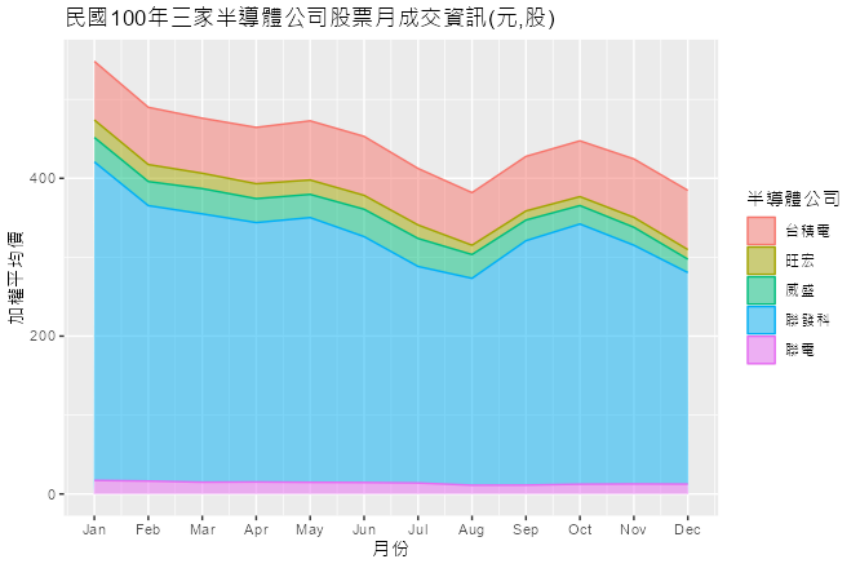


使用 geom_line, geom_area

```
library(ggplot2)
ggplot(stockdata, aes(x = 月份, y = 加權平均價, colour = 半導體公司)) +
  geom_point(aes(size = 週轉率百分比))+
  geom_line(aes(lty = 半導體公司)) +
  scale_x_continuous(breaks = 1:12, labels = month.abb) +
  labs(title="民國100年三家半導體公司股票月成交資訊(元,股)")
```

```
ggplot(stockdata, aes(x = 月份, y = 加權平均價, colour = 半導體公司)) +
  geom_area(aes(fill = 半導體公司), alpha = 0.5) +
  scale_x_continuous(breaks = 1:12, labels = month.abb) +
  labs(title="民國100年三家半導體公司股票月成交資訊(元,股)")
```



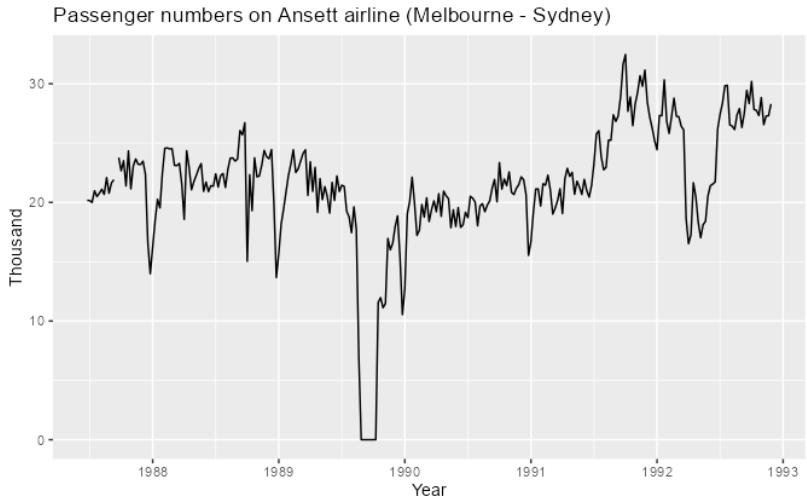



```
stockdata$年月日 <- as.Date(paste0("2011/", stockdata$月份, "/01"))
ggplot(...) +
  scale_x_date(date_labels = "%b/%Y")
```

使用 autoplot {ggplot2} for ts, mts Object

```
> library(forecast)
> library(ggplot2)
> library(fpp) # Forecasting: principles and practice
>
> data(melsyd) # Total weekly air passenger numbers on Ansett airline flights between
Melbourne and Sydney, 1987-1992.
> head(melsyd)
Time Series:
Start = c(1987, 26)
End = c(1987, 31)
Frequency = 52
         First.Class Business.Class Economy.Class
1987.481       1.912             NA        20.167
1987.500       1.848             NA        20.161
1987.519       1.856             NA        19.993
1987.538       2.142             NA        20.986
1987.558       2.118             NA        20.497
1987.577       2.048             NA        20.770
> class(melsyd)
[1] "mts" "ts"
> colnames(melsyd)
[1] "First.Class"    "Business.Class" "Economy.Class"
> time(melsyd)
Time Series:
Start = c(1987, 26)
End = c(1992, 48)
Frequency = 52
  [1] 1987.481 1987.500 1987.519 1987.538 1987.558 1987.577 1987.596 1987.615
1987.635...
```

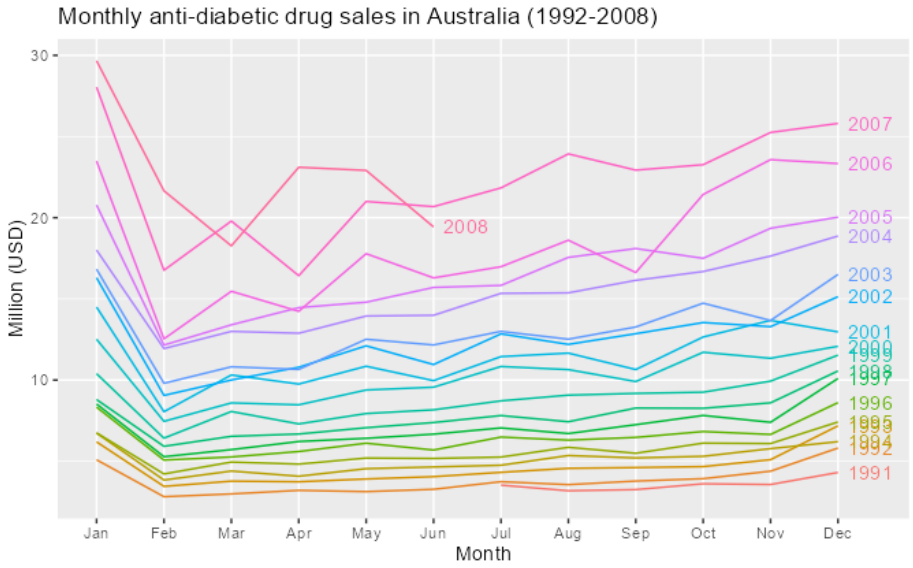
```
> autoplot(melsyd[, "Economy.Class"]) +
+   xlab("Year") +
+   ylab("Thousand") +
+   ggtitle("Passenger numbers (Melbourne - Sydney)")
```

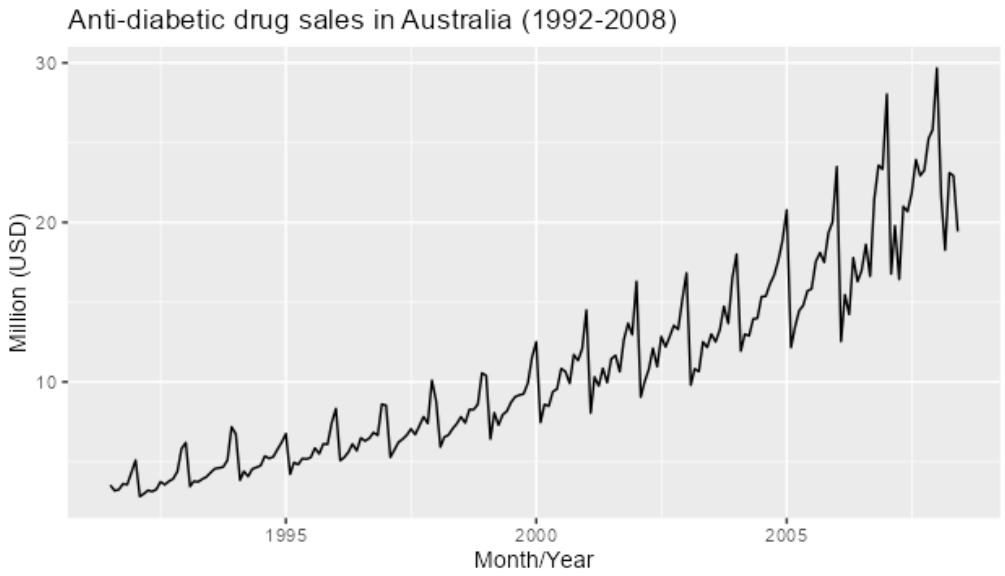


使用 ggseasonplot for ts Object

```
> data(a10) # Monthly anti-diabetic drug (抗糖尿病藥) sales in Australia from 1992 to 2008.
> a10
           Jan       Feb       Mar       Apr       May       Jun       Jul       Aug
1991                                                             3.526591  3.180891
1992  5.088335  2.814520  2.985811  3.204780  3.127578  3.270523  3.737851  3.558776
...
2007 28.038383 16.763869 19.792754 16.427305 21.000742 20.681002 21.834890 23.930204
2008 29.665356 21.654285 18.264945 23.107677 22.912510 19.431740
> class(a10)
[1] "ts"
> ggseasonplot(a10, year.labels = TRUE) +
+   xlab("Month")+
+   ylab("Million (USD)") +
+   ggtitle("Monthly anti-diabetic drug sales in Australia (1992-2008)")
```

```
autoplot(a10) +
  xlab("Month")+
  ylab("Million (USD)") +
  ggtitle("Anti-diabetic drug sales)")
```





plotly package supports the time-series classes: ts, mts, zoo, xts, data.frame, tbl

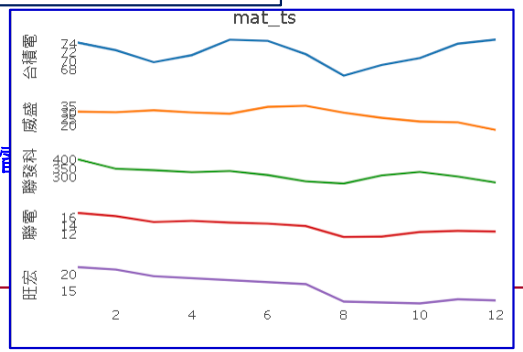
```
library(plotly)
# data.frame with one date/time column
date_mat <- data.frame(date = seq(as.Date("2011/1/1"),
                                  by = "month",
                                  length.out = 12),
                       mat)
sapply(date_mat, class)
ts_plot(date_mat)
```

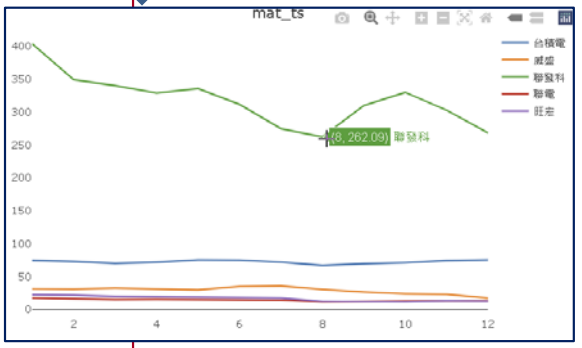
```
> mat
   台積電  威盛 聯發科  聯電  旺宏
1   74.30 30.97 403.55 17.19 22.19
2   72.54 30.54 348.98 16.38 21.49
3   69.74 32.01 339.96 14.92 19.48
4   71.37 30.35 328.65 15.21 18.88
5   74.96 29.40 335.42 14.76 18.25
6   74.70 34.68 311.57 14.51 17.60
7   71.59 35.47 274.39 13.89 17.09
8   66.61 30.13 262.09 11.13 11.84
9   69.11 26.17 309.66 11.25 11.55
10  70.70 23.39 329.66 12.39 11.31
11  74.03 22.74 302.52 12.68 12.54
12  75.00 16.96 268.01 12.51 12.17
```

```
> date_mat
         date 台積電  威盛 聯發科  聯電  旺宏
1  2011-01-01  74.30 30.97 403.55 17.19 22.19
2  2011-02-01  72.54 30.54 348.98 16.38 21.49
3  2011-03-01  69.74 32.01 339.96 14.92 19.48
4  2011-04-01  71.37 30.35 328.65 15.21 18.88
5  2011-05-01  74.96 29.40 335.42 14.76 18.25
6  2011-06-01  74.70 34.68 311.57 14.51 17.60
7  2011-07-01  71.59 35.47 274.39 13.89 17.09
8  2011-08-01  66.61 30.13 262.09 11.13 11.84
9  2011-09-01  69.11 26.17 309.66 11.25 11.55
10 2011-10-01  70.70 23.39 329.66 12.39 11.31
11 2011-11-01  74.03 22.74 302.52 12.68 12.54
12 2011-12-01  75.00 16.96 268.01 12.51 12.17
```

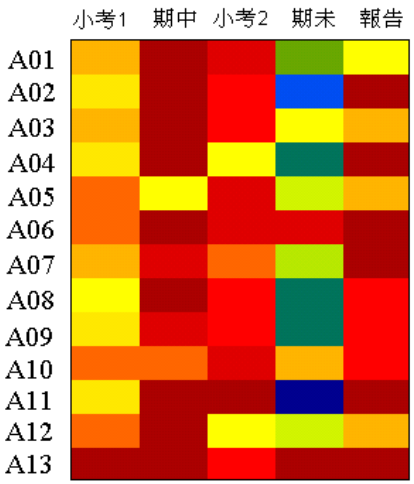
```
> # convert data.frame to ts, and then using ts_plot
> mat_ts <- as.ts(mat, start = c(2011, 1), frequency = 12)
> mat_ts
Time Series:
Start = 1
End = 12
Frequency = 1
   台積電  威盛 聯發科  聯電  旺宏
 1  74.30 30.97 403.55 17.19 22.19
...
12  75.00 16.96 268.01 12.51 12.17
> class(mat_ts)
[1] "mts"    "ts"     "matrix"
> str(mat_ts)
 Time-Series [1:12, 1:5] from 1 to 12: 74.3
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:5] "台積電" "威盛" "聯發科" "聯電
> ts_plot(mat_ts)
> ts_plot(mat_ts, type = "multiple")
```

```
> mat_ts
Time Series:
Start = 1
End = 12
Frequency = 1
   台積電  威盛 聯發科  聯電  旺宏
 1  74.30 30.97 403.55 17.19 22.19
 2  72.54 30.54 348.98 16.38 21.49
 3  69.74 32.01 339.96 14.92 19.48
 4  71.37 30.35 328.65 15.21 18.88
 5  74.96 29.40 335.42 14.76 18.25
 6  74.70 34.68 311.57 14.51 17.60
 7  71.59 35.47 274.39 13.89 17.09
 8  66.61 30.13 262.09 11.13 11.84
 9  69.11 26.17 309.66 11.25 11.55
10  70.70 23.39 329.66 12.39 11.31
11  74.03 22.74 302.52 12.68 12.54
12  75.00 16.96 268.01 12.51 12.17
```

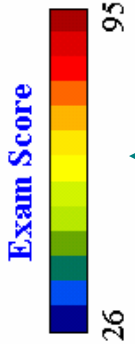




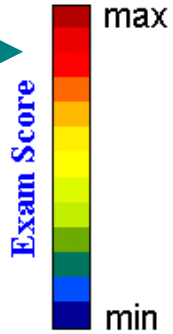
Heat Map: Data Image, Matrix Visualization

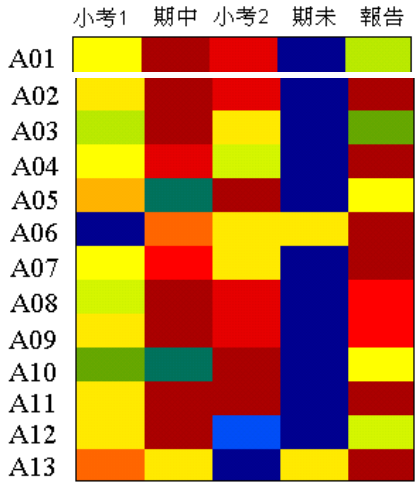


Range Matrix Condition

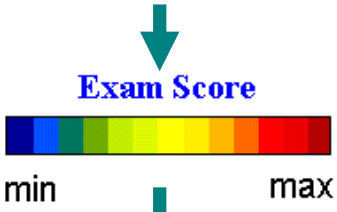


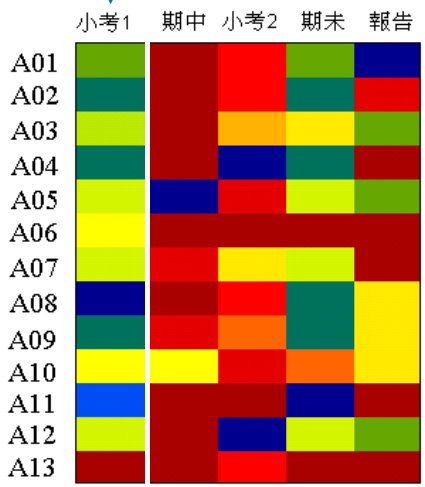
	A	B	C	D	E	F
1	學號	小考1	期中考	小考2	期末考	報告
2	A01	69	92	85	45	62
3	A02	66	90	83	36	90
4	A03	72	92	80	62	70
5	A04	68	90	60	37	95
6	A05	74	60	86	54	70
7	A06	77	90	88	88	95
8	A07	73	88	77	51	95
9	A08	61	90	84	40	82
10	A09	66	88	82	39	80
11	A10	76	75	87	72	80
12	A11	64	90	90	26	95
13	A12	75	90	60	55	70
14	A13	92	90	83	90	95





Range Row Condition

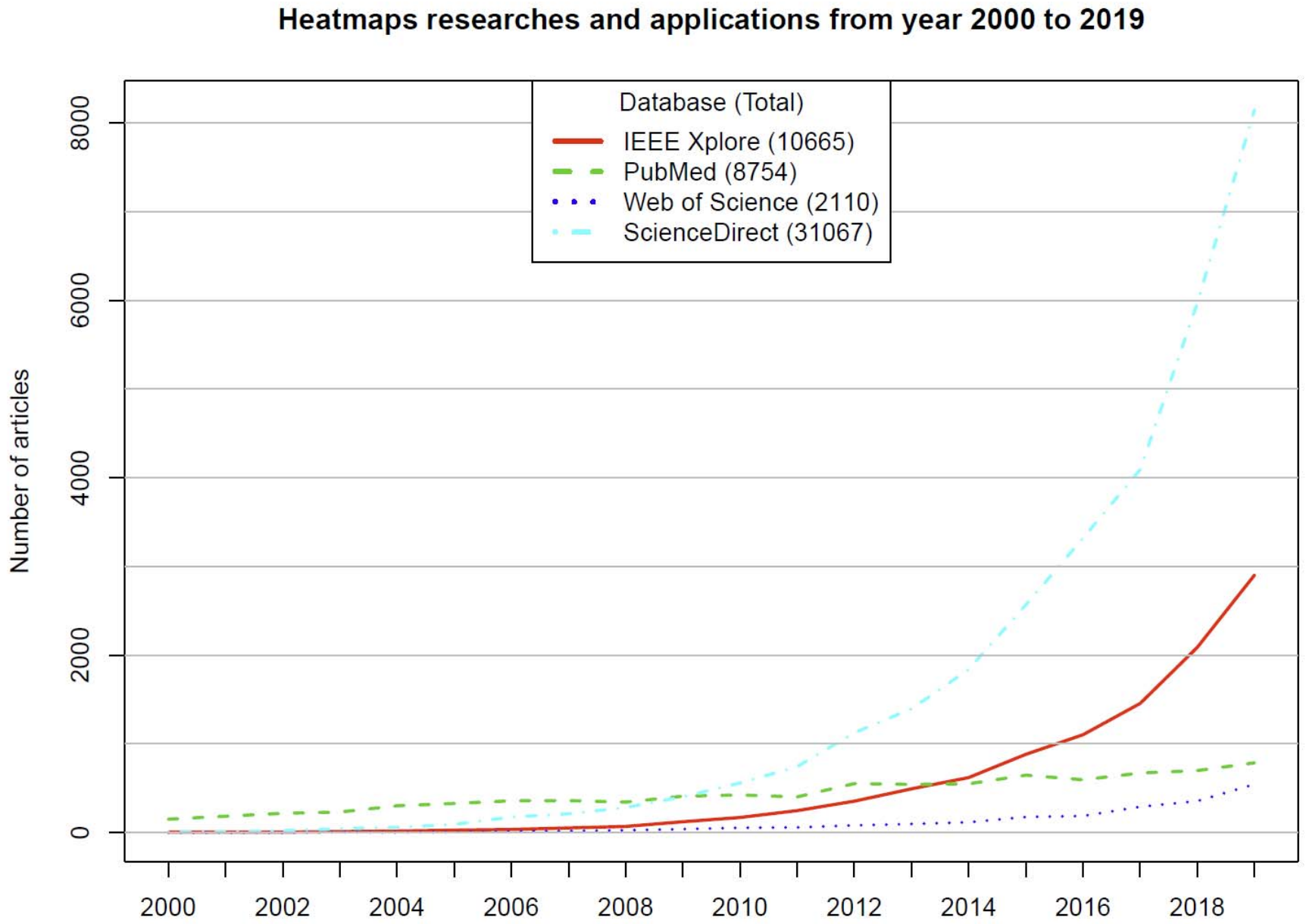




Range Column Condition

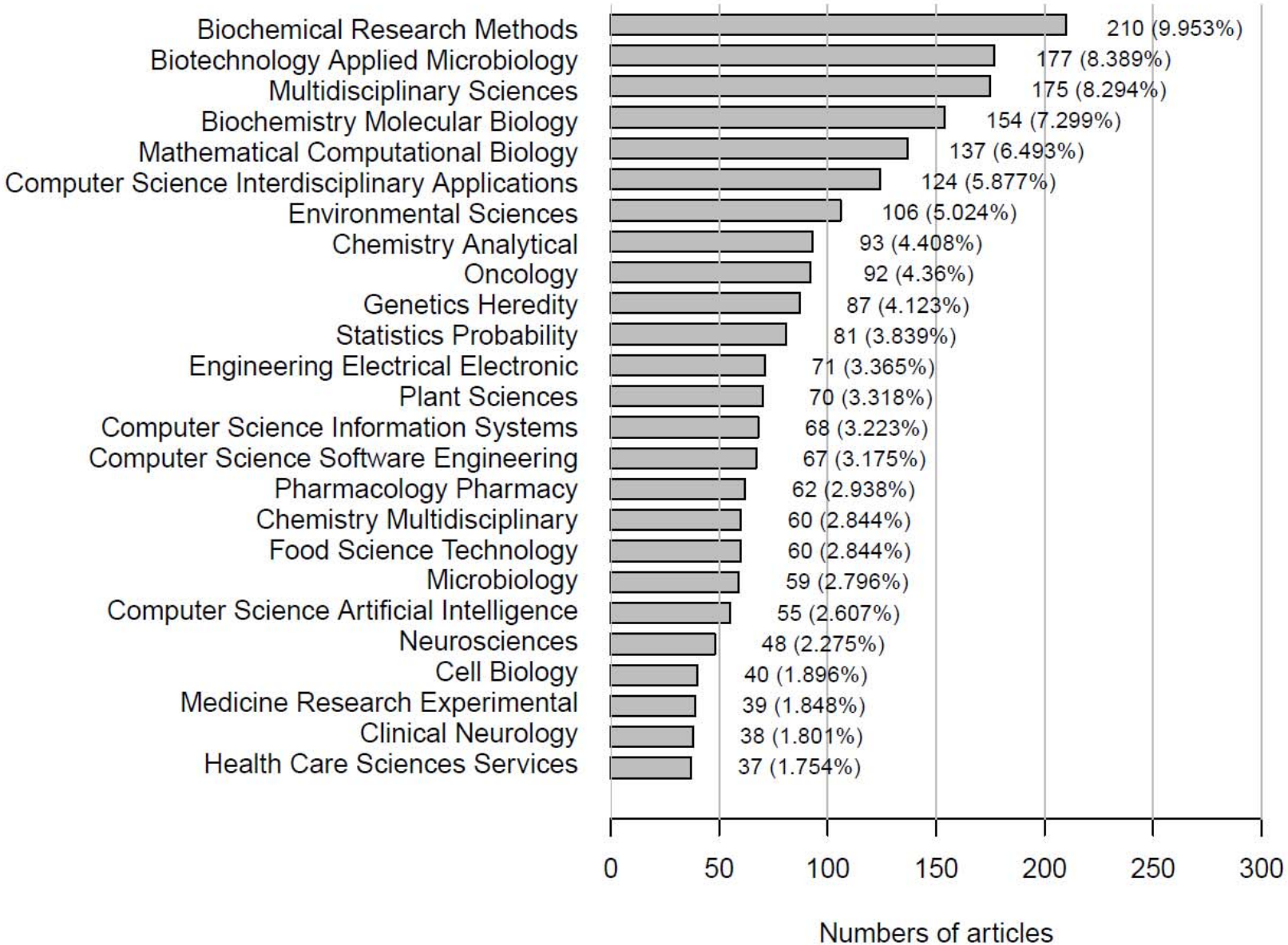
What about this one?

