

R語言 時間序列資料分析

吳漢銘

國立政治大學 統計學系



<http://www.hmwu.idv.tw>



本章大綱&學習目標

2/61

- 學習資源: 參考書目
- R軟體中的時間序列物件
- 時間序列資料輸入與輸出
- 時間序列資料的處理: `lubridate` 套件
- 時間序列圖
- 時間序列資料的平滑化
- 時間序列資料統計模型與預測:
 - Multiple Regression
 - AR, MA, ARMA, ARIMA
- 多變量自迴歸模型: The Stationary Vector Autoregression Model, VAR(p)
- 區間型資料之時間序列分析 (Interval Time Series Analysis)
- 類別(屬質)變數之時間序列分析 (Categorical Time Series Models)
- 財務金融資料分析套件: `quantmod` & `tidyquant`



CRAN Task View: Time Series Analysis

3/61

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.

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

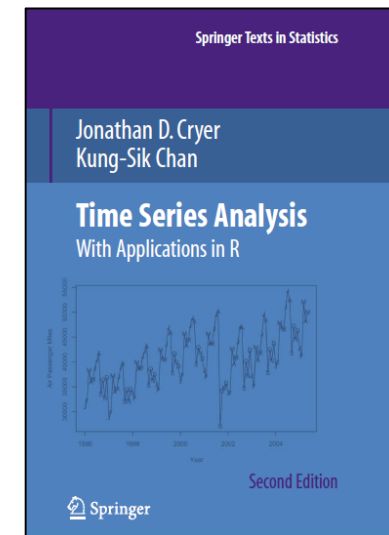
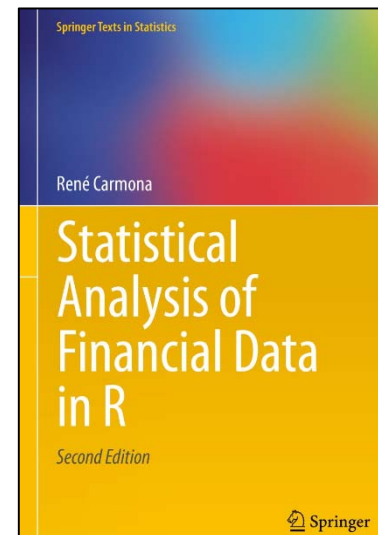
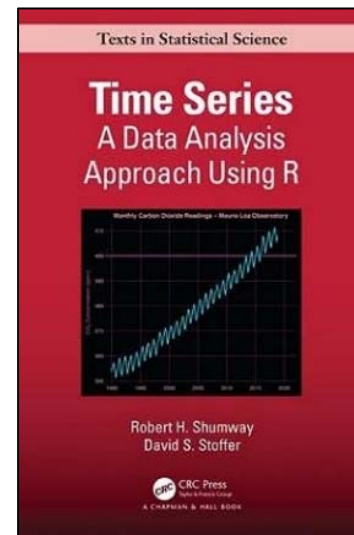
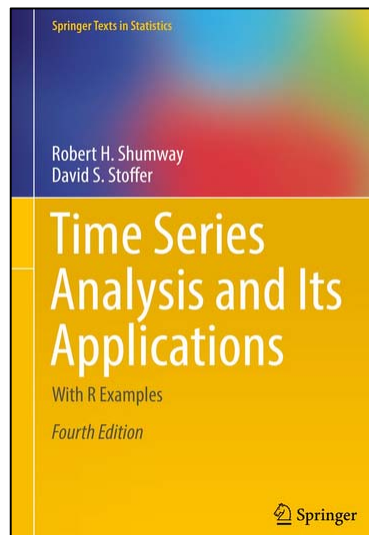
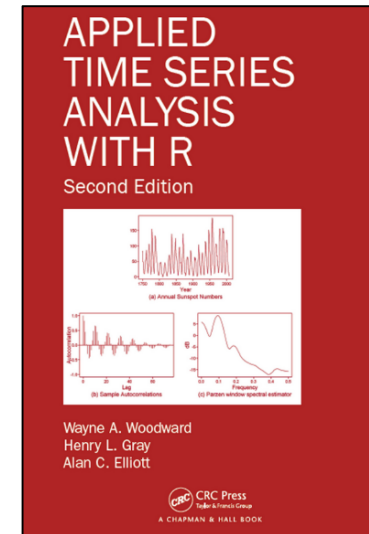
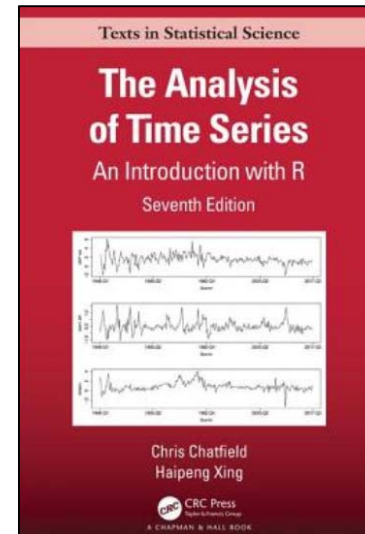
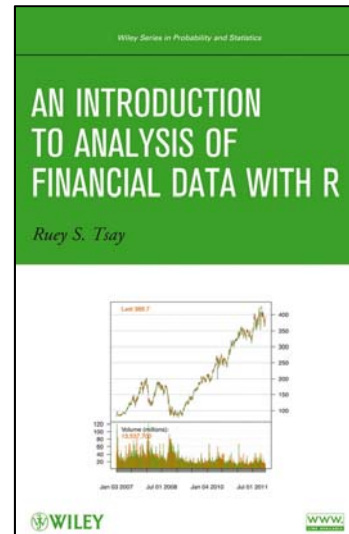
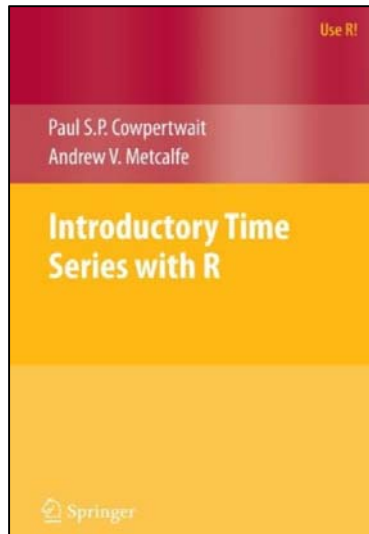
- *Infrastructure* : Base R contains substantial infrastructure for representing and analysing time series data. The fundamental class is "ts" that can represent regularly spaced time series (using numeric time stamps). Hence, it is particularly well-suited for annual,

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

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

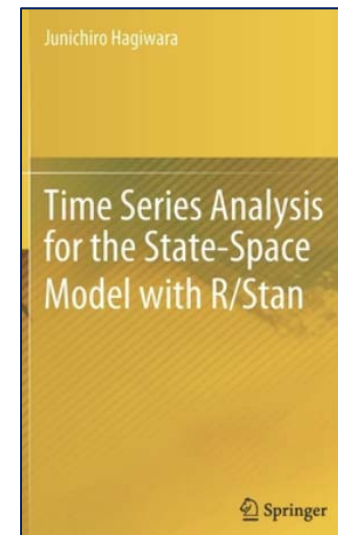
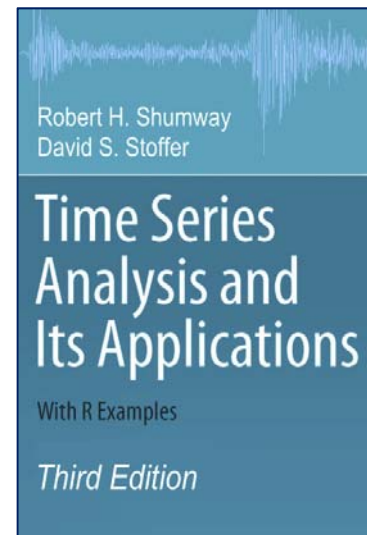
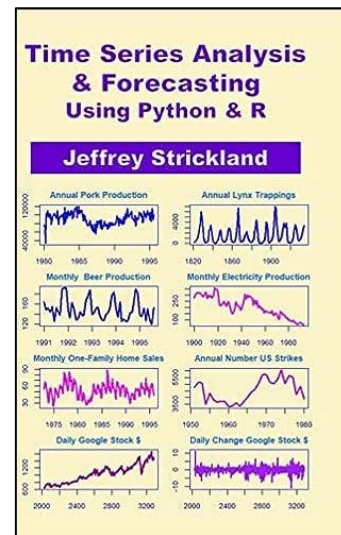
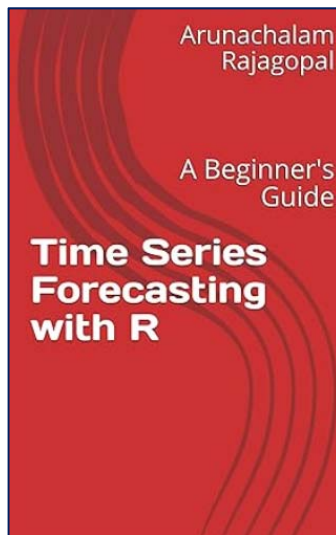
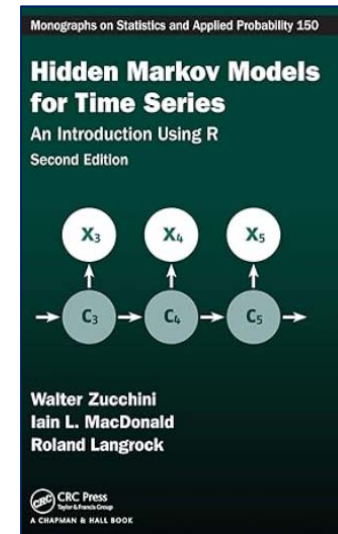
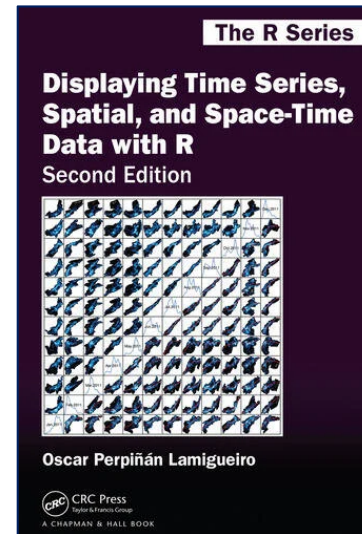
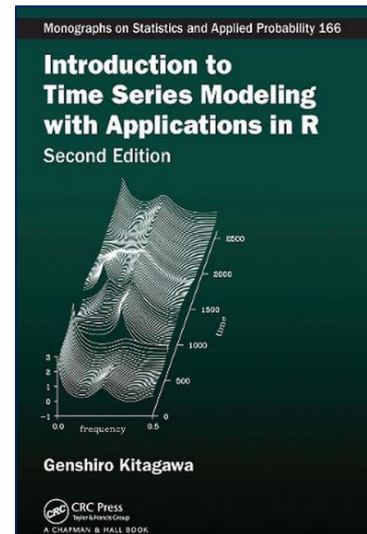
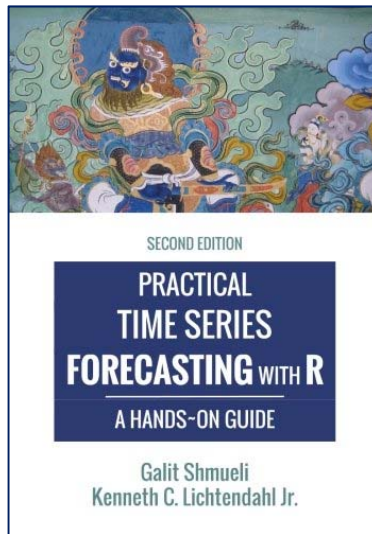
Time Series Analysis Using R

4/61



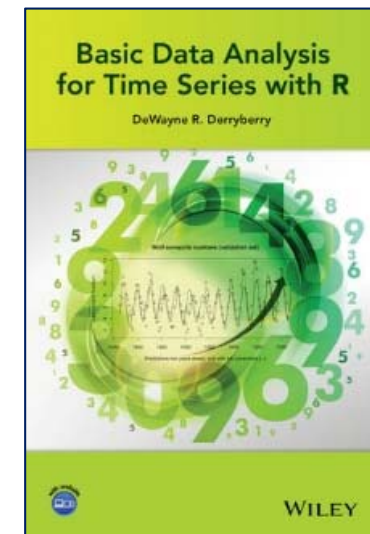
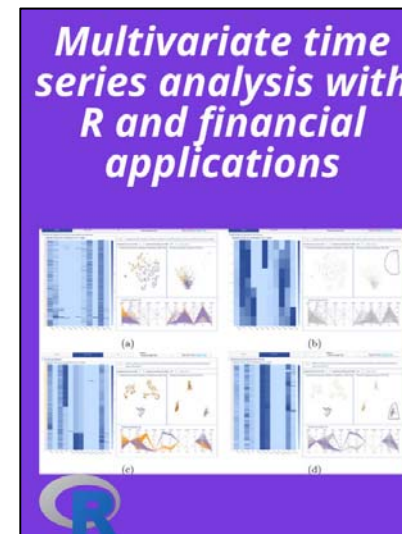
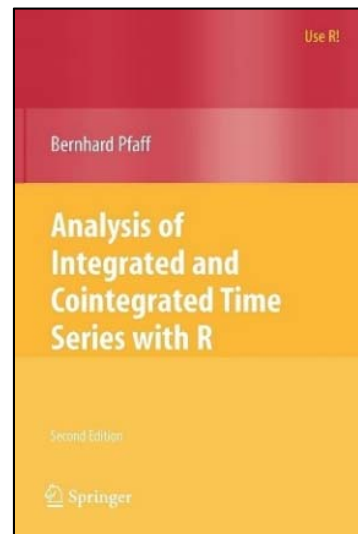
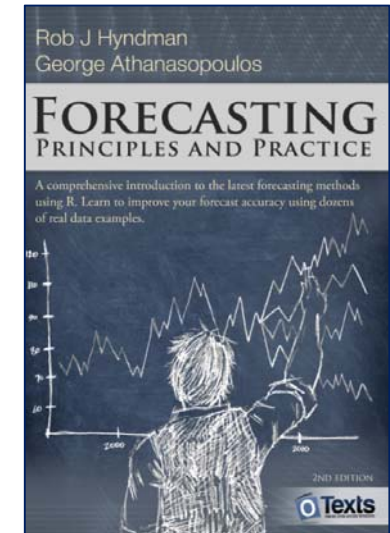
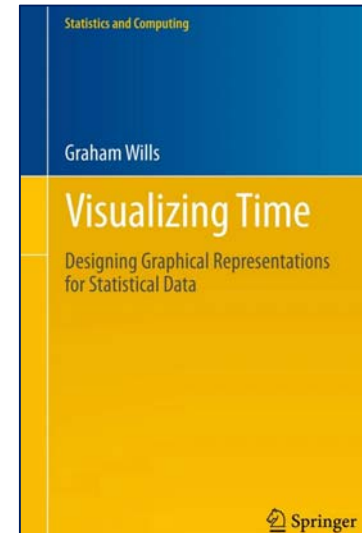
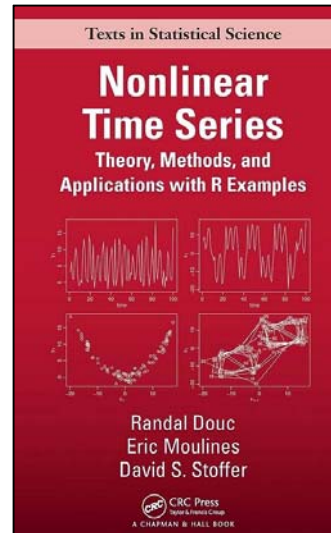
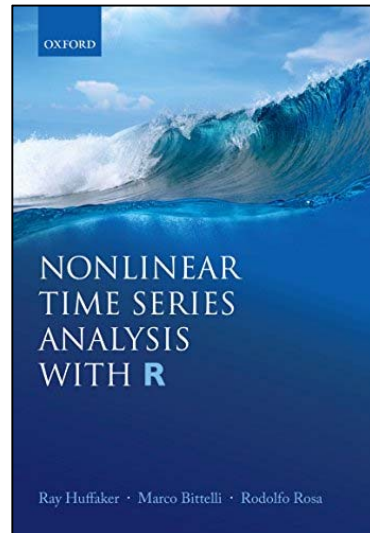
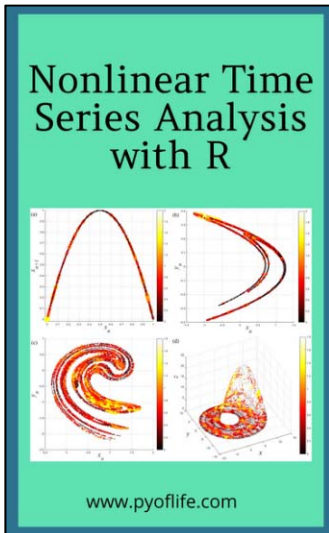
Time Series Analysis Using R

