

# **STATS 326: Applied Time Series Analysis**

# **STATS 786: Time Series Forecasting for Data Science**



**THE UNIVERSITY OF  
AUCKLAND**  
Te Whare Wananga o Tamaki Makaurau  
**NEW ZEALAND**

# Instructor

## Matt Edwards

- Teaches Weeks 1–12
- Email: matt.edwards@auckland.ac.nz
- Office: 303.312



## Help Available

- Ask Me Anything
- Piazza
- STATS Assistance Hub
- Weekly Tutorials

# Meeting times

## Lecture times (NZ time)

- Wed 10-11 AM, Eng1439, Room 401-439
- Fri 10-11 AM, Eng1439, Room 401-439

## Tutorial times (NZ time)

- Fri 12-1 PM, Building 303S, Room 303S-175
- Fri 2-3 PM, Building 302, Room 302-190

- Lectures and tutorials will be in-person. Lectures will be recorded, but tutorials will not be. Overseas students may attend tutorials via Zoom.
- We will be using tutorial times as office hours.

## Learning objectives

- 1 Use appropriate data visualizations to identify the features present in time series.
- 2 Identify the most appropriate time series models for a given problem.
- 3 Fit commonly used time series models using R.
- 4 Interpret and understand the software output for a given time series model.
- 5 Perform model selection and cross-validation.
- 6 Develop computer skills required to forecast time series data.

## Learning objectives

- 1 Use appropriate data visualizations to identify the features present in time series.
- 2 Identify the most appropriate time series models for a given problem.
- 3 Fit commonly used time series models using R.
- 4 Interpret and understand the software output for a given time series model.
- 5 Perform model selection and cross-validation.
- 6 Develop computer skills required to forecast time series data.

Are you interested in learning time series theory...?

Take STATS 726!

# Topics

- 1 Introduction to tidyverse.
- 2 Dates and times.
- 3 Time series visualisation.
- 4 Time series decomposition.
- 5 Forecaster's toolbox.
- 6 Time series regression.
- 7 ETS models.
- 8 ARIMA models.
- 9 Dynamic regression models (if we have time).

## References

- R. H. Hyndman and G. Athanasopoulos. Forecasting: Principles and practice. 3rd edition.
- R. H. Shumway and D. S. Stoffer. Time series analysis and its applications: With R examples.
- P. J. Brockwell and R. A. Davis. Introduction to time series and forecasting.
- R. J. Hyndman, A. B. Koehler, J. K. Ord and R. D. Snyder. Forecasting with exponential smoothing: The state space approach.

# Assessments

<b>Assessment type</b>	<b>STATS 326</b>	<b>STATS 786</b>
Quizzes	10% ( $5 \times 2\%$ )	5% ( $5 \times 1\%$ )
Assignments	20% ( $4 \times 5\%$ )	15% ( $3 \times 5\%$ )
Mid-term test (1 hour)	20%	10%
Group project	NA	20%
Exam (2 hours)	50%	50%

## Software and main packages



## Early preparation

- Are you familiar with tidyverse packages?
- If you have taken STATS 220 or 369, then you do not have anything to do.
- Otherwise,
  - ▶ Read the first four chapters of “ModernDive”: [moderndive.netlify.com](http://moderndive.netlify.com)

## Main packages

```
# Data manipulation and plotting functions  
library(tidyverse)  
  
# Time series manipulation  
library(tsibble)  
  
# Tidy time series data  
library(tsibbledata)  
  
# Time series graphics and statistics  
library(feasts)  
  
# Forecasting functions  
library(fable)
```

## Main packages

```
# Data manipulation and plotting functions  
library(tidyverse)  
  
# Time series manipulation  
library(tsibble)  
  
# Tidy time series data  
library(tsibbledata)  
  
# Time series graphics and statistics  
library(feasts)  
  
# Forecasting functions  
library(fable)
```

```
# All of the above  
library(fpp3)
```

## Installing required packages

```
install.packages(c(  
  "tidyverse",  
  "fpp3"  
)
```

## Call for class reps

- Benefits:
  - ▶ An important and recognised addition to your resume.
  - ▶ Improve your leadership skill set.
  - ▶ Ability to make significant changes to your education.
  - ▶ At the end of the semester, you will be eligible to receive a class rep certificate provided you have registered with AUSA.
- Responsibilities:
  - ▶ Elicit feedback from your classmates.
  - ▶ Attend department student-staff consultative committee meetings (twice per semester).
  - ▶ Help to resolve issues that may arise in the class.



THE UNIVERSITY OF  
**AUCKLAND**  
Te Whare Wānanga o Tāmaki Makaurau  
NEW ZEALAND

# Time series: Dates and times

# Outline

- 1 Time series
- 2 Construct date/times in R
- 3 Extract components from date/times
- 4 Arithmetic with date/times
- 5 Set components in date/times
- 6 Time spans
- 7 Time zones

# Outline

1 Time series

2 Construct date/times in R

3 Extract components from date/times

4 Arithmetic with date/times

5 Set components in date/times

6 Time spans

7 Time zones

# Types of data

There are three most commonly used data types:

1 Time series data

A sequence of observations collected **over time**.

2 Cross-sectional data

A set of observations collected at a **single point in time**.

e.g. Student grades at the end of the current semester for STATS 786.

3 Panel (or longitudinal) data

**A combination** of time and cross-sectional data.

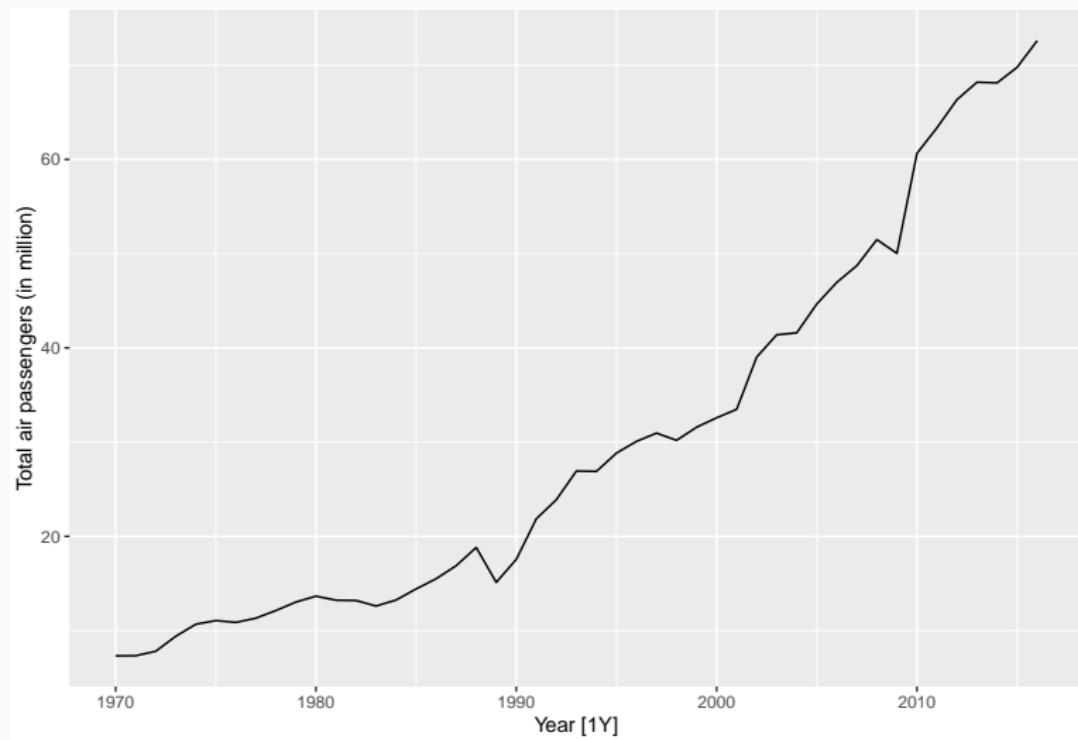
e.g. HIV patients may be followed over time and monthly measures such as CD4 counts, or viral load are collected to measure immunity status and disease burden.

## A few examples of time series data

- **Annual** student enrollments for the University of Auckland.
- **Quarterly** sales of a tableware manufacturing company.
- **Monthly** domestic tourism demand in Australia.
- **Daily** rainfall.
- **Hourly** total customers at McDonald's.
- Number of calls received to a call-center in **every 5 minutes**.

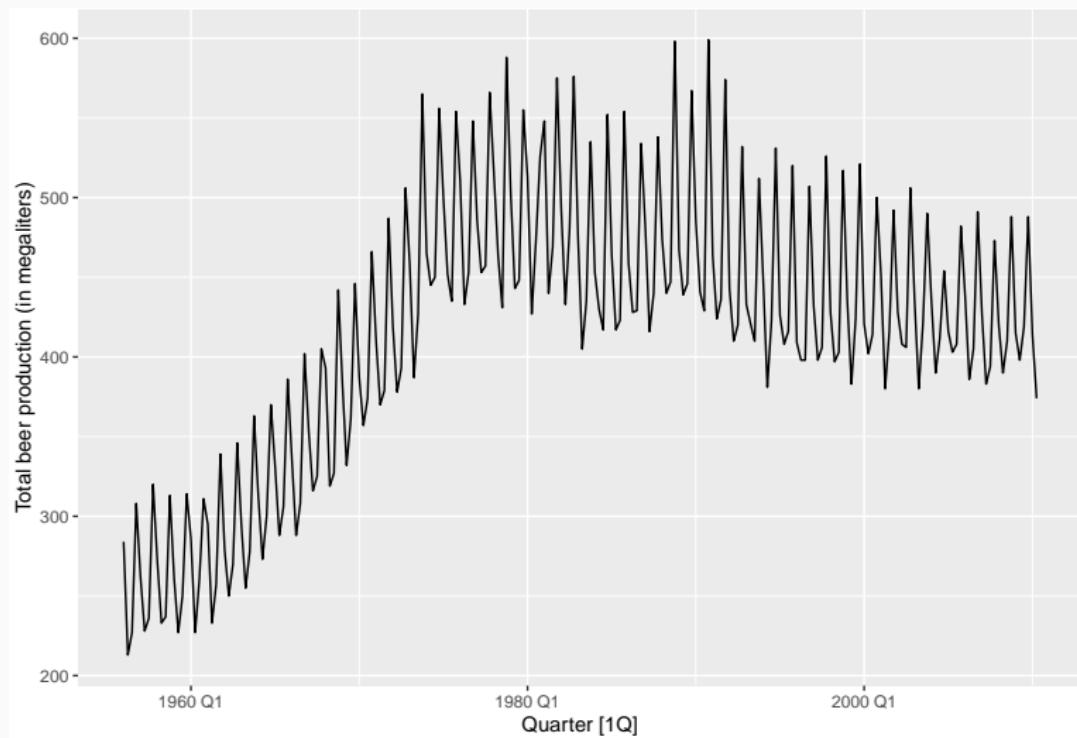
# Annual air passengers in Australia

From 1970 – 2016.