Search “heatmap” (title/abstract) in the academic databases

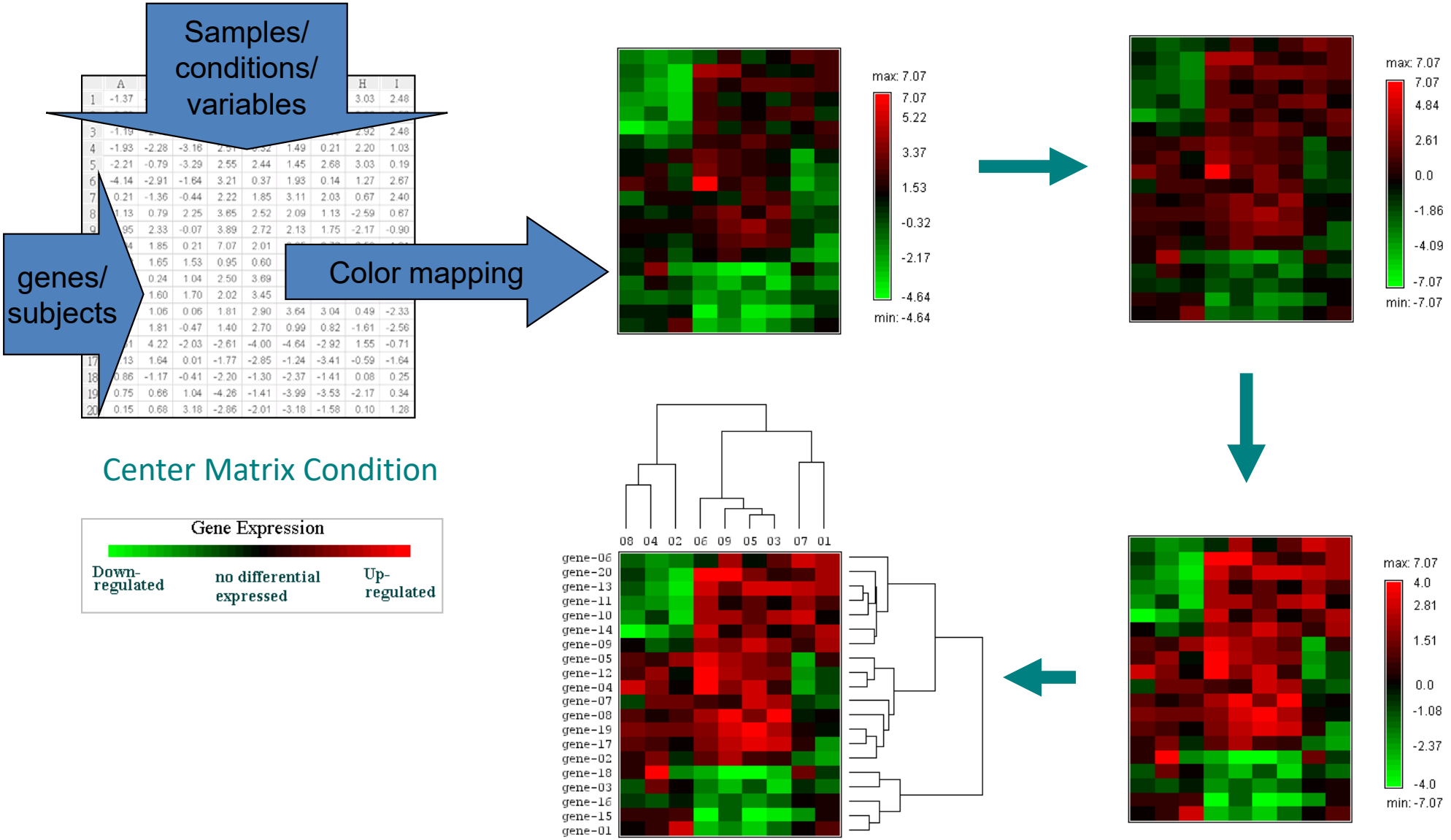


Top25 domains

Top 25 heatmap researches and applications domains (WOS).



Heat Map: Display Conditions



Hierarchical Clustering and Dendrogram (Agglomerative Nesting, AGNES)

Example: Average-Linkage

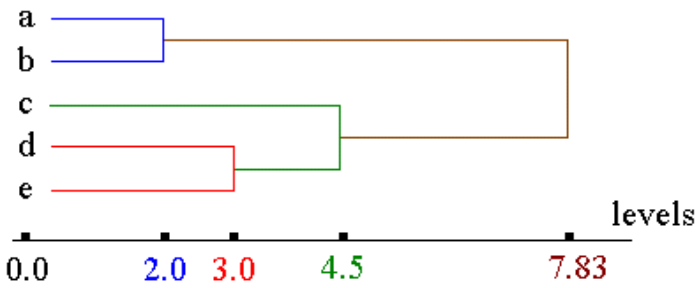
distance matrix

	a	b	c	d	e
a	0	2	6	10	9
b		0	5	9	8
c			0	4	5
d				0	3
e					0

	{a, b}	c	d	e
{a, b}	0	5.5	9.5	8.5
c		0	4	5
d			0	3
e				0

	{a, b}	c	{d, e}
{a, b}	0	5.5	9.0
c		0	4.5
{d, e}			0

	{a, b}	{c, d, e}
{a, b}	0	7.83
{c, d, e}		0



$$D(\{a,b\},\{c\}) = \frac{1}{2}[D(a,c)+D(b,c)]$$

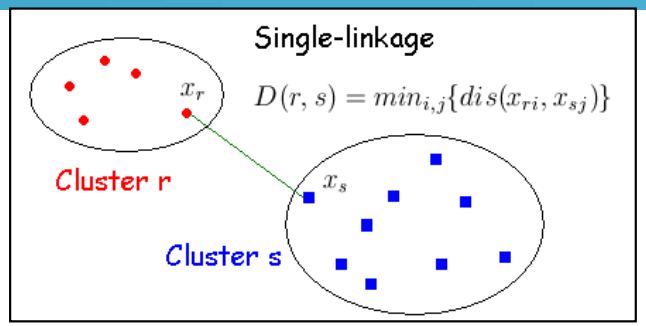
$$= \frac{1}{2}(6+5) = 5.5$$

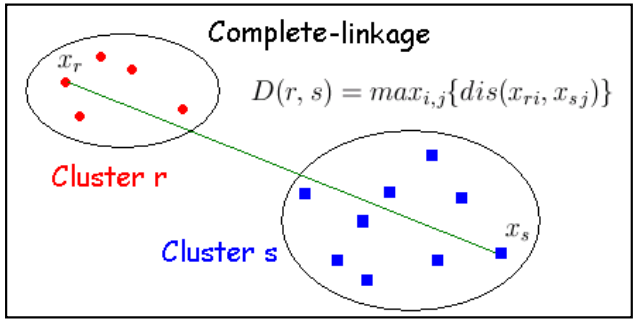
$$D(\{a,b\},\{d,e\})$$

$$= \frac{1}{4}[D(a,d)+D(a,e)+D(b,d)+D(b,e)]$$

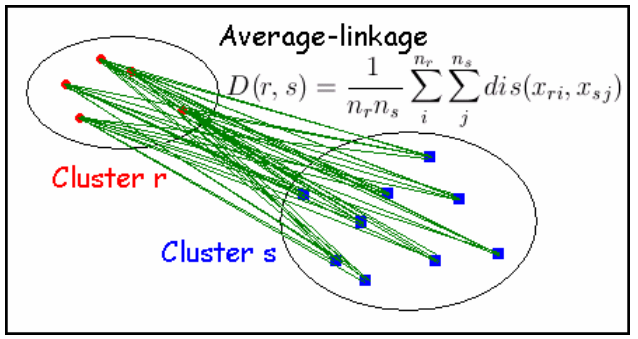
$$= \frac{1}{4}(10+9+9+8) = 9$$

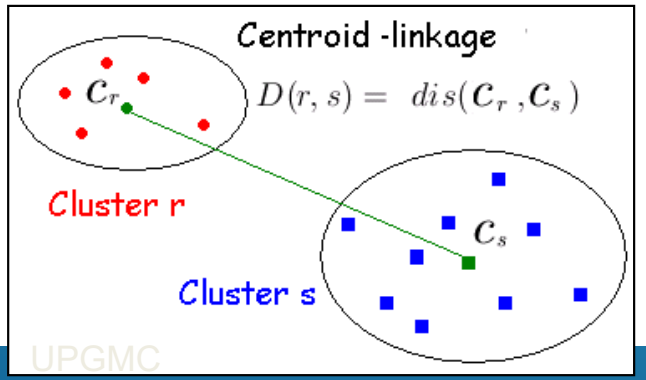
(Kaufman and Rousseeuw, 1990)



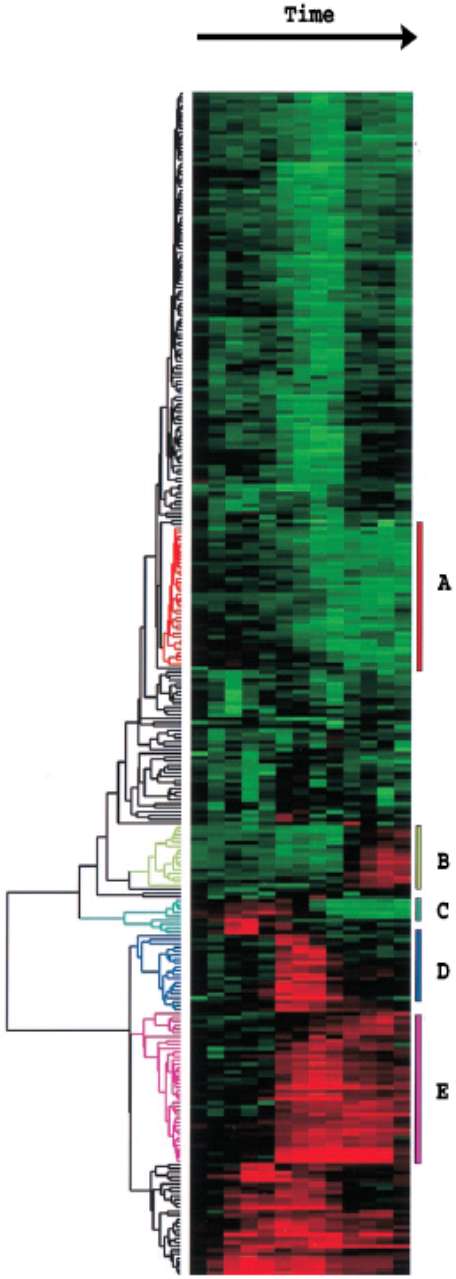


UPGMA (Unweighted Pair-Groups Method Average)





Display of Genome-Wide Expression Patterns



Proc. Natl. Acad. Sci. USA
Vol. 95, pp. 14863–14868, December 1998
Genetics

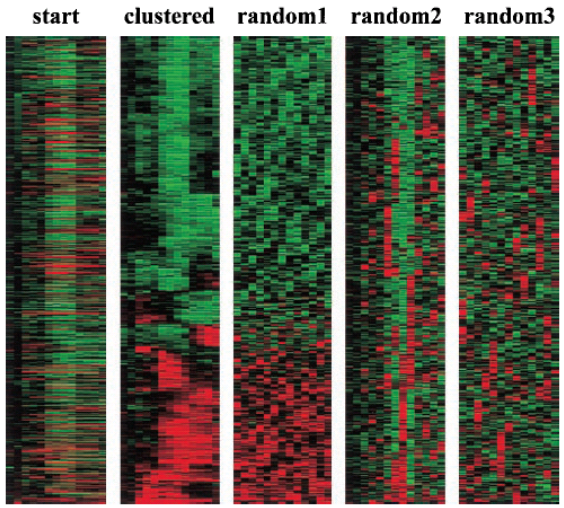
Cluster analysis and display of genome-wide expression patterns

MICHAEL B. EISEN*, PAUL T. SPELLMAN*, PATRICK O. BROWN†, AND DAVID BOTSTEIN*‡

FIG. 1. Clustered display of data from time course of serum stimulation of primary human fibroblasts. Experimental details are described elsewhere (11). Briefly, foreskin fibroblasts were grown in culture and were deprived of serum for 48 hr. Serum was added back and samples taken at time 0, 15 min, 30 min, 1 hr, 2 hr, 3 hr, 4 hr, 8 hr, 12 hr, 16 hr, 20 hr, 24 hr. The final datapoint was from a separate unsynchronized sample. Data were measured by using a cDNA microarray with elements representing approximately 8,600 distinct human genes. All measurements are relative to time 0. Genes were selected for this analysis if their expression level deviated from time 0 by at least a factor of 3.0 in at least 2 time points. The dendrogram and colored image were produced as described in the text; the color scale ranges from saturated green for log ratios −3.0 and below to saturated red for log ratios 3.0 and above. Each gene is represented by a single row of colored boxes; each time point is represented by a single column. Five separate clusters are indicated by colored bars and by identical coloring of the corresponding region of the dendrogram. As described in detail in ref. 11, the sequence-verified named genes in these clusters contain multiple genes involved in (*A*) cholesterol biosynthesis, (*B*) the cell cycle, (*C*) the immediate–early response, (*D*) signaling and angiogenesis, and (*E*) wound healing and tissue remodeling. These clusters also contain named genes not involved in these processes and numerous uncharacterized genes. A larger version of this image, with gene names, is available at http://rana.stanford.edu/clustering/serum.html.

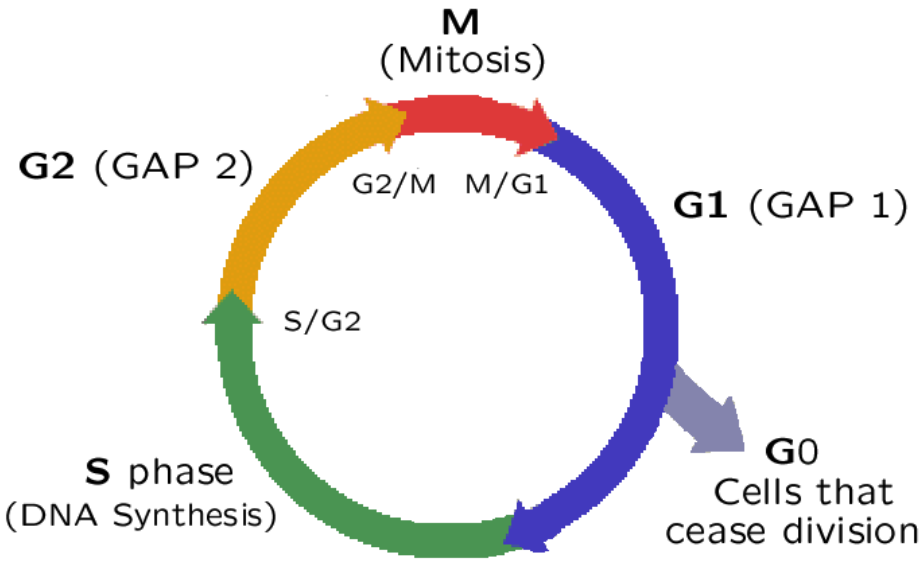
Software:
Cluster and TreeView

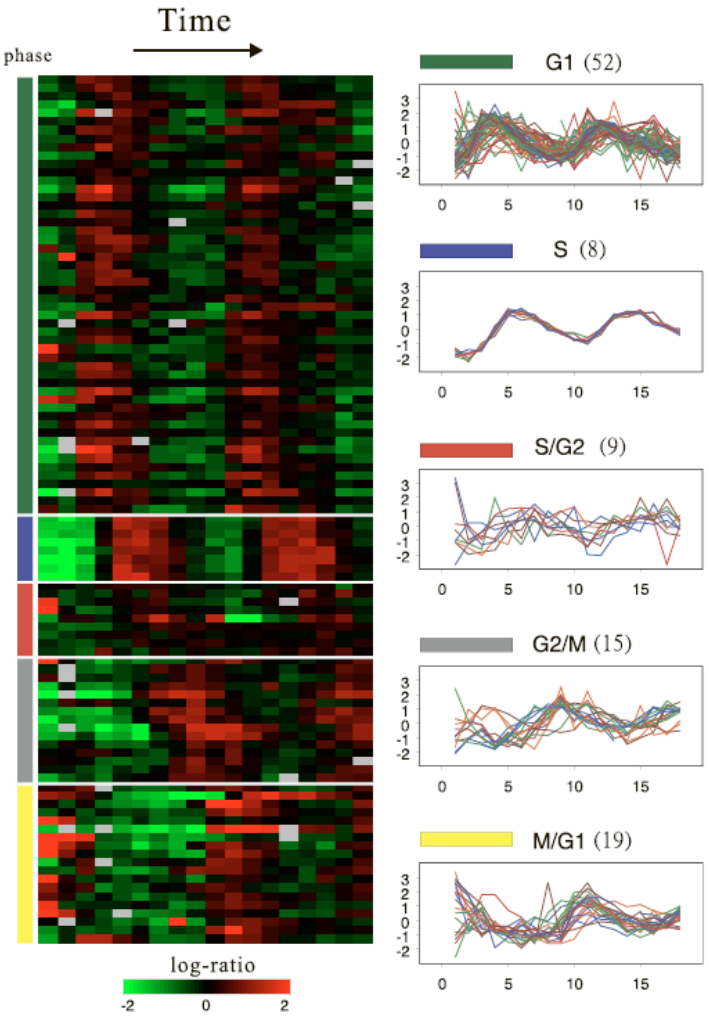
FIG. 3. To demonstrate the biological origins of patterns seen in Figs. 1 and 2, data from Fig. 1 were clustered by using methods described here before and after random permutation within rows (random 1), within columns (random 2), and both (random 3).



Microarray Data

- Lu and Wu (2010)
  - Time course data: every 7 minutes and totally 18 time points.
  - Known genes: there are 103 cell cycle-regulated genes by traditional method in G1, S, S/G2, G2/M, or M/G1. (Remove NA's: 79.)





See also: Using R to draw a Heatmap from Microarray Data
http://www2.warwick.ac.uk/fac/sci/moac/people/students/peter_cock/r/heatmap/

Heatmaps in R

Table 1: R packages for static MV/heatmaps.

Packages	Description	Reference
Autoimage	Multiple heatmaps for projected coordinates	French (2017)
ComplexHeatmap	Make complex heatmaps	Gu *et al.* (2016)
corrplot	Visualization of a correlation matrix	Wei *et al.* (2017)
fheatmap	Fantastic heatmap	Tumulu and Sindiri (2015)
heatmap3	An improved heatmap package	Zhao *et al.* (2014)
heatmap.plus	Heatmap with more sensible behavior	Day (2015)
Heatplus	Heatmaps with row and/or column covariates and colored clusters	Ploner (2020)
NeatMap	Nonclustering heatmap alternatives in R	Rajaram and Oono (2010)
mcheatmaps	Multiple matrices heatmap visualization	Chenard and Najmanovich (2014)
pheatmap	Pretty heatmaps	Kolde (2015)
superheat	A graphical tool for exploring complex datasets using heatmaps	Barter and Yu (2018)

Table 2: R packages and R-based tools for interactive and/or web-based MV/heatmaps.

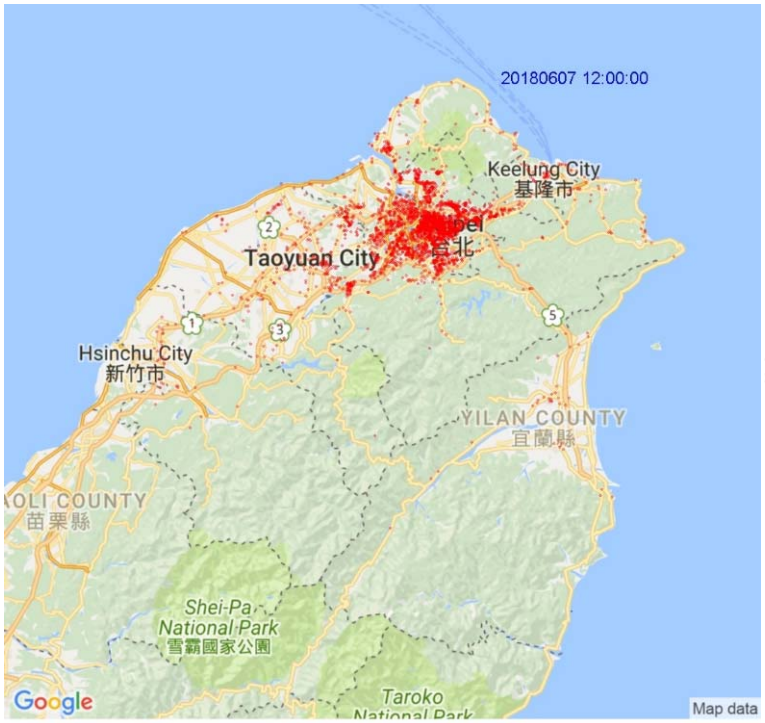
Packages	Description	Reference
clustvis	A web tool for visualizing clustering of multivariate data	Metsalu and Vilo (2015)
d3heatmap	Interactive heatmaps using htmlwidgets and D3.js	Cheng and Galili (2018)
gapmap	Drawing gapped cluster heatmaps with ggplot2	Sakai (2015)
heatmaply	Interactive cluster heatmaps using plotly	Galili *et al.* (2018)
Heatmapper	Web-enabled heat mapping for all	Babicki *et al.* (2016)
iheatmapr	Interactive, complex heatmaps	Schep and Kummerfeld (2017)
shinyheatmap	Ultrafast low-memory heatmap web interface for big data genomics	Khomtchouk *et al.* (2017)

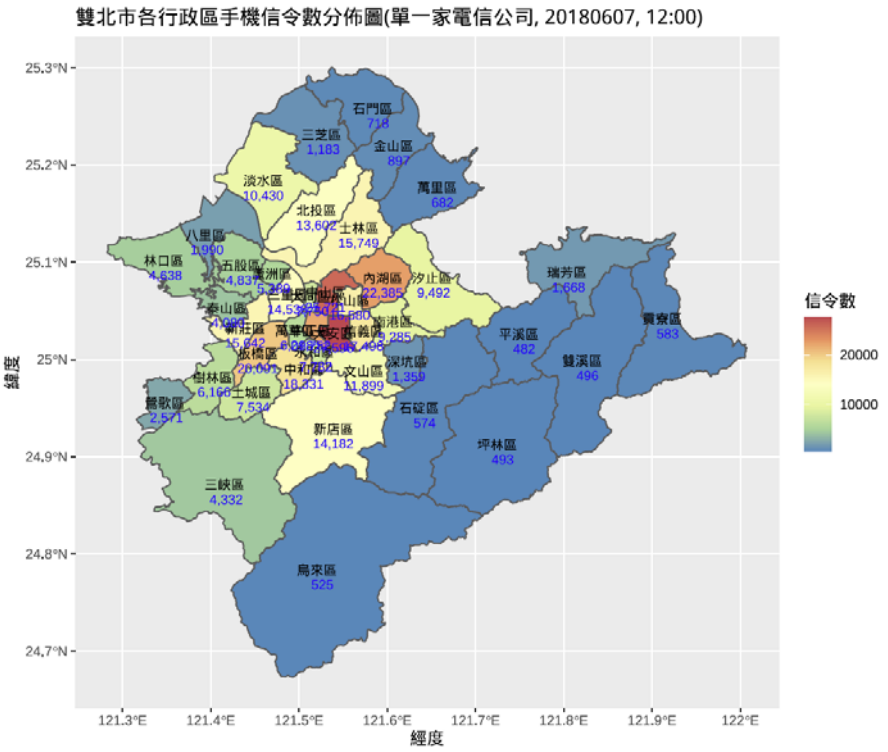
Wu, H. M. and Chen, C.H.* (2021), Matrix visualization: a review and perspective, WIRES Computational Statistics.