5/61



Time Series Analysis Using R

6/61





The multiple time series and multi-variate time series

7/61



You

多重時間序列(multiple time series)和多變量時間序列(multi-variate time series)有什麼差別?



ChatGPT

"多重時間序列" (Multiple Time Series) 與"多變量時間序列" (Multivariate Time Series) 這兩個術語在時間序列分析中有明顯的差異：

1. 多重時間序列 (Multiple Time Series) :

- 指的是多個時間序列數據集的集合。
- 每個時間序列代表隨時間變化的單一變量的數據序列。
- 其特點在於每個時間序列都是單變量的 (即隨時間跟踪單一變量) , 但同時分析多個這樣的序列。
- 例如, 你可能有特定地點的溫度、濕度和風速的時間序列數據。每一個都是單變量時間序列, 但將它們作為一組同時分析時, 就構成了多重時間序列。
- 分析的重點通常在於比較這些獨立的序列或者在彼此的背景下理解每一個序列。

2. 多變量時間序列 (Multivariate Time Series) :

- 多變量時間序列涉及在相同時間間隔觀察和記錄的多個變量或測量值。
- 在多變量時間序列中, 這些變量通常是相互依賴的, 分析的重點在於理解它們隨時間的相互關係。
- 例如, 一個多變量時間序列可能包括每小時記錄的溫度和濕度數據。這裡, 你不只是跟踪兩個分開的變量, 而是對它們在每個時間點的相互關係感興趣。
- 這種類型的分析更為複雜, 因為它涉及理解多個變量之間隨時間變化的動態相關性和交互作用。

總結來說, 兩者的區別在於變量的處理方式: 「多重時間序列」涉及獨立但並行地分析多個單變量時間序列, 而「多變量時間序列」涉及分析包含多個相互關聯變量的單一序列。

editGPT

```
> 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
```

Time	Temperature	cloud cover	humidity	wind
5:00 am	59 °F	97%	74%	8 mph SSE
6:00 am	59 °F	89%	75%	8 mph SSE
7:00 am	58 °F	79%	76%	7 mph SSE
8:00 am	58 °F	74%	77%	7 mph S
9:00 am	60 °F	74%	74%	7 mph S
10:00 am	62 °F	74%	70%	8 mph S
11:00 am	64 °F	76%	65%	8 mph SSW
12:00 pm	66 °F	80%	60%	8 mph SSW



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

9/61

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.



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.

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

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.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"
```

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),
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$year + 1900
[1] 2028
> my.date$yday
[1] 2
> my.date$yday
[1] 287
```



Two POSIXt sub-classes: POSIXlt / POSIXct 類別

12/61

- 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

13/61

```
> 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"
```

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
```

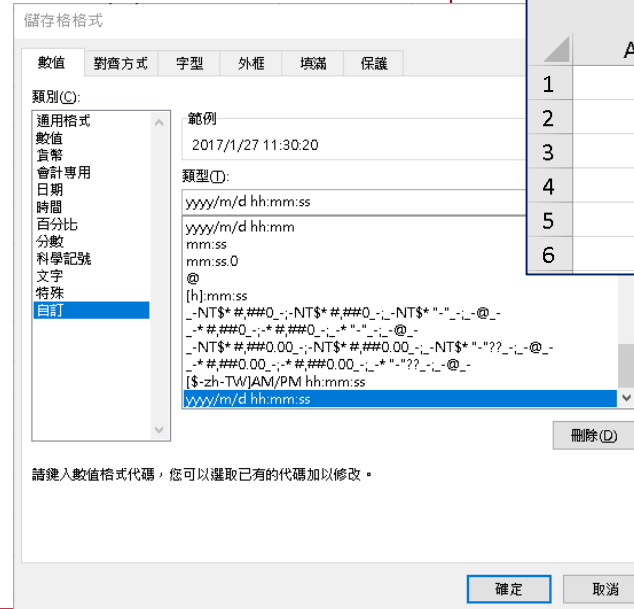
時間序列資料輸入與輸出: EXCEL檔

15/61

```
> library(readxl)
> mydate <- read_excel("mydate.xlsx", col_names = FALSE)
> mydate
# A tibble: 6 x 3
  ...1 ...2 ...3
  <dbl> <dbl> <dtm>
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

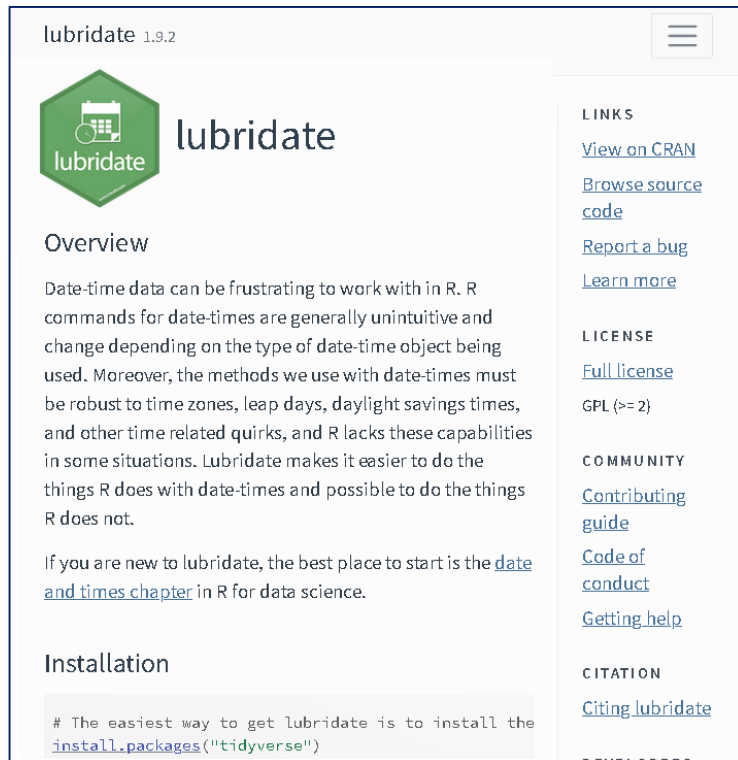
	A	B	C	D
1	1	73	2017/1/27	11:30:20
2	2	52	2017/3/5	12:01:40
3	3	57	2017/5/12	03:20:00
4	1	74	2017/8/27	14:00:00
5	2	51	2017/10/17	21:03:50
6	3	60	2017/12/8	08:40:30

```
> mydate2 <- read_excel("mydate2.xlsx", col_names = FALSE)
> mydate2
# A tibble: 6 x 4
  ...1 ...2 ...3 ...4
  <dbl> <dbl> <dtm> <dtm>
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")
```

lubridate: Make Dealing with Dates a Little Easier



lubridate 1.9.2

lubridate

Overview

Date-time data can be frustrating to work with in R. R commands for date-times are generally unintuitive and change depending on the type of date-time object being used. Moreover, the methods we use with date-times must be robust to time zones, leap days, daylight savings times, and other time related quirks, and R lacks these capabilities in some situations. Lubridate makes it easier to do the things R does with date-times and possible to do the things R does not.

If you are new to lubridate, the best place to start is the [date and times chapter](#) in R for data science.

Installation

```
# The easiest way to get lubridate is to install the
install.packages("tidyverse")
```

LINKS

- [View on CRAN](#)
- [Browse source code](#)
- [Report a bug](#)
- [Learn more](#)

LICENSE

- [Full license](#)
- GPL (>= 2)

COMMUNITY

- [Contributing guide](#)
- [Code of conduct](#)
- [Getting help](#)

CITATION

- [Citing lubridate](#)

DEVELOPERS

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
<pre>date <- as.POSIXct("01-01-2010", format = "%d-%m-%Y", tz = "UTC") as.numeric(format(date, "%m")) or as.POSIXlt(date)\$month + 1 date <- as.POSIXct(format(date, "%Y-2-%d"), tz = "UTC")</pre>	<pre>date <- dmy("01-01-2010") month(date) month(date) <- 2</pre>

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
<pre>date <- seq(date, length = 2, by = "-1 day")[2] as.POSIXct(format(as.POSIXct(date), tz = "UTC"), tz = "GMT")</pre>	<pre>date <- date - days(1) with_tz(date, "GMT")</pre>

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>

Dates and times with lubridate :: CHEATSHEET



Date-times



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"
```

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

- Identify the order of the year (y), month (m), day (d), hour (h), minute (m) and second (s) elements in your data.
- 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().` `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")`



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
```

GET AND SET COMPONENTS

Use an accessor function to get a component. Assign into an accessor function to change a component in place.

```
d ## "2017-11-28"
day(d) ## 28
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)`

2018-01-31 11:59:59 UTC `week(x)` Week of the year. `week(dt)`
`isoweek()` ISO 8601 week.
`epiweek()` Epidemiological week.

2018-01-31 11:59:59 UTC `quarter(x)` Quarter. `quarter(dt)`

2018-01-31 11:59:59 UTC `semester(x, with_year = FALSE)`
Semester. `semester(dt)`

2018-01-31 11:59:59 UTC `am(x)` Is it in the am? `am(dt)`

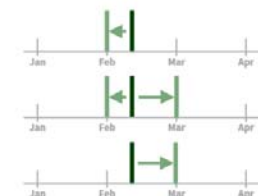
2018-01-31 11:59:59 UTC `pm(x)` Is it in the pm? `pm(dt)`

2018-01-31 11:59:59 UTC `dst(x)` Is it daylight savings? `dst(d)`

2018-01-31 11:59:59 UTC `leap_year(x)` Is it a leap year?
`leap_year(d)`

2018-01-31 11:59:59 UTC `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")
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()**.

- Derive a template, create a function
`sl <- stamp("Created Sunday, Jan 17, 1999 3:34")`

- Apply the template to dates
`sl(ymd("2010-04-05"))`
`## [1] "Created Monday, Apr 05, 2010 00:00"`

Tip: use a date with day > 12

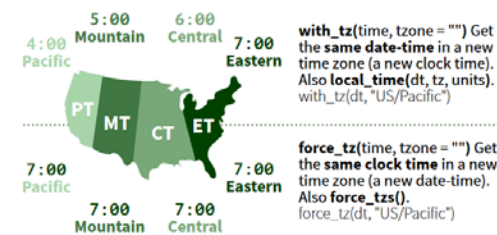
Time Zones

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.



Cheatsheet, lubridate

18/61

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")
```



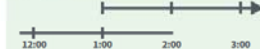
The start of daylight savings (spring forward)

```
gap <- ymd_hms("2018-03-11 01:30:00", tz = "US/Eastern")
```



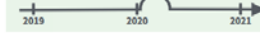
The end of daylight savings (fall back)

```
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)

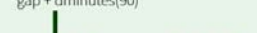


Durations track the passage of physical time, which deviates from clock time when irregularities occur.

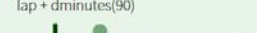
nor + dminutes(90)



gap + dminutes(90)



lap + dminutes(90)

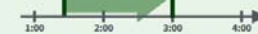


leap + dyears(1)



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

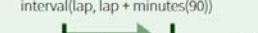
interval(nor, nor + minutes(90))



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)
```

```
"3m 12d 0h 0m 0s"
```

Number of months Number of days etc.

years(x = 1) x years.

months(x = 1) 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. **Diffimes** 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)
```

```
"1209600s (~2 weeks)"
```

Exact length in seconds Equivalent in common units

dyears(x = 1) 31536000x seconds.

dmomths(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 x 10⁻³ seconds.

dmicroseconds(x = 1) x x 10⁻⁶ seconds.

dnanoseconds(x = 1) x x 10⁻⁹ seconds.

dpicoseconds(x = 1) x x 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.diffime()**.
as.duration(i)

make_diffime(x) Make diffime with the specified number of units.
make_diffime(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.