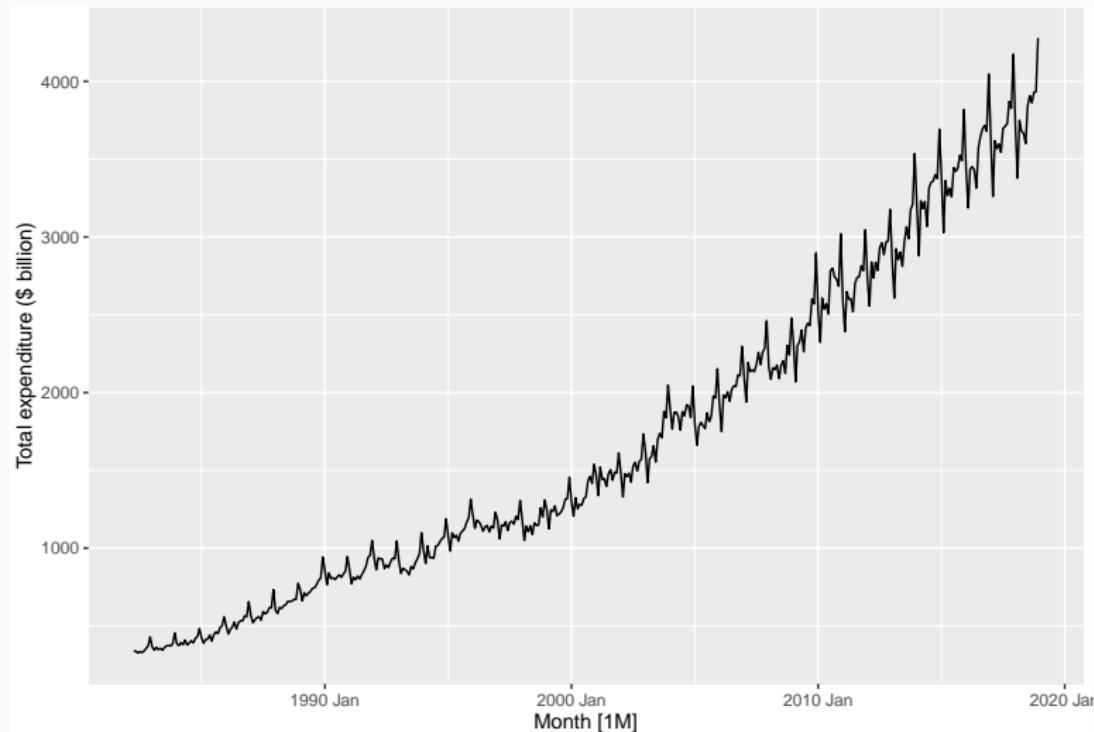
# Quarterly beer production in Australia

From Q1 1956 – Q2 2010.



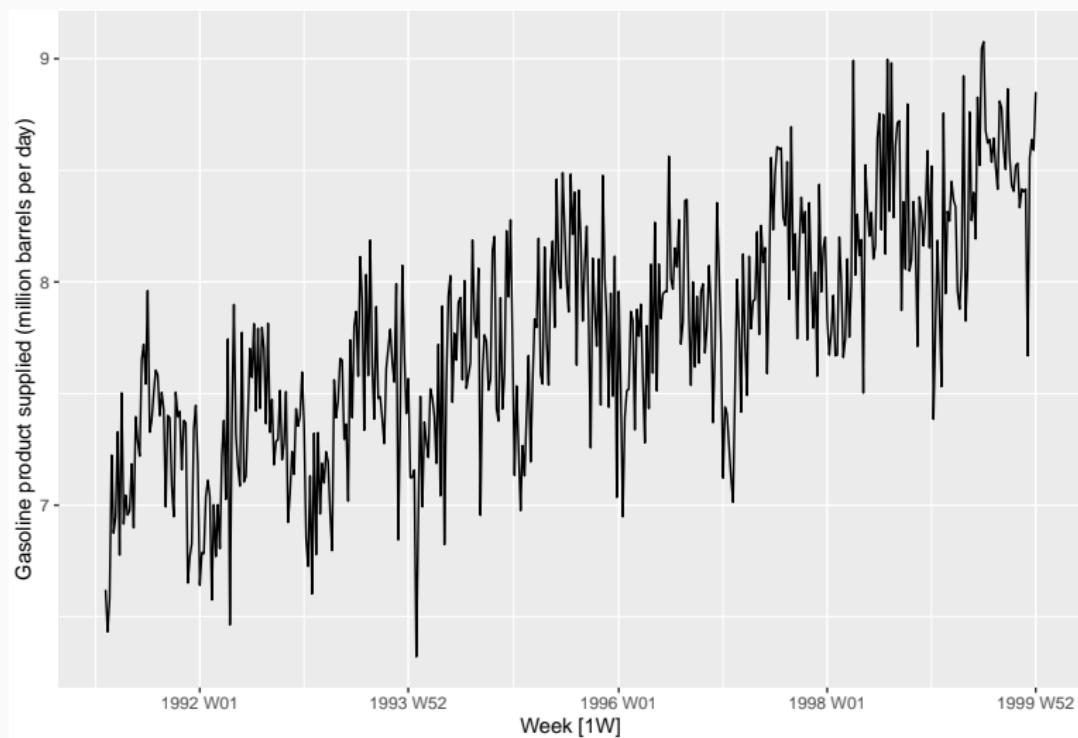
# Monthly expenditure on eating out

From Apr 1982 – Dec 2018.



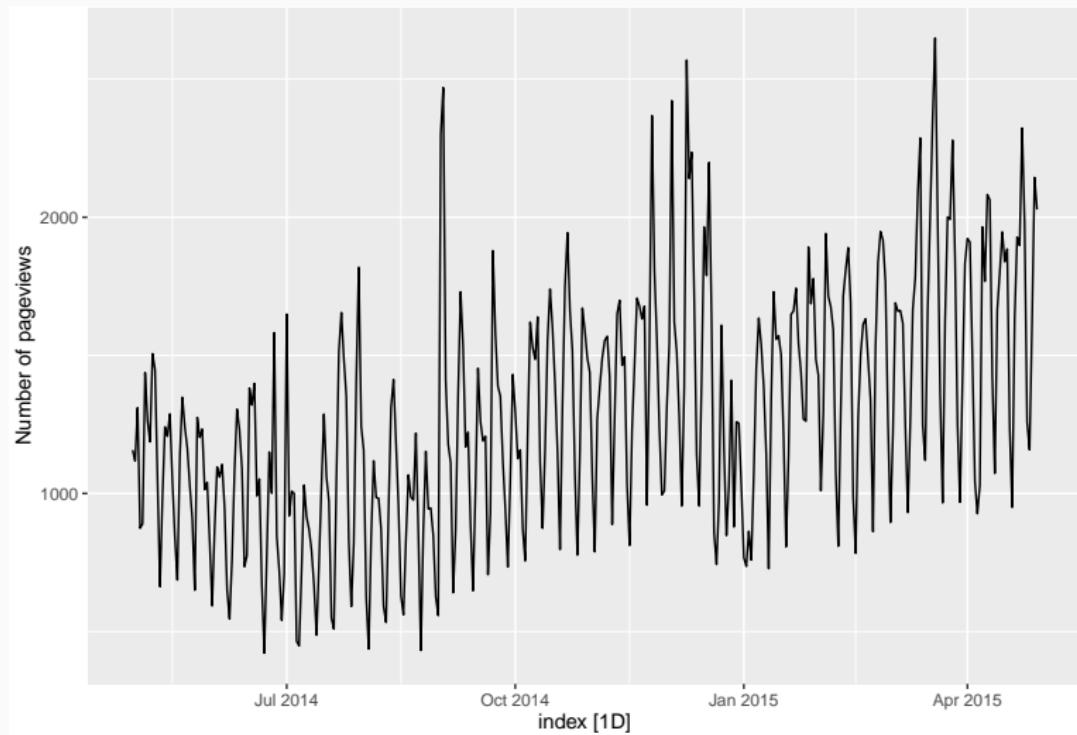
# Weekly gasoline product supplied in USA

From 2 Feb 1991 – 31 Dec 1999.