電信大數據

id,date,time,latitude(緯度),longitude(經度)

raw data





應用:

公共部門(例如: 資源配置、風險管理、績效評估及建置規畫)

民間部門(例如: 市場行銷、人力安排、商業選址及置產出租)

協同主持人，2020/04-2020/10，計畫名稱: 內政部「109 年電信信令資料整合」委外服務案。產學合作單位: 內政部統計處。[產學]

協同主持人 (2019/04－2019/12)，計畫名稱: 內政部「108 年國土資訊系統社會經濟資料庫擴建及資料服務平台推動計畫」委外服務案: 以電信手機信令資料推估人口參數及人口移動行為之研究。產學合作單位: 財團法人空間及環境科技文教基金會。委託單位：內政部統計處。

自2018 年起，主辦**台灣燈會**節日的各縣市政府(例如: 2018 年嘉義縣、2019 年屏東縣及 2020 年台中市)已應用此項技術分析遊客人群移動特徵，據此來調整活動資源配置及交通行程的安排規劃。

電信大數據

D2.01表示: 2018/06/05，00:00(含)~01:00共4個時間點(00:00，00:15，00:30，00:45)之停留地區，取最多停留地區為此時段之停留地區。D7.02表示: 2018/06/10，01:00(含)~02:00。

D7.02 D7.03 D7.04 D7.05 D7.06 D7.07 D7.08 D7.09 D7.10 D7.11

D2.01 D2.02 D2.03 D2.04 D2.05 D2.06 D2.07 D2.08 D2.09 D2.10

白色為遺失

id,date,time,latitude,longitude

raw data

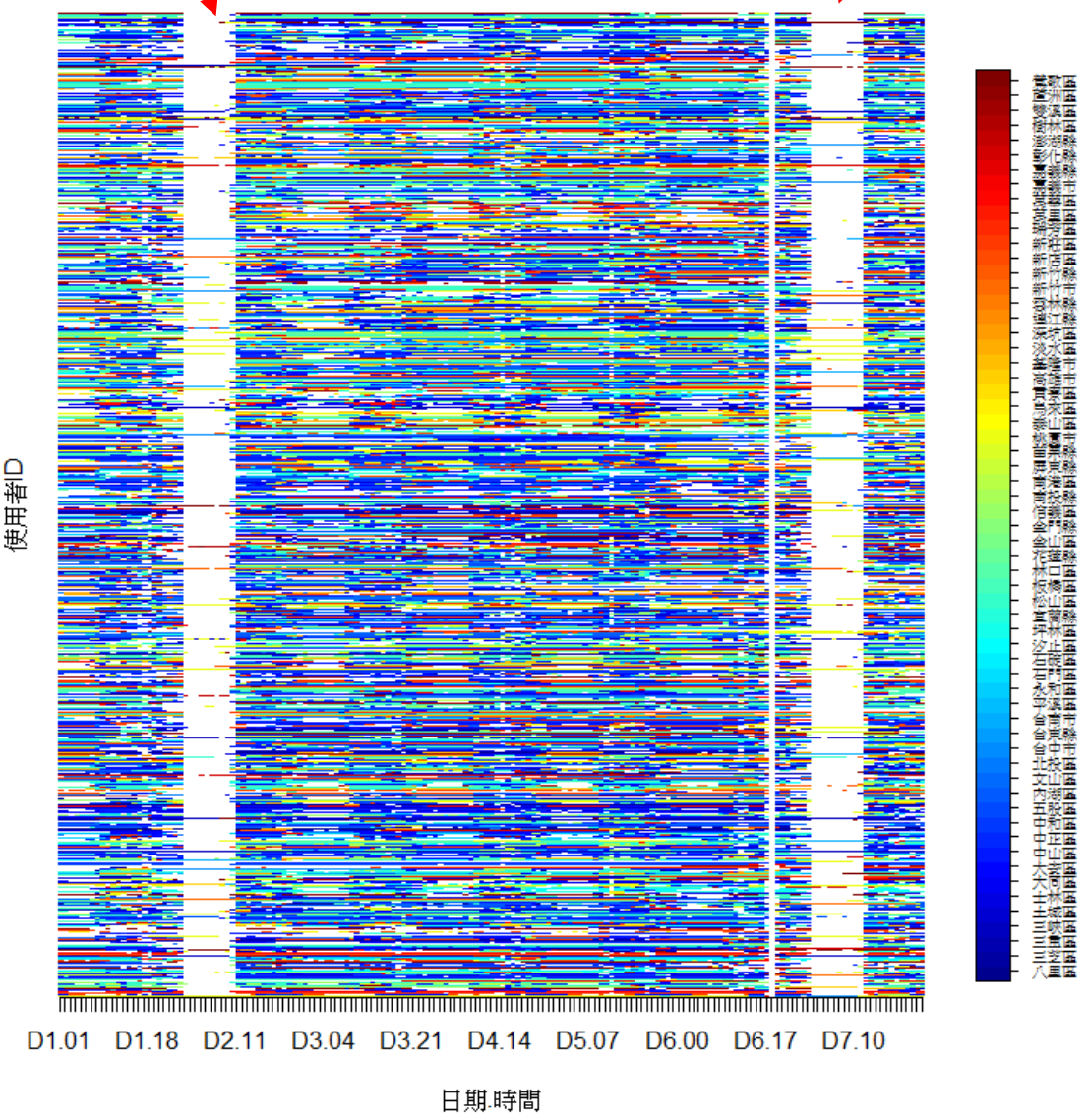
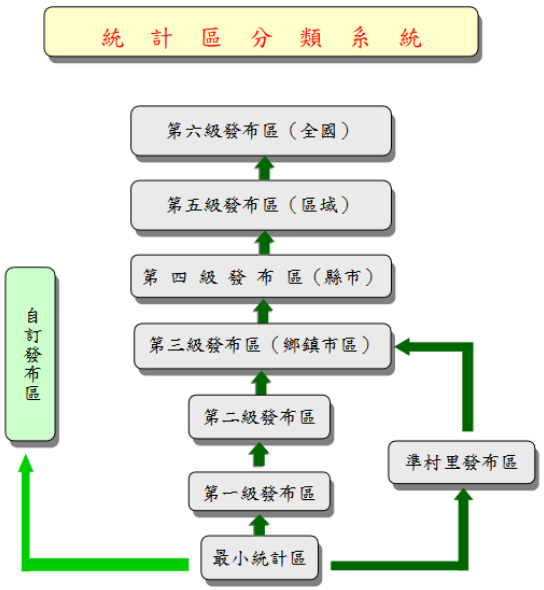
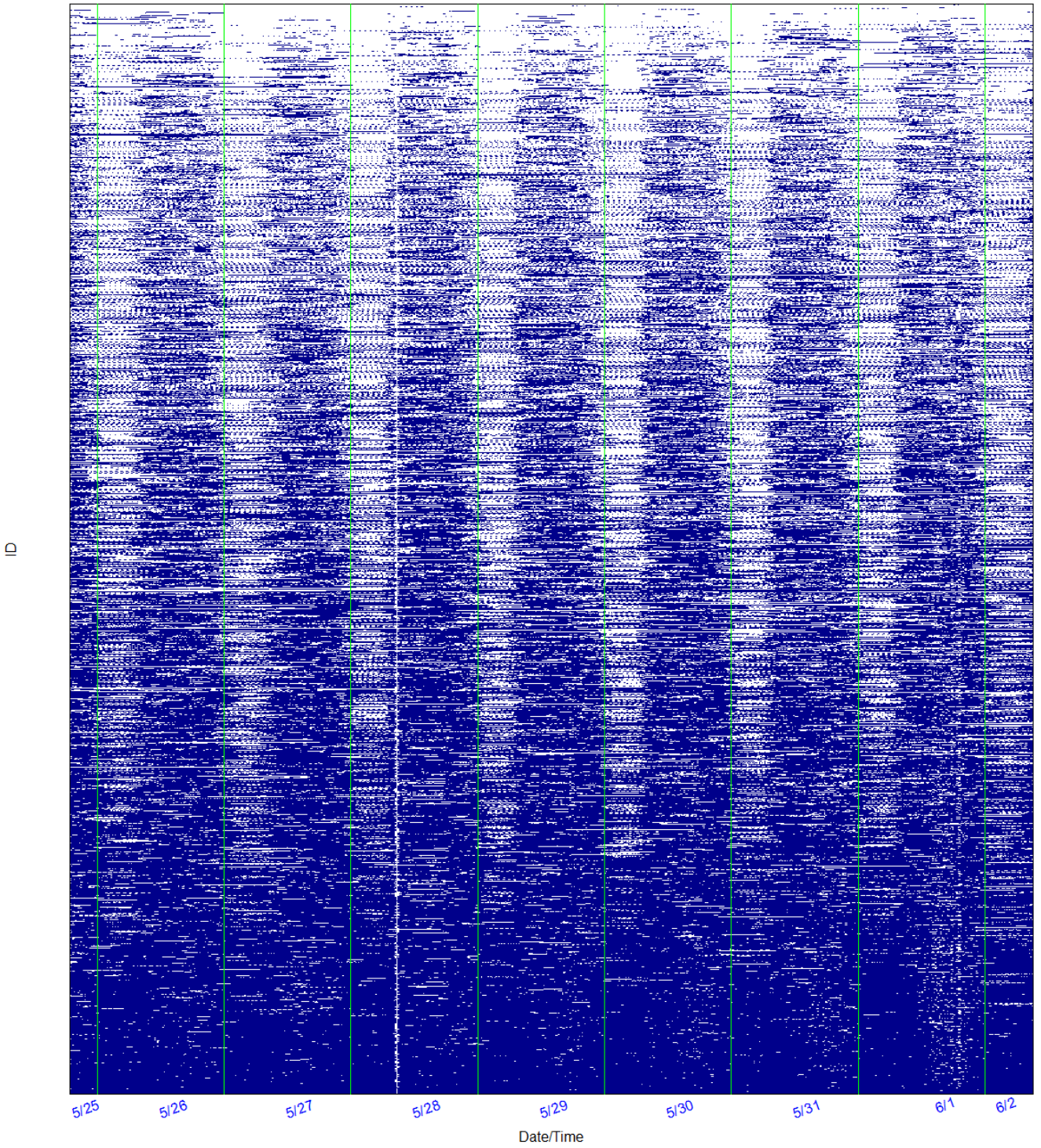
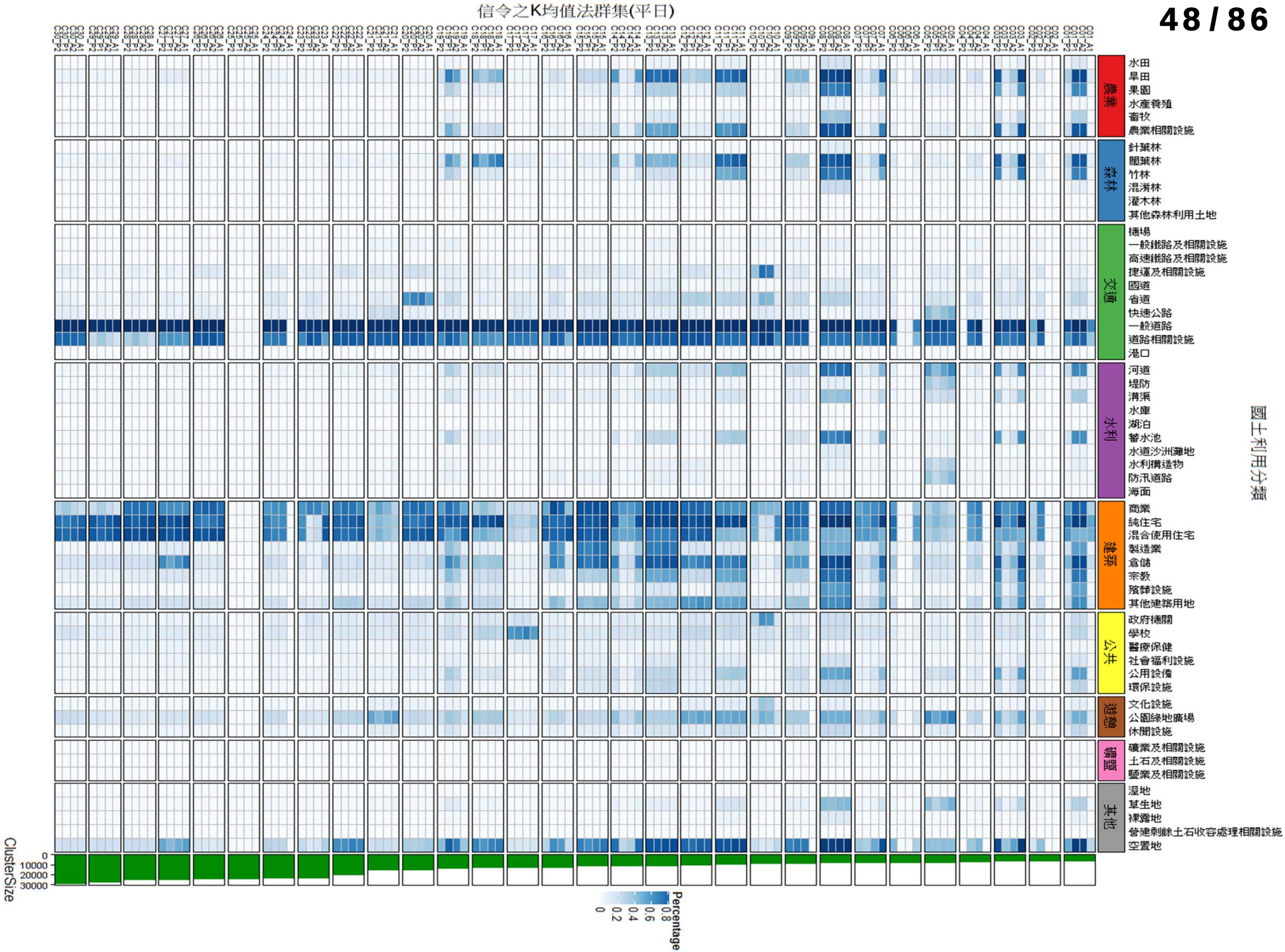


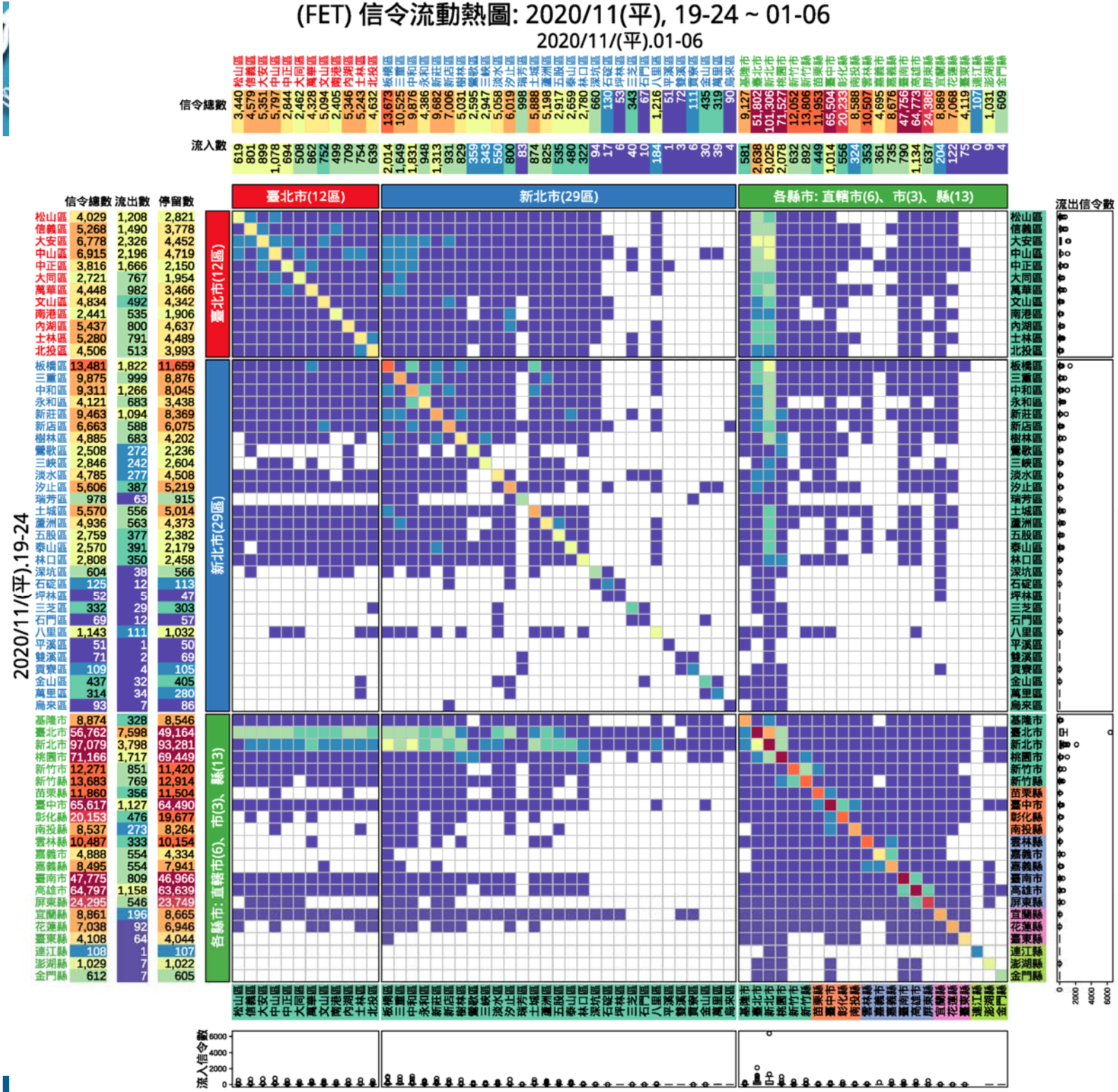
圖 1、統計區分類系統架構



Missing pattern for all 433743 IDs (2018/05/25~06/02)




信令之K均值法群集(平日)
國土利用分類
農業
水田
旱田
果園
水產養殖
畜牧
農業相關設施
森林
針葉林
闊葉林
竹林
混淆林
灌木林
其他森林利用土地
交通
機場
一般鐵路及相關設施
高速鐵路及相關設施
捷運及相關設施
國道
省道
快速公路
一般道路
道路相關設施
港口
水利
河道
堤防
溝渠
水庫
湖泊
蓄水池
水道沙洲灘地
水利構造物
防汛道路
海面
建築
商業
純住宅
混合使用住宅
製造業
倉儲
宗教
殯葬設施
其他建築用地
公共
政府機關
學校
醫療保健
社會福利設施
公用設備
環保設施
遊憩
文化設施
公園綠地廣場
休閒設施
礦鹽
礦業及相關設施
土石及相關設施
鹽業及相關設施
其他
溼地
草生地
裸露地
營建剩餘土石收容處理相關設施
空置地
ClusterSize
0
10000
20000
30000
Percentage
0
0.2
0.4
0.6
0.8



中正區
(t+1)

新店區
(t)

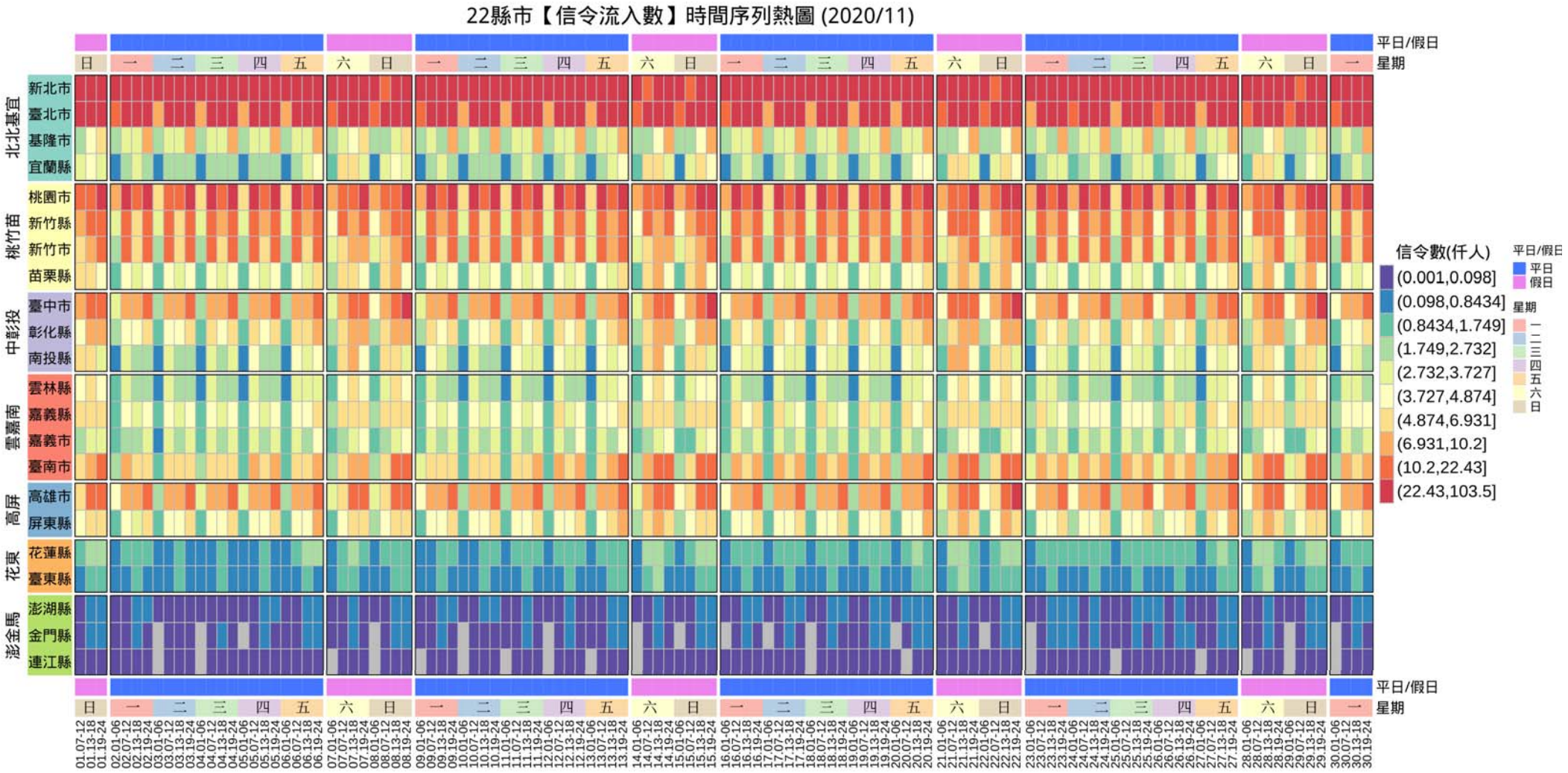
624

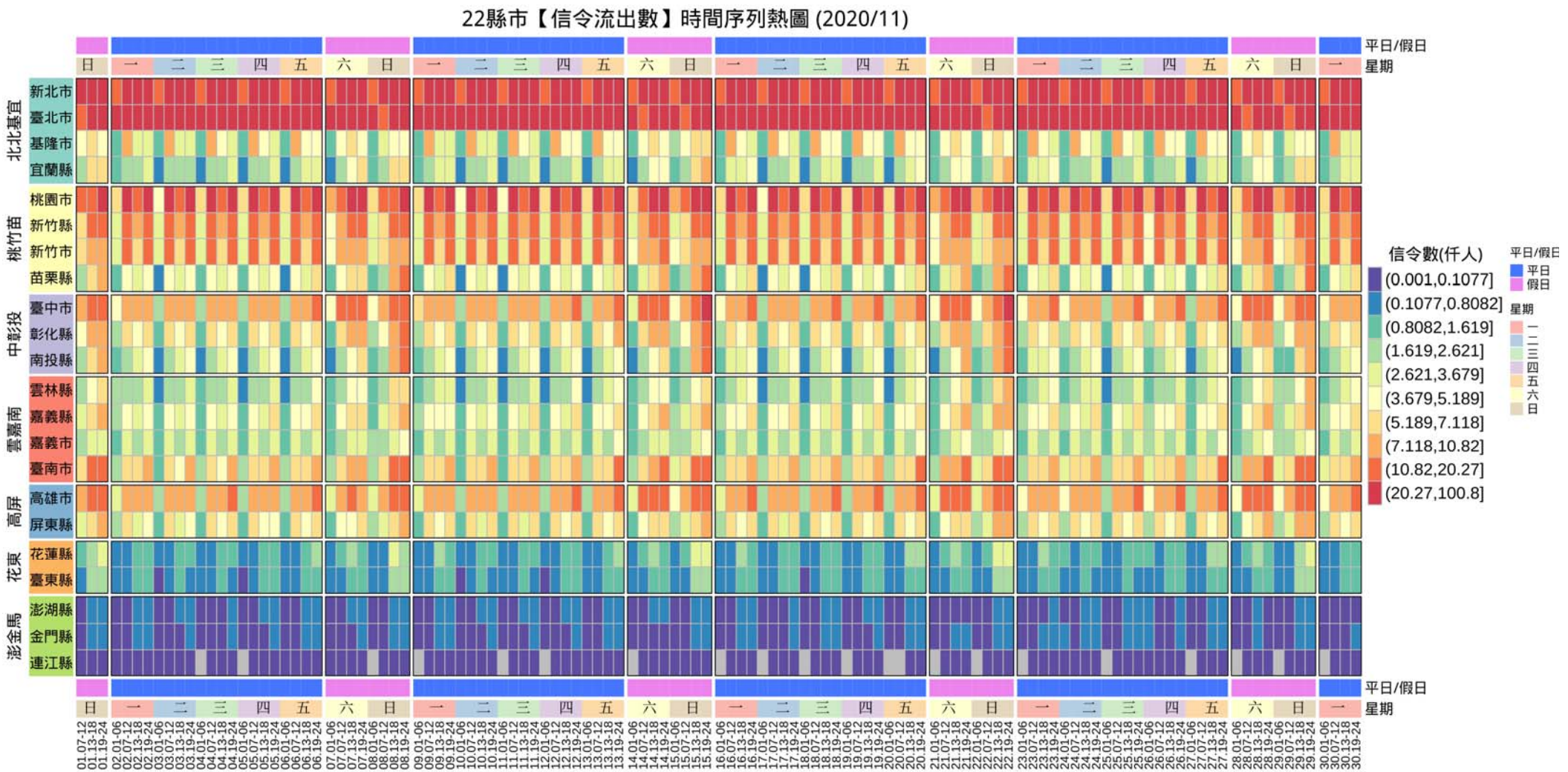
- 共624人從新店區(時間t)移動到中正區(時間t+1)。
- 對角線為停留在原區域之使用者人數。

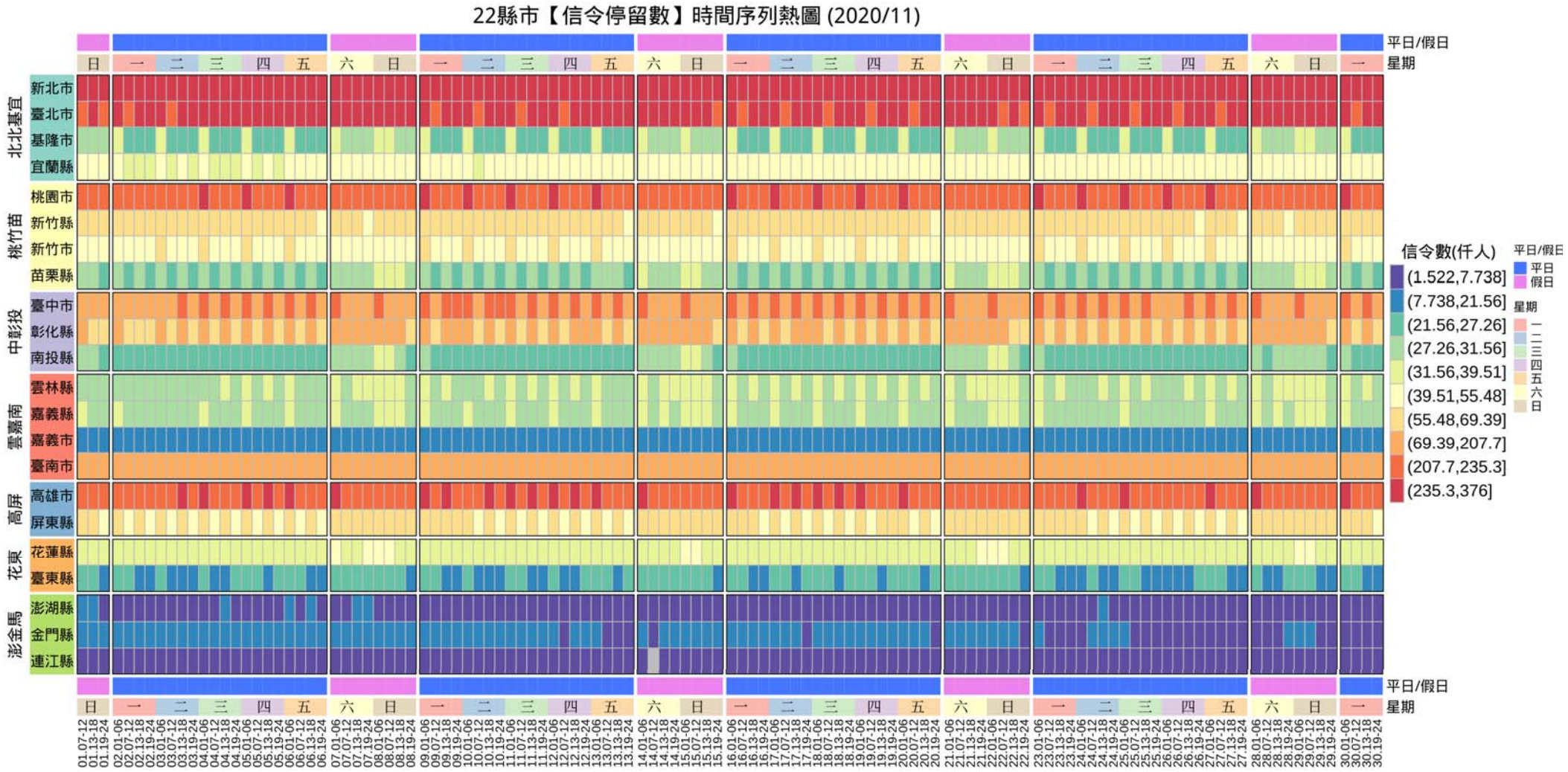
信令數
(0,100]
(100,300]
(300,500]
(500,1000]
(1000,3000]
(3000,5000]
(5000,10000]
(10000,20000]
(20000,40000]
(40000,5e+05]