```
i <- interval(ymd("2017-01-01"), d)
```

```
j <- d %~% ymd("2017-12-31")
```

Start Date End Date
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())**



CC BY SA Posit Software, PBC • info@posit.co • posit.co • Learn more at lubridate.tidyverse.org • HTML cheatsheets at pos.it/cheatsheets • lubridate 1.9.2 • Updated: 2023-07



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)

20/61

- 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) 英文: August, Aug
[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
```

```
> 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

25/61

```
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

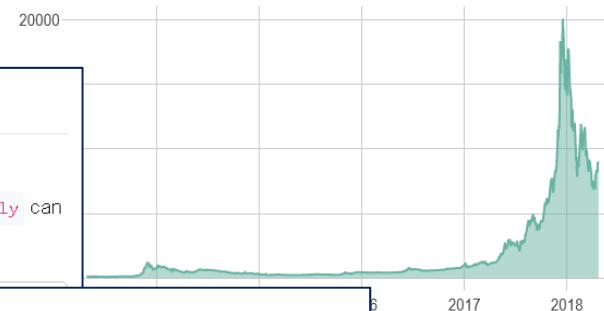
Time Series



INTERACTIVE VERSION: **PLOTLY**

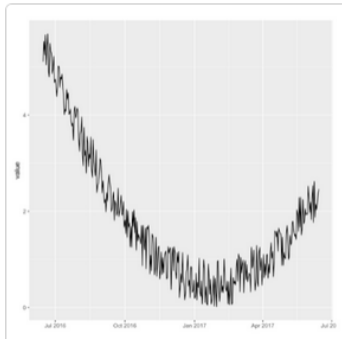
The `ggplotly()` function of the `plotly` library makes it a breeze to build an interactive version. Try to hover circles to get a tooltip, or select an area of interest for zooming. Double click to reinitialize.

GET CODE



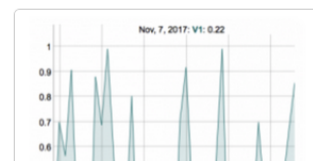
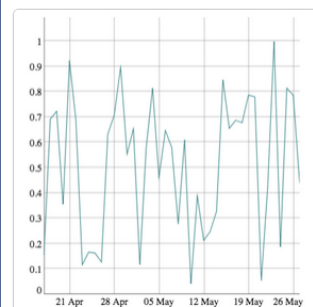
TIME SERIES WITH **GGPLOT2**

`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/>



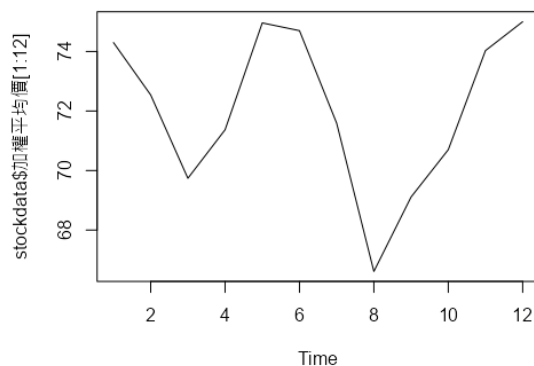
matplot {graphics}, geom_line {ggplot2}

27/61

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
台積電	100	2	77	69.9	72.54	235,159	74,985,055,548
台積電	100	3	72.2	65.7	69.74	276,434	88,459,924,495
台積電	100	4	73.9	68	71.37	211,611	70,177,023,098
台積電	100	5	76.9	73	74.96	213,185	74,005,599,560
台積電	100	6	78.2	70.4	74.7	260,507	96,761,306,205
台積電	100	7	73.9	68.5	71.59	238,386	73,569,965,426
台積電	100	8	72.8	62.2	66.61	305,409	84,617,942,159
台積電	100	9	72.1	65.9	69.11	266,720	74,225,030,814
台積電	100	10	74	68.1	70.7	181,361	59,947,670,693
台積電	100	11	76	71.3	74.03	197,579	65,432,526,407
台積電	100	12	76.8	72	75	179,107	53,687,756,290
威盛	100	1	33.4	29.3	30.97	55,107	4,580,913,795
威盛	100	2	32.65	28.35	30.54	26,901	2,060,809,696
威盛	100	3	35.45	28.5	32.01	55,882	4,355,434,678
威盛	100	4	32.8	27.55	30.35	136,058	651,18,81
威盛	100	5	32.6	25.95	29.4		
威盛	100	6	37.25	31.2	34.68		
威盛	100	7	38.15	32.45	35.47		
威盛	100	8	35.4	26.6	30.13		
威盛	100	9	29	23.1	26.17		
威盛	100	10	25.15	20.4	23.39		
威盛	100	11	25.7	18.7	22.74		
威盛	100	12	20.2	14.8	16.96		
聯發科	100	1	424	378	403.55	106,530	57,621,649,341
聯發科	100	2	380	325.5	348.98	97,339	46,409,931,806
聯發科	100	3	355	312.5	339.96	117,960	52,887,228,668

```
stockdata <- read.table("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}

28/61

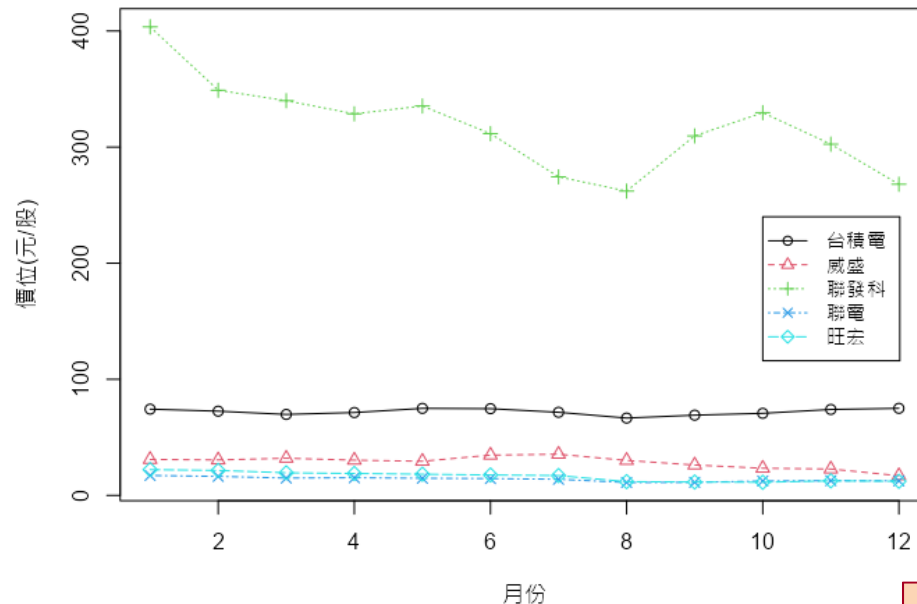
```
mat <- as.data.frame(matrix(stockdata$加權平均價, ncol = 5))  
colnames(mat) <- unique(stockdata$半導體公司)  
mat
```

`type = "l"`

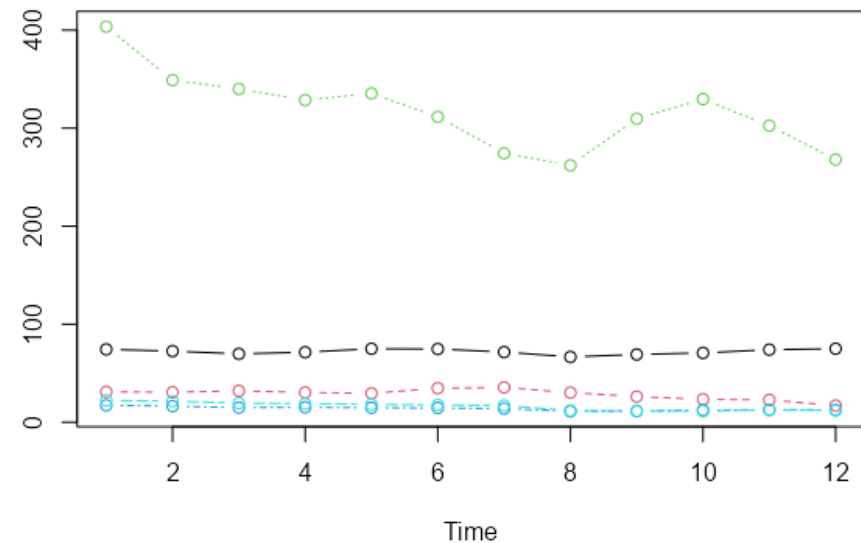
```
matplot(1:12, mat, type = "o", lty = 1:5, col = 1:5, pch = 1:5,  
        main = "民國100年5家半導體公司股票月成交資訊(元,股)",  
        xlab = "月份", ylab = "價位(元/股)")  
legend(10, 240, legend = colnames(mat), lty = 1:5, col = 1:5,  
       pch = 1:5, cex = 0.9)
```

```
> 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
```

民國100年5家半導體公司股票月成交資訊(元,股)



multiple time series data



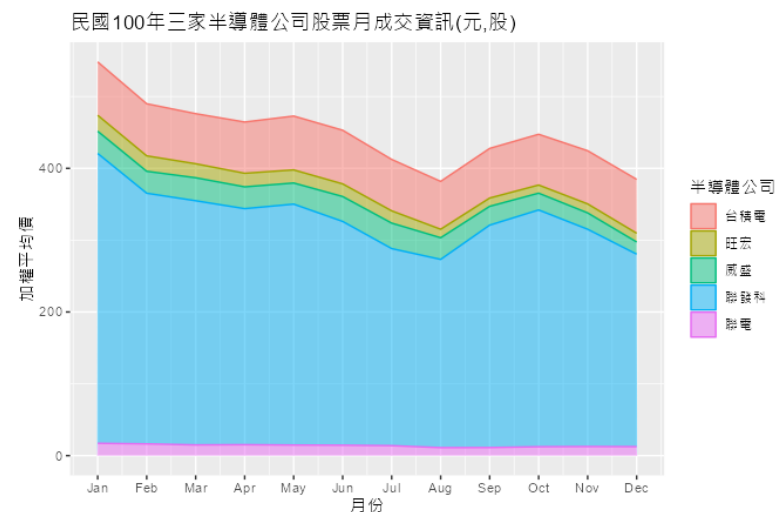
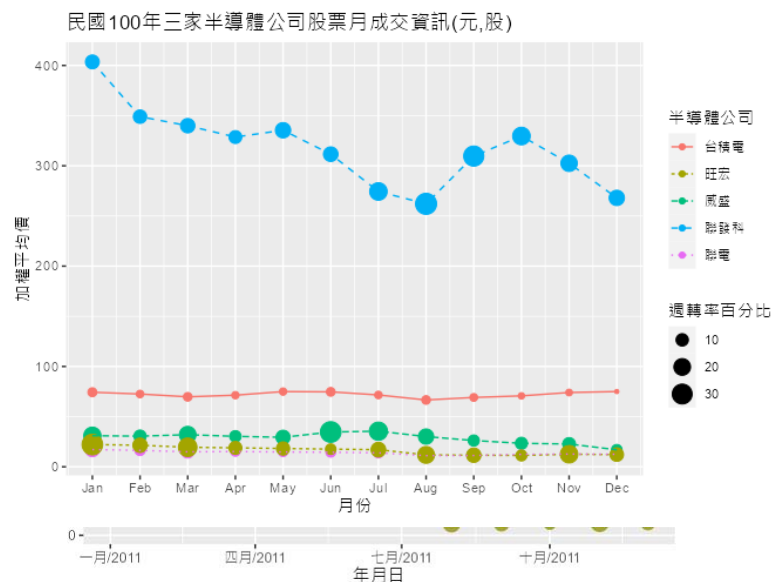
```
ts.plot(mat, lty = 1:5, col = 1:5, type = "b")
```

使用 geom_line {ggplot2}

29/61

```
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** for **ts**, **mts** Object

30/61

```
> 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:

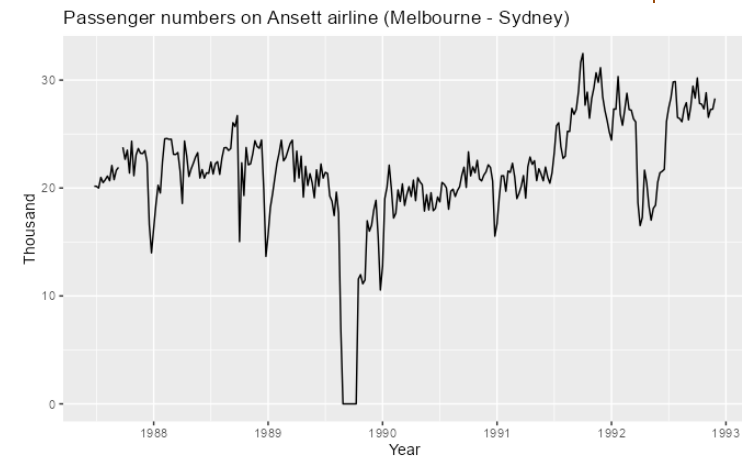
Start = c(1987, 26)

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"
```

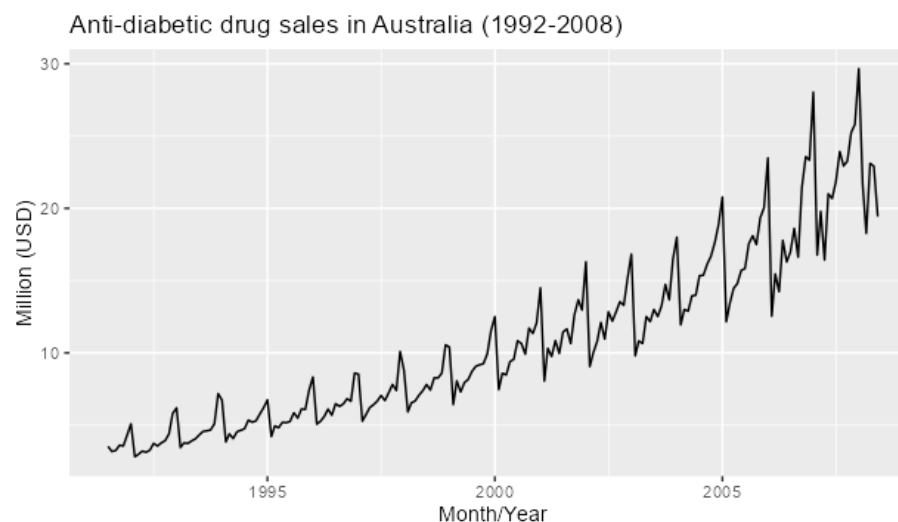
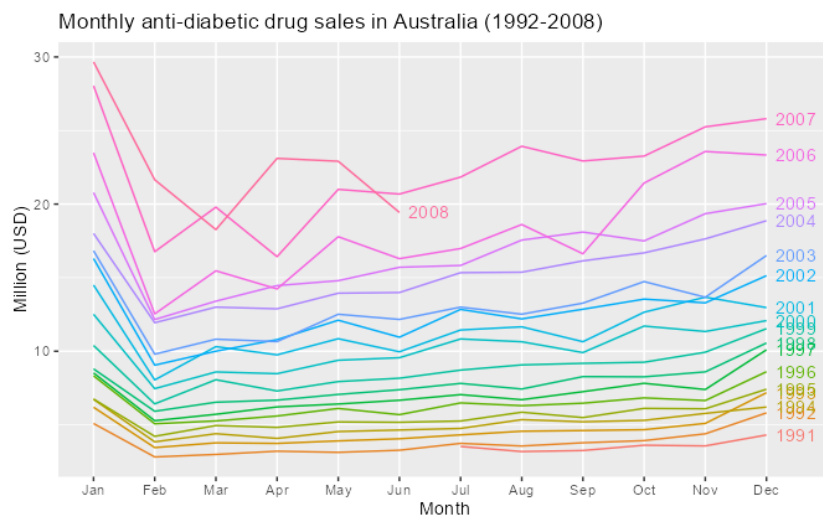
```
> ggseasonplot(a10, year.labels = TRUE) +
```

```
+ 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

32/61