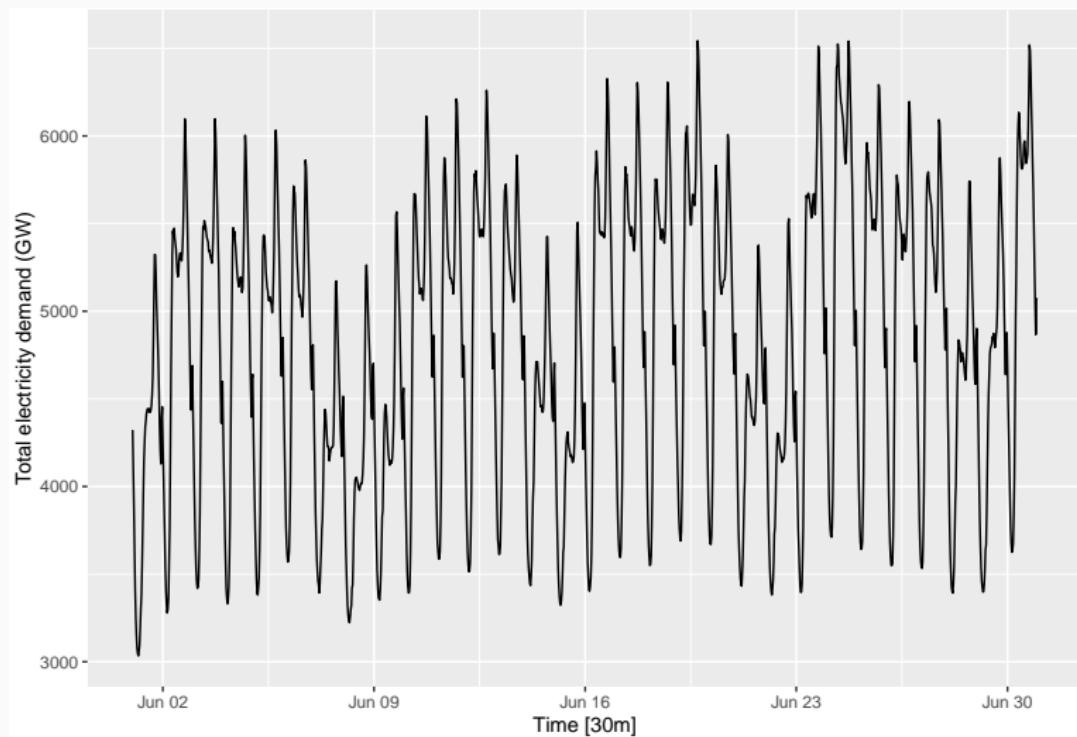
# Daily pageviews for Hyndsite blog

From 30 Apr 2014 – 29 Apr 2015.



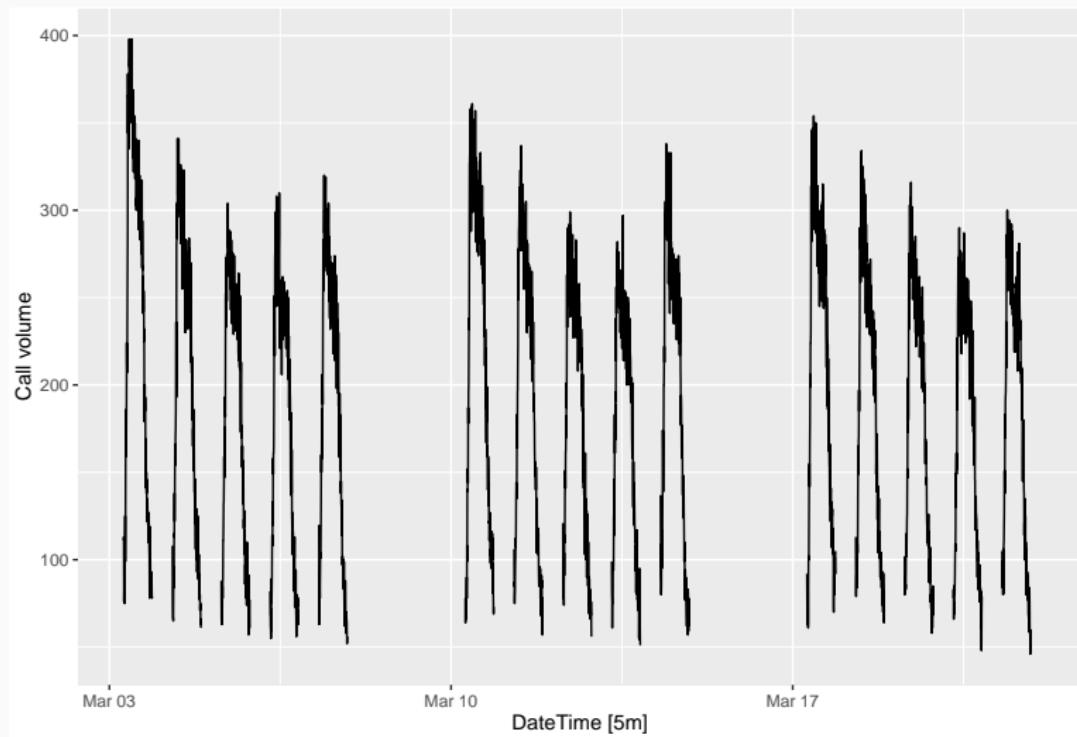
# Half-hourly electricity demand in Victoria

From 1 – 30 Jun 2014.



## Five-minute call volume on weekdays

Between 7:00am and 9:05pm, from 3 Mar 2003 to 21 Mar 2003.



# Outline

1 Time series

2 Construct date/times in R

3 Extract components from date/times

4 Arithmetic with date/times

5 Set components in date/times

6 Time spans

7 Time zones

## Dates and times

- We use dates and times all the time in our regular life and they don't look very complicated.
- However, when we learn more about dates and times it becomes more and more complicated.
  - ▶ Not every year has 365 days.
  - ▶ Not every day has 24 hours.
  - ▶ Not every minute has 60 seconds.
- In this course, we won't discuss every thing about dates and times, but will provide a set of necessary tools to handle common time series analysis challenges.

- We use  .

## Create date/times in R

There are three types of date/time data that refer to a time instant:

- A **date**: tibbles print as <date>.
- A **time** within a day: tibbles print as <time>.
- A **date-time** (date + time): tibbles print this as <dttm>.
- We will discuss only dates and date-times.
- If you need to handle times try using **hms** package.

## Create dates from a string

```
# identify the order in which  
# year (y), month (m) and day (d) appear  
ymd("2021-02-03")
```

```
## [1] "2021-02-03"
```

```
dmy("03-02-2021")
```

```
## [1] "2021-02-03"
```

```
mdy("Feb 3rd, 2021")
```

```
## [1] "2021-02-03"
```

```
# accepts unquoted numbers  
ymd(20210203)
```

```
## [1] "2021-02-03"
```

## Create date-time from a string

```
# add an underscore and one or more of  
# "h" (hours), "m" (mintues) and "s" (seconds)  
ymd_hms("2021-02-03 13:42:05")
```

```
## [1] "2021-02-03 13:42:05 UTC"
```

```
mdy_hm("02-03-2021 13:42")
```

```
## [1] "2021-02-03 13:42:00 UTC"
```

```
# You can also provide a time zone  
tzzone <- "Pacific/Auckland"  
ymd_hms("2021-02-03 13:42:05", tz = tzzone)
```

```
## [1] "2021-02-03 13:42:05 NZDT"
```

```
mdy("02-03-2021", tz = tzzone)
```

```
## [1] "2021-02-03 NZDT"
```

## Create date/time from individual components

- Suitable when the individual components of date/time are spread across multiple columns

```
make_date(year = 2021, month = 2, day = 3)
```

```
## [1] "2021-02-03"
```

```
make_datetime(year = 2021, month = 2,
              day = 3, hour = 13,
              min = 42, sec = 5,
              tz = tzone)
```

```
## [1] "2021-02-03 13:42:05 NZDT"
```

## From other types

- If you want to switch between a date-time and a date then use `as_datetime()` and `as_date()`.

```
today() # current date
```

```
## [1] "2022-02-26"
```

```
now() # current date-time
```

```
## [1] "2022-02-26 21:19:54 NZDT"
```

```
as_datetime(today())
```

```
## [1] "2022-02-26 UTC"
```

```
as_date(now())
```

```
## [1] "2022-02-26"
```

## From other types

- Sometimes we get date/time as a numeric offset from the “Unix Epoch”, 1970-01-01.
- If the offset is in seconds use `as_datetime()`.
- If the offset is in days use `as_date()`.

```
# 51 years since 1970, 13 leap years until 2021
# and another 33 days to 3rd Feb
as_date(365 * 51 + 13 + 33)
```

```
## [1] "2021-02-03"
```

```
as_datetime(60 * 60 * 5)
```

```
## [1] "1970-01-01 05:00:00 UTC"
```

```
as_datetime(60 * 60 * 5, tz = tzone) # NZ is 12 hours ahead
```

```
## [1] "1970-01-01 17:00:00 NZST"
```

# Outline

- 1 Time series
- 2 Construct date/times in R
- 3 Extract components from date/times
- 4 Arithmetic with date/times
- 5 Set components in date/times
- 6 Time spans
- 7 Time zones

## Getting components from a date/time

```
datetime <- ymd_hms("2021-02-03 13:42:05")
```

```
year(datetime)
```

```
## [1] 2021
```

```
month(datetime)
```

```
## [1] 2
```

```
mday(datetime) # day of the month
```

```
## [1] 3
```

```
yday(datetime) # day of the year
```

```
## [1] 34
```

## Getting components from a date/time

- `wday()` and `month()` support two more arguments: `label` and `abbr`.

```
wday(datetime) # day of the week
```

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

```
month(datetime, label = TRUE)
```

```
## [1] Feb
```

```
## 12 Levels: Jan < Feb < Mar < Apr < May < Jun < Jul < Aug < Sep < ... < Dec
```

```
wday(datetime, label = TRUE, abbr = FALSE)
```

```
## [1] Wednesday
```

```
## 7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

- We can use `hour()`, `minute()` and `second()` functions to extract the hour, minute and second, respectively.

# Outline

- 1 Time series
- 2 Construct date/times in R
- 3 Extract components from date/times
- 4 Arithmetic with date/times
- 5 Set components in date/times
- 6 Time spans
- 7 Time zones

## Rounding

- `round_date()` takes a date-time object and time unit and rounds it to the nearest value of the specified time unit.
- For date-times which are halfway between two consecutive units are rounded up.
- We can round to the nearest unit or a multiple of a unit.
- Meaningful specifications of unit in English are supported: secs, min, mins, 3 minutes.

## round\_date() in R

```
datetime <- ymd_hms("2021-02-03 12:05:59.23", tz = tzone)
round_date(datetime, ".5s")
```

```
## [1] "2021-02-03 12:05:59 NZDT"
```

```
round_date(datetime, "sec")
```

```
## [1] "2021-02-03 12:05:59 NZDT"
```

```
round_date(datetime, "minute")
```

```
## [1] "2021-02-03 12:06:00 NZDT"
```

```
round_date(datetime, "5 mins")
```

```
## [1] "2021-02-03 12:05:00 NZDT"
```

- The unit can be "hour", "day", "week", "month", "season", "bimonth", "quarter", "halfyear", "year" etc.

## **floor\_date() and ceiling\_date() in R**

- `floor_date()` takes a date-time object and rounds it down to the nearest boundary of the specified time unit.
- `ceiling_date()` takes a date-time object and rounds it up to the nearest boundary of the specified time unit.