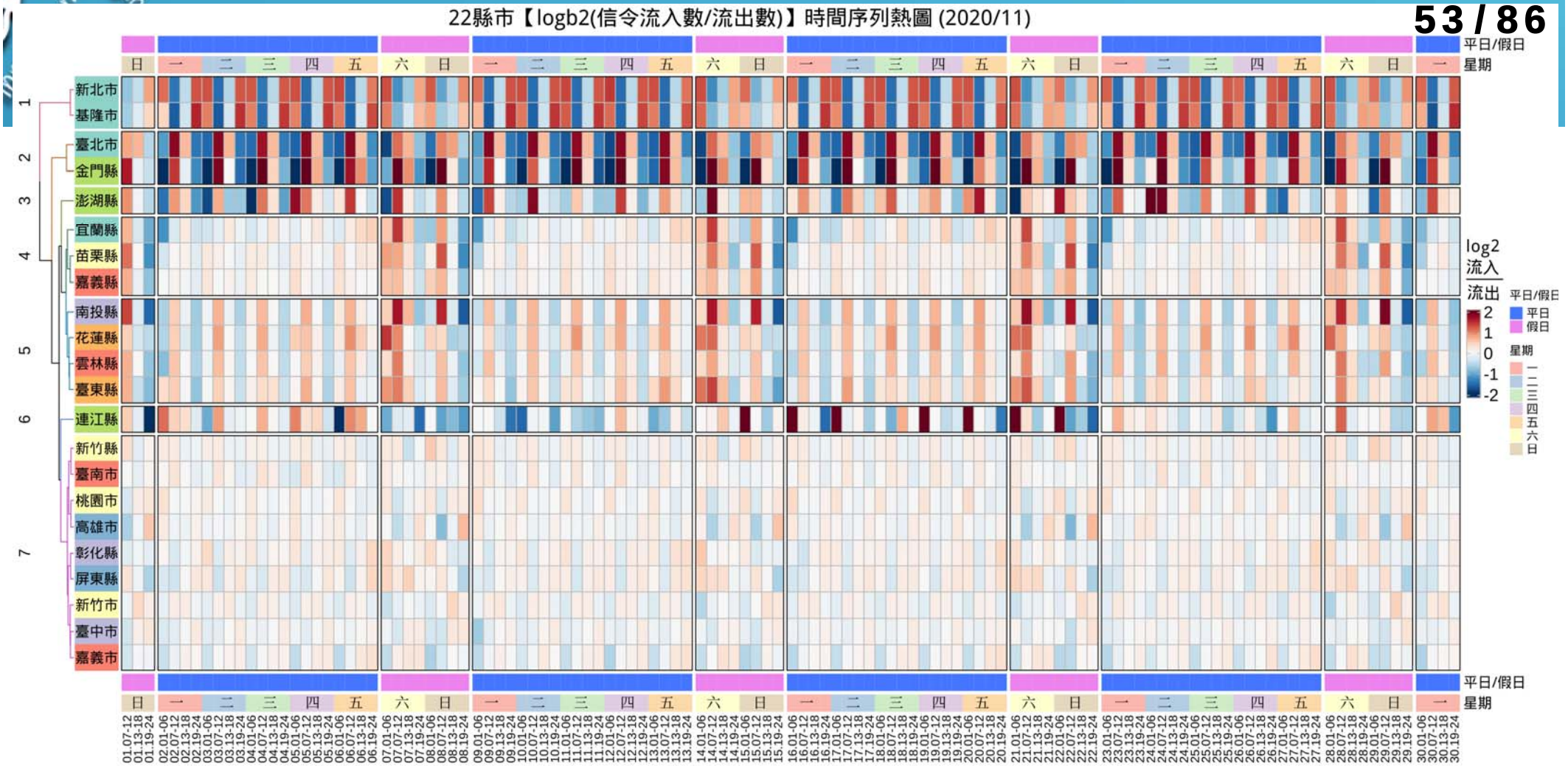
地區
北部
中部
南部
東部
外島

(FET, TWM, CHT)
2,152,725 by 4,320
(30 x 24 x 6)


22縣市【信令流入數】時間序列熱圖 (2020/11)
北北基宜
新北市
臺北市
基隆市
宜蘭縣
桃竹苗
桃園市
新竹縣
新竹市
苗栗縣
中彰投
臺中市
彰化縣
南投縣
雲嘉南
雲林縣
嘉義縣
嘉義市
臺南市
高屏
高雄市
屏東縣
花東
花蓮縣
臺東縣
澎金馬
澎湖縣
金門縣
連江縣
日 一 二 三 四 五 六 日
平日/假日
星期
信令數(仟人)
(0.001,0.098]
(0.098,0.8434]
(0.8434,1.749]
(1.749,2.732]
(2.732,3.727]
(3.727,4.874]
(4.874,6.931]
(6.931,10.2]
(10.2,22.43]
(22.43,103.5]
平日
假日
一 二 三 四 五 六 日


22縣市【信令流出數】時間序列熱圖 (2020/11)
北北基宜
新北市
臺北市
基隆市
宜蘭縣
桃竹苗
桃園市
新竹縣
新竹市
苗栗縣
中彰投
臺中市
彰化縣
南投縣
雲嘉南
雲林縣
嘉義縣
嘉義市
臺南市
高屏
高雄市
屏東縣
花東
花蓮縣
臺東縣
澎金馬
澎湖縣
金門縣
連江縣
日 一 二 三 四 五 六 日
平日/假日
星期
信令數(仟人)
(0.001,0.1077]
(0.1077,0.8082]
(0.8082,1.619]
(1.619,2.621]
(2.621,3.679]
(3.679,5.189]
(5.189,7.118]
(7.118,10.82]
(10.82,20.27]
(20.27,100.8]
平日
假日
一 二 三 四 五 六 日


22縣市【信令停留數】時間序列熱圖 (2020/11)
北北基宜
新北市
臺北市
基隆市
宜蘭縣
桃竹苗
桃園市
新竹縣
新竹市
苗栗縣
中彰投
臺中市
彰化縣
南投縣
雲嘉南
雲林縣
嘉義縣
嘉義市
臺南市
高屏
高雄市
屏東縣
花東
花蓮縣
臺東縣
澎金馬
澎湖縣
金門縣
連江縣
日 一 二 三 四 五 六 日
平日/假日
星期
信令數(仟人)
(1.522,7.738]
(7.738,21.56]
(21.56,27.26]
(27.26,31.56]
(31.56,39.51]
(39.51,55.48]
(55.48,69.39]
(69.39,207.7]
(207.7,235.3]
(235.3,376]
平日
假日
一 二 三 四 五 六 日



- 當新北市是偏向流入時，同時段台北市是偏向流出，反之亦是。兩城市的平日及假日信令數流向樣態也有明顯的不同。

- 基隆市與新北市(第1群)、台北市與金門縣(第2群)，不論平日或假日皆有高度相似的信令數流向樣態;
- 信令數流向樣態非常相似: 第4群，第5群。
- 第4群地區與第5群地區的平日信令數流向樣態有明顯不同，第5群地區有明顯的流入高於流出或流出高於流入的時段，兩群地區在假日就較相似，皆有較大的流入現象，這符合大眾對第4群地區(宜苗嘉)及第5群地區(南雲花東)是熱門旅遊區的印象。
- 第7群: 不論平日或假日，流入數與流出數大約一致的縣市。
- 澎湖縣與連江縣各成一群(第3群、第6群)，信令數流向樣態與其它縣市的有很大不同，且兩縣市包含相對較高或較低的流入流出比值是因為於某些時段，信令數過少所致。

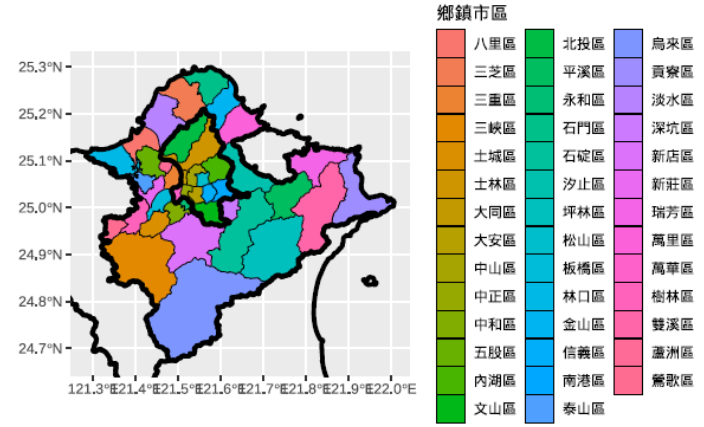
Matrix visualization and clustering of the mobile phone data based on symbolic data analysis

象徵性資料分析法於電信信令資料的矩陣視覺化與分群

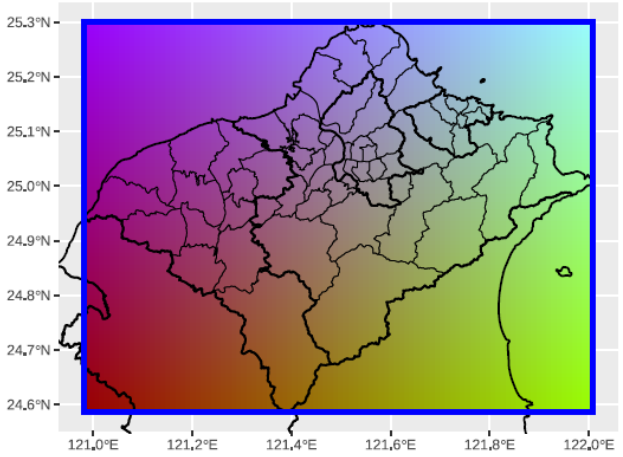
陳逸瑄[1]、王鴻龍[1]、吳漢銘[2]
[1]國立臺北大學統計學系，[2]國立政治大學統計學系

經緯度區間資料 $([\text{long}_{P_{12.5}}, \text{long}_{P_{87.5}}], [\text{lat}_{P_{12.5}}, \text{lat}_{P_{87.5}}])$

(a)



(b)



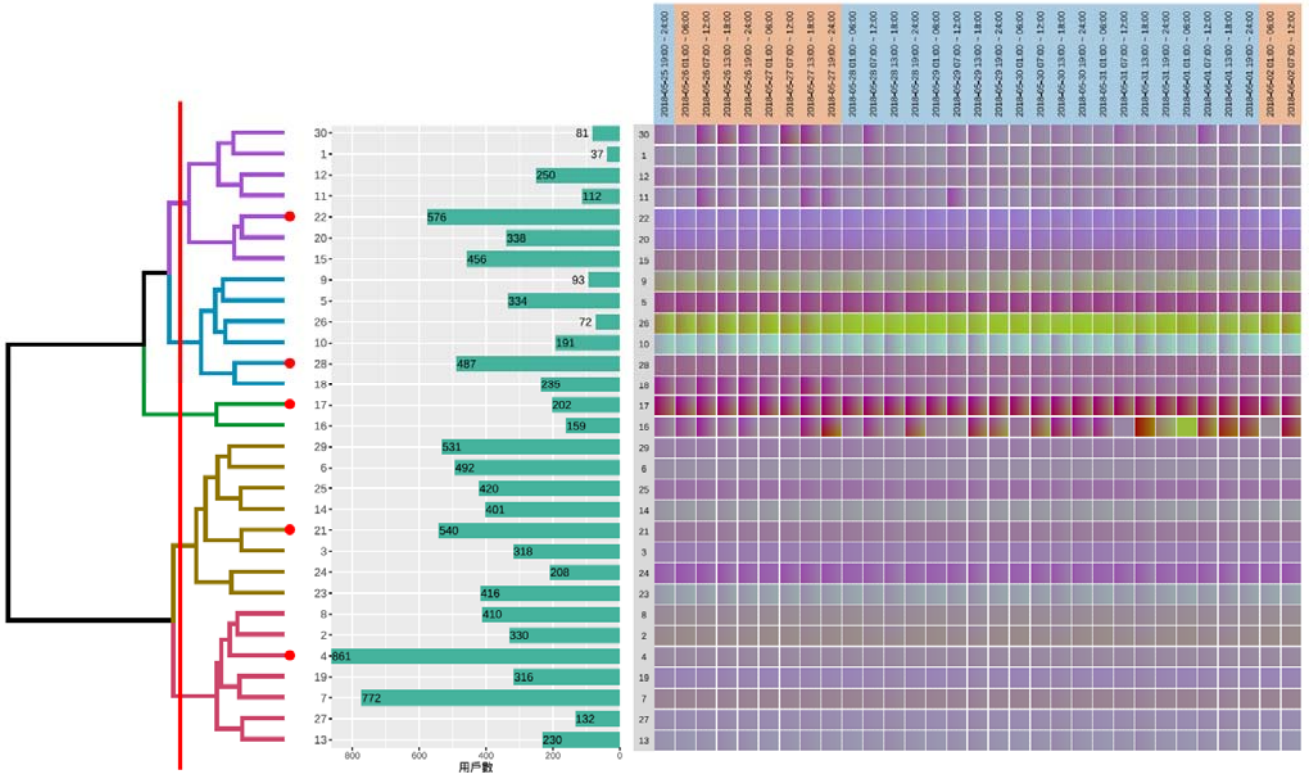
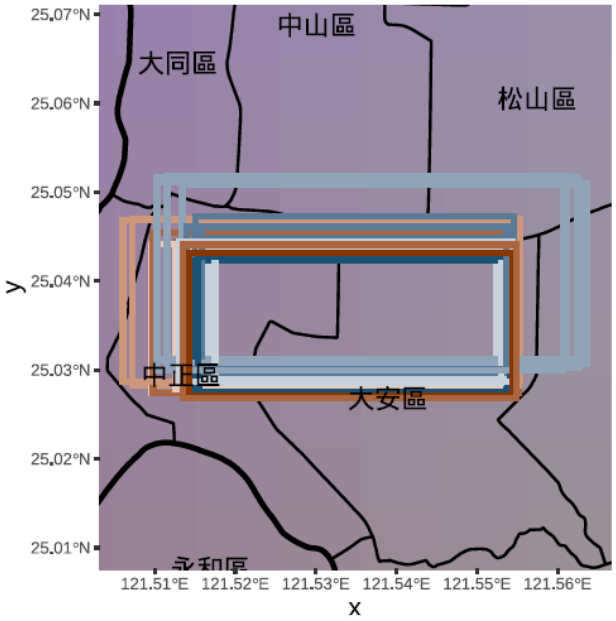


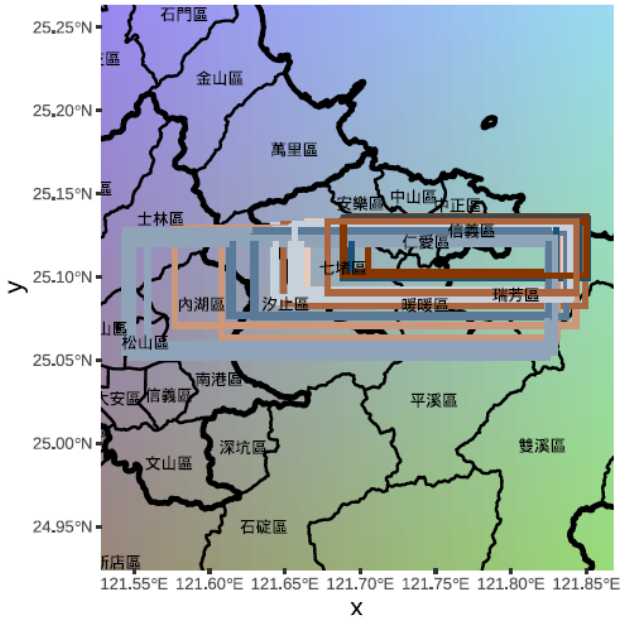
圖 11: 以區間型矩陣視覺化技術呈現人群移動樣態之熱圖，其中區間型經緯度資料來自第一階段的動態時間校正演算法搭配 K 均值法，並且區間座標的產製是以經緯度之 $P_{12.5}$ 和 $P_{87.5}$ 為依據。

區間軌跡地圖視覺化

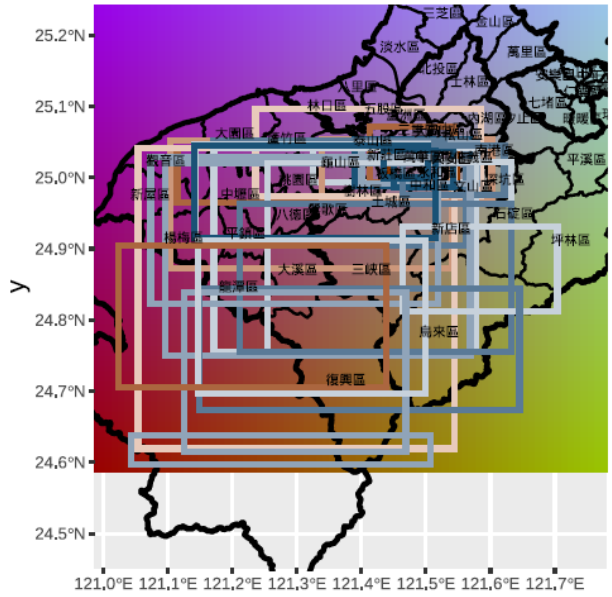
第 4 群的移動範圍



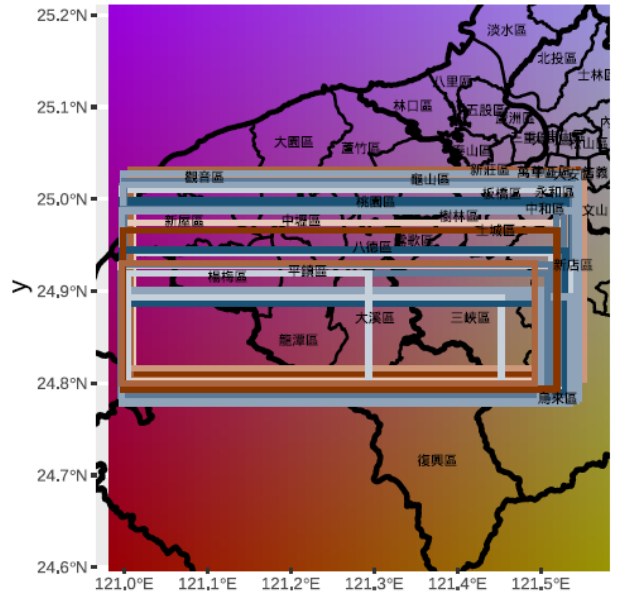
第 10 群的移動範圍



第 16 群的移動範圍



第 17 群的移動範圍



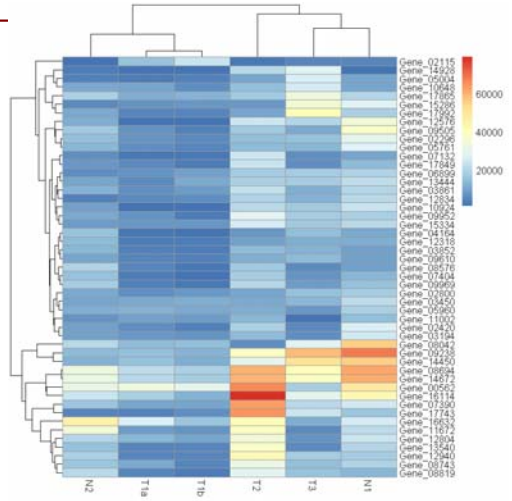
時間區段

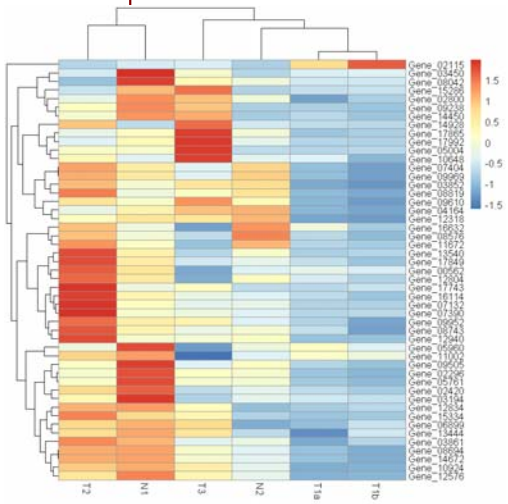
- 平日 01:00 ~ 06:00
- 平日 07:00 ~ 12:00
- 平日 13:00 ~ 18:00
- 平日 19:00 ~ 24:00
- 假日 01:00 ~ 06:00
- 假日 07:00 ~ 12:00
- 假日 13:00 ~ 18:00
- 假日 19:00 ~ 24:00

pheatmap: Pretty Heatmaps

Implementation of heatmaps that offers more control over dimensions and appearance.

```
> library(pheatmap)
> DESeq_subset <- read.csv("DESeq_subset.csv")
> dim(DESeq_subset)
[1] 49  7
> head(DESeq_subset)
           X   T1a   T1b    T2    T3    N1    N2
1 Gene_00562 32314 29693 66140 17973 47994 30878
2 Gene_02115 15261 23301  1944  4578  4087  1072
...
6 Gene_03194  7611  6806 13506  5727 25020  9235
> DESeq.X <- as.matrix(DESeq_subset[,2:ncol(DESeq_subset)])
> colnames(DESeq.X)
[1] "T1a" "T1b" "T2"  "T3"  "N1"  "N2"
> rownames(DESeq.X) <- DESeq_subset[,1]
> dimnames(DESeq.X)
[[1]]
 [1] "Gene_00562" "Gene_02115" "Gene_02296" "Gene_02420" ...
[46] "Gene_17743" "Gene_17849" "Gene_17865" "Gene_17992"

[[2]]
[1] "T1a" "T1b" "T2"  "T3"  "N1"  "N2"

> str(DESeq.X)
 int [1:49, 1:6] 32314 15261 6730 ...
 - attr(*, "dimnames")=List of 2
  ..$ : chr [1:49] "Gene_00562"...
  ..$ : chr [1:6] "T1a" "T1b" "T2" "T3" ...
> pheatmap(DESeq.X)
```



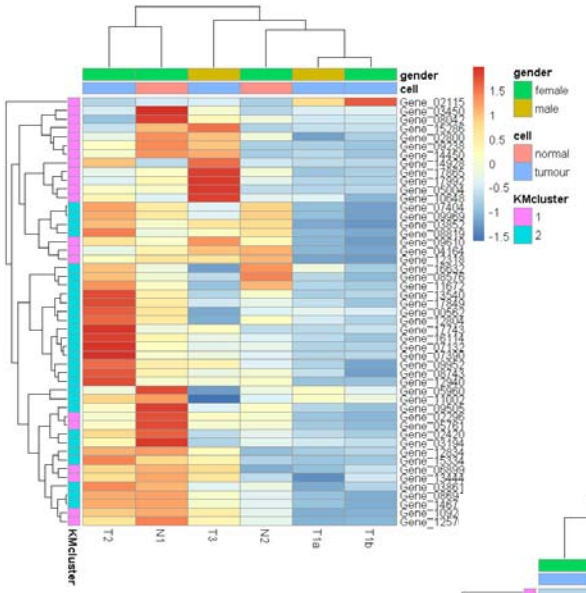


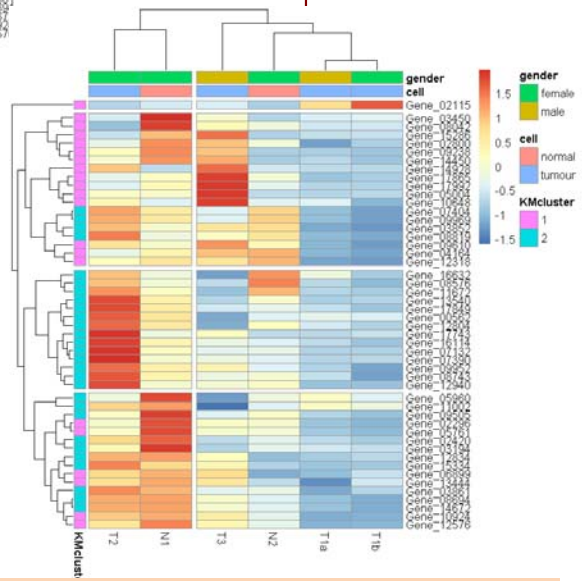
```
> DESeq.X.std <- t(apply(DESeq.X, 1, scale))
> class(DESeq.X.std)
[1] "matrix"
> dimnames(DESeq.X.std) <- dimnames(DESeq.X)
> pheatmap(DESeq.X.std) # note the color spectrum
```

Making a heatmap in R with the pheatmap package
https://davetang.org/muse/2018/05/15/making-a-heatmap-in-r-with-the-pheatmap-package/

pheatmap: Pretty Heatmaps

```
> sample.group <- data.frame(cell = rep(c("tumour", "normal"), c(4,2)),
+                            gender = sample(c("male", "female"), 6, replace=T))
> row.names(sample.group) <- colnames(DESeq.X)
> sample.group
      cell gender
T1a tumour   male
T1b tumour female
T2  tumour female
T3  tumour   male
N1  normal female
N2  normal female
> km <- as.character(kmeans(DESeq.X.std, 2)$cluster
> gene.cluster <- data.frame(KMcluster = km)
> row.names(gene.cluster) <- rownames(DESeq.X)
> head(gene.cluster)
           KMcluster
Gene_00562         2
Gene_02115         1
Gene_02296         1
Gene_02420         2
Gene_02800         1
Gene_03194         2
>
> pheatmap(DESeq.X.std,
+          annotation_row = gene.cluster,
+          annotation_col = sample.group)
>
> pheatmap(DESeq.X.std,
+          annotation_row = gene.cluster,
+          annotation_col = sample.group,
+          cutree_rows = 4,
+          cutree_cols = 2)
```






```
display_numbers = TRUE
```