```
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)
```

```
> 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
```

```
> # 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
```

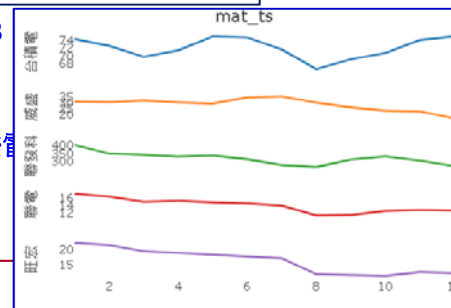
```
..$ : NULL
```

```
..$ : chr [1:5] "台積電" "威盛" "聯發科" "聯電" "旺宏"
```

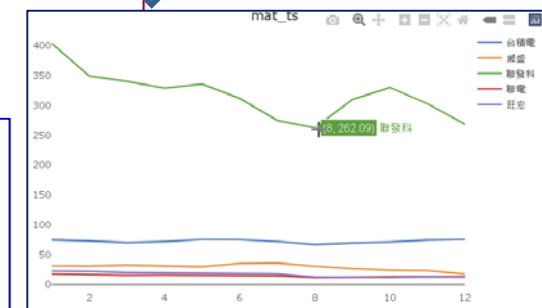
```
> ts_plot(mat_ts)
```

```
> 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
```



```
> 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
```



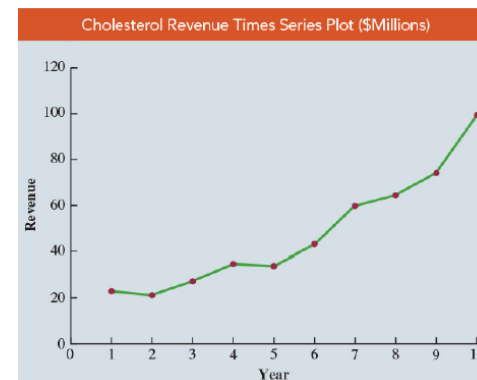
基本時間序樣態 (Basic Time Series Patterns)

33/61

Horizontal Pattern

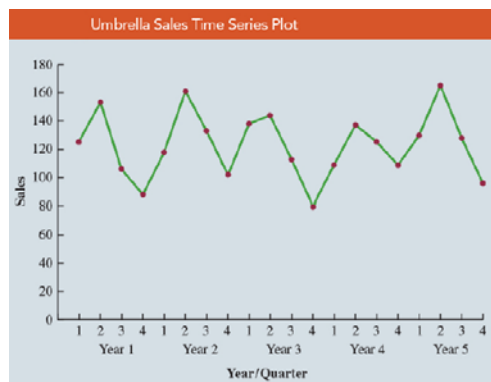


Trend Pattern

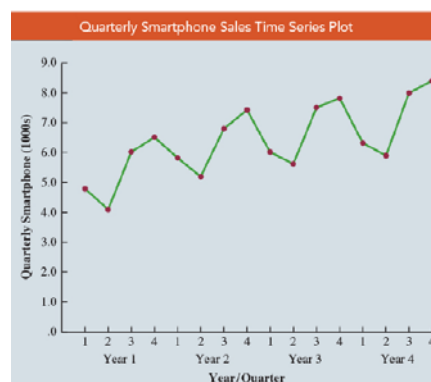


Images Source: Anderson et al., 2019, Statistics for Business & Economics (14th Edition), Cengage Learning Ltd

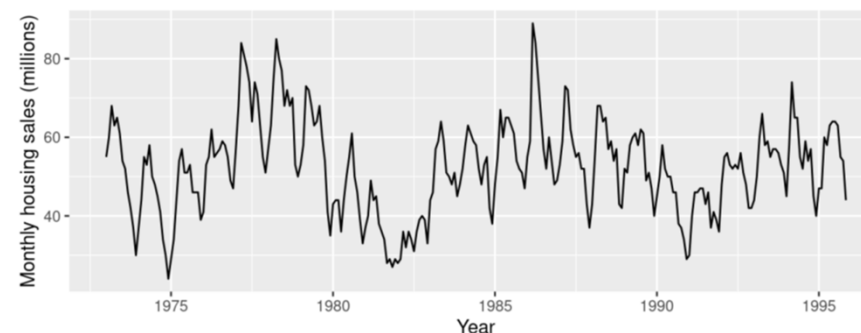
Seasonal Pattern



Trend and Seasonal Pattern



Cyclical Pattern



- If the frequency is unchanging and associated with some aspect of the calendar, then the pattern is seasonal.
- If the fluctuations are not of a fixed frequency then they are cyclic.
- In general, the average length of cycles is longer than the length of a seasonal pattern.
- The magnitudes of cycles tend to be more variable than the magnitudes of seasonal patterns.

(Autocorrelation Function, ACF)

- Autocorrelation is the correlation between two observations at different points in a time series.
- A time series y_1, y_2, \dots, y_T .
- mean: $\bar{y} = \frac{\sum_{t=1}^T y_t}{T}$.
- The autocorrelation function at lag k :

$$ACF(k) = \frac{\sum_{t=1}^{T-k} (y_t - \bar{y})(y_{t+k} - \bar{y})}{\sum_{t=1}^T (y_t - \bar{y})^2} = \frac{\sum_{t=k+1}^T (y_t - \bar{y})(y_{t-k} - \bar{y})}{\sum_{t=1}^T (y_t - \bar{y})^2}$$

Week	Sales
1	68
2	84
3	76
4	92
5	72
6	64
7	80
8	72
9	88
10	80
11	60
12	88

$lag(k = 1)$

Week	Sales
1	68
2	84
3	76
4	92
5	72
6	64
7	80
8	72
9	88
10	80
11	60
12	88

$lag(k = 2)$

Week	Sales
1	68
2	84
3	76
4	92
5	72
6	64
7	80
8	72
9	88
10	80
11	60
12	88

$lag(k = 3)$

Week	Sales
1	68
2	84
3	76
4	92
5	72
6	64
7	80
8	72
9	88
10	80
11	60
12	88

$lag(k = 4)$

correlogram:
 $ACF(k)$ against k



自我相關函數 (Autocorrelation Function, ACF)

35/61

■ Randomness/White Noise:

- ACF should be **near zero for all lags**.
- Non-random data have at least **one significant lag**.
- When the data are not random, you need to **incorporate lags into a regression analysis** to model the data appropriately.

■ Stationarity:

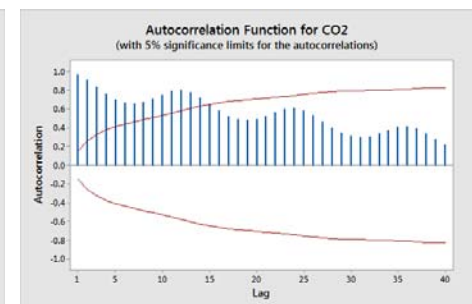
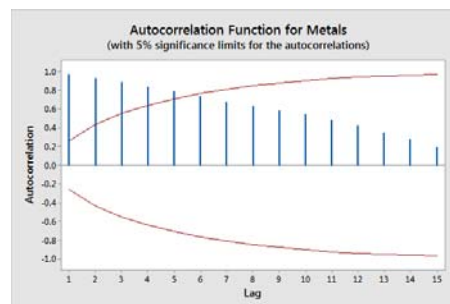
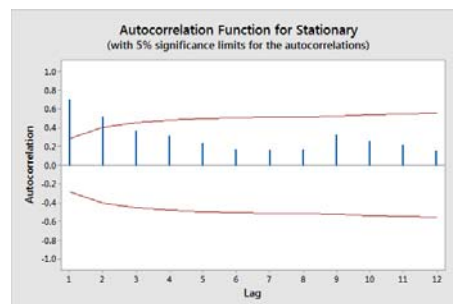
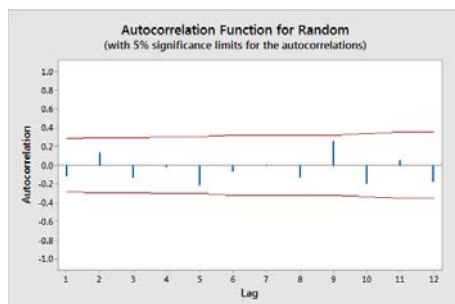
- The time series does not have a trend, has a constant variance, a constant autocorrelation pattern, and no seasonal pattern.
- Stationary time series : ACF **declines to near zero** rapidly.
- ACF drops slowly for a non-stationary time series.

■ Trends:

- When trends are present in a time series, shorter lags typically have large positive correlations.
- ACF taper off slowly as the lags increase.

■ Seasonality:

- ACF are larger for lags at multiples of the seasonal frequency than for other lags. (**wavy correlations**)
- When a time series has both a trend and seasonality, the ACF plot displays a **mixture** of both effects.



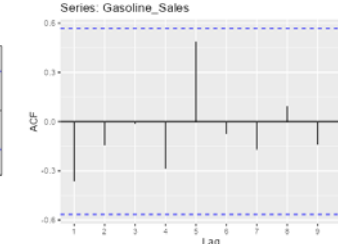
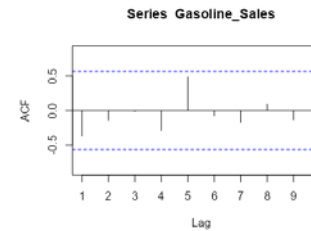
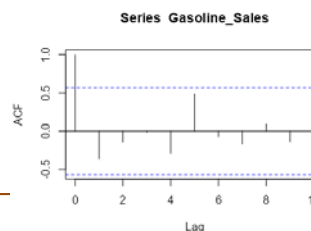
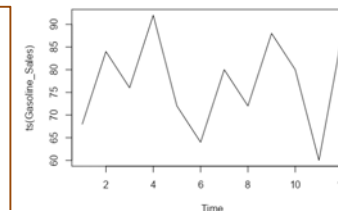
Images source: <https://statisticsbyjim.com/time-series/autocorrelation-partial-autocorrelation/>

自我相關函數 (Autocorrelation Function, ACF)

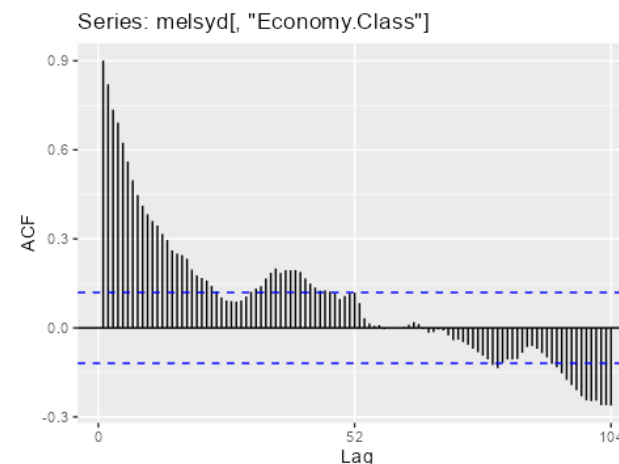
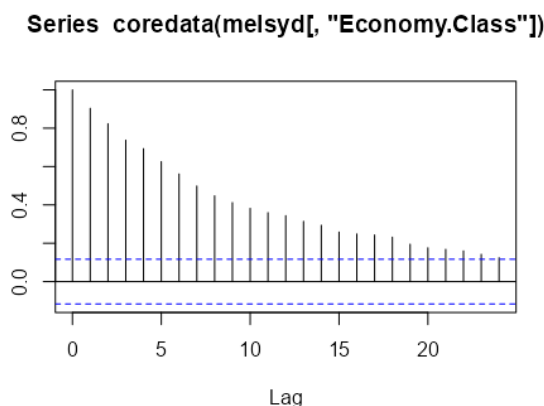
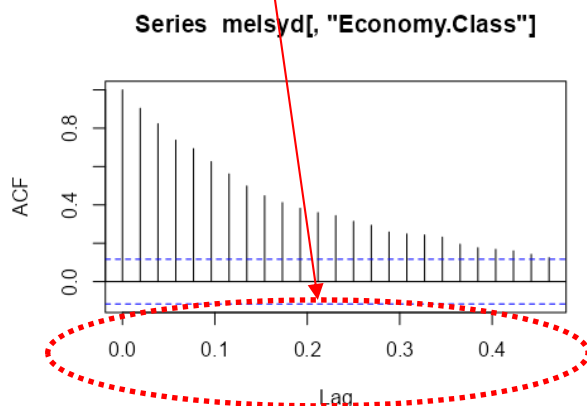
36/61

```
> Gasoline_Sales <- ts(c(68, 84, 76, 92, 72, 64, 80, 72, 88, 80, 60, 88))
> Gasoline_Sales
Time Series:
Start = 1
End = 12
Frequency = 1
[1] 68 84 76 92 72 64 80 72 88 80 60 88
> acf(Gasoline_Sales) # base package
> Acf(Gasoline_Sales) # forecast package
> ggAcf(Gasoline_Sales) # forecast package
```

```
plot(ts(Gasoline_Sales))
```



```
class(melsyd[, "Economy.Class"])
acf(melsyd[, "Economy.Class"], na.action = na.pass)
acf(coredata(melsyd[, "Economy.Class"]), na.action = na.pass)
Acf(melsyd[, "Economy.Class"])
ggAcf(melsyd[, "Economy.Class"])
```





Durbin Watson Test

(An autocorrelation test)

37/61

- One of the key assumptions in linear regression is that there is no correlation between the residuals, e.g. the residuals are independent. Perform a Durbin-Watson test to detect the presence of autocorrelation in the residuals of a regression.
- The Durbin Watson Test is a measure of autocorrelation (i.e, serial correlation), the **AR(1) process**, in residuals from regression analysis.

Hypothesis:

$H_0 = \text{no first order autocorrelation.}$

$H_1 = \text{first order correlation exists.}$

(For a first order correlation, the lag is one time unit).

Assumptions:

- The errors are $N(0, \sigma^2)$.
- The errors are stationary.

Test statistic:

$$DW = \frac{\sum_{t=2}^T (e_t - e_{t-1})^2}{\sum_{t=1}^T e_t^2}$$

where e_t are residuals from an OLS regression.

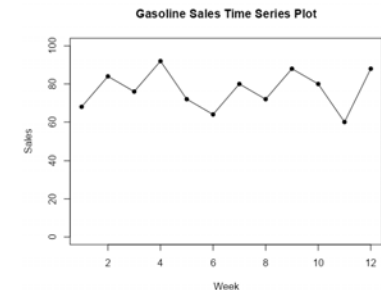
DW = 2: is no autocorrelation.

DW = 0 ~ 2 is positive autocorrelation.

DW > 2 ~ 4 is negative autocorrelation.

A rule of thumb is that test statistic values in the range of 1.5 to 2.5 are relatively normal.