```
floor_date(datetime, "season")
```

```
## [1] "2020-12-01 NZDT"
```

```
ceiling_date(datetime, "season")
```

```
## [1] "2021-03-01 NZDT"
```

# Outline

- 1 Time series
- 2 Construct date/times in R
- 3 Extract components from date/times
- 4 Arithmetic with date/times
- 5 Set components in date/times
- 6 Time spans
- 7 Time zones

## Setting components

```
datetime <- ymd_hms("2021-02-03 12:05:59.23", tz = tzone)
year(datetime) <- 2000
datetime
```

```
## [1] "2000-02-03 12:05:59 NZDT"
```

```
month(datetime) <- 12
datetime
```

```
## [1] "2000-12-03 12:05:59 NZDT"
```

- We can also use `update()` to create a new date-time object.

```
update(datetime, year = 2021, month = 12, mday = 10)
```

```
## [1] "2021-12-10 12:05:59 NZDT"
```

- If the values are too large, they will roll over.

```
update(datetime, mday = 30)
```

# Outline

- 1 Time series
- 2 Construct date/times in R
- 3 Extract components from date/times
- 4 Arithmetic with date/times
- 5 Set components in date/times
- 6 Time spans
- 7 Time zones

## Time spans

There are three important classes that represent time spans:

- **durations**: represent an exact number of seconds.
- **periods**: represent human expressive units such as weeks, months, years.
- **intervals**: represent a starting and ending point.

## Durations

- Durations always record the span in seconds.
- When the duration is large, it displays an estimate in larger units, but the underlying object is always recorded in seconds.
- For display and creation purposes, units are converted to seconds using their common lengths in seconds:  
 $1 \text{ minute} = 60\text{s}$ ,  $1 \text{ hour} = 3600\text{s}$ ,  $1 \text{ day} = 86400\text{s}$ ,  $1 \text{ week} = 604800\text{s}$ .
- The duration of month and year assume that each year consists of 365.25 days.

# Constructors of duration in R

```
dseconds(10)
```

```
## [1] "10s"
```

```
dminutes(10)
```

```
## [1] "600s (~10 minutes)"
```

```
dhours(2)
```

```
## [1] "7200s (~2 hours)"
```

```
ddays(1:2)
```

```
## [1] "86400s (~1 days)"  "172800s (~2 days)"
```

```
dweeks(3)
```

```
## [1] "1814400s (~3 weeks)"
```

## Constructors of duration in R

```
dyears(1)
```

```
## [1] "31557600s (~1 years)"
```

```
ddays(365.25)
```

```
## [1] "31557600s (~1 years)"
```

- We can add, subtract and multiply durations.

```
2 * dyears(1)
```

```
## [1] "63115200s (~2 years)"
```

```
dyears(1) + dweeks(2) + dhours(15)
```

```
## [1] "32821200s (~1.04 years)"
```

```
tomorrow <- today() + ddays(1)  
last_year <- today() - dyears(1)
```

## Constructors of duration in R

```
(datetime <- ymd_hms("2020-09-26 10:00:10", tz = tzone))
```

```
## [1] "2020-09-26 10:00:10 NZST"
```

```
datetime + ddays(1)
```

```
## [1] "2020-09-27 11:00:10 NZDT"
```

- What has happened here?

## Periods

- Periods are time spans but they don't have a fixed length in seconds.
- They work with "human" times such as years, months, weeks, days, hours, minutes and seconds.
- They don't consider leap years, daylight savings.
- For example, `months(2)` always has the length of 2 months even though the length of two months vary depending on when the time period begins.

## Constructors of periods in R

```
seconds(10)
```

```
## [1] "10S"
```

```
minutes(10)
```

```
## [1] "10M 0S"
```

```
hours(2)
```

```
## [1] "2H 0M 0S"
```

```
days(1:2)
```

```
## [1] "1d 0H 0M 0S" "2d 0H 0M 0S"
```

```
weeks(3)
```

```
## [1] "21d 0H 0M 0S"
```

## Constructors of periods in R

```
years(1)
```

```
## [1] "1y 0m 0d 0H 0M 0S"
```

- We can add and multiply periods.

```
5 * (months(3) + days(2))
```

```
## [1] "15m 10d 0H 0M 0S"
```

```
days(5) + hours(2) + minutes(10)
```

```
## [1] "5d 2H 10M 0S"
```

## Periods added to time instants

- We can add them to dates. Unlike durations, periods most likely do what you expect.

```
# Daylight savings time  
(datetime <- ymd_hms("2020-09-26 10:00:10", tz = tzone))  
  
## [1] "2020-09-26 10:00:10 NZST"
```

```
datetime + ddays(1)  
  
## [1] "2020-09-27 11:00:10 NZDT"
```

```
datetime + days(1)  
  
## [1] "2020-09-27 10:00:10 NZDT"
```

## Periods added to time instants

```
# A leap year  
ymd("2020-01-01", tz = tzone) + dyears(1)
```

```
## [1] "2020-12-31 06:00:00 NZDT"
```

```
ymd("2020-01-01", tz = tzone) + years(1)
```

```
## [1] "2021-01-01 NZDT"
```

## Intervals

- An interval is a duration with a starting point that makes it precise so we can determine exactly how long it is.
- `interval()` creates an interval object with the specified start and end dates.
- We can also define an interval using the `%--%` operator.

```
# Daylight saving
start <- ymd_hms("2021-04-03 13:42:40", tz = tzone)
end <- ymd_hms("2021-04-04 13:42:40", tz = tzone)
(time_interval <- start %--% end)

## [1] 2021-04-03 13:42:40 NZDT--2021-04-04 13:42:40 NZST
```

## Example

- How many weeks are there between 2021-01-01 and 2021-10-11?

```
start <- ymd("2021-01-01")
end <- ymd("2021-10-11")
time_interval <- start %--% end
time_interval / dweeks(1)
```

```
## [1] 40.42857
```

```
time_interval %/% dweeks(1)
```

```
## [1] 40
```

# Outline

- 1 Time series
- 2 Construct date/times in R
- 3 Extract components from date/times
- 4 Arithmetic with date/times
- 5 Set components in date/times
- 6 Time spans
- 7 Time zones

## Time zones

- R uses the international standard IANA (Internet Assigned Numbers Authority) time zones.
- They use a consistent naming scheme, usually in the form <continent>/<city>.
- There are exceptions to this.
- For example, "America/New\_York", "Pacific/Auckland".

```
# Find out what R knows as your current time zone  
# If R doesn't know you will get an NA  
Sys.timezone()
```

```
## [1] "Pacific/Auckland"
```

```
head(OlsonNames()) # gives a complete list of all time zones
```

## [1] "Africa/Abidjan"	"Africa/Accra"	"Africa/Addis_Ababa"
## [4] "Africa/Algiers"	"Africa/Asmara"	"Africa/Asmera"

## Time zones

- In R, the time zone is an attribute of the date/time that only controls printing.

```
(time_nyk <- ymd_hms("2021-02-03 12:00:00", tz = "America/New_York"))
```

```
## [1] "2021-02-03 12:00:00 EST"
```

```
(time_uk <- ymd_hms("2021-02-03 17:00:00", tz = "Europe/London"))
```

```
## [1] "2021-02-03 17:00:00 GMT"
```

```
(time_aku <- ymd_hms("2021-02-04 06:00:00", tz = "Pacific/Auckland"))
```

```
## [1] "2021-02-04 06:00:00 NZDT"
```

```
# # All of them represent the same time instant  
time_nyk - time_uk # time_nyk - time_aku
```

```
## Time difference of 0 secs
```

## Time zones

- Unless otherwise specified, lubridate package always uses UTC (Coordinated Universal Time).
- Operations which combine date-times such as `c()`, will display the time zone based on the first element of the vector.

```
(times <- c(time_nyk, time_uk, time_aku))
```

```
## [1] "2021-02-03 12:00:00 EST" "2021-02-03 12:00:00 EST"  
## [3] "2021-02-03 12:00:00 EST"
```

## Time zone conversions

- lubridate package provides two functions for time zone conversions.
- `with_tz()` keeps the instant in time the same, and changes how it is displayed.
- This function is useful when the time instant is correct, but we want a more meaningful display.

```
(times_mel <- with_tz(times, tzone = "Australia/Melbourne"))
```

```
## [1] "2021-02-04 04:00:00 AEDT" "2021-02-04 04:00:00 AEDT"  
## [3] "2021-02-04 04:00:00 AEDT"
```

```
times - times_mel
```

```
## Time differences in secs  
## [1] 0 0 0
```

## Time zone conversions

- `force_tz()` changes the underlying instant in time.
- This function is useful when you know that the label of the time zone is wrong, and you need to fix it.