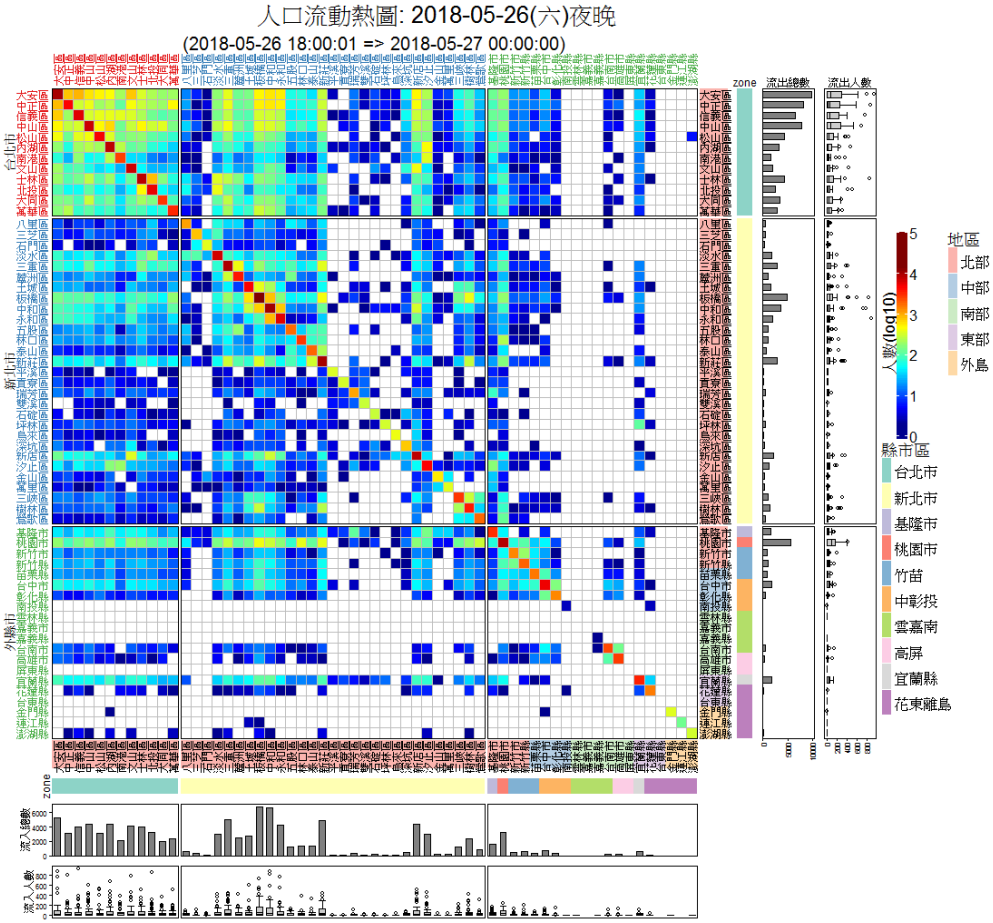
more examples:
https://www.jianshu.com/p/86ae39a227f4

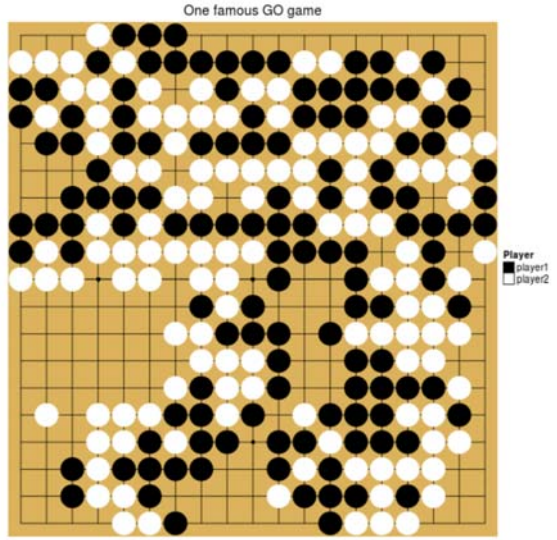
ComplexHeatmap

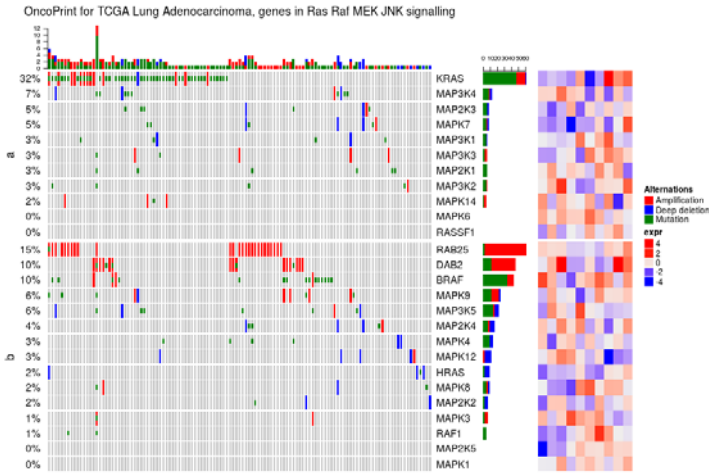
Zuguang Gu, Roland Eils, Matthias Schlesner, Complex heatmaps reveal patterns and correlations in multidimensional genomic data, Bioinformatics, Volume 32, Issue 18, 15 September 2016, Pages 2847–2849.

https://jokergoo.github.io/ComplexHeatmap-reference/book/

```
> if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
> BiocManager::install("ComplexHeatmap")
> library(ComplexHeatmap)
> Heatmap(exprs(selected.eset))
```







visualize multiple genomic alteration events by heatmap

Example: ComplexHeatmap

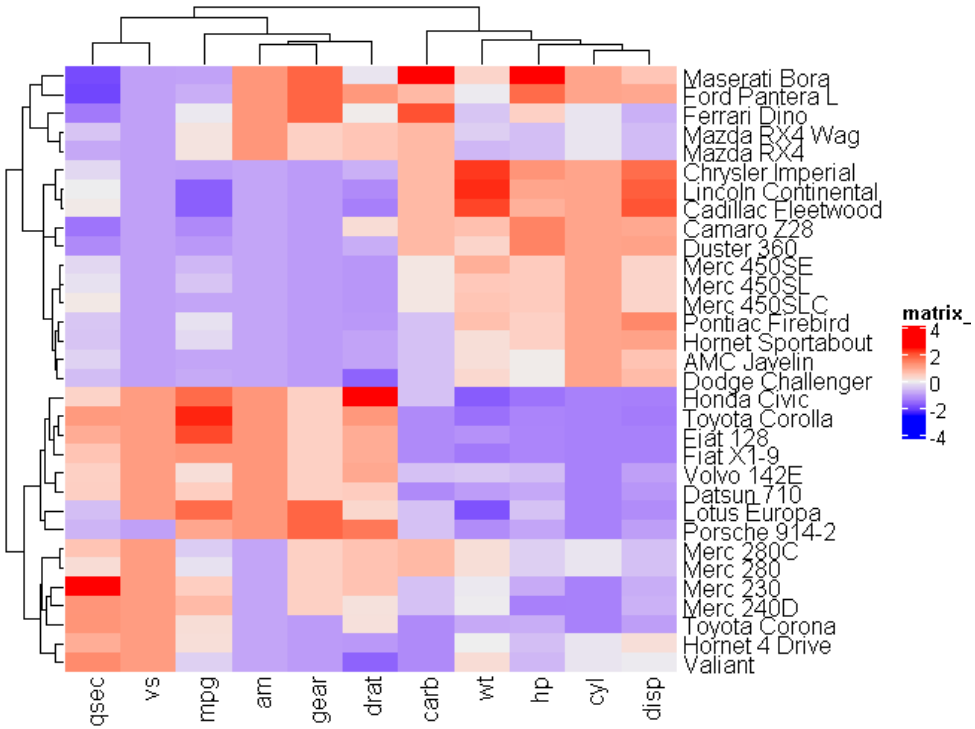
```
> head(mtcars)
                   mpg cyl disp  hp drat    wt  qsec vs am gear carb
Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
> str(mtcars)
'data.frame':    32 obs. of  11 variables:
 $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
 $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
 $ disp: num  160 160 108 258 360 ...
 $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
 $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
 $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
 $ qsec: num  16.5 17 18.6 19.4 17 ...
 $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
 $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
 $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
 $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
> mtcars.df <- scale(mtcars)
> class(mtcars.df)
[1] "matrix" "array"
> library(ComplexHeatmap)
> ? Heatmap
```

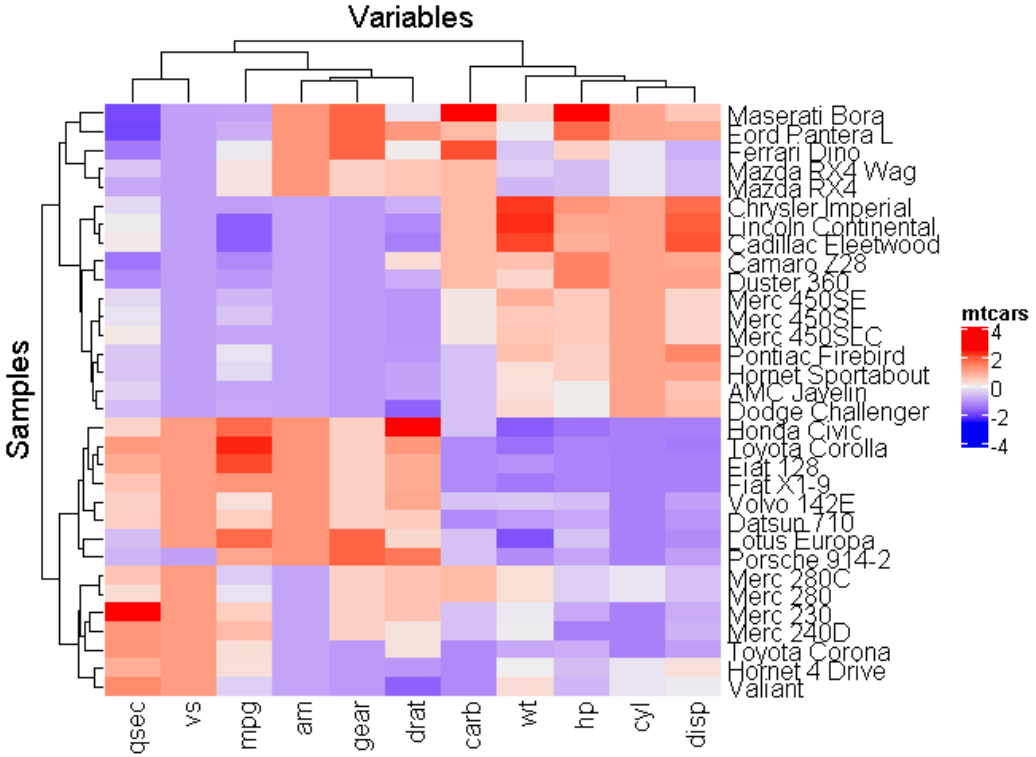
Reference
https://www.datanovia.com/en/lessons/heatmap-in-r-static-and-interactive-visualization/

ComplexHeatmap: 列、行標題

```
Heatmap(mtcars.df)
```

```
Heatmap(mtcars.df, name = "mtcars",
        row_title = "Samples",
        column_title = "Variables")
```

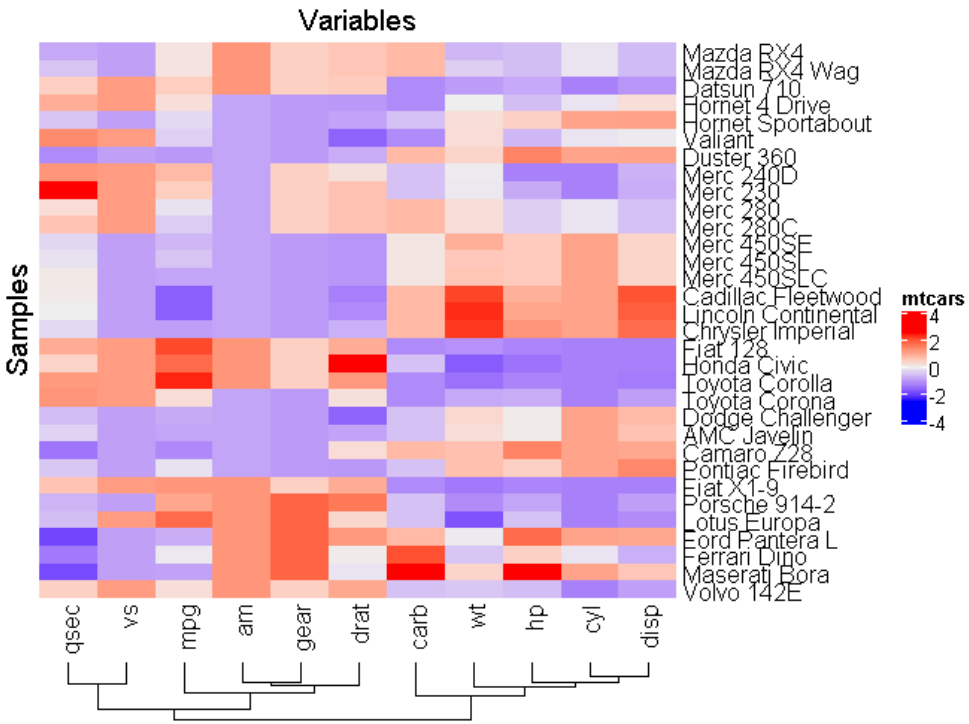


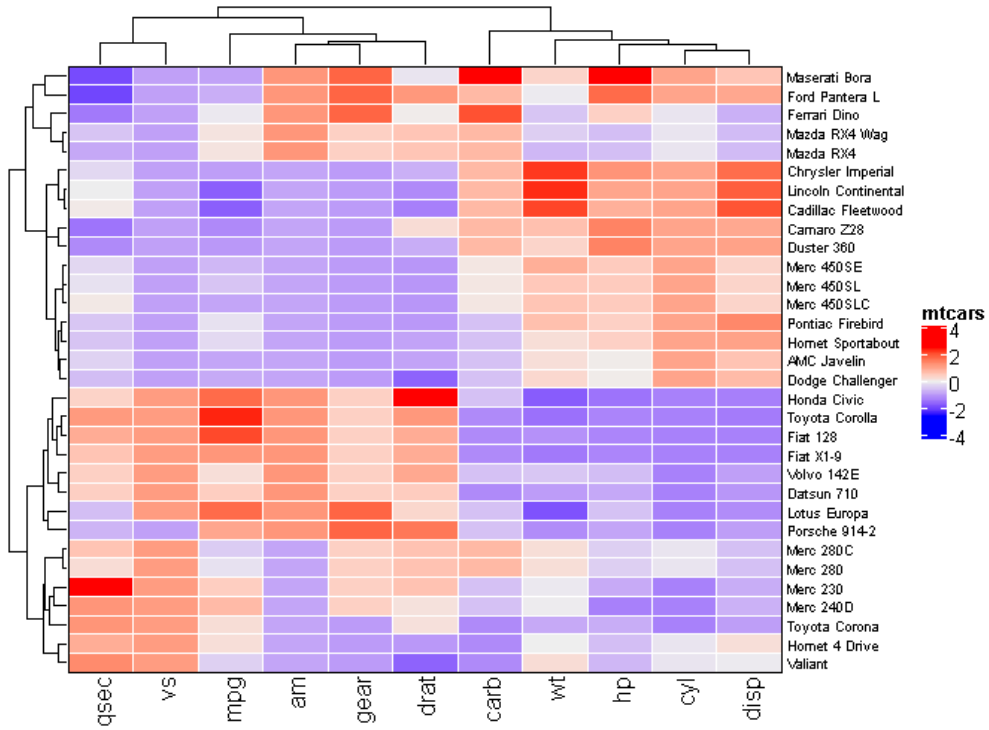


ComplexHeatmap:
樹狀圖位置、格線及字體大小

```
Heatmap(mtcars.df, name = "mtcars",
        row_title = "Samples",
        column_title = "Variables",
        cluster_rows = FALSE,
        column_dend_side = "bottom")
```

```
Heatmap(mtcars.df, name = "mtcars",
        rect_gp = gpar(col = "white", lwd = 1),
        border = TRUE,
        row_names_gp = gpar(fontsize = 7))
```

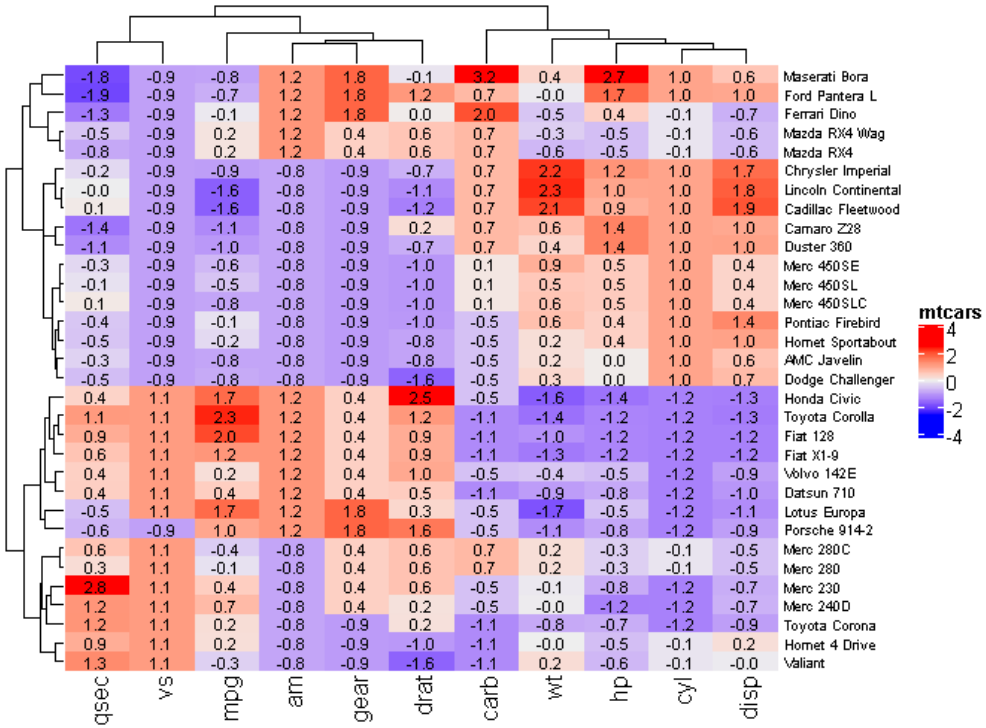


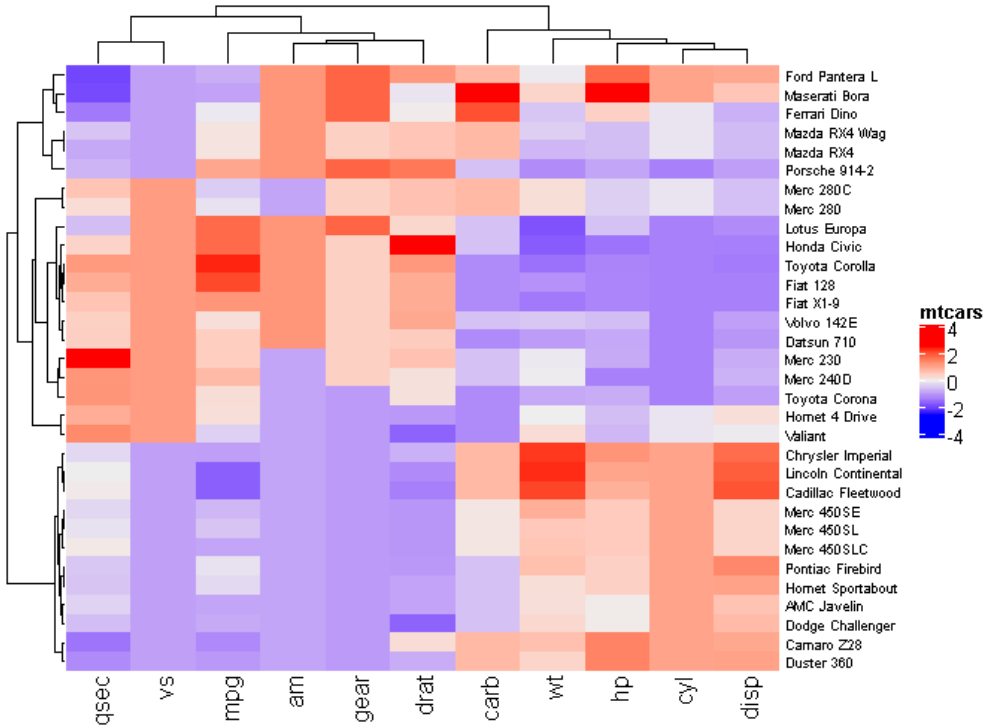


ComplexHeatmap: 顯示數字、指定分群法

```
Heatmap(mtcars.df, name = "mtcars",
        cell_fun = function(j, i, x, y, width, height, fill) {
          grid.text(sprintf("%.1f", mtcars.df[i, j]), x, y,
                    gp = gpar(fontsize = 8))},
        row_names_gp = gpar(fontsize = 7))
```

```
Heatmap(mtcars.df, name = "mtcars",
        clustering_distance_rows = "pearson",
        clustering_method_rows = "average",
        row_names_gp = gpar(fontsize = 7))
```

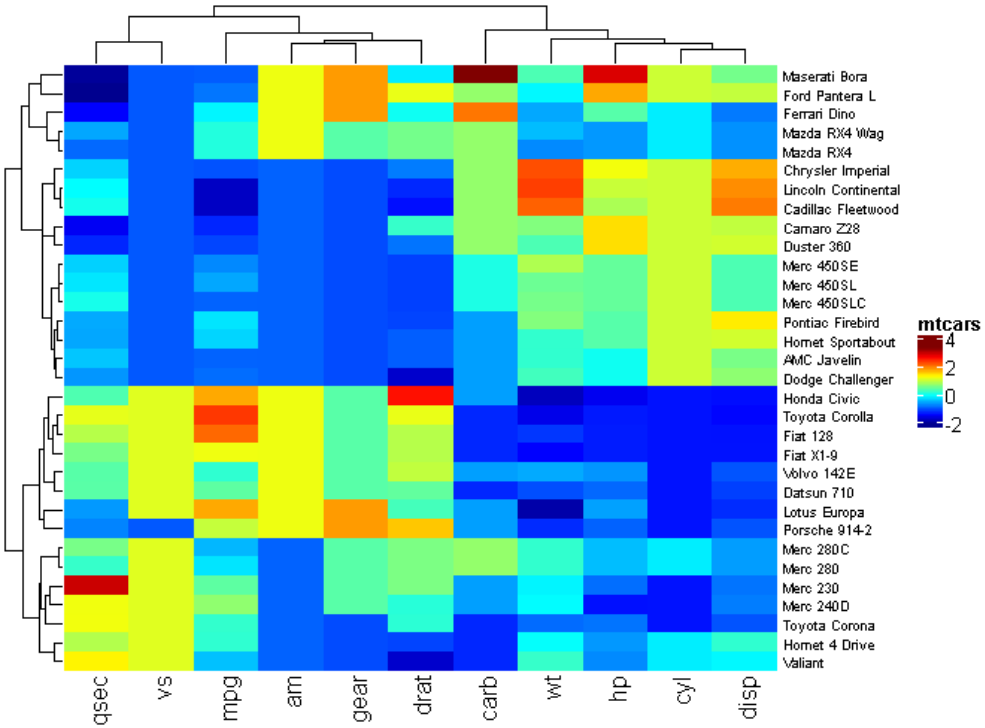


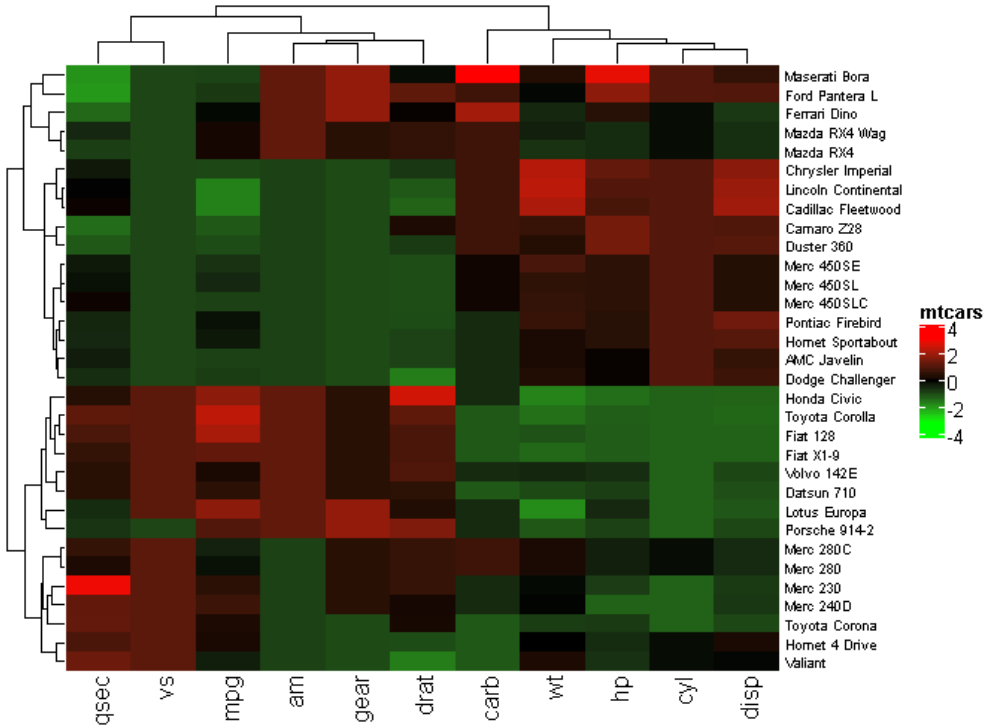


ComplexHeatmap: 自定色階

```
library(fields)
Heatmap(mtcars.df, name = "mtcars",
        col = tim.colors(),
        row_names_gp = gpar(fontsize = 7))
```