```
> library(readxl)
> Gasoline <- read_excel("Gasoline.xlsx")
> Gasoline
# A tibble: 12 x 2
  Week Sales
  <dbl> <dbl>
1     1     68
2     2     84
...
12    12     88
> plot(Sales ~ Week, data = Gasoline, type = "o",
+       main = "Gasoline Sales Time Series Plot",
+       ylim = c(0, 100), pch = 16)
>
> Gasoline_lm <- lm(Sales ~ Week, data = Gasoline)
> library(lmtest)
> dwtest(formula = Gasoline_lm,
+         alternative = "two.sided")
```



```
      Durbin-Watson test
data:  Gasoline_lm
DW = 2.5478, p-value = 0.5156
alternative hypothesis: true autocorrelation is not 0
```

移動平均 (moving average，簡稱均線)

- 移動平均法使用 n 期的資料進行平均。
- **簡單移動平均** $SMA_t = \frac{x_t + x_{t-1} + x_{t-2} + \cdots + x_{t-n}}{n+1}$
- 期數選擇影響很大，該如何選擇：
 - 事件發展有無週期性。若有，應以此週期為期數，藉以消除週期影響。
 - 對趨勢平均性的要求。期數愈多，修勻效果愈平均。
 - 對趨勢反映近期變化敏感程度的要求。想得到長期趨勢，期數選擇要大。短期趨勢則選擇小期數。
- **加權移動平均** $WMA_t = w_0x_t + w_1x_{t-1} + \cdots + w_nx_{t-n}$
- **指數移動平均** $EMA_t = \alpha x_t + \alpha(1 - \alpha)x_{t-1} + \alpha(1 - \alpha)^2x_{t-2} + \cdots$

$$F_{t+1} = \alpha Y_t + (1 - \alpha)F_t \quad F_1 = Y_1.$$

F_{t+1} : forecast of the time series for period $(t + 1)$

Y_t : actual value of the time series in period t

F_t : forecast of the time series for period t

α : smoothing constant $(0 \leq \alpha \leq 1)$

$$F_4 = \alpha Y_3 + (1 - \alpha)F_3$$

$$= \alpha Y_3 + \alpha(1 - \alpha)Y_2 + (1 - \alpha)^2Y_1$$



Smoothing and Moving Average in R

smooth: Forecasting Using Smoothing Functions

<https://cran.r-project.org/web/packages/smooth/index.html>

es() - Exponential Smoothing;

ssarima() - State-Space ARIMA, also known as Several Seasonalities ARIMA;

ces() - Complex Exponential Smoothing;

ges() - Generalised Exponential Smoothing;

ves() - Vector Exponential Smoothing;

sma() - Simple Moving Average in state-space form;

TTR: Technical Trading Rules

<https://cran.r-project.org/web/packages/TTR/index.html>

SMA(x, n = 10, ...)

EMA(x, n = 10, wilder = FALSE, ratio = NULL, ...)

DEMA(x, n = 10, v = 1, wilder = FALSE, ratio = NULL)

WMA(x, n = 10, wts = 1:n, ...)

EVWMA(price, volume, n = 10, ...)

ZLEMA(x, n = 10, ratio = NULL, ...)

VWAP(price, volume, n = 10, ...)

VMA(x, w, ratio = 1, ...)

HMA(x, n = 20, ...)

ALMA(x, n = 9, offset = 0.85, sigma = 6, ...)

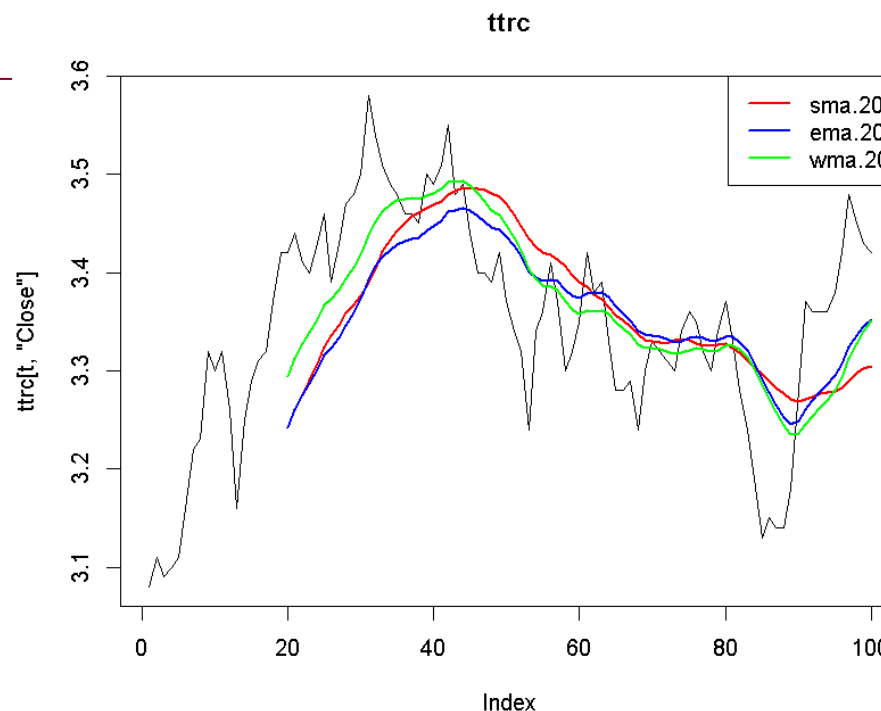
- SMA: Simple moving average.
- EMA: Exponential moving average.
- WMA: Weighted moving average.
- DEMA: Double-exponential moving average.
- EVWMA: Elastic, volume-weighted moving average.
- ZLEMA: Zero lag exponential moving average.
- VWMA: Volume-weighted moving average (same as VWAP).
- VWAP: Volume-weighted average price (same as VWMA).
- VWA: Variable-length moving average.
- HMA: Hull moving average.
- ALMA: Arnaud Legoux moving average.

ttrc {TTR}: Technical Trading Rule Composite data

Historical Open, High, Low, Close, and Volume data for the periods January 2, 1985 to December 31, 2006. Randomly generated.

```
> # install.packages("TTR")
> library(TTR)
> data(ttrc)
> dim(ttrc)
[1] 5550    6
> head(ttrc)
      Date Open High Low Close Volume
1 1985-01-02 3.18 3.18 3.08  3.08 1870906
2 1985-01-03 3.09 3.15 3.09  3.11 3099506
3 1985-01-04 3.11 3.12 3.08  3.09 2274157
4 1985-01-07 3.09 3.12 3.07  3.10 2086758
5 1985-01-08 3.10 3.12 3.08  3.11 2166348
6 1985-01-09 3.12 3.17 3.10  3.16 3441798
```

```
> t <- 1:100
> sma.20 <- SMA(ttrc[t, "Close"], 20)
> ema.20 <- EMA(ttrc[t, "Close"], 20) # Arms' ----
> wma.20 <- WMA(ttrc[t, "Close"], 20)
>
> plot(ttrc[t, "Close"], type = "l", main = "ttrc")
> lines(sma.20, col = "red", lwd = 2)
> lines(ema.20, col = "blue", lwd = 2)
> lines(wma.20, col = "green", lwd = 2)
> legend("topright", legend = c("sma.20", "ema.20", "wma.20"),
+       col = c("red", "blue", "green"), lty = 1, lwd = 2)
```



預測正確率 (Forecast Accuracy)

- **ForecastError** = $ActualValue - Forecast$
- **PercentageError** = $ForecastError / ActualValue * 100\%$
- **The mean absolute error (MAE)**: the average of the absolute values of the forecast errors.
- **The mean squared error (MSE)**: the average of the sum of squared forecast errors.
- **The mean absolute percentage error (MAPE)**: average of the absolute value of percentage forecast errors.

Naive forecasting method: the simplest of all the forecasting methods that uses the most recent week's sales volume as the forecast for the next week.

Week	Time Series Value	Forecast	Forecast Error	Absolute Value of Forecast Error	Squared Forecast Error	Percentage Error	Absolute Value of Percentage Error
1	68						
2	84	68	16	16	256	19.05	19.05
3	76	84	-8	8	64	-10.53	10.53
4	92	76	16	16	256	17.39	17.39
5	72	92	-20	20	400	-27.78	27.78
6	64	72	-8	8	64	-12.50	12.50
7	80	64	16	16	256	20.00	20.00
8	72	80	-8	8	64	-11.11	11.11
9	88	72	16	16	256	18.18	18.18
10	80	88	-8	8	64	-10.00	10.00
11	60	80	-20	20	400	-33.33	33.33
12	88	60	28	28	784	31.82	31.82
		Totals	20	164	2864	1.19	211.69

$$MAE = \underline{164/11=14.91} \quad MSE = \underline{2864/11=260.36} \quad MAPE = \underline{211.69/11 = 19.24\%}$$

Images Source: Anderson et al., 2019, Statistics for Business & Economics (14th Edition), Cengage Learning Ltd

範例: Naive Method

42/61

```
> library(readxl)
> Gasoline <- read_excel("Gasoline.xlsx")
> Gasoline
```

```
# A tibble: 12 x 2
```

```
  Week Sales
  <dbl> <dbl>
```

```
1     1    68
2     2    84
3     3    76
4     4    92
5     5    72
6     6    64
7     7    80
8     8    72
9     9    88
10    10    80
11    11    60
12    12    88
```

```
>
> Gasoline_fc_naive <- naive(Gasoline_ts)
> Gasoline_fc_naive$fitted
```

```
Time Series:
```

```
Start = 1
```

```
End = 12
```

```
Frequency = 1
```

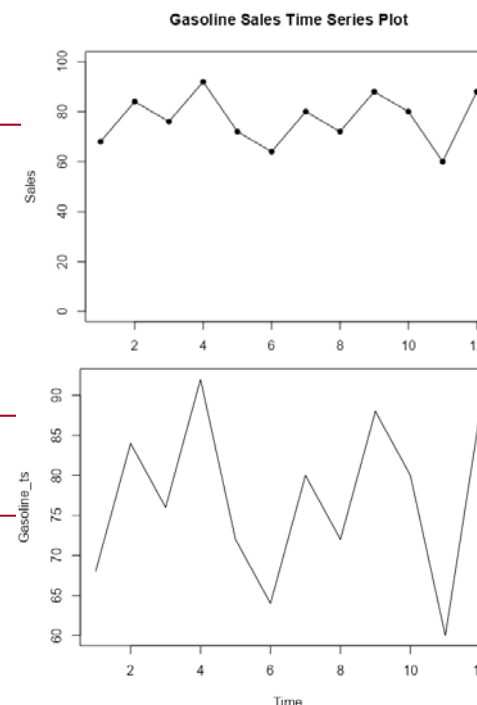
```
[1] NA 68 84 76 92 72 64 80 72 88 80 60
```

```
> summary(Gasoline_fc_naive)
```

```
> accuracy(Gasoline_fc_naive)
```

	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1
Training set	1.818182	16.13579	14.90909	0.1082169	19.24431	1	-0.4553872

```
> plot(Sales ~ Week, data = Gasoline, type = "o",
+       main = "Gasoline Sales Time Series Plot",
+       ylim = c(0, 100), pch = 16)
>
> library(forecast)
> Gasoline_ts <- ts(Gasoline$Sales)
> plot(Gasoline_ts)
```



```
> summary(Gasoline_fc_naive)
```

```
Forecast method: Naive method
```

```
Model Information:
```

```
Call: naive(y = Gasoline_ts)
```

```
Residual sd: 16.1358
```

```
Error measures:
```

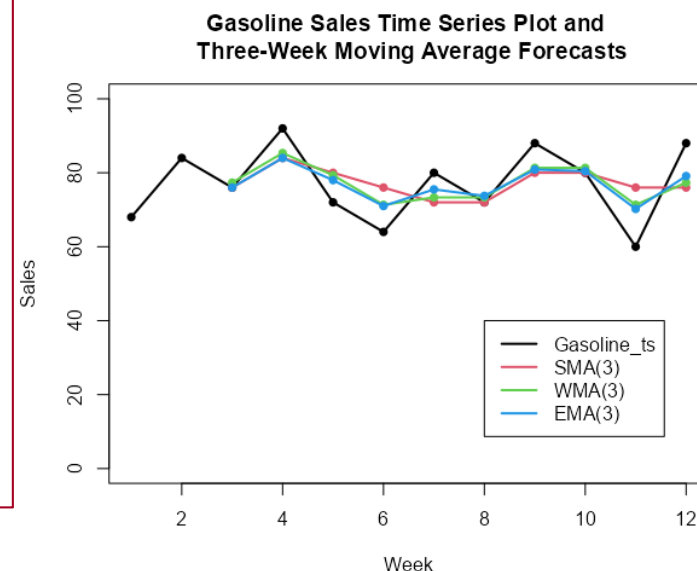
	ME	RMSE	MAE	MPE	MAPE	MASE	ACF1
Training set	1.818182	16.13579	14.90909	0.1082169	19.24431	1	-0.4553872

```
Forecasts:
```

	Point Forecast	Lo 80	Hi 80	Lo 95	Hi 95
13	88	67.32116	108.6788	56.374438	119.6256
14	88	58.75570	117.2443	43.274701	132.7253
22	88	22.60775	153.3922	-12.008809	188.0088

範例: Moving Average and Forecast Accuracy