```
(times_mel <- force_tz(times, tzone = "Australia/Melbourne"))
```

```
## [1] "2021-02-03 12:00:00 AEDT" "2021-02-03 12:00:00 AEDT"  
## [3] "2021-02-03 12:00:00 AEDT"
```

```
times = times_mel
```

```
## Time differences in hours  
## [1] 16 16 16
```

## References

- 1 Wickham, H., and Grolemund, G. (2017). R for Data Science.  
<https://r4ds.had.co.nz/>



THE UNIVERSITY OF  
**AUCKLAND**  
Te Whare Wānanga o Tāmaki Makaurau  
NEW ZEALAND

# Time series visualization

# Outline

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Time series patterns
- 5 Lag plots and autocorrelation
- 6 White noise

# Outline

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Time series patterns
- 5 Lag plots and autocorrelation
- 6 White noise

## Time series graphics

- The first step in any data analysis task is to visualize the data.
- Various types of graphs enable us to observe many features of the data such as trends, unusual observations, structural breaks, and relationships between variables.
- The features we observe from these plots should be incorporated as much as possible into the forecasting methods to be used.
- Before we produce any plots, we need to prepare and set up time series data in R.



- We use .

## tsibble package

- The **tsibble** package extends the tidyverse to handle temporal data.
- It is built on top of the tibble, and is a data- and model-oriented object.
- In comparison to conventional time series objects in R such as `ts`, `zoo` and `xts`, the tsibble preserves time indices as an essential data column and allows heterogeneous data structures possible.
- It also allows storage and manipulation of multiple time series.
- A tsibble contains three main components:
  - ▶ **Key** : comprised of single or multiple variables which uniquely identify each series
  - ▶ **Index** : time information about the observations
  - ▶ **Measured variable(s)** : numbers of interest

## tsibble objects

```
global_economy
```

```
## # A tsibble: 15,150 x 6 [1Y]
## # Key:     Country [263]
## #       Year Country          GDP Imports Exports Population
## #   <dbl> <fct>      <dbl>    <dbl>    <dbl>      <dbl>
## 1 1960 Afghanistan 537777811.    7.02     4.13    8996351
## 2 1961 Afghanistan 548888896.    8.10     4.45    9166764
## 3 1962 Afghanistan 546666678.    9.35     4.88    9345868
## 4 1963 Afghanistan 751111191.   16.9     9.17    9533954
## 5 1964 Afghanistan 800000044.   18.1     8.89    9731361
## 6 1965 Afghanistan 1006666638.   21.4    11.3    9938414
## 7 1966 Afghanistan 1399999967.   18.6     8.57   10152331
## 8 1967 Afghanistan 1673333418.   14.2     6.77   10372630
## 9 1968 Afghanistan 1373333367.   15.2     8.90   10604346
## 10 1969 Afghanistan 1408888922.   15.0    10.1   10854428
## # ... with 15,140 more rows
```

## tsibble objects

```
global_economy
```

```
## # A tsibble: 15,150 x 6 [1Y]
## # Key:     Country [263]
## # 
## #   Year    Country      GDP Imports Exports Population
## #   Index    <fct>     <dbl>   <dbl>   <dbl>     <dbl>
## # 1 1960 Afghanistan 537777811.    7.02    4.13 8996351
## # 2 1961 Afghanistan 548888896.    8.10    4.45 9166764
## # 3 1962 Afghanistan 546666678.    9.35    4.88 9345868
## # 4 1963 Afghanistan 751111191.   16.9    9.17 9533954
## # 5 1964 Afghanistan 800000044.   18.1    8.89 9731361
## # 6 1965 Afghanistan 1006666638.   21.4   11.3 9938414
## # 7 1966 Afghanistan 1399999967.   18.6    8.57 10152331
## # 8 1967 Afghanistan 1673333418.   14.2    6.77 10372630
## # 9 1968 Afghanistan 1373333367.   15.2    8.90 10604346
## # 10 1969 Afghanistan 1408888922.   15.0   10.1 10854428
## # ... with 15,140 more rows
```

## tsibble objects

```
global_economy
```

```
## # A tsibble: 15,150 x 6 [1Y]
## # Key:     Country [263]
## #       Year Country          GDP Imports Exports Population
## #   Index   Key      <dbl>    <dbl>    <dbl>    <dbl>
## 1 1960 Afghanistan 537777811.    7.02     4.13  8996351
## 2 1961 Afghanistan 548888896.    8.10     4.45  9166764
## 3 1962 Afghanistan 546666678.    9.35     4.88  9345868
## 4 1963 Afghanistan 751111191.   16.9     9.17  9533954
## 5 1964 Afghanistan 800000044.   18.1     8.89  9731361
## 6 1965 Afghanistan 1006666638.   21.4    11.3   9938414
## 7 1966 Afghanistan 1399999967.   18.6     8.57  10152331
## 8 1967 Afghanistan 1673333418.   14.2     6.77  10372630
## 9 1968 Afghanistan 1373333367.   15.2     8.90  10604346
## 10 1969 Afghanistan 1408888922.   15.0    10.1   10854428
## # ... with 15,140 more rows
```

## tsibble objects

```
global_economy
```

```
## # A tsibble: 15,150 x 6 [1Y]
## # Key:     Country [263]
## #       Year Country          GDP Imports Exports Population
## #   Index   Key
## #   1 1960 Afghanistan 537777811.    7.02    4.13 8996351
## #   2 1961 Afghanistan 548888896.    8.10    4.45 9166764
## #   3 1962 Afghanistan 546666678.    9.35    4.88 9345868
## #   4 1963 Afghanistan 751111191.   16.9     9.17 9533954
## #   5 1964 Afghanistan 800000044.   18.1     8.89 9731361
## #   6 1965 Afghanistan 1006666638.   21.4    11.3 9938414
## #   7 1966 Afghanistan 1399999967.   18.6     8.57 10152331
## #   8 1967 Afghanistan 1673333418.   14.2     6.77 10372630
## #   9 1968 Afghanistan 1373333367.   15.2     8.90 10604346
## #  10 1969 Afghanistan 1408888922.   15.0    10.1 10854428
## # ... with 15,140 more rows
```

# tsibble objects

tourism

```
## # A tsibble: 24,320 x 5 [1Q]
## # Key:      Region, State, Purpose [304]
##   Quarter Region  State Purpose  Trips
##   <qtr>  <chr>    <chr>  <chr>    <dbl>
## 1 1998   Q1 Adelaide SA  Business  135.
## 2 1998   Q2 Adelaide SA  Business  110.
## 3 1998   Q3 Adelaide SA  Business  166.
## 4 1998   Q4 Adelaide SA  Business  127.
## 5 1999   Q1 Adelaide SA  Business  137.
## 6 1999   Q2 Adelaide SA  Business  200.
## 7 1999   Q3 Adelaide SA  Business  169.
## 8 1999   Q4 Adelaide SA  Business  134.
## 9 2000   Q1 Adelaide SA  Business  154.
## 10 2000  Q2 Adelaide SA  Business  169.
## # ... with 24,310 more rows
```

# tsibble objects

tourism

```
## # A tsibble: 24,320 x 5 [1Q]
## # Key:      Region, State, Purpose [304]
## #   Quarter Region  State Purpose  Trips
## #   Index    <chr>    <chr> <chr>    <dbl>
## # 1 1998 Q1 Adelaide SA  Business  135.
## # 2 1998 Q2 Adelaide SA  Business  110.
## # 3 1998 Q3 Adelaide SA  Business  166.
## # 4 1998 Q4 Adelaide SA  Business  127.
## # 5 1999 Q1 Adelaide SA  Business  137.
## # 6 1999 Q2 Adelaide SA  Business  200.
## # 7 1999 Q3 Adelaide SA  Business  169.
## # 8 1999 Q4 Adelaide SA  Business  134.
## # 9 2000 Q1 Adelaide SA  Business  154.
## # 10 2000 Q2 Adelaide SA  Business  169.
## # ... with 24,310 more rows
```

# tsibble objects

tourism

```
## # A tsibble: 24,320 x 5 [1Q]
## # Key:      Region, State, Purpose [304]
## #   Quarter Region  State Purpose Trips
## #   Index    Keys          <dbl>
## # 1 1998 Q1 Adelaide SA Business 135.
## # 2 1998 Q2 Adelaide SA Business 110.
## # 3 1998 Q3 Adelaide SA Business 166.
## # 4 1998 Q4 Adelaide SA Business 127.
## # 5 1999 Q1 Adelaide SA Business 137.
## # 6 1999 Q2 Adelaide SA Business 200.
## # 7 1999 Q3 Adelaide SA Business 169.
## # 8 1999 Q4 Adelaide SA Business 134.
## # 9 2000 Q1 Adelaide SA Business 154.
## # 10 2000 Q2 Adelaide SA Business 169.
## # ... with 24,310 more rows
```

# tsibble objects

tourism

```
## # A tsibble: 24,320 x 5 [1Q]
## # Key:      Region, State, Purpose [304]
## #   Quarter Region  State Purpose Trips
## #   Index    Keys          Measure
## # 1 1998 Q1 Adelaide SA Business 135.
## # 2 1998 Q2 Adelaide SA Business 110.
## # 3 1998 Q3 Adelaide SA Business 166.
## # 4 1998 Q4 Adelaide SA Business 127.
## # 5 1999 Q1 Adelaide SA Business 137.
## # 6 1999 Q2 Adelaide SA Business 200.
## # 7 1999 Q3 Adelaide SA Business 169.
## # 8 1999 Q4 Adelaide SA Business 134.
## # 9 2000 Q1 Adelaide SA Business 154.
## # 10 2000 Q2 Adelaide SA Business 169.
## # ... with 24,310 more rows
```

# Creating a tsibble object

## 1 Using the `tsibble()` function

### Example

```
mydata <- tsibble(  
  year = 2018:2021,  
  observations = c(10, 45, 36, 89),  
  index = year  
)  
mydata
```

```
## # A tsibble: 4 x 2 [1Y]  
##   year observations  
##   <int>       <dbl>  
## 1 2018         10  
## 2 2019         45  
## 3 2020         36  
## 4 2021         89
```

# Creating a tsibble object

## 1 Using the `tsibble()` function

### Example