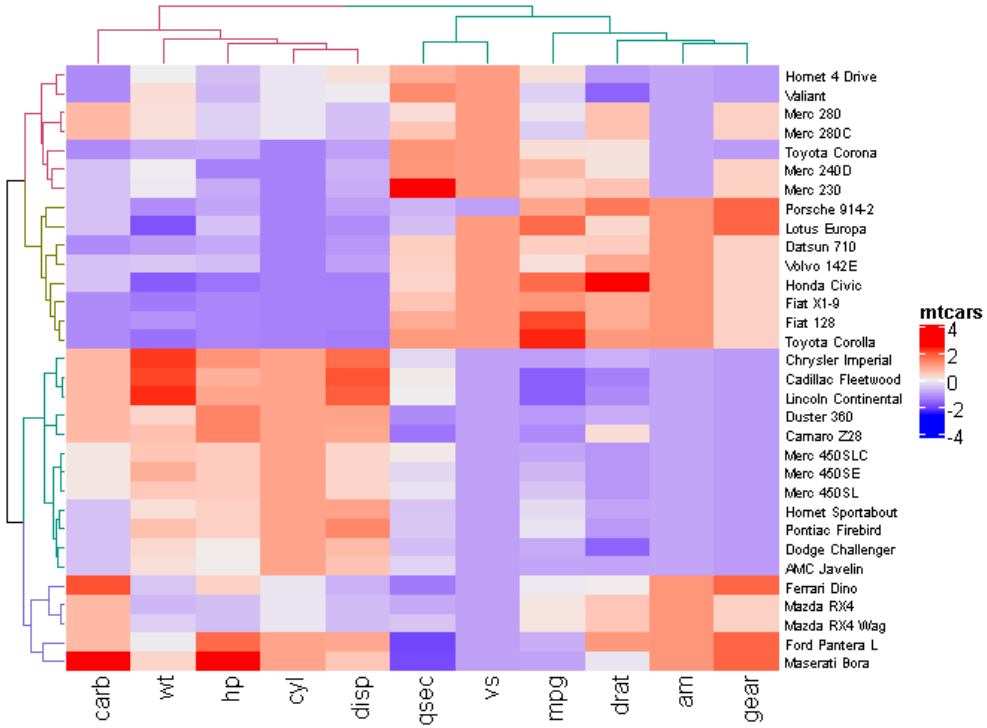
```
library(circlize) # Circular Visualization
mycolorRamp <- colorRamp2(c(-3, 0, 3), c("green", "black", "red"))
Heatmap(mtcars.df, name = "mtcars",
        col = mycolorRamp,
        row_names_gp = gpar(fontsize = 7))
```



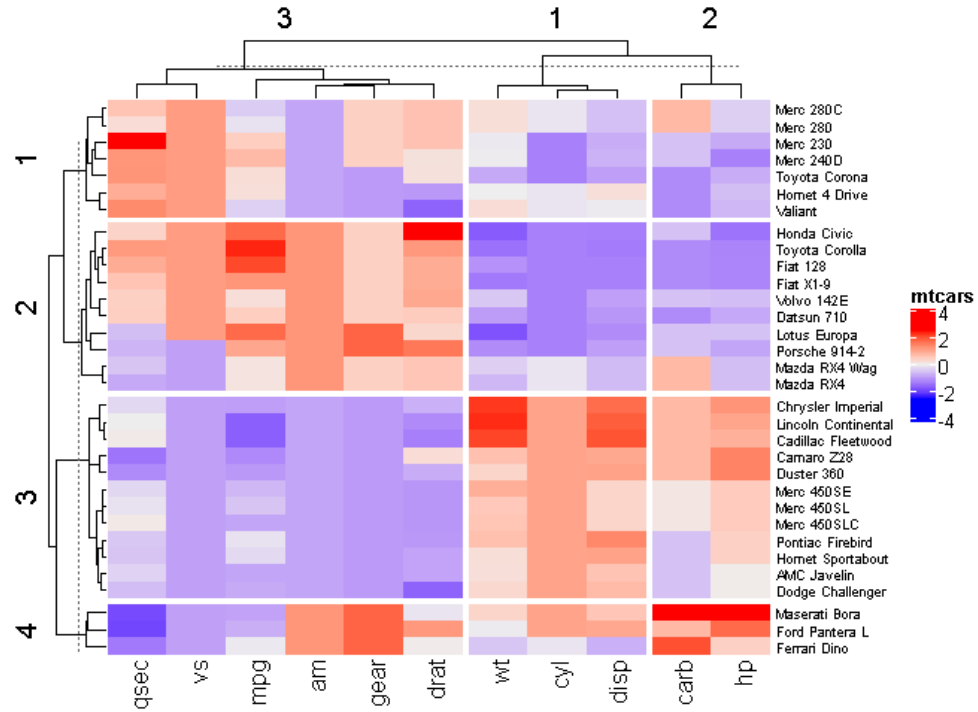


ComplexHeatmap: 以顏色標記樹狀圖、熱圖分割

```
library(dendextend)
row.dend <- hclust(dist(mtcars.df))
column.dend <- hclust(dist(t(mtcars.df)))
Heatmap(mtcars.df, name = "mtcars",
        cluster_rows = color_branches(row.dend, k = 4),
        cluster_columns = color_branches(column.dend, k = 2),
        row_names_gp = gpar(fontsize = 7))
```



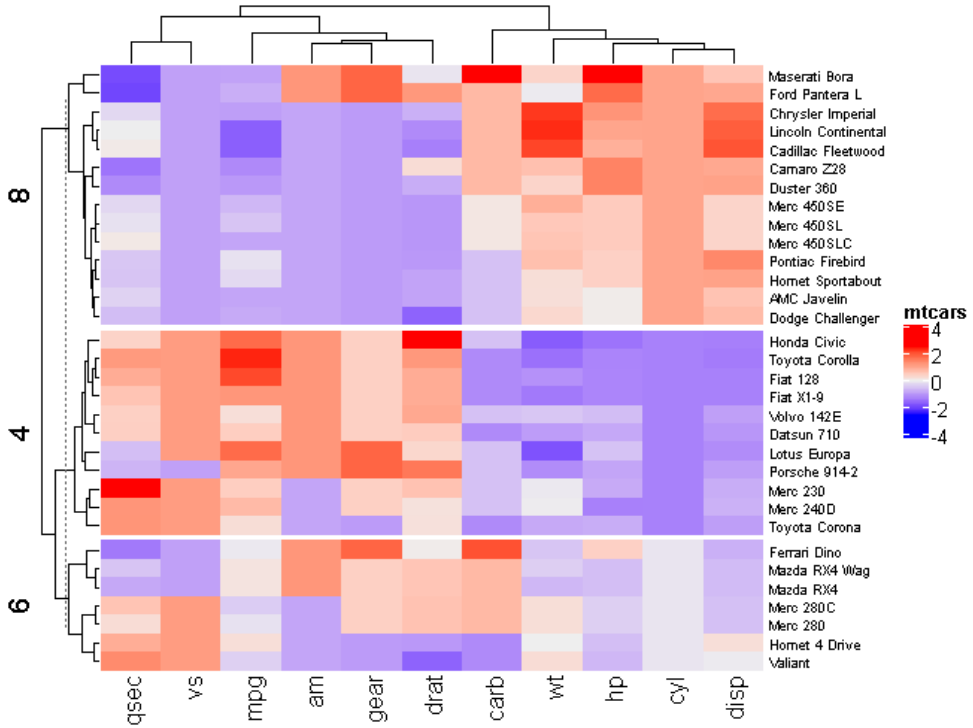

```
# plit the dendrogram using k-means
Heatmap(mtcars.df, name = "mtcars",
        row_km = 4,
        column_km = 3,
        row_names_gp = gpar(fontsize = 7))
```

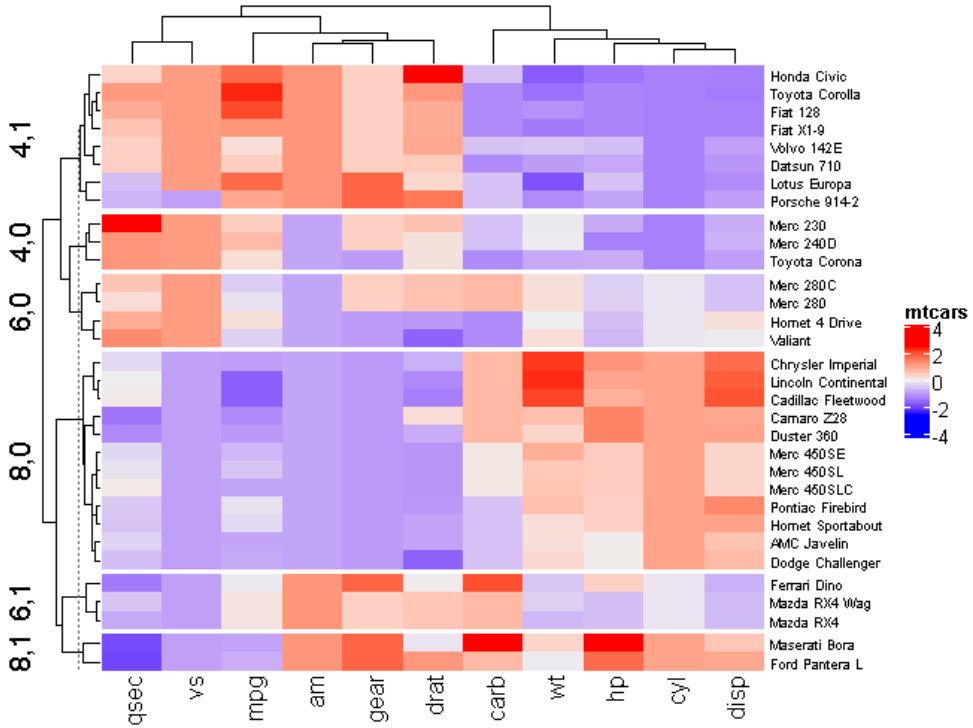


ComplexHeatmap: 依據類別變數分割熱圖

```
# split by a vector specifying rowgroups
Heatmap(mtcars.df, name = "mtcars",
        row_split = mtcars$cyl,
        row_names_gp = gpar(fontsize = 7))
```

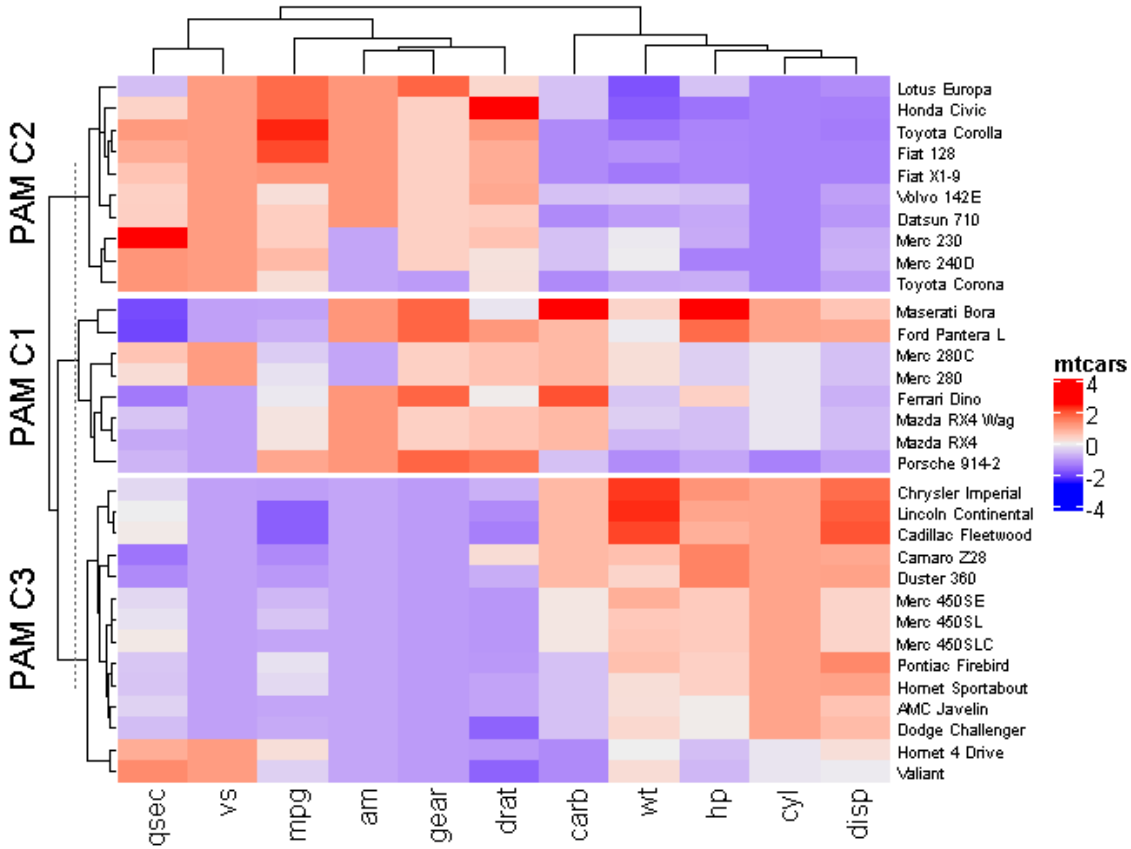
```
# Split by combining multiple variables
Heatmap(mtcars.df, name ="mtcars",
        row_split = data.frame(cyl = mtcars$cyl, am = mtcars$am),
        row_names_gp = gpar(fontsize = 7))
```





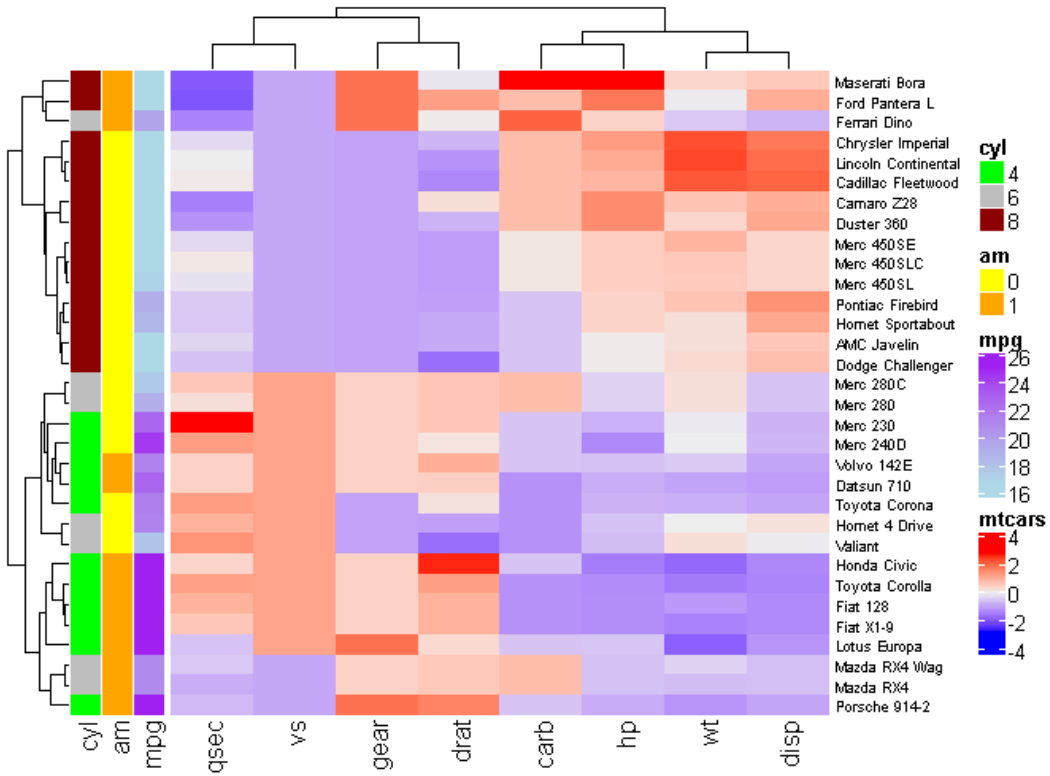
ComplexHeatmap: 依據群集分析法分割熱圖

```
# install.packages("cluster")
library("cluster")
set.seed(12345)
pa <- pam(mtcars.df, k = 3)
Heatmap(mtcars.df, name = "mtcars",
        row_split = paste0("PAM C", pa$clustering),
        row_names_gp = gpar(fontsize = 7))
```



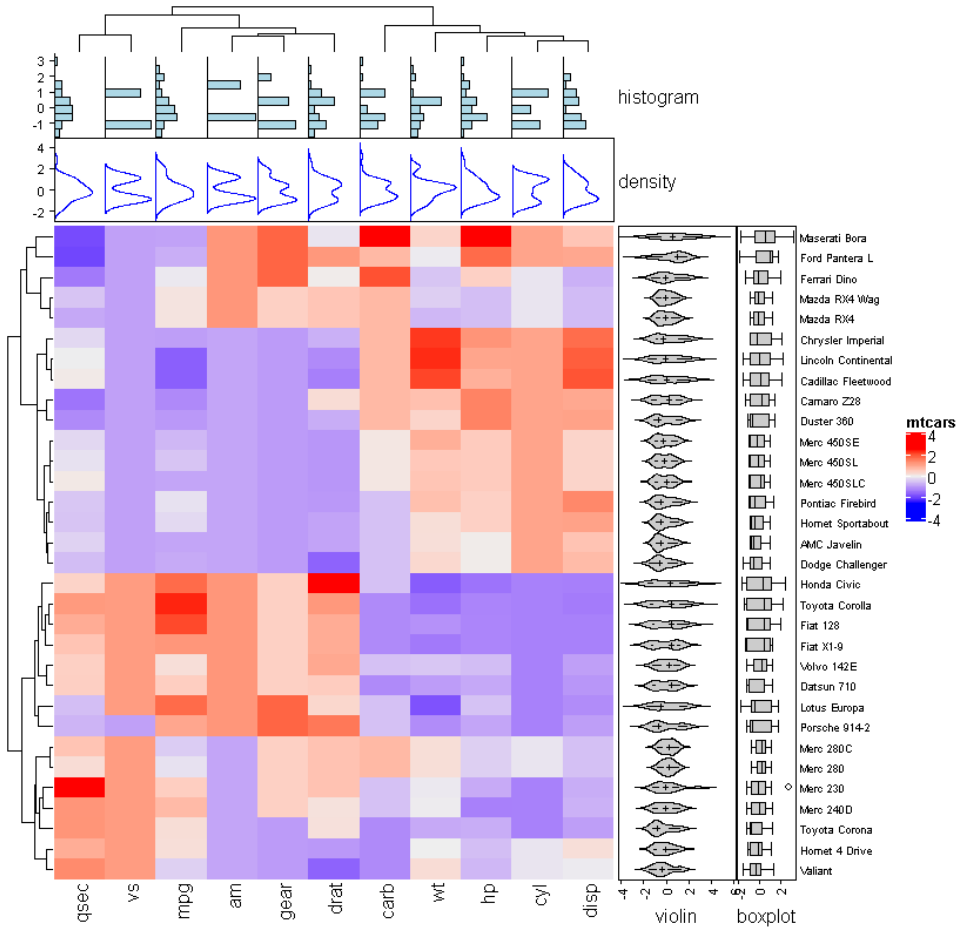
ComplexHeatmap: 以類別變數標記註解並指定顏色

```
# HeatmapAnnotation for "top_annotation", "bottom_annotation"
# rowAnnotation for "left_annotation", "right_annotation"
var.colors <- list(cyl = c("4" = "green", "6" = "gray", "8" = "darkred"),
                   am = c("0" = "yellow", "1" = "orange"),
                   mpg = colorRamp2(c(17, 25), c("lightblue", "purple")))

ha <- rowAnnotation(cyl = mtcars$cyl, am = mtcars$am, mpg = mtcars$mpg,
                    col = var.colors)

myvars <- colnames(mtcars.df) %in% c("cyl", "am", "mpg")
Heatmap(mtcars.df[, !myvars], name = "mtcars",
        left_annotation = ha,
        row_names_gp = gpar(fontsize = 7))
```



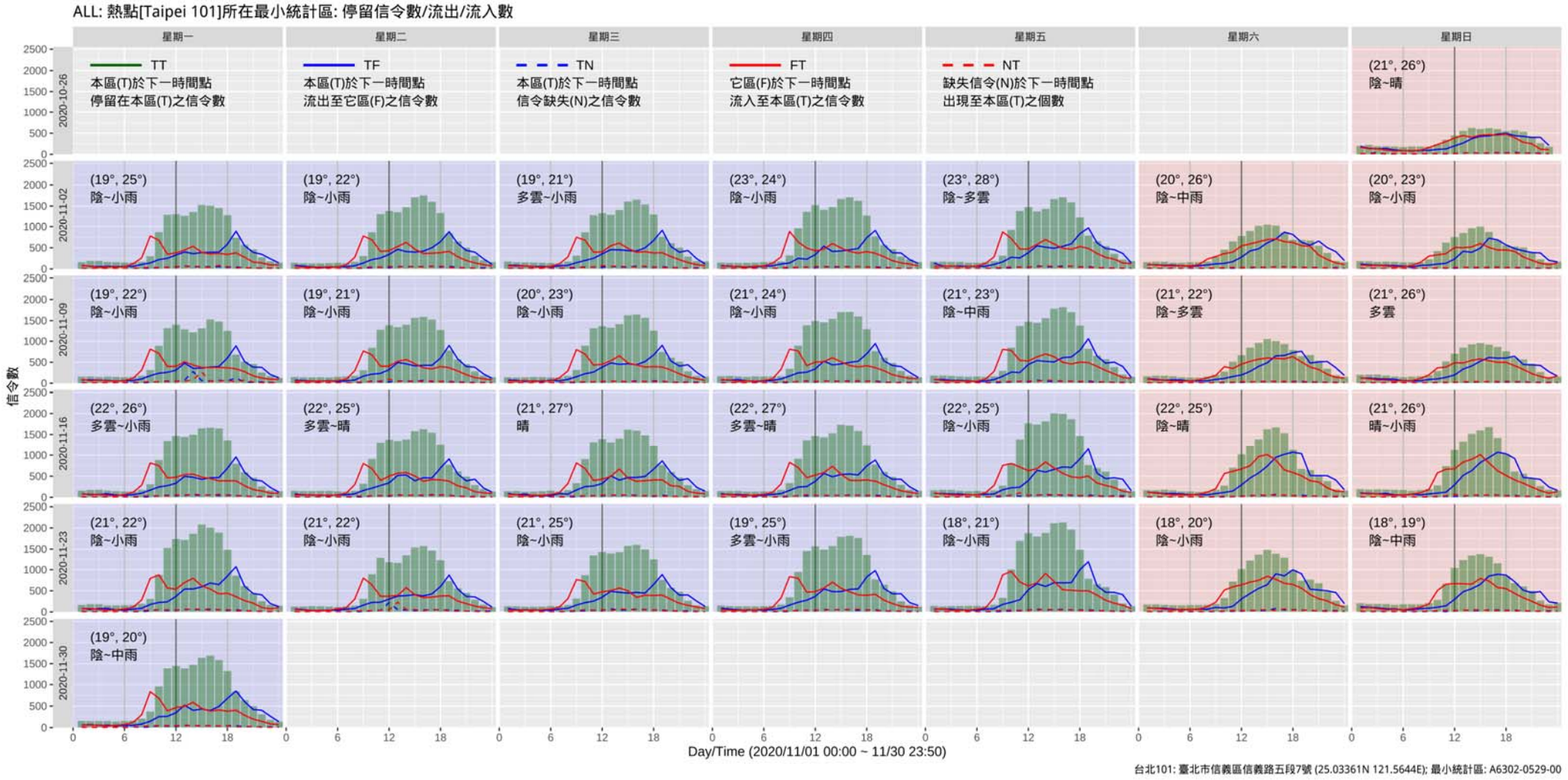
ComplexHeatmap: 標記統計圖為註解

```
# anno_points, anno_barplot, anno_boxplot, anno_density, anno_histogram
h <- anno_histogram(mtcars.df, gp = gpar(fill = "lightblue"))
d <- anno_density(mtcars.df, type = "line", gp = gpar(col = "blue"))
ha.top <- HeatmapAnnotation(histogram = h, density = d, height = unit(3.8, "cm"))

v <- anno_density(mtcars.df, type = "violin", which = "row")
b <- anno_boxplot(mtcars.df, which = "row")
ha.right <- HeatmapAnnotation(violin = v, boxplot = b, which = "row", width = unit(4, "cm"))

Heatmap(mtcars.df, name = "mtcars",
        top_annotation = ha.top,
        right_annotation = ha.right,
        row_names_gp = gpar(fontsize = 7))
```

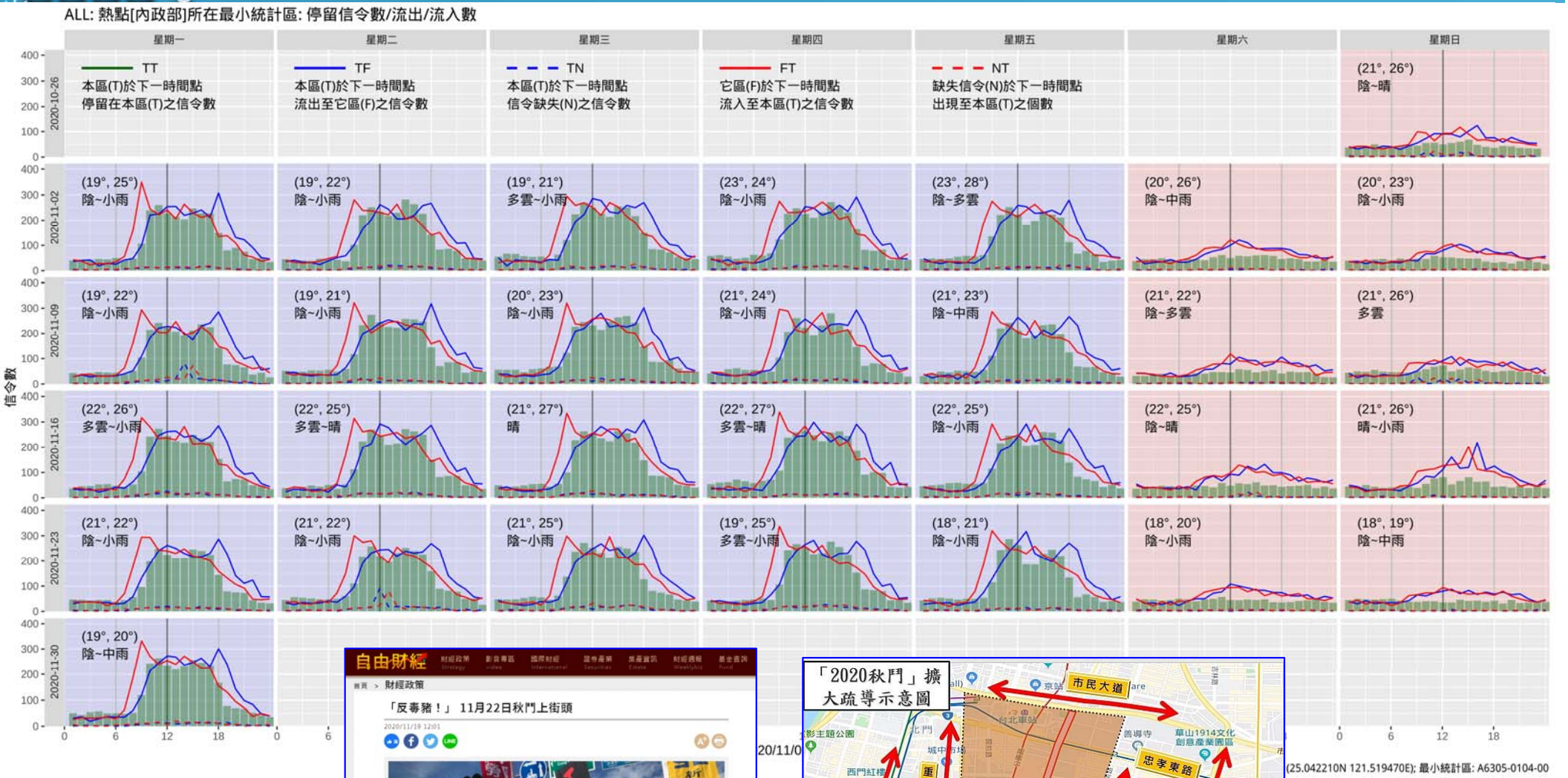


Translate from pheatmap to ComplexHeatmap
https://jokergoo.github.io/2020/05/06/translate-from-pheatmap-to-complexheatmap/