```
> library(TTR)
> Gasoline_fc_sma3 <- SMA(Gasoline_ts, n = 3)
> Gasoline_fc_wma3 <- WMA(Gasoline_ts, n = 3)
> Gasoline_fc_ema3 <- EMA(Gasoline_ts, n = 3)
>
> accuracy(Gasoline_fc_sma3, Gasoline_ts[2:12])
      ME      RMSE      MAE      MPE      MAPE
Test set 0 12.78888 10.66667 -2.310057 14.35661
> accuracy(Gasoline_fc_wma3, Gasoline_ts[2:12])
      ME      RMSE      MAE      MPE      MAPE
Test set 0.2222222 13.5592 11.92593 -2.129945 15.99248
> accuracy(Gasoline_fc_ema3, Gasoline_ts[2:12])
      ME      RMSE      MAE      MPE      MAPE
Test set 0.6909722 13.44512 11.98264 -1.502064 15.9555
```

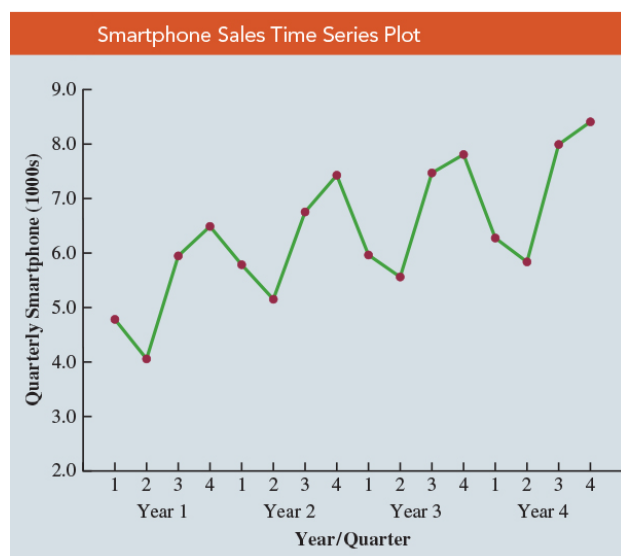


```
> # Gasoline Sales Time Series Plot and Three-Week Moving Average Forecasts
> plot(Gasoline_ts, type = "o", ylim = c(0, 100), lwd = 2, pch = 16,
+       xlab = "Week", ylab = "Sales",
+       main = "Gasoline Sales Time Series Plot and \n Three-Week Moving Average Forecasts")
> points(Gasoline_fc_sma3, type = "o", lwd = 2, col = 2, pch = 16)
> points(Gasoline_fc_wma3, type = "o", lwd = 2, col = 3, pch = 16)
> points(Gasoline_fc_ema3, type = "o", lwd = 2, col = 4, pch = 16)
> legend(8, 40, legend = c("Gasoline_ts", "SMA(3)", "WMA(3)", "EMA(3)"),
+       col = 1:4, lty = 1, lwd = 2)
```

- The general form of the estimated **multiple regression equation** for modeling both the quarterly seasonal effects and the linear trend in the smartphone time series

$$\hat{Y}_t = b_0 + b_1 Qtr1 + b_2 Qtr2 + b_3 Qtr3 + b_4 t$$

Seasonality and Trend



\hat{Y}_t = estimate or forecast of sales in time period t

dummy variable

$Qtr1 = 1$ ($Qtr2 = 1$) ($Qtr3 = 1$)

if time period t corresponds to the first
(second) (third) quarter of the year; 0 otherwise.

The estimated multiple regression equation

$$Sales(1000s) = 6.069 - 1.363 Qtr1 - 2.034 Qtr2 - 0.304 Qtr3 + 0.1456 t$$

Forecast for Time Period 17 (Quarter 1 in Year 5):

$$Sales(1000s) = 6.069 + 1.363(1) + 2.034(0) + 0.304(0) + 0.1456(17) = 7.18$$



範例: Fit A Multiple Regression Model to Smartphone Sales Time Series Data

45/61

```
> SmartphoneSales <- read_excel("SmartphoneSales.xlsx")
```

```
> SmartphoneSales
```

```
# A tibble: 16 x 3
```

	Year	Quarter	`Sales (1000s)`
	<dbl>	<dbl>	<dbl>
1	1	1	4.8
2	1	2	4.1
3	1	3	6
4	1	4	6.5
5	2	1	5.8
6	2	2	5.2
7	2	3	6.8
8	2	4	7.4
9	3	1	6
10	3	2	5.6
11	3	3	7.5
12	3	4	7.8
13	4	1	6.3
14	4	2	5.9
15	4	3	8
16	4	4	8.4

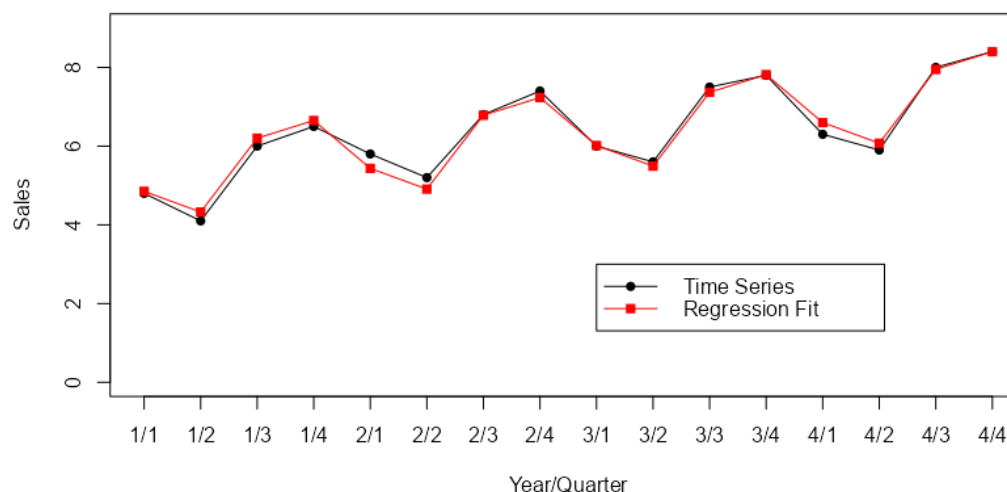
```
> SmartphoneSales_ts <- ts(SmartphoneSales$`Sales (1000s)`,  
+                           start =1, frequency = 4)
```

```
>
```

```
> plot(SmartphoneSales_ts, type = "o", pch = 16,  
+       ylim = c(0, 9), xlab = "Year/Quarter", ylab = "Sales",  
+       xaxt = "n",  
+       main = "Smartphone Sales Time Series Plot")
```

```
> axis(1, at = seq(from = 1, by = 0.25, length.out = length(SmartphoneSales_ts)),  
+       labels = paste0(rep(1:4, each = 4), "/", rep(1:4, 4)))
```

Smartphone Sales Time Series Plot





範例: Smartphone Sales Time Series

```
> SmartphoneSales$Quarter <- factor(SmartphoneSales$Quarter, levels = 4:1)
> SmartphoneSales$Period <- 1:nrow(SmartphoneSales)
> SmartphoneSales_lm <- lm(`Sales (1000s)` ~ Quarter + Period, data = SmartphoneSales)
> summary(SmartphoneSales_lm)
```

Call:

```
lm(formula = `Sales (1000s)` ~ Quarter + Period, data = SmartphoneSales)
```

Residuals:

	Min	1Q	Median	3Q	Max
	-0.2988	-0.1569	-0.0075	0.1150	0.3663

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	6.06875	0.16250	37.347	6.12e-13	***
Quarter3	-0.30438	0.15368	-1.981	0.0732	.
Quarter2	-2.03375	0.15511	-13.112	4.66e-08	***
Quarter1	-1.36313	0.15745	-8.657	3.06e-06	***
Period	0.14563	0.01211	12.023	1.14e-07	***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.2167 on 11 degrees of freedom

Multiple R-squared: 0.9763, Adjusted R-squared: 0.9676

F-statistic: 113.2 on 4 and 11 DF, p-value: 7.376e-09

```
> points(x = seq(from = 1, by = 0.25, length.out = length(SmartphoneSales_ts)),
+        y = SmartphoneSales_lm$fitted.values,
+        type = "o", pch = 15, col = "red")
> legend(3, 3, legend = c("Time Series", "Regression Fit"), lty = 1,
+        col = c("black", "red"), pch = c(16, 15))
```

$$\text{Sales}(1000s) = 6.069 - 1.363 \text{ Qtr1} - 2.034 \text{ Qtr2} - 0.304 \text{ Qtr3} + 0.1456 t$$

Forecast quarterly sales for next year. Next year is year 5 for the smartphone sales time series; that is, time periods 17, 18, 19, and 20.

```
> predict(SmartphoneSales_lm,
+         newdata = data.frame(Quarter =
+                               factor(1:4),
+                               Period = 17:20))
           1           2           3           4
7.18125 6.65625 8.53125 8.98125
```



時間序列模型基本概念: 穩定、平穩、穩態 (Stationary)

47/61

The foundation of time series analysis is stationarity.

A time series $\{r_t\}$ is said to be *strictly stationary* if the joint distribution of $(r_{t_1}, \dots, r_{t_k})$ is identical to that of $(r_{t_1+t}, \dots, r_{t_k+t})$ for all t , where k is an arbitrary positive integer and (t_1, \dots, t_k) is a collection of k positive integers.



WIKIPEDIA
The Free Encyclopedia

ADF Test (Augmented Dickey-Fuller)

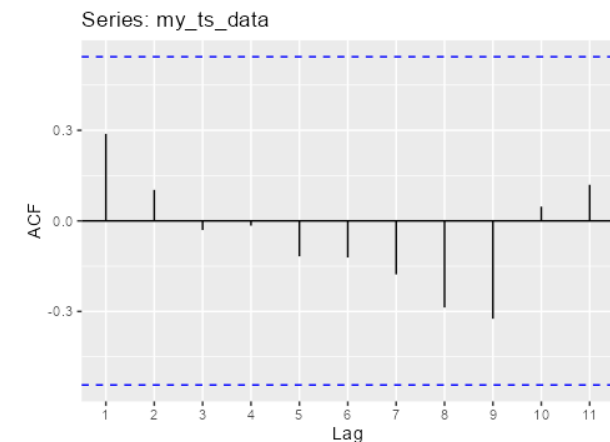
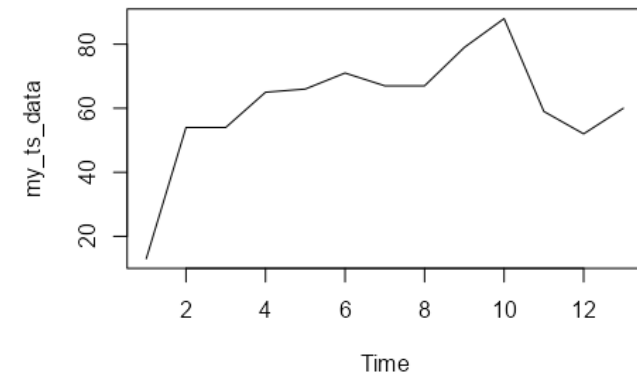
https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93Fuller_test

H_0 : The time series is non-stationary.

```
> library(tseries)
> my_ts_data <- ts(c(13, 54, 54, 65, 66, 71, 67, 67,
+                   79, 88, 59, 52, 60))
> plot(my_ts_data)
> ggAcf(my_ts_data)
> adf.test(my_ts_data)
```

Augmented Dickey-Fuller Test

```
data: my_ts_data
Dickey-Fuller = -1.6549, Lag order = 2, p-value = 0.7039
alternative hypothesis: stationary
```





時間序列模型基本概念

AR, MA, ARMA, ARIMA 模型

48/61

自迴歸模型: Autoregressive Model, $AR(p)$

$$Y_t = c + \sum_{k=1}^p \phi_k Y_{t-k} + \epsilon_t$$

A time series r_t is called a white noise if $\{r_t\}$ is a sequence of independent and identically distributed random variables with finite mean and variance.

$\{\epsilon_t\}$ is a white noise process
with mean 0 and variance σ^2

移動平均模型: Moving Average Model, $MA(q)$

$$Y_t = c + \sum_{i=1}^q \theta_i \epsilon_{t-i} + \epsilon_t$$

移動平均自迴歸模型

Autoregressive Moving Average Model, $ARMA(p, q)$

$$Y_t = c + \sum_{k=1}^p \phi_k Y_{t-k} + \sum_{i=1}^q \theta_i \epsilon_{t-i} + \epsilon_t$$

有些書是用減號的

$$ARMA(1, 1) \quad Y_t = \phi Y_{t-1} + \epsilon_t + \theta \epsilon_{t-1}$$



(差分)整合移動平均自迴歸模型， $ARIMA(p, d, q)$

Box-Jenkins backshift operator $\Rightarrow B^p X_t = X_{t-p}$

$$ARMA(p, q) \quad Y_t = \sum_{i=1}^p \phi_i Y_{t-i} + a_t - \sum_{j=1}^q \theta_j a_{t-j}$$

$$\Rightarrow \left(1 - \sum_{i=1}^p \phi_i B^i\right) Y_t = \left(1 - \sum_{j=1}^q \theta_j B^j\right) a_t$$

$$\Rightarrow \phi_p(B) Y_t = \theta_q(B) a_t$$

$$\text{where } \phi_p(B) = \left(1 - \sum_{i=1}^p \phi_i B^i\right) \text{ and } \theta_q(B) = \left(1 - \sum_{j=1}^q \theta_j B^j\right).$$

- p 為自我迴歸項數。
- q 為移動平均項數。
- d 為使之成為平穩序列所做的差分次數(階數)。

When time series exhibit nonstationary behavior, then the ARMA model presented above can be extended and written using differences:

$$\begin{aligned} W_t &= Y_t - Y_{t-1} = (1 - B)Y_t \\ W_t - W_{t-1} &= Y_t - 2Y_{t-1} + Y_{t-2} \\ &= (1 - B)^2 Y_t \\ &\vdots \end{aligned}$$

$$W_t - \sum_{k=1}^d W_{t-k} = (1 - B)^d Y_t,$$

Replacing in the ARMA model with the differences defined above yields the formal

$ARIMA(p, d, q)$ model:

$$\phi_p(B)(1 - B)^d Y_t = \theta_q(B) a_t$$

ARIMA(p, d, q)建模步驟

1. 繪製時間序列圖並進行初步分析：

- 觀察是否存在異常值或顯著的季節性和趨勢成分。
- 如果方差隨時間顯著波動，可以考慮使用Box-Cox變換來穩定方差。

2. 檢測和確保時間序列的平穩性：

- 通過ADF (Augmented Dickey-Fuller Test)檢定或KPSS (Kwiatkowski-Phillips-Schmidt-Shin Test)檢定檢測時間序列的平穩性。
- 如果時間序列不是平穩的，進行一階或更高階的差分，直到序列變得平穩。

3. 識別ARIMA模型的階數：

- 對平穩的時間序列繪製ACF圖和PACF圖。
- 根據ACF和PACF圖，判別可能的ARIMA(p, d, q)模型。

4. 模型配適和選擇：

- 使用選定的 p, d, q 值來配適ARIMA模型。
- 通過AIC或AICc來選擇最佳模型。

5. 模型診斷：

- 繪製模型殘差的ACF圖。
- 進行白噪檢定來評估模型的適合度。
- 如果模型殘差不滿足白噪條件，可以考慮重新選擇模型並進行配適。

6. 模型預測：

- 找到滿足條件的模型，可以使用該模型來進行未來的預測。
- 評估模型的預測效能，計算預測誤差的度量（如MAE, RMSE等）。

7. 結果報告和解釋：

- 對模型的結果進行解釋和報告。
- 提供模型的預測結果和相應的信賴區間。

model	acf	pacf
AR(p)	dies away geometrically	zero after lag p
MA(q)	zero after lag q	dies away geometrically
ARMA(p, q)	dies away geometrically	dies away geometrically



ARIMA 範例:

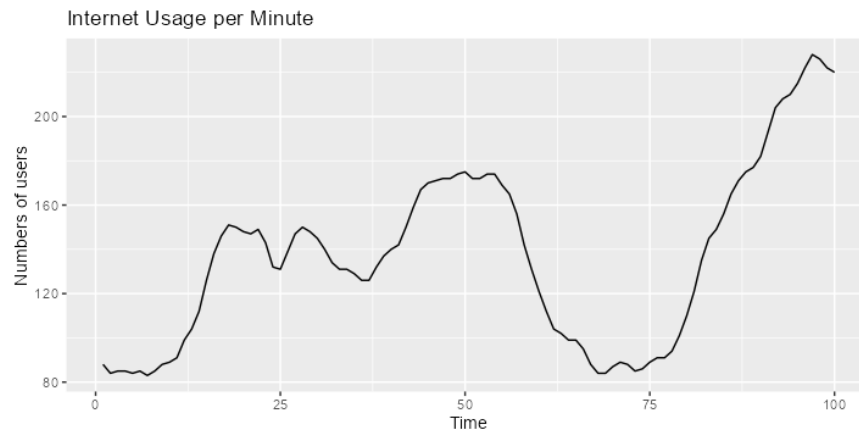
51/61

WWWusage {datasets}: Internet Usage per Minute

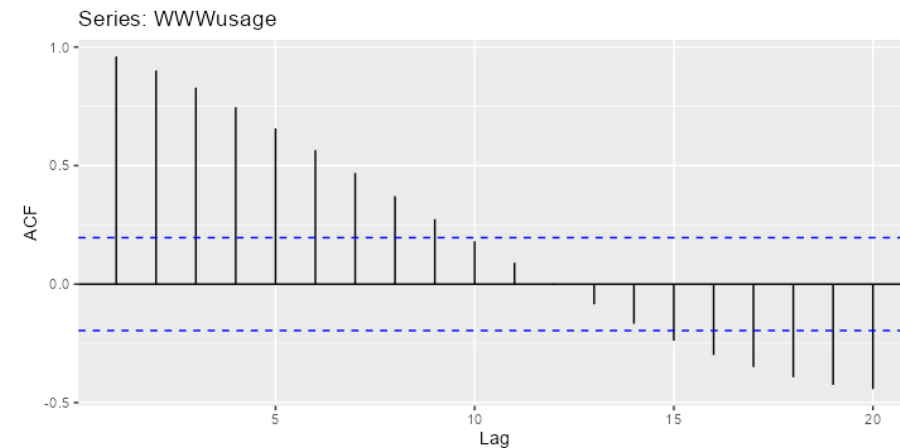
WWWusage {datasets}:

A time series of the numbers of users connected to the Internet through a server every minute.