```
mydata <- tsibble(  
  year = 2018:2021,  
  observations = c(10, 45, 36, 89),  
  index = year  
)  
  
mydata  
  
## # A tsibble: 4 x 2 [1Y]  
##   year observations  
##   <int>       <dbl>  
## 1 2018         10  
## 2 2019        45  
## 3 2020        36  
## 4 2021        89
```

By default, `key = NULL` and `regular = TRUE`. In this course, we focus only on regular time series.

# The tsibble index

- If the observations have been collected more frequently than once per year then we need to use a time class function on the index.

## Example

```
z

## # A tibble: 4 x 2
##   month    observations
##   <chr>        <dbl>
## 1 2020     Jan      40.0
## 2 2020     Feb      94.3
## 3 2020     Mar      63.7
## 4 2020     Apr      79.7
```

# The tsibble index

```
z %>%  
  mutate(month = yearmonth(month)) %>%  
  as_tsibble(index = month)
```

```
## # A tsibble: 4 x 2 [1M]  
##       month observations  
##     <mth>      <dbl>  
## 1 2020 Jan      40.0  
## 2 2020 Feb      94.3  
## 3 2020 Mar      63.7  
## 4 2020 Apr      79.7
```

## `as_tsibble()` function

- `as_tsibble()` coerce other objects to a tsibble.
- If the object is a tibble or data frame, `as_tsibble()` requires a little more information to declare the index and key variables.
- If the object is `ts` or `mts`, we can convert it to a tsibble without any further information.

## Other useful time class functions

- Depending on the frequency at which we have collected the observations we can use the following time class functions.

Frequency	Function
Annual	<code>start:end</code>
Quarterly	<code>yearquarter()</code>
Monthly	<code>yearmonth()</code>
Weekly	<code>yearweek()</code>
Daily	<code>as_date(), ymd()</code>
Sub-daily	<code>as_datetime()</code>

# Creating a tsibble object

## 2 By importing a file into R

- In practice, mostly the available data reside in databases, MS-Excel or csv files.
- We need to first read this file into R, and then identify the index and key variables.

```
prison <- readr::read_csv("data/prison_population.csv")
prison %>% print(n = 4)
```

```
## # A tibble: 3,072 x 6
##   Date      State Gender Legal    Indigenous Count
##   <date>    <chr> <chr>  <chr>    <chr>        <dbl>
## 1 2005-03-01 ACT   Female Remanded ATSI         0
## 2 2005-03-01 ACT   Female Remanded Non-ATSI     2
## 3 2005-03-01 ACT   Female Sentenced ATSI        0
## 4 2005-03-01 ACT   Female Sentenced Non-ATSI    5
## # ... with 3,068 more rows
```

## Form a `tsibble` from a csv file

```
prison <- prison %>%
  mutate(Quarter = yearquarter(Date)) %>%
  select(-Date)
prison %>% print(n = 5)
```

```
## # A tibble: 3,072 x 6
##   State Gender Legal     Indigenous Count Quarter
##   <chr> <chr>  <chr>      <chr>      <dbl>    <qtr>
## 1 ACT   Female Remanded ATSI        0 2005 Q1
## 2 ACT   Female Remanded Non-ATSI    2 2005 Q1
## 3 ACT   Female Sentenced ATSI       0 2005 Q1
## 4 ACT   Female Sentenced Non-ATSI   5 2005 Q1
## 5 ACT   Male   Remanded  ATSI       7 2005 Q1
## # ... with 3,067 more rows
```

## Form a tsibble from a csv file

```
prison <- prison %>%  
  mutate(Quarter = yearquarter(Date)) %>%  
  select(-Date)  
prison %>% print(n = 5)
```

```
## # A tibble: 3,072 x 6  
##   State Gender Legal  
##   <chr>  <chr>  <chr>    <chr>      <dbl>  <qtr>  
## 1 ACT    Female Remanded ATSI          0 2005 Q1  
## 2 ACT    Female Remanded Non-ATSI      2 2005 Q1  
## 3 ACT    Female Sentenced ATSI          0 2005 Q1  
## 4 ACT    Female Sentenced Non-ATSI      5 2005 Q1  
## 5 ACT    Male   Remanded  ATSI          7 2005 Q1  
## # ... with 3,067 more rows
```

This is a tibble NOT a tsibble!

## Form a tsibble from a csv file

```
prison <- prison %>%  
  as_tsibble(index = Quarter,  
             key = c(State, Gender, Legal, Indigenous))  
prison %>% print(n = 5)
```

```
## # A tsibble: 3,072 x 6 [1Q]  
## # Key:      State, Gender, Legal, Indigenous [64]  
##   State Gender Legal   Indigenous Count Quarter  
##   <chr> <chr> <chr>     <chr>     <dbl>    <qtr>  
## 1 ACT   Female Remanded ATSI         0 2005 Q1  
## 2 ACT   Female Remanded ATSI         1 2005 Q2  
## 3 ACT   Female Remanded ATSI         0 2005 Q3  
## 4 ACT   Female Remanded ATSI         0 2005 Q4  
## 5 ACT   Female Remanded ATSI         1 2006 Q1  
## # ... with 3,067 more rows
```

## Useful dplyr verbs

`filter()` selects **a subset of rows** from a data frame.

`select()` selects **a subset of columns** from a data frame.

`summarise()` allows to summarise data across keys.

`mutate()` creates new variables.

- PBS tsibble containing monthly sales data (scripts and costs) on pharmaceutical products in Australia.
- Scripts/costs are disaggregated by drug type (ATC1: 15 categories, ATC2: 84 categories), concession type (concession vs general) and type (co-payments vs safety net).
- Altogether there are  $84 \times 2 \times 2 = 336$  time series.
- You can find more information about the data set by typing ?PBS in R.

## filter() function

We extract sales data for drugs used in diabetes.

```
PBS %>%  
  filter(ATC2 == "A10") %>%  
  print(n = 8)
```

```
## # A tsibble: 816 x 9 [1M]  
## # Key:      Concession, Type, ATC1, ATC2 [4]  
##   Month Concession Type  ATC1  ATC1_desc ATC2  ATC2_desc  
##   <mth> <chr>     <chr> <chr> <chr>     <chr> <chr>  
## 1 1991 Jul Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## 2 1991 Aug Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## 3 1991 Sep Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## 4 1991 Oct Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## 5 1991 Nov Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## 6 1991 Dec Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## 7 1992 Jan Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## 8 1992 Feb Concession~ Co-p~ A    Alimenta~ A10  ANTIDIAB~  
## # ... with 808 more rows, and 2 more variables:
```

## select() function

```
PBS %>%  
  filter(ATC2 == "A10") %>%  
  select(Month, Concession, Type, Cost) %>%  
  print(n = 8)
```

```
## # A tsibble: 816 x 4 [1M]  
## # Key:      Concession, Type [4]  
##       Month Concession     Type        Cost  
##       <mth> <chr>        <chr>       <dbl>  
## 1 1991 Jul Concessional Co-payments 2092878  
## 2 1991 Aug Concessional Co-payments 1795733  
## 3 1991 Sep Concessional Co-payments 1777231  
## 4 1991 Oct Concessional Co-payments 1848507  
## 5 1991 Nov Concessional Co-payments 1686458  
## 6 1991 Dec Concessional Co-payments 1843079  
## 7 1992 Jan Concessional Co-payments 1564702  
## 8 1992 Feb Concessional Co-payments 1732508  
## # ... with 808 more rows
```

## select() function

```
PBS %>%  
  filter(ATC2 == "A10") %>%  
  select(Month, Concession, Type, Cost) %>%  
  print(n = 8)
```

```
## # A tsibble: 816 x 4 [1M]  
## # Key:      Concession  Type [4]  
##      Month Concession  Type  
##      <mth> <chr>  
## 1 1991 Jul Concessional Co-payments 1795755  
## 2 1991 Aug Concessional Co-payments 1777231  
## 3 1991 Sep Concessional Co-payments 1777231  
## 4 1991 Oct Concessional Co-payments 1848507  
## 5 1991 Nov Concessional Co-payments 1686458  
## 6 1991 Dec Concessional Co-payments 1843079  
## 7 1992 Jan Concessional Co-payments 1564702  
## 8 1992 Feb Concessional Co-payments 1732508  
## # ... with 808 more rows
```

Index variable Month will be returned even if it was not explicitly selected.  
Because it is needed for a tsibble.

## summarise() function

```
PBS %>%  
  filter(ATC2 == "A10") %>%  
  select(Month, Concession, Type, Cost) %>%  
  summarise(Total_cost = sum(Cost)) %>%  
  print(n = 8)
```

```
## # A tsibble: 204 x 2 [1M]  
##   Month Total_cost  
##   <mth>     <dbl>  
## 1 1991    Jul     3526591  
## 2 1991    Aug     3180891  
## 3 1991    Sep     3252221  
## 4 1991    Oct     3611003  
## 5 1991    Nov     3565869  
## 6 1991    Dec     4306371  
## 7 1992    Jan     5088335  
## 8 1992    Feb     2814520  
## # ... with 196 more rows
```

## mutate() function