熱點分析: Taipei 101



熱點分析: 內政部

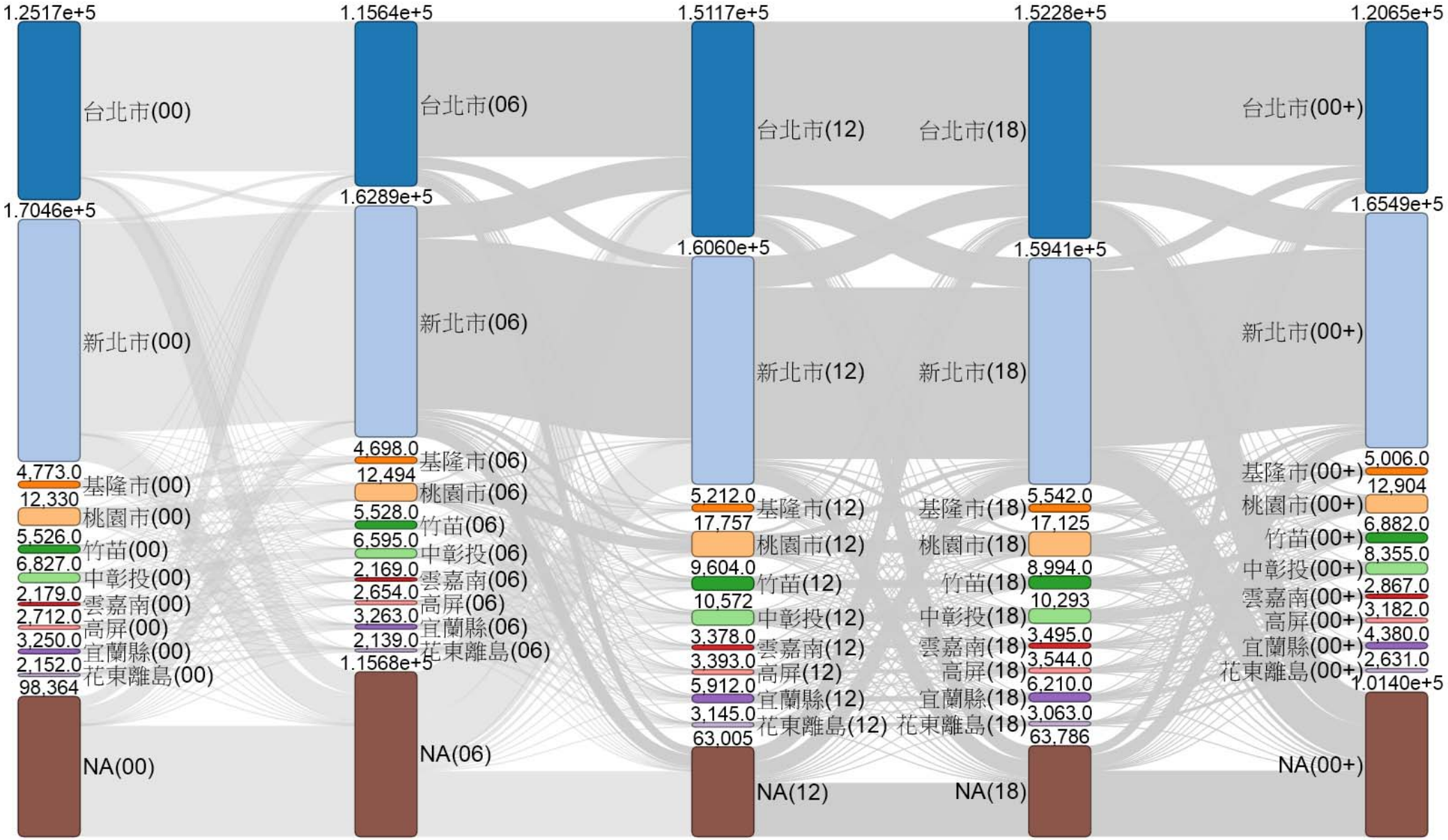

ALL: 熱點[內政部]所在最小統計區: 停留信令數/流出/流入數
星期一
星期二
星期三
星期四
星期五
星期六
星期日
TT
本區(T)於下一時間點
停留在本區(T)之信令數
TF
本區(T)於下一時間點
流出至它區(F)之信令數
TN
本區(T)於下一時間點
信令缺失(N)之信令數
FT
它區(F)於下一時間點
流入至本區(T)之信令數
NT
缺失信令(N)於下一時間點
出現至本區(T)之個數
2020-10-26
2020-11-02
2020-11-09
2020-11-16
2020-11-23
2020-11-30
信令數
(21°, 26°) 陰~晴
(19°, 25°) 陰~小雨
(19°, 22°) 陰~小雨
(19°, 21°) 多雲~小雨
(23°, 24°) 陰~小雨
(23°, 28°) 陰~多雲
(20°, 26°) 陰~中雨
(20°, 23°) 陰~小雨
(19°, 22°) 陰~小雨
(19°, 21°) 陰~小雨
(20°, 23°) 陰~小雨
(21°, 24°) 陰~小雨
(21°, 23°) 陰~中雨
(21°, 22°) 陰~多雲
(21°, 26°) 多雲
(22°, 26°) 多雲~小雨
(22°, 25°) 多雲~晴
(21°, 27°) 晴
(22°, 27°) 多雲~晴
(22°, 25°) 陰~小雨
(22°, 25°) 陰~晴
(21°, 26°) 晴~小雨
(21°, 22°) 陰~小雨
(21°, 22°) 陰~小雨
(21°, 25°) 陰~小雨
(19°, 25°) 多雲~小雨
(18°, 21°) 陰~小雨
(18°, 20°) 陰~小雨
(18°, 19°) 陰~中雨
(19°, 20°) 陰~中雨
0 6 12 18
20/11/0
(25.042210N 121.519470E); 最小統計區: A6305-0104-00


自由財經
財經政策
「反毒豬！」 11月22日秋鬥上街頭
2020年秋鬥大遊行
反毒豬!反雙標!反黨國!
2020秋鬥遊行將在11月22日上街頭，（記者李雅雯攝）


「2020秋鬥」擴大疏導示意圖
市民大道
忠孝東路
中華路
重慶南路
林森南路
金山南路
愛國東、西路
和平東、西路
影響範圍
擴大疏導範圍
改道動線
Google

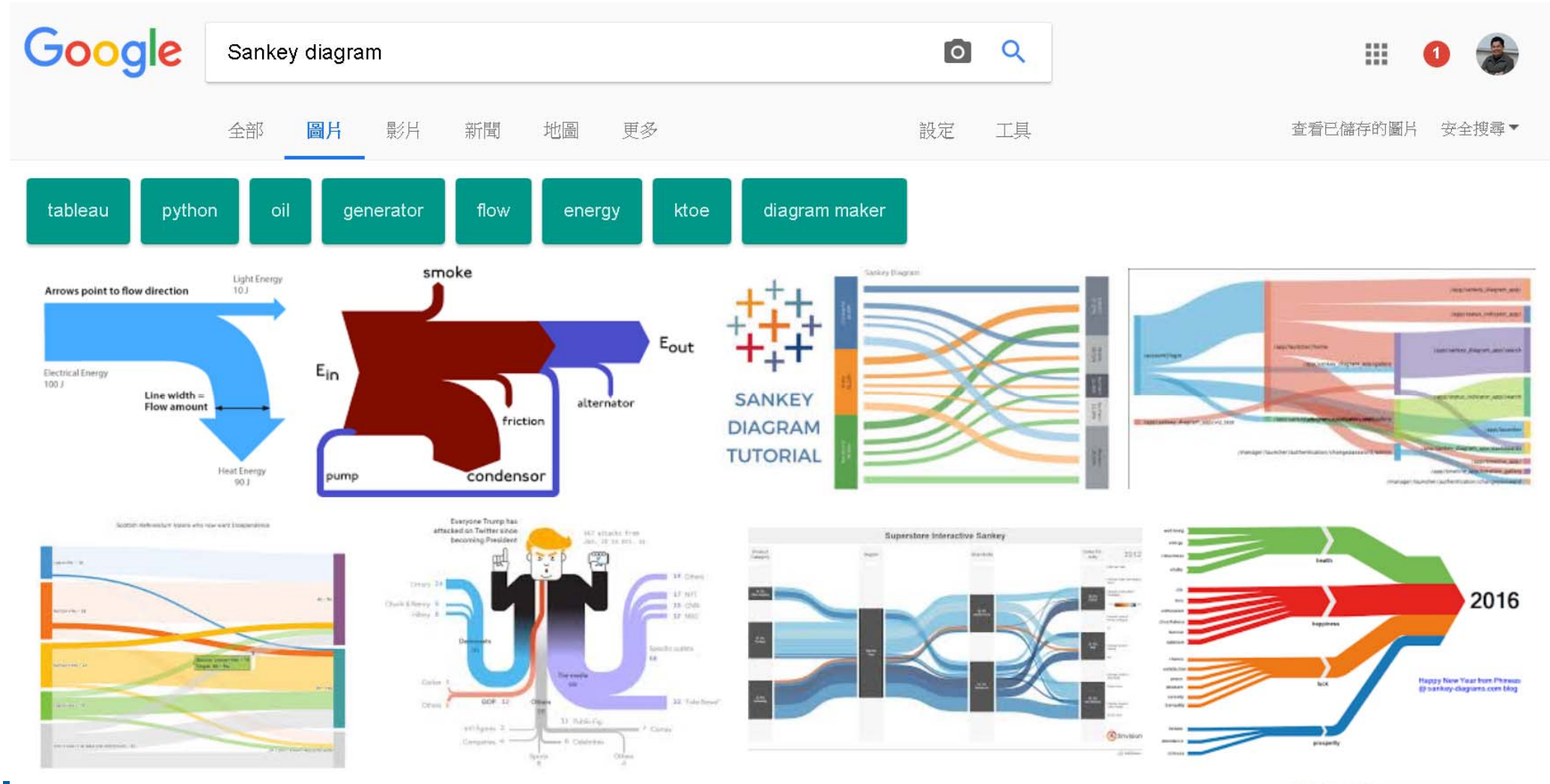
桑基圖 (Sankey Diagram)

zone-sankey-na-20180526(六)



Sankey Diagram

http://www.sankey-diagrams.com/



sankeyNetwork {networkD3}:
Create a D3 JavaScript Sankey diagram

Usage

```
sankeyNetwork(Links, Nodes, Source, Target, Value, NodeID, NodeGroup = NodeID,
  LinkGroup = NULL, units = "",
  colourScale = JS("d3.scaleOrdinal(d3.schemeCategory20);"), fontSize = 7,
  fontFamily = NULL, nodeWidth = 15, nodePadding = 10, margin = NULL,
  height = NULL, width = NULL, iterations = 32, sinksRight = TRUE)
```

Arguments

- **Links**: a data frame object with the links between the nodes. It should have include the Source and Target for each link. An optional Value variable can be included to specify how close the nodes are to one another.
- **Nodes**: a data frame containing the node id and properties of the nodes. If no ID is specified then the nodes must be in the same order as the Source variable column in the Links data frame. Currently only grouping variable is allowed.
- **Source**: character string naming the network source variable in the Links data frame.
- **Target**: character string naming the network target variable in the Links data frame.
- **Value**: character string naming the variable in the Links data frame for how far away the nodes are from one another.
- **NodeID**: character string specifying the node IDs in the Nodes. data frame. Must be 0-indexed.
- **NodeGroup**: character string specifying the node groups in the Nodes. Used to color the nodes in the network.

sankeyNetwork {networkD3}:
Create a D3 JavaScript Sankey diagram

```
> URL <- paste0(
+         "https://cdn.rawgit.com/christophergandrud/networkD3/",
+         "master/JSONdata/energy.json")
> Energy <- jsonlite::fromJSON(URL)
> str(Energy)
List of 2
 $ nodes:'data.frame':  48 obs. of  1 variable:
  ..$ name: chr [1:48] "Agricultural 'waste'" "Bio-conversion" "Liquid" "Losses" ...
 $ links:'data.frame':  68 obs. of  3 variables:
  ..$ source: int [1:68] 0 1 1 1 1 6 7 8 10 9 ...
  ..$ target: int [1:68] 1 2 3 4 5 2 4 9 9 4 ...
  ..$ value : num [1:68] 124.729 0.597 26.862 280.322 81.144 ...
> lapply(Energy, head)
$nodes
                  name
1 Agricultural 'waste'
2       Bio-conversion
3               Liquid
4               Losses
5                Solid
6                  Gas

$links
  source target   value
1      0      1 124.729
2      1      2   0.597
3      1      3  26.862
4      1      4 280.322
5      1      5  81.144
6      6      2  35.000
```

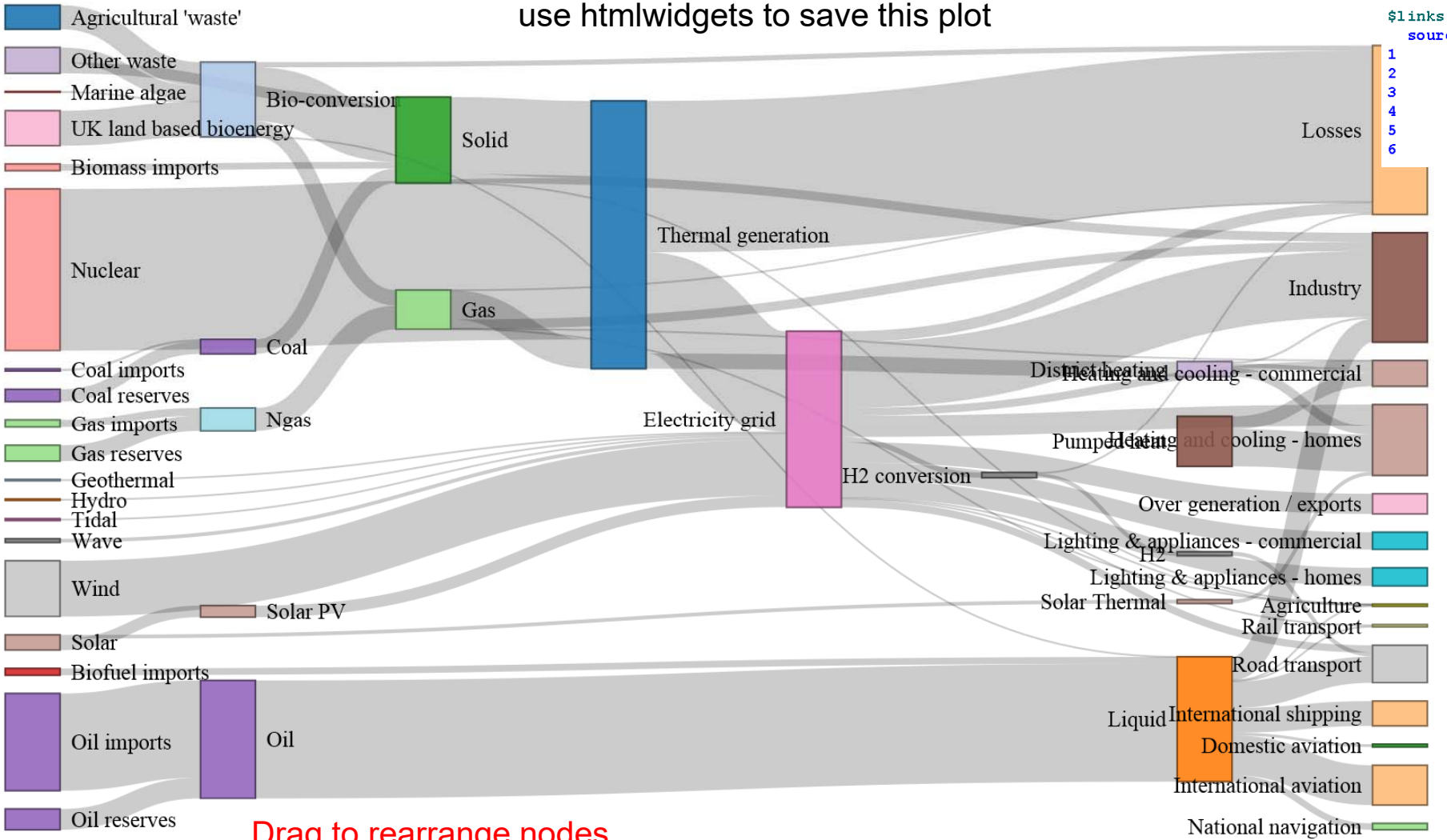
```
> sankeyNetwork(Links = Energy$links, Nodes = Energy$nodes,
+               Source = "source", Target = "target",
+               Value = "value", NodeID = "name",
+               fontSize = 12, nodeWidth = 30)
```

sankeyNetwork

```
> sankeyNetwork(Links = Energy$links, Nodes = Energy$nodes,
+               Source = "source", Target = "target",
+               Value = "value", NodeID = "name",
+               fontSize = 12, nodeWidth = 30)
```

```
> lapply(Energy, head)
$nodes
                    name
1 Agricultural 'waste'
2       Bio-conversion
3               Liquid
4               Losses
5                Solid
6                  Gas

$links
  source target    value
1      0      1 124.729
2      1      2   0.597
3      1      3  26.862
4      1      4 280.322
5      1      5  81.144
6      6      2  35.000
```

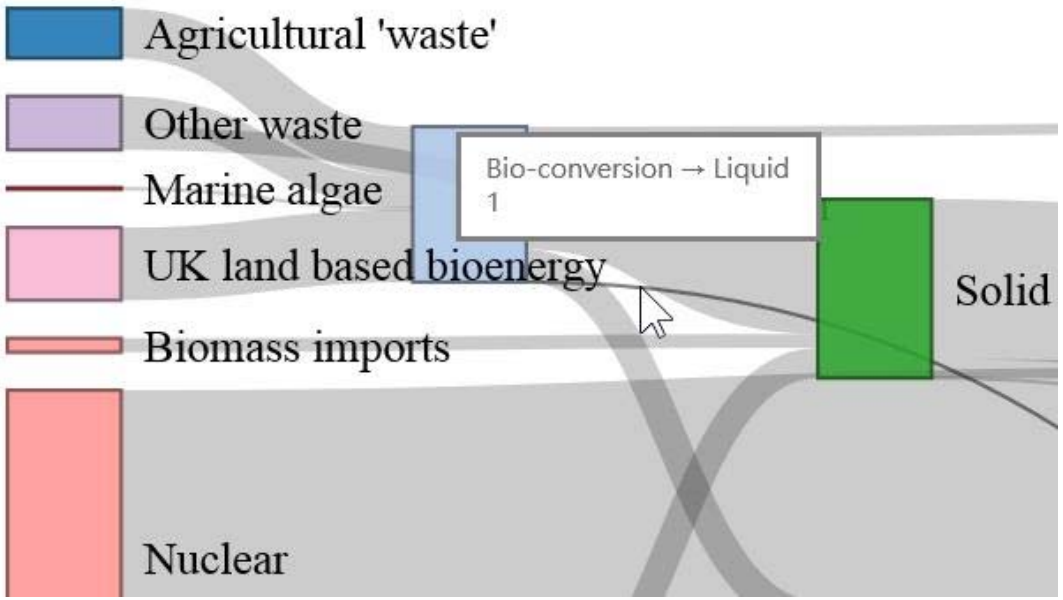
use htmlwidgets to save this plot



Drag to rearrange nodes.

sankeyNetwork

```
> Energy
$nodes
                                    name
1                  Agricultural 'waste'
2                        Bio-conversion
3                                Liquid
4                                Losses
5                                 Solid
6                                   Gas
7                        Biofuel imports
8                        Biomass imports
...
46               UK land based bioenergy
47                                  Wave
48                                  Wind

$links
   source target   value
1       0      1 124.729
2       1      2   0.597
3       1      3  26.862
4       1      4 280.322
5       1      5  81.144
6       6      2  35.000
7       7      4  35.000
...
67     46     15  19.013
68     47     15 289.366
```



Sankey diagrams are closely related to alluvial diagrams, which show how network structure changes over time.
https://en.wikipedia.org/wiki/Alluvial_diagram

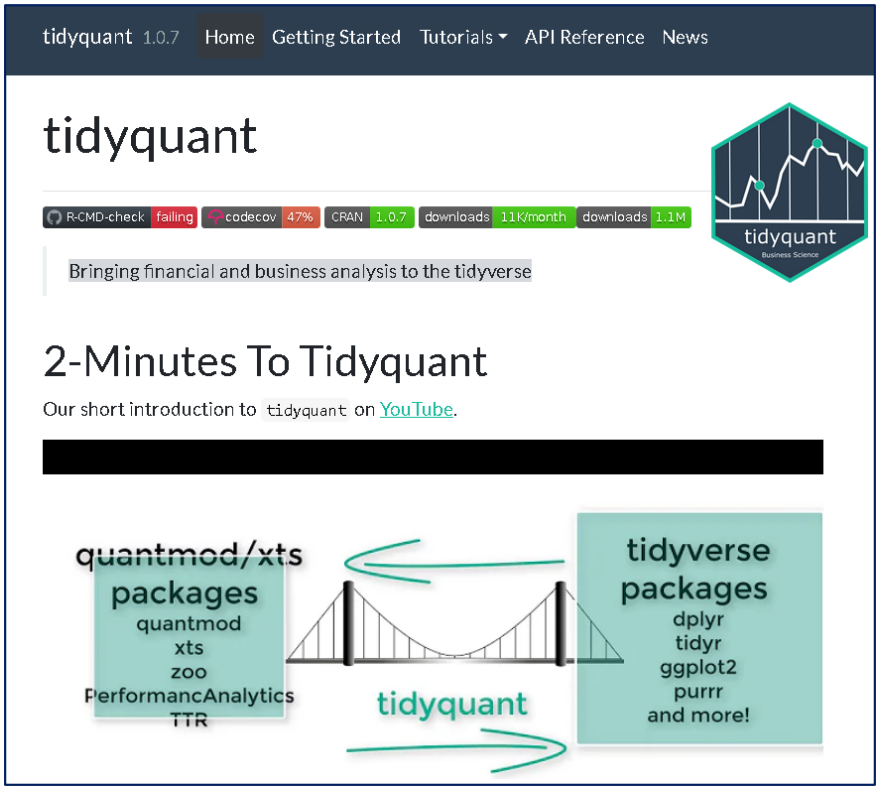
quantmod: Quantitative Financial Modeling Framework

tidyquant: Bringing financial and business analysis to the tidyverse

quantmod: Specify, build, trade, and analyse quantitative financial trading strategies.

http://www.quantmod.com/

https://business-science.github.io/tidyquant/



- Core Functions:
  - Getting Financial Data from the web: **tq_get()**
  - Manipulating Financial Data: **tq_transmute()** and **tq_mutate()**
  - Performance Analysis and Portfolio Analysis: **tq_performance()** and **tq_portfolio()**
- Comparing Stock Prices, Evaluating Stock Performance, Evaluating Portfolio Performance
- https://cran.r-project.org/web/packages/tidyquant/vignettes/TQ03-scaling-and-modeling-with-tidyquant.html

Create Your Own Visualization Methods: Path point approach

Computational Statistics (2024) 39:1937–1969
https://doi.org/10.1007/s00180-023-01440-7

ORIGINAL PAPER

Dimension reduction and visualization of multiple time series data: a symbolic data analysis approach

Emily Chia-Yu Su[1] · Han-Ming Wu[2]

Received: 2 December 2022 / Accepted: 9 November 2023 / Published online: 6 December 2023

Abstract
Exploratory analysis and visualization of multiple time series data are essential for discovering the underlying dynamics of a series before attempting modeling and forecasting. This study extends two dimension reduction methods - principal component analysis (PCA) and sliced inverse regression (SIR) - to multiple time series data. This is achieved through the innovative path point approach, a new addition to the symbolic data analysis framework. By transforming multiple time series data into time-dependent intervals marked by starting and ending values, each series is geometrically represented as successive directed segments with unique path points. These path points serve as the foundation of our novel representation approach. PCA and SIR are then applied to the data table formed by the coordinates of these path points, enabling visualization of temporal trajectories of objects within a reduced-dimensional subspace. Empirical studies encompassing simulations, microarray time series data from a yeast cell cycle, and financial data confirm the effectiveness of our path point approach in revealing the structure and behavior of objects within a 2D factorial plane. Comparative analyses with existing methods, such as the applied vector approach for PCA and SIR on time-dependent interval data, further underscore the strength and versatility of our path point representation in the realm of time series data.