```
library(forecast)
autoplot(WWWusage) +
  xlab("Time")+
  ylab("Numbers of users") +
  ggtitle("Internet Usage per Minute")
```



```
ggAcf(WWWusage)
```





ARIMA 範例:

52/61

WWWusage {datasets}: Internet Usage per Minute

```
> WWWusage_arma <- Arima(WWWusage, order = c(2, 1, 2))
> WWWusage_arma
Series: WWWusage
ARIMA(2,1,2)
```

Coefficients:

	ar1	ar2	ma1	ma2
	0.0215	0.3407	1.2001	0.4394
s.e.	0.2981	0.2356	0.2799	0.1729

```
sigma^2 = 10.1: log likelihood = -253.68
AIC=517.36 AICc=518.01 BIC=530.34
```

auto.arima: Fit best ARIMA model to univariate time series. Returns best ARIMA model according to either AIC, AICc or BIC value. The function conducts a search over possible model within the order constraints provided.

```
> WWWusage_arma_best <- auto.arima(WWWusage)
> WWWusage_arma_best
Series: WWWusage
ARIMA(1,1,1)
```

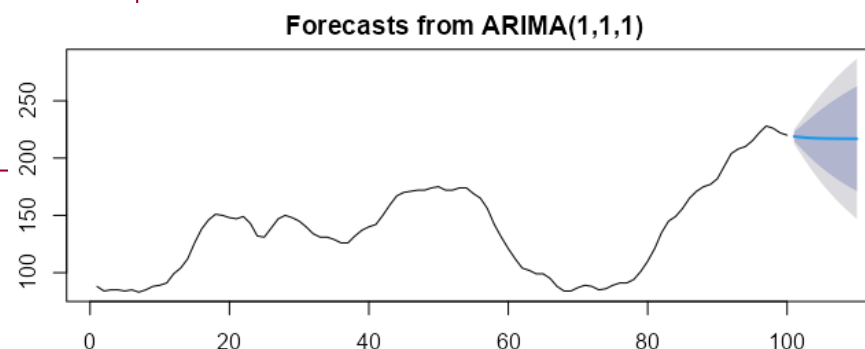
Coefficients:

	ar1	ma1
	0.6504	0.5256
s.e.	0.0842	0.0896

```
sigma^2 = 9.995: log likelihood = -254.15
AIC=514.3 AICc=514.55 BIC=522.08
```

```
> forecast(WWWusage_arma_best, h = 10)
      Point Forecast      Lo 80      Hi 80      Lo 95      Hi 95
101             218.8805 214.8288 222.9322 212.6840 225.0770
...
110             216.8413 171.1478 262.5348 146.9592 286.7235
```

```
> plot(forecast(WWWusage_arma_best, h = 10))
```





The Stationary Vector Autoregression Model , VAR(p)

VAR(1):

$$\begin{aligned}y_{1t} &= a_{11}y_{1t-1} + a_{12}y_{2t-1} + \epsilon_{1t} \\y_{2t} &= a_{21}y_{1t-1} + a_{22}y_{2t-1} + \epsilon_{2t}\end{aligned}$$
$$\begin{pmatrix} y_{1t} \\ y_{2t} \end{pmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{pmatrix} y_{1t-1} \\ y_{2t-1} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \end{pmatrix}$$

VAR(2):

$$\begin{pmatrix} y_{1t} \\ y_{2t} \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \end{pmatrix} + \begin{pmatrix} \pi_{11}^1 & \pi_{12}^1 \\ \pi_{21}^1 & \pi_{22}^1 \end{pmatrix} \begin{pmatrix} y_{1t-1} \\ y_{2t-1} \end{pmatrix} + \begin{pmatrix} \pi_{11}^2 & \pi_{12}^2 \\ \pi_{21}^2 & \pi_{22}^2 \end{pmatrix} \begin{pmatrix} y_{1t-2} \\ y_{2t-2} \end{pmatrix} + \begin{pmatrix} \epsilon_{1t} \\ \epsilon_{2t} \end{pmatrix}$$

民政局-生育獎勵金加倍送規劃說明

附件4—本市100年起發放生育獎勵金·歷年執行情形

年度	預算(元)	執行數(元)	執行率	核發人數(人)	出生人數(人)
104	5億5,000萬	5億6,244萬	102.3%	28,122	28,987
105	5億7,000萬	5億4,940萬	96.4%	27,470	27,992
106	5億7,000萬	4億9,290萬	86.5%	24,645	25,042
107	5億5,000萬	4億5,022萬	81.9%	22,511	22,849
108	5億2,000萬	4億2,666萬	82.05%	21,333	21,468
109	4億8,000萬	3億7,838萬	78.83%	18,919	19,029
110	4億6,000萬	3億3,626萬5,000元	73.1%	16,698	16,695
111	3億8,000萬	3億3,534萬	88.25%	15,094	14,528
112	4億1,000萬				推估18,000

VARMA(1):

$$\mathbf{x}_t = \Gamma \mathbf{u}_t + \phi \mathbf{x}_{t-1} + \mathbf{w}_t$$

$\mathbf{u}_t = (1, t)'$ includes terms to simultaneously fit the constant and trend.

google "ARIMAX (AutoRegressive Integrated Moving Average with Exogenous variables)"

一般工作: 參數估計、信賴區間、假設檢定、模型檢測、模型選擇、預測



R Package **vars**: VAR Modelling

54/61

Canada {vars}: Macroeconomic time series

The original time series are published by the OECD. The sample range is from the 1stQ 1980 until 4thQ 2000.

- **prod**: a measure of labour productivity (勞動生產率的衡量指標)
- **e**: Canadian civil employment (民間就業)
- **U**: Canadian unemployment rate (失業率)
- **rw**: Canadian manufacturing real wage (製造業實際工資)

```
> Canada
      e      prod      rw      U
1980 Q1 929.6105 405.3665 386.1361 7.53
1980 Q2 929.8040 404.6398 388.1358 7.70
1980 Q3 930.3184 403.8149 390.5401 7.47
1980 Q4 931.4277 404.2158 393.9638 7.27
1981 Q1 932.6620 405.0467 396.7647 7.37
1981 Q2 933.5509 404.4167 400.0217 7.13
1981 Q3 933.5315 402.8191 400.7515 7.40
1981 Q4 933.0769 401.9773 405.7335 8.33
1982 Q1 932.1238 402.0897 409.0504 8.83
```

```
1999 Q3 958.7166 415.1678 467.6281 7.53
1999 Q4 959.4881 415.7016 467.7026 6.93
2000 Q1 960.3625 416.8674 469.1348 6.80
2000 Q2 960.7834 417.6104 469.3364 6.70
2000 Q3 961.0290 418.0030 470.0117 6.93
2000 Q4 961.7657 417.2667 469.6472 6.87
```

```
> class(Canada)
[1] "mts" "ts"
> str(Canada)
Time-Series [1:84, 1:4] from 1980 to 2001: 930 930 930 931 933 ...
- attr(*, "dimnames")=List of 2
 ..$ : NULL
 ..$ : chr [1:4] "e" "prod" "rw" "U"
```

VAR {vars}

Estimation of a VAR by utilising OLS per equation.

Usage

```
VAR(y, p = 1, type = c("const", "trend", "both", "none"),
     season = NULL, exogen = NULL, lag.max = NULL,
     ic = c("AIC", "HQ", "SC", "FPE"))
```

- **y**: data item containing the endogenous variables
- **p**: integer for the lag order.
- **type**: type of deterministic regressors to include.
- **season**: inclusion of centered seasonal dummy variables.
- **exogen**: inclusion of exogenous variables.
- **ic**: selects the information criteria.

$$y_t = A_1 y_{t-1} + \dots + A_p y_{t-p} + CD_t + u_t$$



Macroeconomic time series data

```
> var2c_model <- VAR(Canada, p = 2, type = "const")
> var2c_model
```

VAR Estimation Results:

=====

Estimated coefficients for equation e:

=====

Call:

```
e = e.l1 + prod.l1 + rw.l1 + U.l1 + e.l2 + prod.l2 + rw.l2 + U.l2 + const
```

e.l1	prod.l1	rw.l1	U.l1	e.l2
1.637821e+00	1.672717e-01	-6.311863e-02	2.655848e-01	-4.971338e-01
prod.l2	rw.l2	U.l2	const	
-1.016501e-01	3.844492e-03	1.326893e-01	-1.369984e+02	

...

Estimated coefficients for equation U:

=====

Call:

```
U = e.l1 + prod.l1 + rw.l1 + U.l1 + e.l2 + prod.l2 + rw.l2 + U.l2 + const
```

e.l1	prod.l1	rw.l1	U.l1	e.l2
-0.58076382	-0.07811707	0.01866214	0.61893150	0.40981822
prod.l2	rw.l2	U.l2	const	
0.05211668	0.04180115	-0.07116885	149.78056487	

```
> predict(var2c_model, n.ahead = 3, ci = 0.95)
```

```
> predict(var2c_model, n.ahead = 3, ci = 0.95)
```

```
$e
      fcst    lower    upper    CI
[1,] 962.6557 961.9446 963.3668 0.7111044
[2,] 963.6538 962.3422 964.9654 1.3116050
[3,] 964.6932 962.8261 966.5603 1.8670903
```

```
$prod
      fcst    lower    upper    CI
[1,] 417.2623 415.9835 418.5411 1.278808
[2,] 417.7410 415.7854 419.6965 1.955532
[3,] 418.2196 415.7674 420.6717 2.452134
```

```
$rw
      fcst    lower    upper    CI
[1,] 470.2954 468.7660 471.8247 1.529348
[2,] 470.8948 468.8195 472.9701 2.075289
[3,] 471.5360 469.0592 474.0128 2.476757
```

```
$U
      fcst    lower    upper    CI
[1,] 6.428832 5.880708 6.976957 0.5481244
[2,] 5.903919 5.017510 6.790327 0.8864083
[3,] 5.396177 4.219319 6.573035 1.1768580
```

See also: [VAR.etp](#): VAR Modelling: Estimation, Testing, and Prediction



Information criteria and FPE for different VAR(p)

VARselect {vars}: The function returns information criteria and final prediction error for sequential increasing the lag order up to a VAR(p)-process. which are based on the same sample size.

Usage

```
VARselect(y, lag.max = 10, type = c("const", "trend", "both", "none"),  
          season = NULL, exogen = NULL)
```