```
PBS %>%
  filter(ATC2 == "A10") %>%
  select(Month, Concession, Type, Cost) %>%
  summarise(Total_cost = sum(Cost)) %>%
  mutate(Total_cost = Total_cost/1e6) -> a10
```

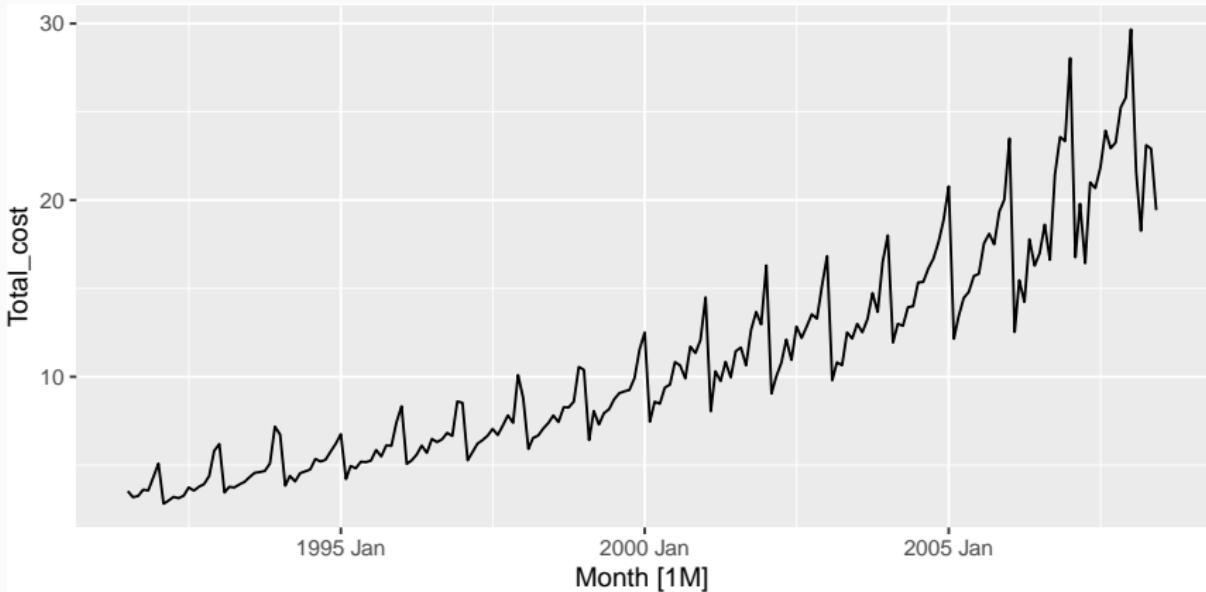
# Outline

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Time series patterns
- 5 Lag plots and autocorrelation
- 6 White noise

## Time plots

- The observations are plotted against the time of observation where the consecutive observations are joined by straight lines.

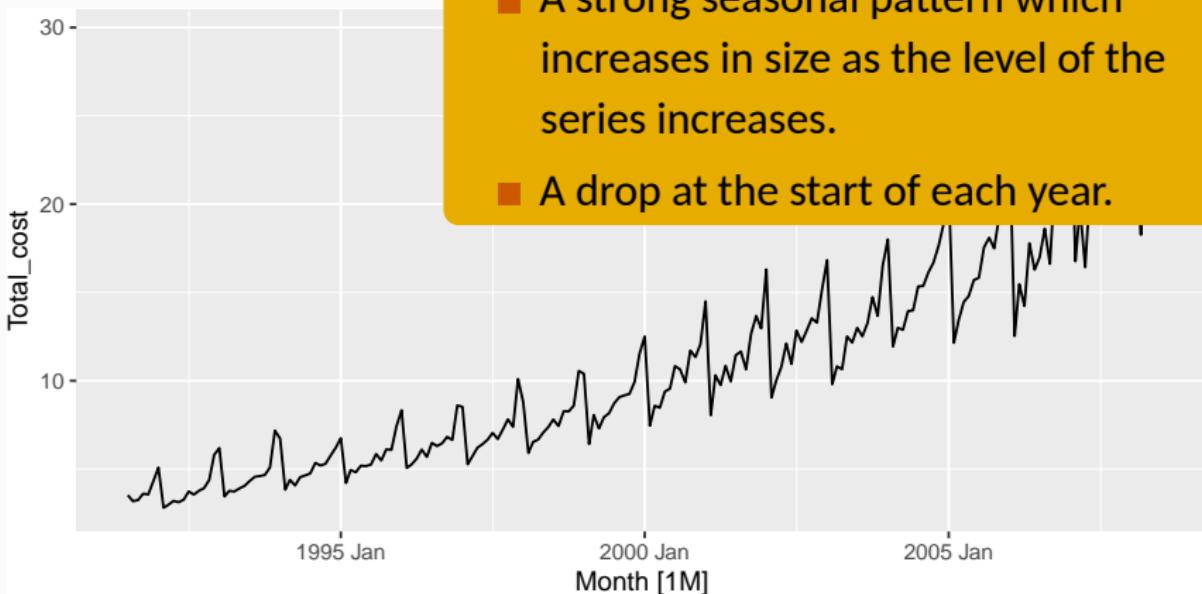
```
a10 %>%  
  autoplot(Total_cost)
```



## Time plots

- The observations are plotted against the time of observation where the consecutive observations are joined by straight lines.

```
a10 %>%  
  autoplot(Total_cost)
```



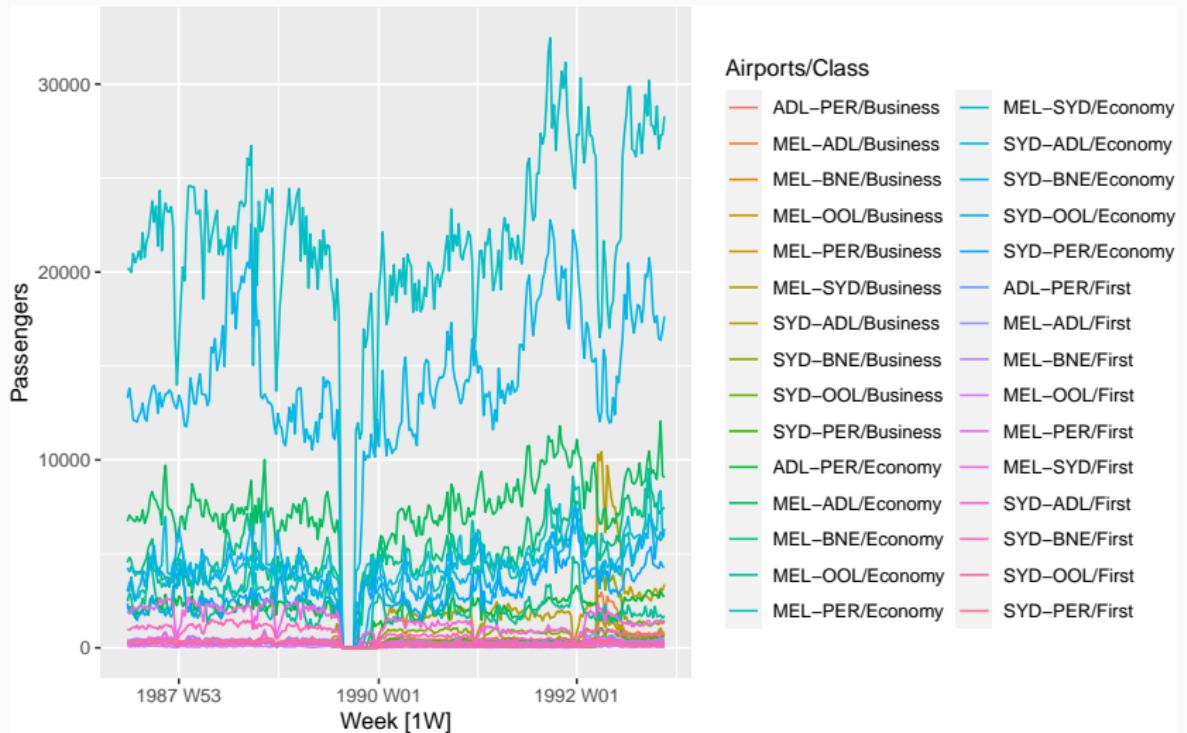
# Ansett Airlines

```
ansett
```

```
## # A tsibble: 7,407 x 4 [1W]
## # Key:      Airports, Class [30]
## # Week Airports Class   Passengers
## # <week> <chr>    <chr>     <dbl>
## 1 1989 W28 ADL-PER Business     193
## 2 1989 W29 ADL-PER Business     254
## 3 1989 W30 ADL-PER Business     185
## 4 1989 W31 ADL-PER Business     254
## 5 1989 W32 ADL-PER Business     191
## 6 1989 W33 ADL-PER Business     136
## 7 1989 W34 ADL-PER Business      0
## 8 1989 W35 ADL-PER Business      0
## 9 1989 W36 ADL-PER Business      0
## 10 1989 W37 ADL-PER Business     0
## # ... with 7,397 more rows
```

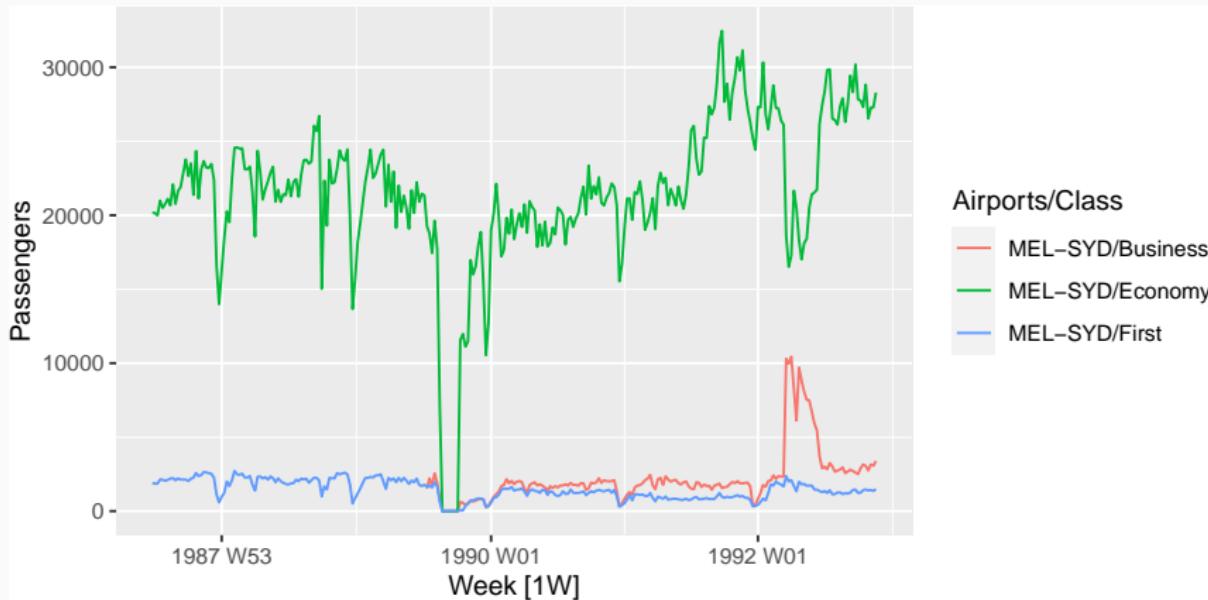
# Ansett Airlines