Keywords Exploratory data analysis · Data visualization · PCA · Sliced inverse regression · Symbolic data analysis · Time dependent interval-valued data

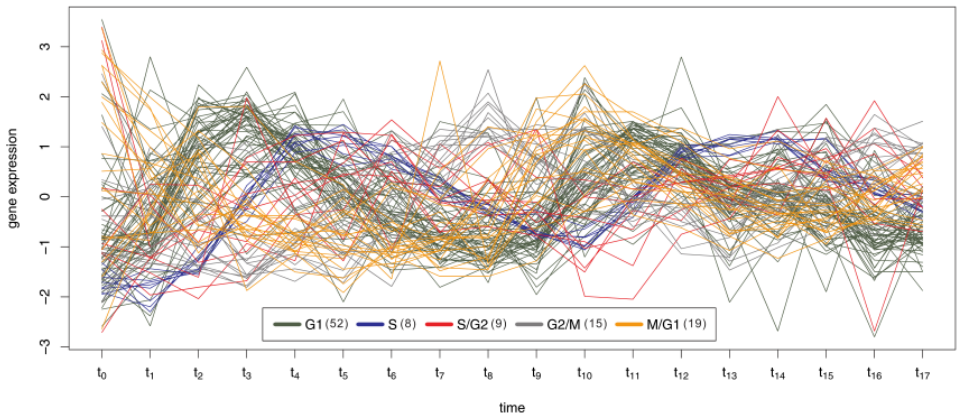
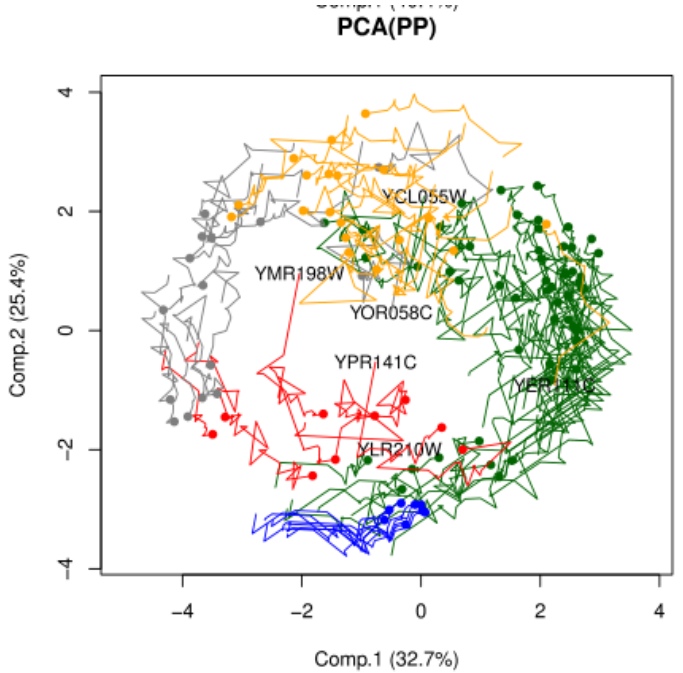
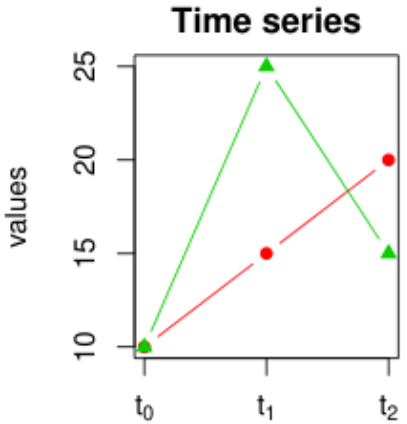


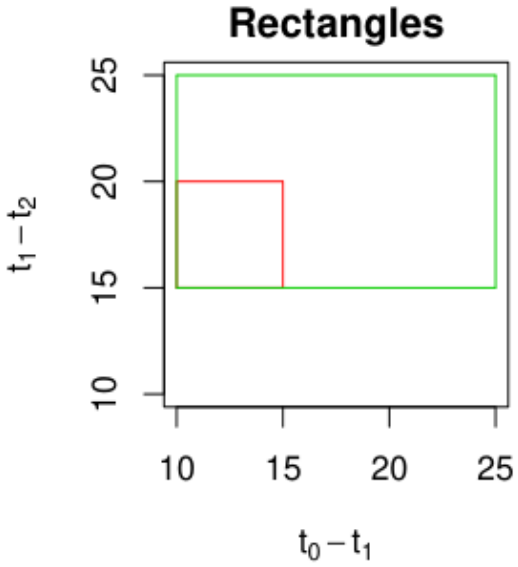
Fig. 7 The time series microarray data consists of 103 cell cycle-regulated yeast genes. Each expression profile is normalized to have a mean equal to zero and a variance of one

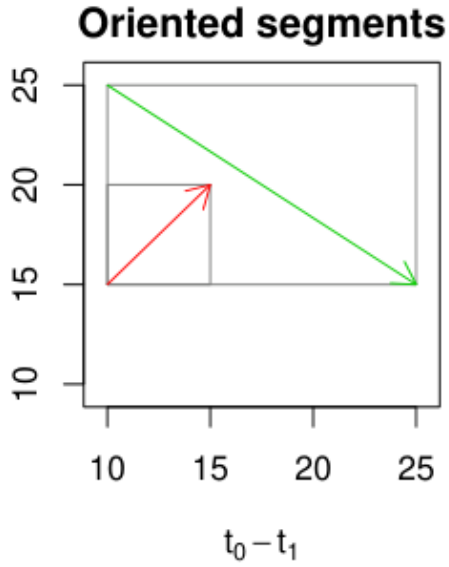


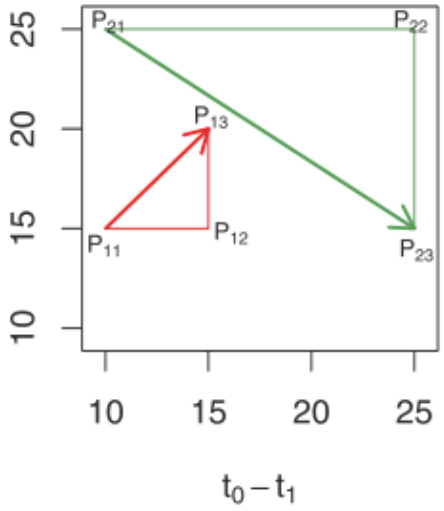
Representation of Time series: 2D (Irpino, 2006)



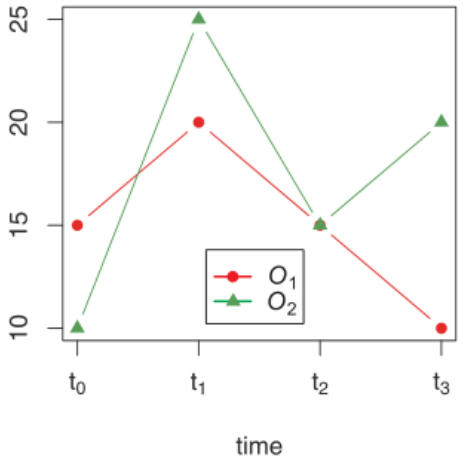
(a) Time series				(b) Intervals		(c) Time dep. intervals	
				period	period	period	period
				t_0 - t_1	t_1 - t_2	t_0 - t_1	t_1 - t_2
	t_0	t_1	t_2	t_0 - t_1	t_1 - t_2	start - end	start - end
O_1	10	15	20	[10, 15]	[15, 20]	[10; 15]	[15; 20]
O_2	10	25	15	[10, 25]	[15, 25]	[10; 25]	[25; 15]







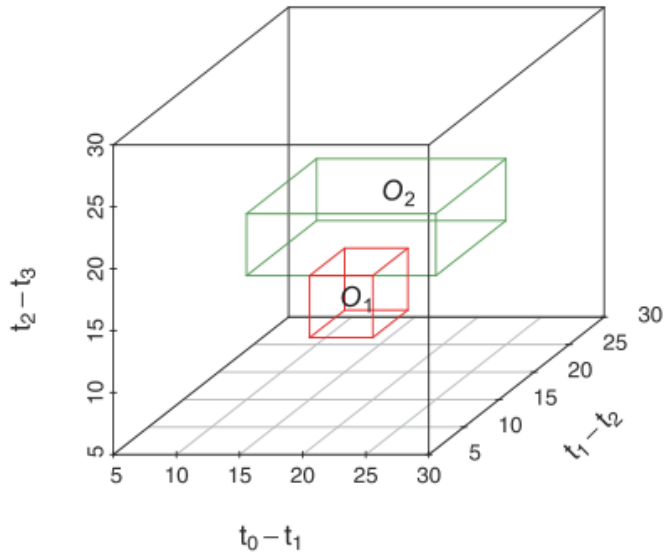
4-TIME POINTS



(a) Time series

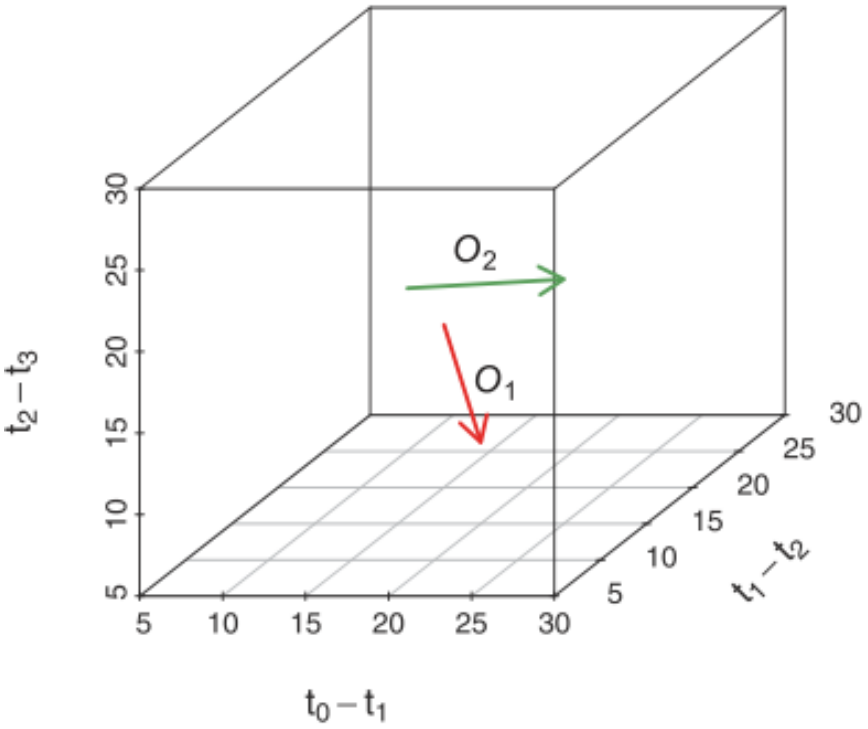
	t_0	t_1	t_2	t_3
O_1	15	20	15	10
O_2	10	25	15	20

BOXES



(b) Intervals

Variables	X_1	X_2	X_3
Period	t_0 - t_1	t_1 - t_2	t_2 - t_3
Objects	start - end	start - end	start - end
O_1	[15, 20]	[15, 20]	[10, 15]
O_2	[10, 25]	[15, 25]	[15, 20]



Directed segments

(c) Time-dependent intervals

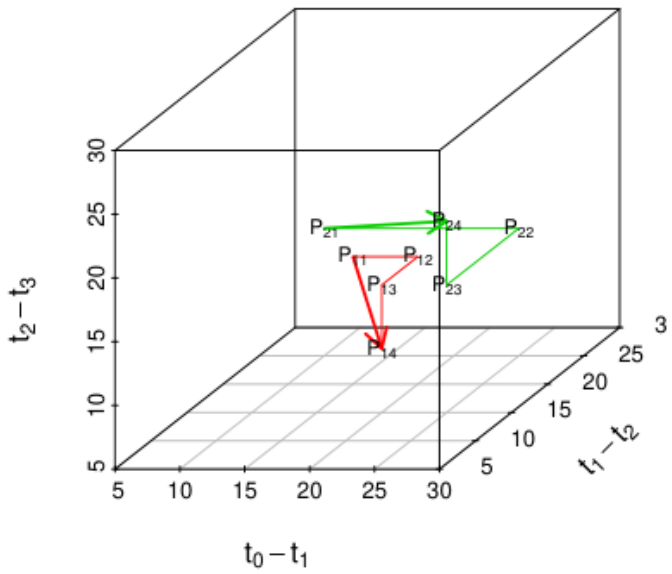
Variables	X_1	X_2	X_3
Period	t_0 - t_1	t_1 - t_2	t_2 - t_3
Objects	start - end	start - end	start - end
O_1	[15; 20]	[20; 15]	[15; 10]
O_2	[10; 25]	[25; 15]	[15; 20]

1. The position is due to the values assumed in the different times.
2. The length is related to the period variability.
3. The direction is related to the shape of a time series.

(a) Time series

	t_0	t_1	t_2	t_3
O_1	15	20	15	10
O_2	10	25	15	20

Representation of Time series: path points



(d) Time period variables with path points

Obj.	Variables / Period / Path points	X_1 t_0 - t_1 start - end	X_2 t_1 - t_2 start - end	X_3 t_2 - t_3 start - end
O_1	P_{11}	15	20	15
	P_{12}	20	20	15
	P_{13}	20	15	15
	P_{14}	20	15	10
O_2	P_{21}	10	25	15
	P_{22}	25	25	15
	P_{23}	25	15	15
	P_{24}	25	15	20

(a) Time series

	t_0	t_1	t_2	t_3
O_1	15	20	15	10
O_2	10	25	15	20

(b) Intervals

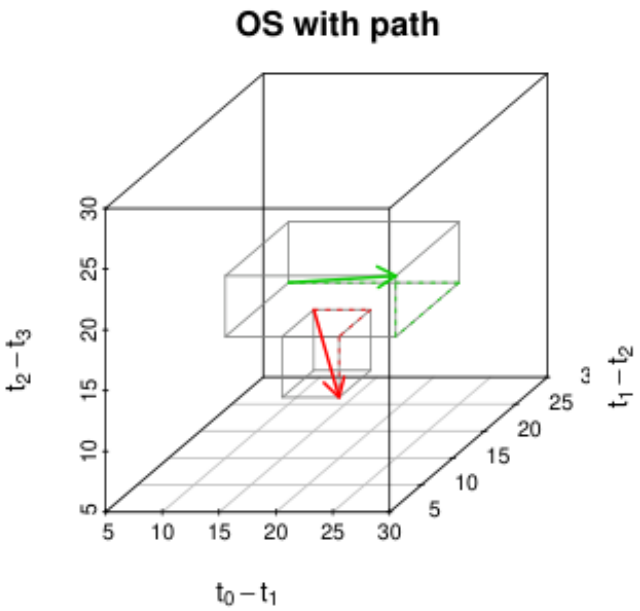
Variables / Period / Objects	X_1 t_0 - t_1 start - end	X_2 t_1 - t_2 start - end	X_3 t_2 - t_3 start - end
O_1	[15, 20]	[15, 20]	[10, 15]
O_2	[10, 25]	[15, 25]	[15, 20]

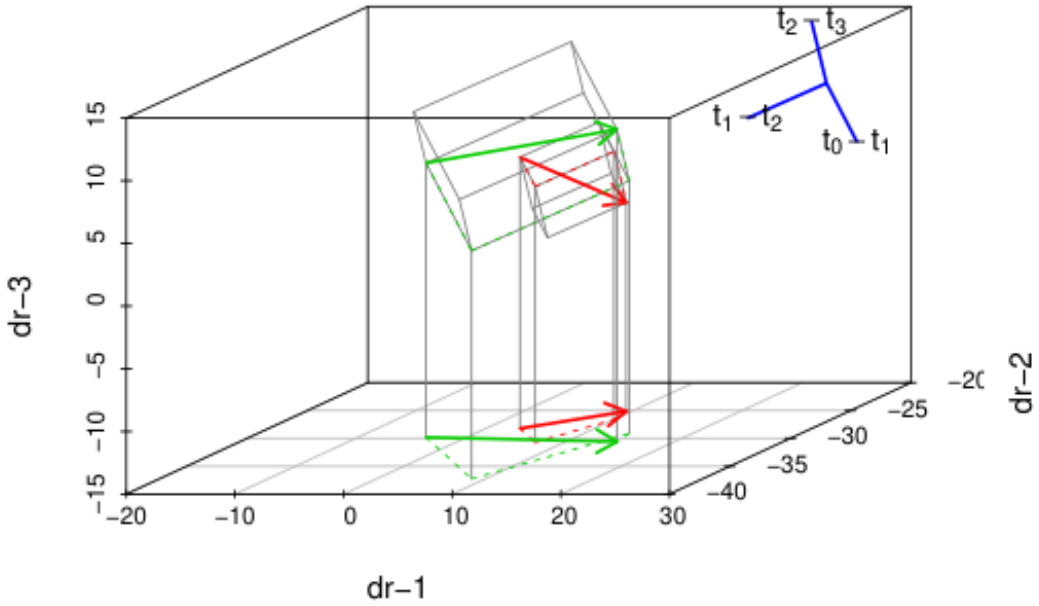
(c) Time-dependent intervals

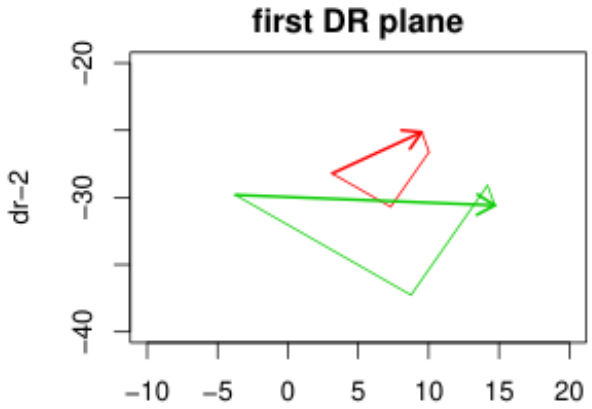
Variables / Period / Objects	X_1 t_0 - t_1 start - end	X_2 t_1 - t_2 start - end	X_3 t_2 - t_3 start - end
O_1	[15; 20]	[20; 15]	[15; 10]
O_2	[10; 25]	[25; 15]	[15; 20]

Projections and Visualization in DR Space

PP approach

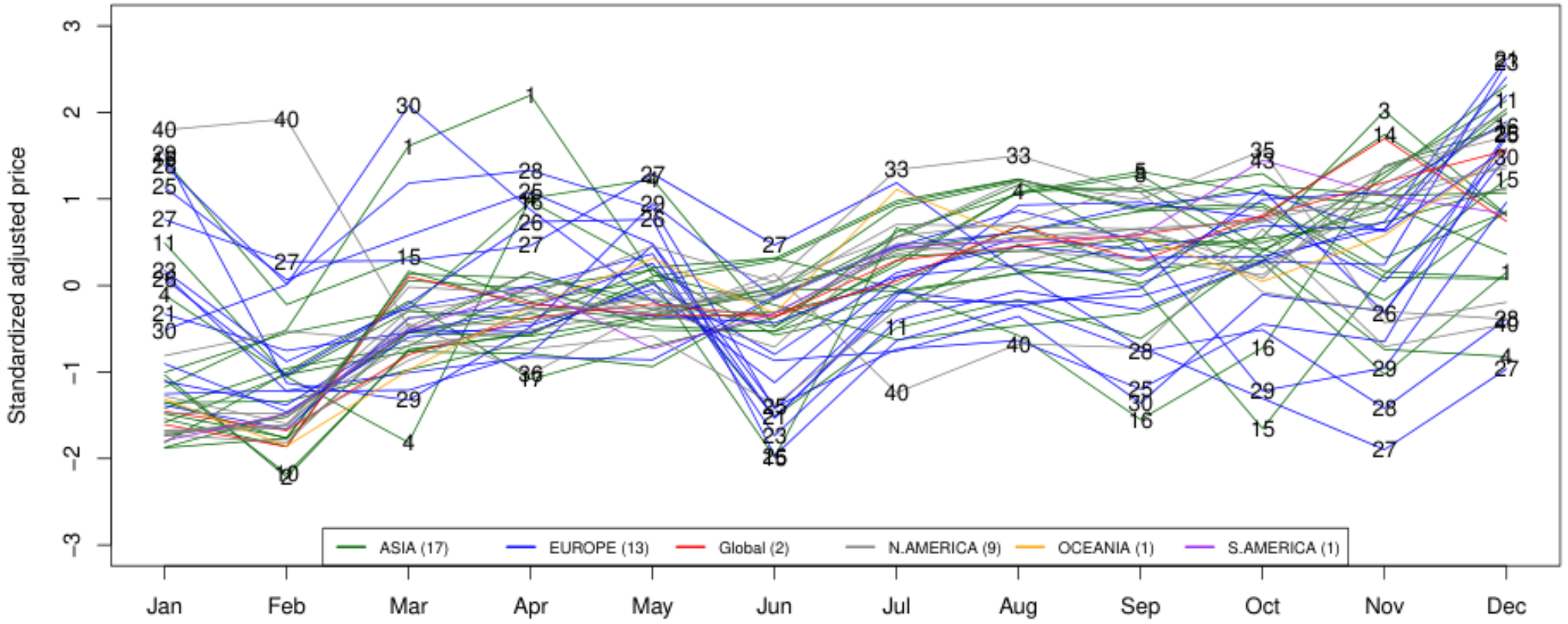






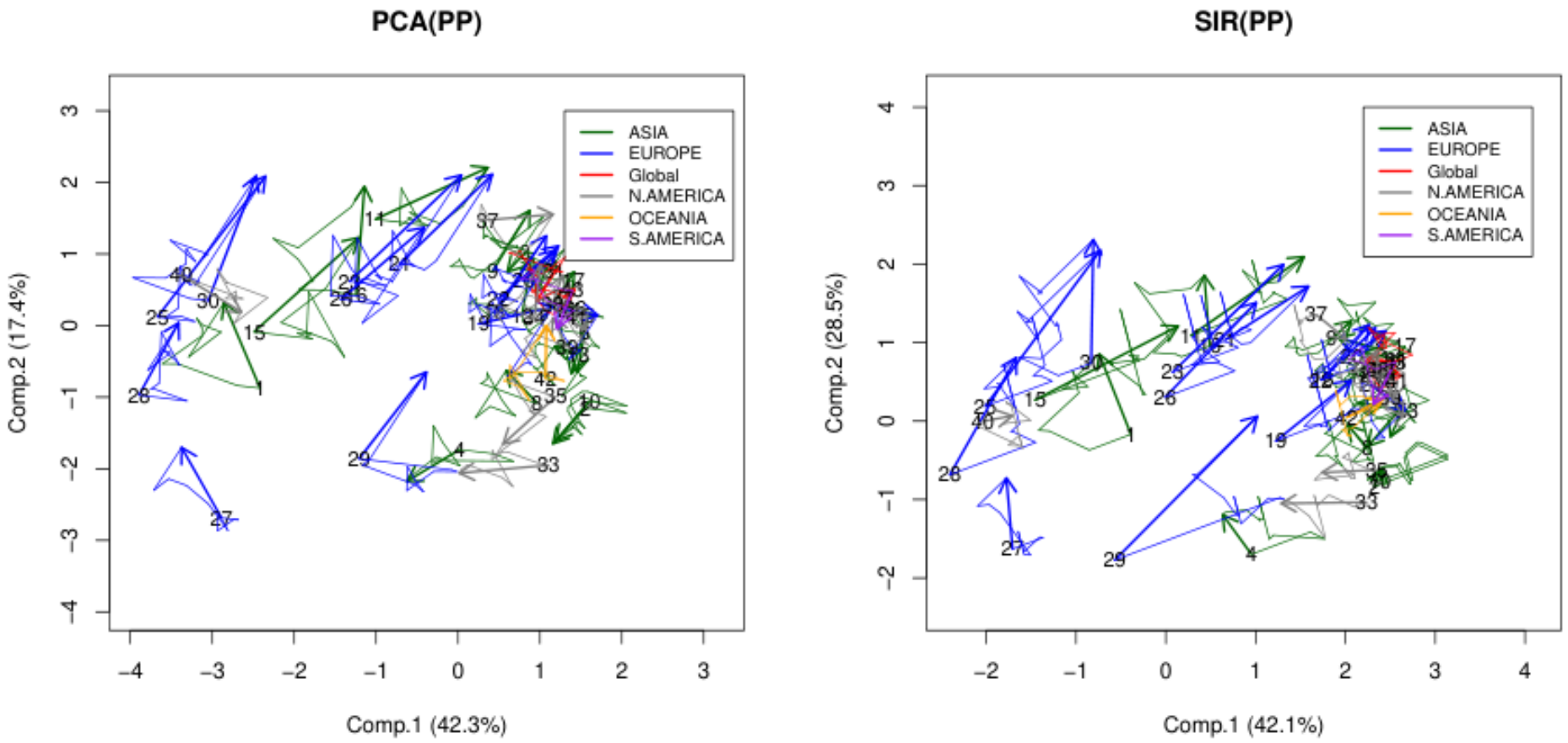
Major world indices from Investing.com.

Market	n_i	No.: name of indices
ASIA	17	1: BIST 100, 2: BSE Sensex, 3: China A50, 4: CSE All-Share, 5: Hang Seng, 6: IDX Composite, 7: Karachi 100, 8: KOSPI, 9: MICEX, 10: Nifty 50, 11: Nikkei 225, 12: RTSI, 13: SET, 14: Shanghai, 15: TA35, 16: Tadawul All Share, 17: Taiwan Weighted.
EUROPE	13	18: AEX, 19: BEL 20, 20: Budapest SE, 21: CAC 40, 22: DAX, 23: Euro Stoxx 50, 24: FTSE 100, 25: FTSE MIB, 26: IBEX 35, 27: OMXC20, 28: PSI20, 29: SMI, 30: WIG20.
Global	2	31: DJ Shanghai, 32: TR Canada 50.
NORTH AMERICA	9	33: DJ New Zealand, 34: Dow 30, 35: IPC, 36: Nasdaq, 37: OMXS30, 38: SmallCap 2000, 39: S&P500, 40: S&P500 VIX, 41: S&P/TSX.
OCEANIA	1	42: S&P/ASX 200.
SOUTH AMERICA	1	43: Bovespa.



The normalized monthly adjusted prices of the 43 major world indices from `Investing.com` of the year 2016.

The financial data of the major world indices



The first two PCA/SIR projections of the path point (PP) approaches applied to the financial data of the 43 major world indices.