```
> VARselect(Canada, lag.max = 5, type = "const")
```

```
$selection
```

```
AIC(n)   HQ(n)   SC(n) FPE(n)  
      3       2       2      3
```

```
$criteria
```

	1	2	3	4	5
AIC(n)	-5.817851996	-6.35093701	-6.397756084	-6.145942174	-5.926500201
HQ(n)	-5.577529641	-5.91835677	-5.772917961	-5.328846166	-4.917146309
SC(n)	-5.217991781	-5.27118862	-4.838119523	-4.106417440	-3.407087295
FPE(n)	0.002976003	0.00175206	0.001685528	0.002201523	0.002811116

$$AIC(n) = \ln \det(\tilde{\Sigma}_u(n)) + \frac{2}{T} nK^2 \quad ,$$

$$SC(n) = \ln \det(\tilde{\Sigma}_u(n)) + \frac{\ln(T)}{T} nK^2 \quad ,$$

$$HQ(n) = \ln \det(\tilde{\Sigma}_u(n)) + \frac{2 \ln(\ln(T))}{T} nK^2 \quad ,$$

$$FPE(n) = \left(\frac{T + n^*}{T - n^*} \right)^K \det(\tilde{\Sigma}_u(n)) \quad ,$$

區間型資料之時間序列分析 Interval Time Series Analysis

57/61

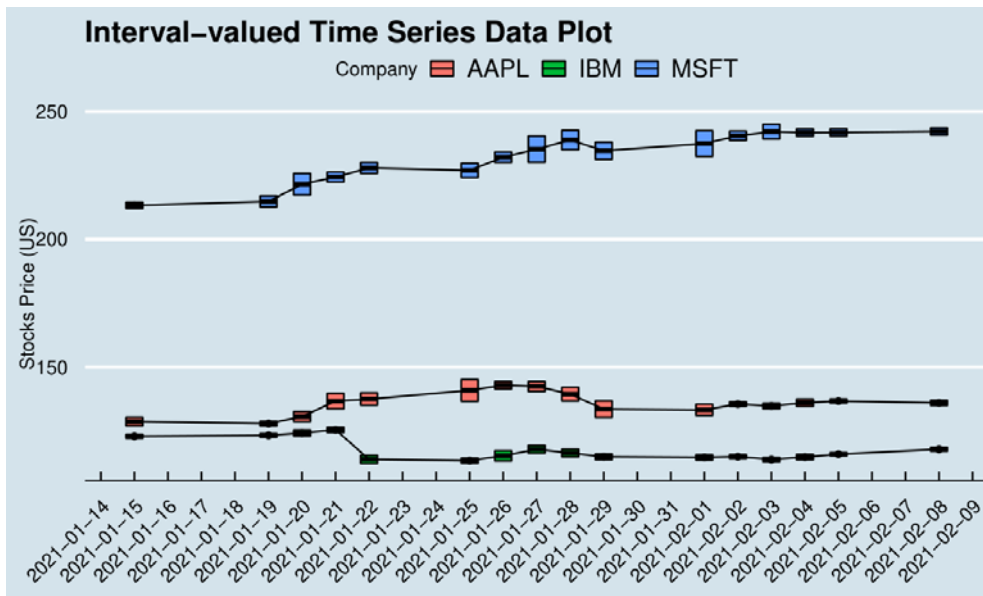
Received: 14 July 2022 | Revised: 29 June 2023 | Accepted: 19 August 2023
DOI: 10.1002/for.3024

RESEARCH ARTICLE

WILEY

Interval time series forecasting: A systematic literature review

Piao Wang¹ | Shahid Hussain Gurmani² | Zhifu Tao^{1,3,4} | Jinpei Liu⁵ | Huayou Chen^{1,4}

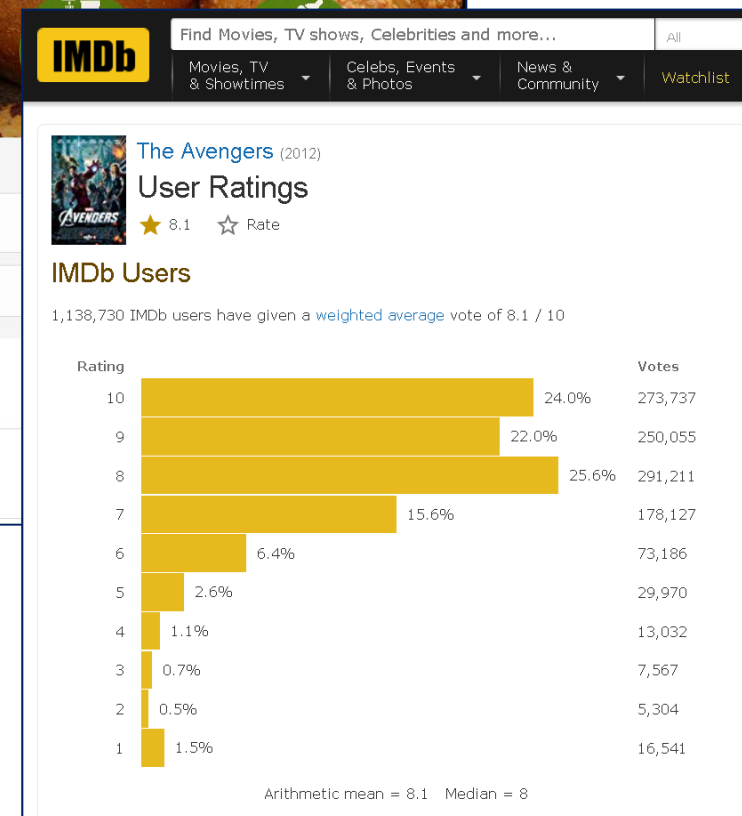
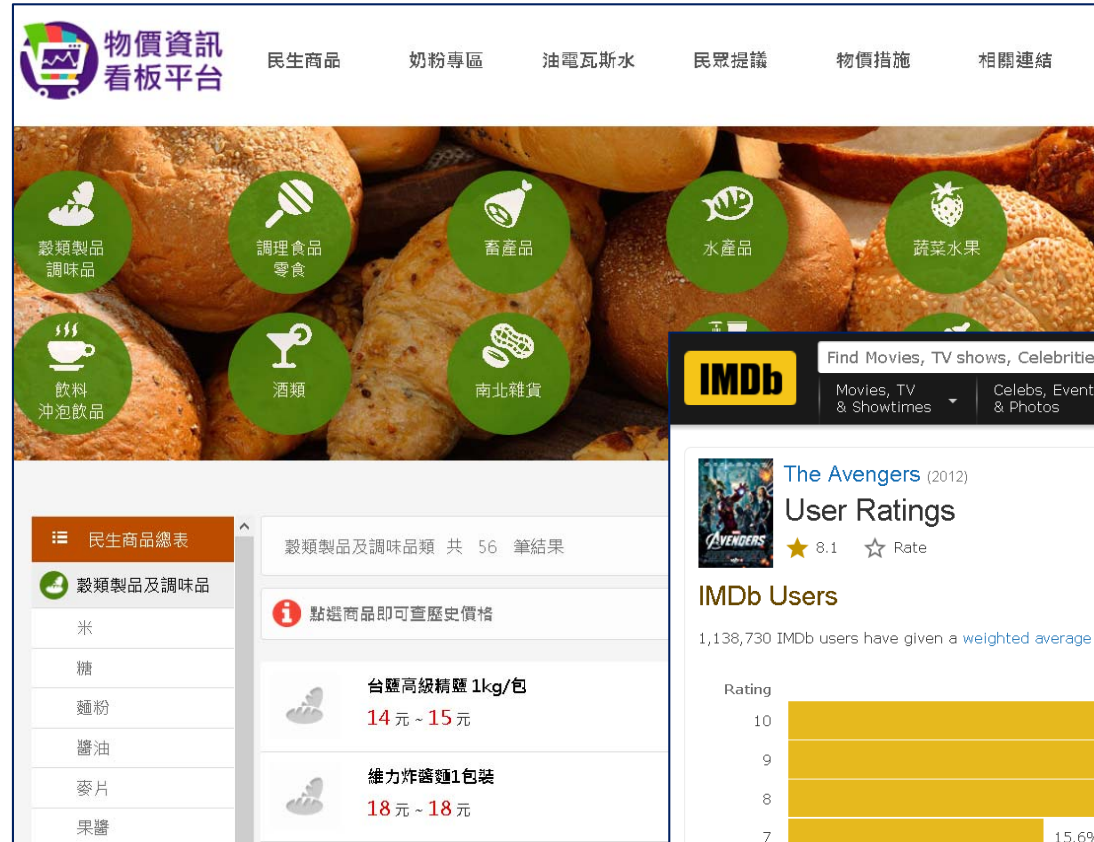


NOTE: No R package is available on CRAN for Interval Time Series Analysis."

- Wang, P., Gurmani, S. H., Tao, Z., Liu, J., & Chen, H. [Interval time series forecasting: A systematic literature review](#). Journal of Forecasting. First published: 20 September 2023.
- Zhang, H., Su, J., Tang, L., & Srivastava, A. (2023). Elastic statistical analysis of interval-valued time series. Journal of Applied Statistics, 50(1), 60-85.
- Lin, L. C., Sung, H., & Lee, S. (2023). Comprehensive interval-valued time series model with application to the S&P 500 index and PM2. 5 level data analysis. Applied Stochastic Models in Business and Industry, 39(2), 198-218.
- Zhou, W., Shan, D., Yang, J., & Lu, W. (2019). The modeling of interval-valued time series: A method based on fuzzy set theory and artificial neural networks. International Journal of Computational Intelligence and Applications, 18(01), 1950002.
- Islamoglu, E., Islamoglu, A., & Bulut, H. (2015). Comparison of forecasting methods for interval-valued time series. International Journal of Statistics and Applications, 5(6), 317-337.
- Maia, A. L. S., de Carvalho, F. D. A., & Ludermir, T. B. (2008). Forecasting models for interval-valued time series. Neurocomputing, 71(16-18), 3344-3352. (#citations: 191)

象徵性資料 (Symbolic Data)

58/61



Prices of commodities platform
National Development Council, TW,
<http://price.nat.gov.tw>

Edwin Diday, 2016, Thinking by classes in data science: the symbolic data analysis paradigm. *WIREs Computational Statistics*, 8:172–205.

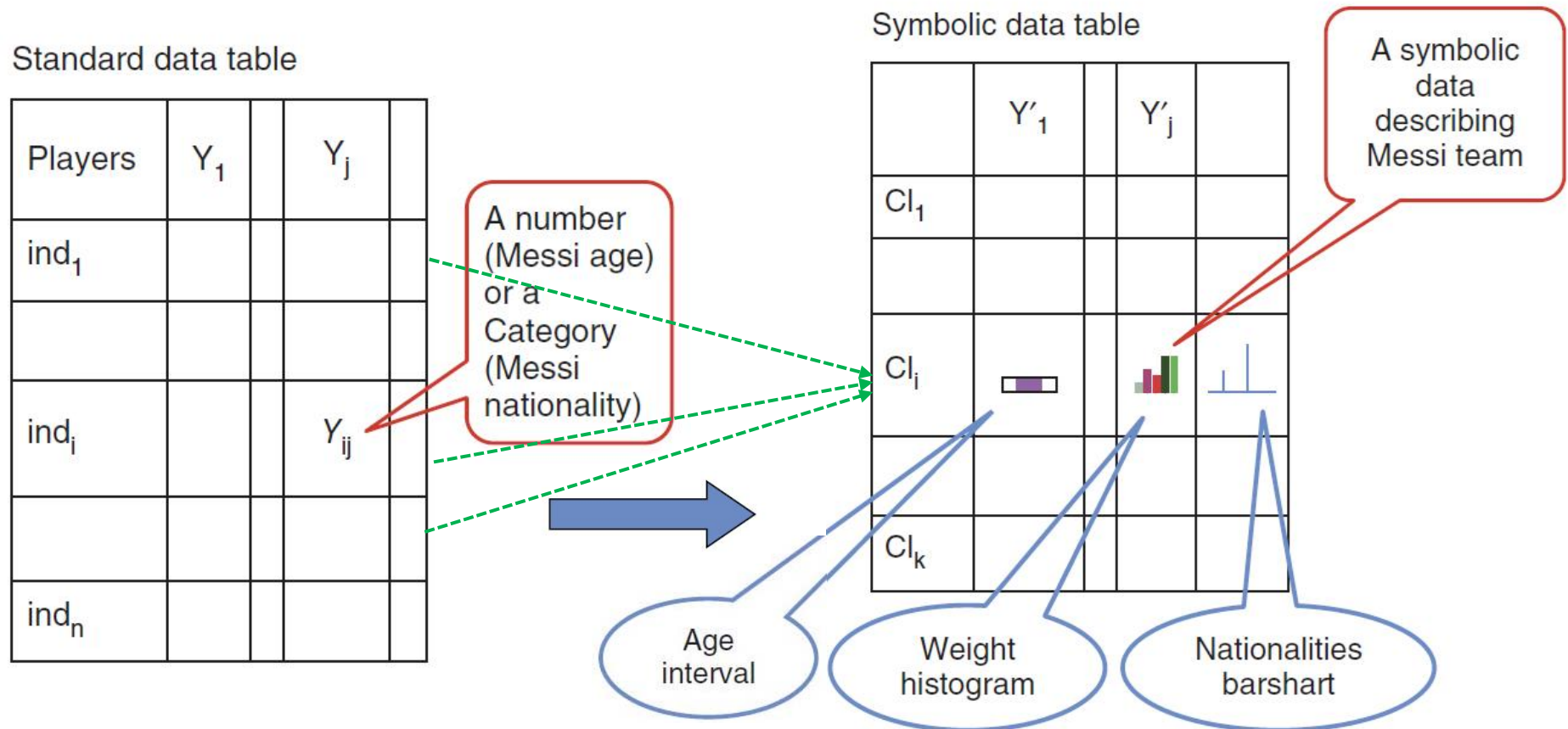


FIGURE 1 | From a standard data table (X, Y) describing a set of individuals X by a set of standard variables Y , to a symbolic data table (X', Y') describing a set of teams X' by a set of symbolic variables Y' .

類別(屬質)變數之時間序列分析

Categorical Time Series Models

60/61



ELSEVIER



Available online at www.sciencedirect.com

ScienceDirect

Stochastic Processes and their Applications 129 (2019) 3446–3462

stochastic
processes
and their
applications

www.elsevier.com/locate/spa

On categorical time series models with covariates

Konstantinos Fokianos^{a,*}, Lionel Truquet^b

^a Lancaster University, Department of Mathematics & Statistics, Fylde College, Lancaster, LA1 4YF, United Kingdom

^b UMR 9194 CNRS CREST, ENSAI, Campus de Ker-Lann, rue Blaise Pascal, BP 37203, 35172 Bruz cedex, France

Received 5 October 2017; received in revised form 27 July 2018; accepted 26 September 2018

Available online 9 October 2018

Abstract

We study the problem of stationarity and ergodicity for autoregressive multinomial logistic time series models which possibly include a latent process and are defined by a GARCH-type recursive equation. We improve considerably upon the existing conditions about stationarity and ergodicity of those models. Proofs are based on theory developed for chains with complete connections. A useful coupling technique is employed for studying ergodicity of infinite order finite-state stochastic processes which generalize finite-state Markov chains. Furthermore, for the case of finite order Markov chains, we discuss ergodicity properties of a model which includes strongly exogenous but not necessarily bounded covariates.

© 2018 Elsevier B.V. All rights reserved.

MSC: primary 62M10; secondary 60G10 ; 60B12

Keywords: Autoregression; Categorical data; Chains with complete connection; Coupling; Covariates; Ergodicity; Markov chains

to be continuous...

tidyquant: Bringing financial and business analysis to the tidyverse

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

<http://www.quantmod.com/>

quantmod

Quantitative Financial Modelling & Trading Framework for R

<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>