```
ansett %>%  
  autoplot(Passengers)
```



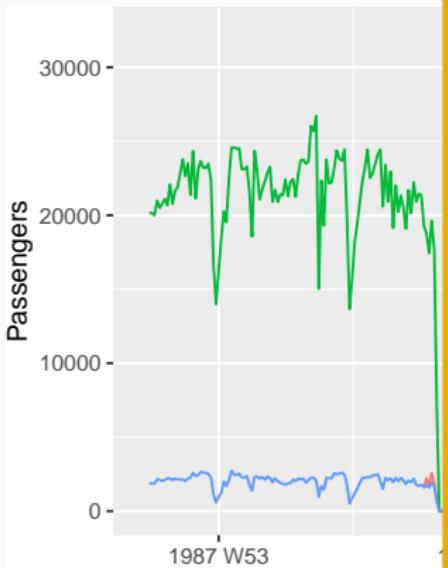
# Ansett Airlines

```
ansett %>%  
  filter(Airports == "MEL-SYD") %>%  
  autoplot(Passengers)
```



# Ansett Airlines

```
ansett %>%  
  filter(Airports == "MEL-SYD") %>%  
  autoplot(Passengers)
```



- A period in 1989 with no passengers carried.
- A large increase in passenger load occurred in the second half of 1991 for the Economy class.
- A period of reduced load in 1992 for the Economy class.
- Some large dips in load around the start of each year due to holiday effects.
- Contains missing observations such as in 1987 w38.

# Ansett Airlines

```
ansett %>%  
  filter(Airports == "MEL-SYD") %>%  
  scan_gaps()  
  
## # A tsibble: 2 x 3 [1W]  
## # Key:     Airports, Class [2]  
##   Airports Class      Week  
##   <chr>    <chr>    <week>  
## 1 MEL-SYD  Economy 1987 W38  
## 2 MEL-SYD  First    1987 W38
```

# Outline

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Time series patterns
- 5 Lag plots and autocorrelation
- 6 White noise

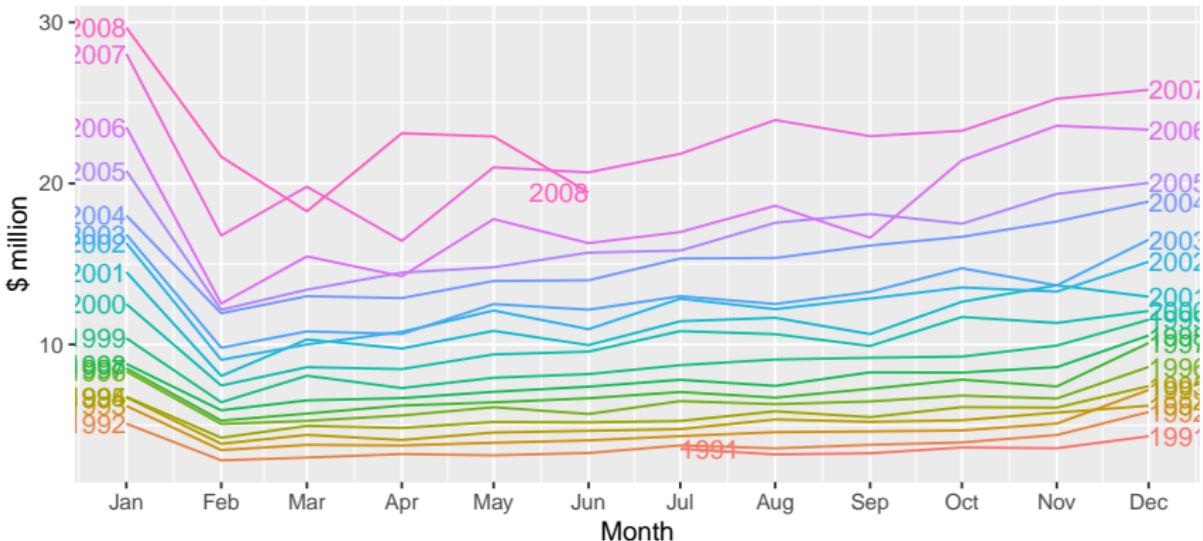
## Seasonal plots

- Similar to a time plot, but the data are plotted against the individual seasons in which the data were observed.
- These plots enable us to observe the underlying seasonal pattern more clearly.
- It is also useful to identify years in which substantial departures from the seasonal pattern has occurred.

# Seasonal plots

```
a10 %>%  
  gg_season(Total_cost, labels = "both") +  
  labs(y = "$ million",  
        title = "Seasonal plot for diabetic drug sales")
```

Seasonal plot for diabetic drug sales



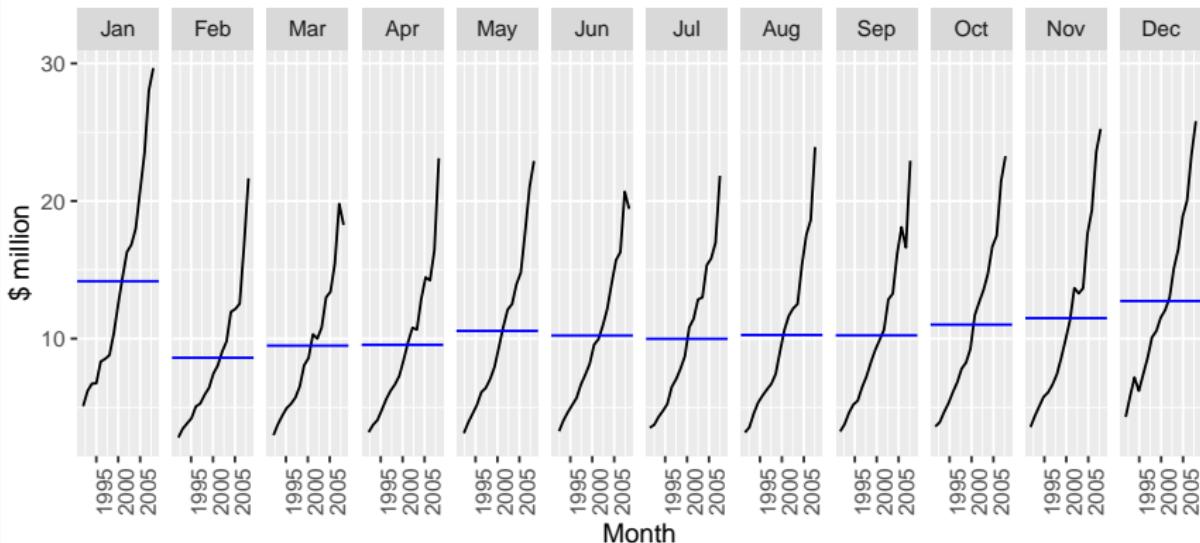
## Seasonal subseries plots

- Data for each season are gathered and plotted as separate time plots.
- From these plots we can clearly observe the underlying seasonal pattern.
- It also shows the changes in seasonality over time.

# Seasonal subseries plots

```
a10 %>%  
  gg_subseries(Total_cost) +  
  labs(y = "$ million",  
        title = "Subseries plot for diabetic drug sales")
```

Subseries plot for diabetic drug sales

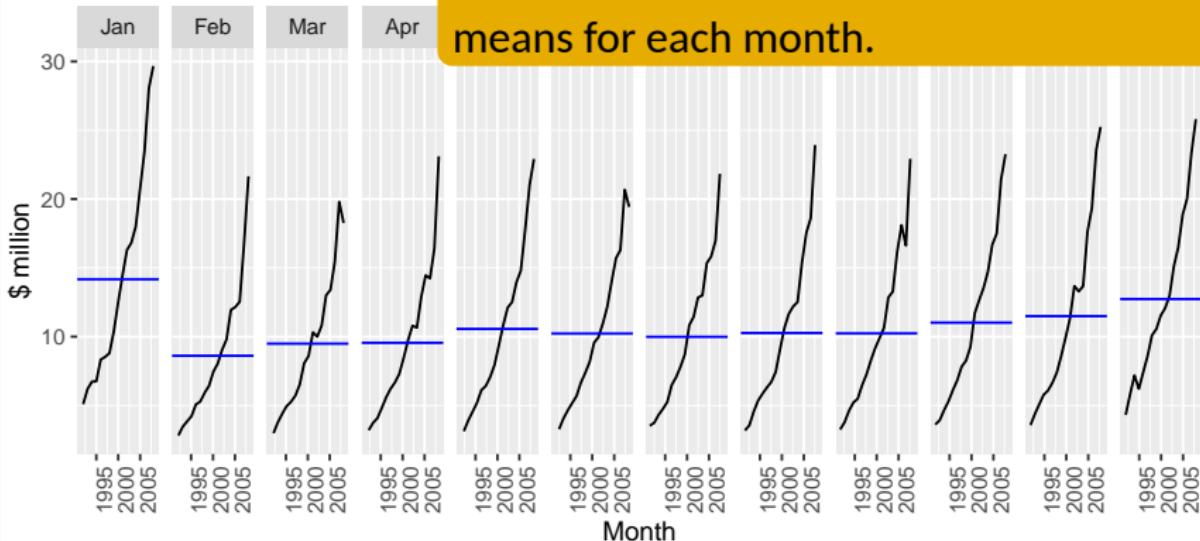


# Seasonal subseries plots

```
a10 %>%  
  gg_subseries(Total_cost) +  
  labs(y = "$ million",  
        title = "Subseries plot for diabetic drug sales")
```

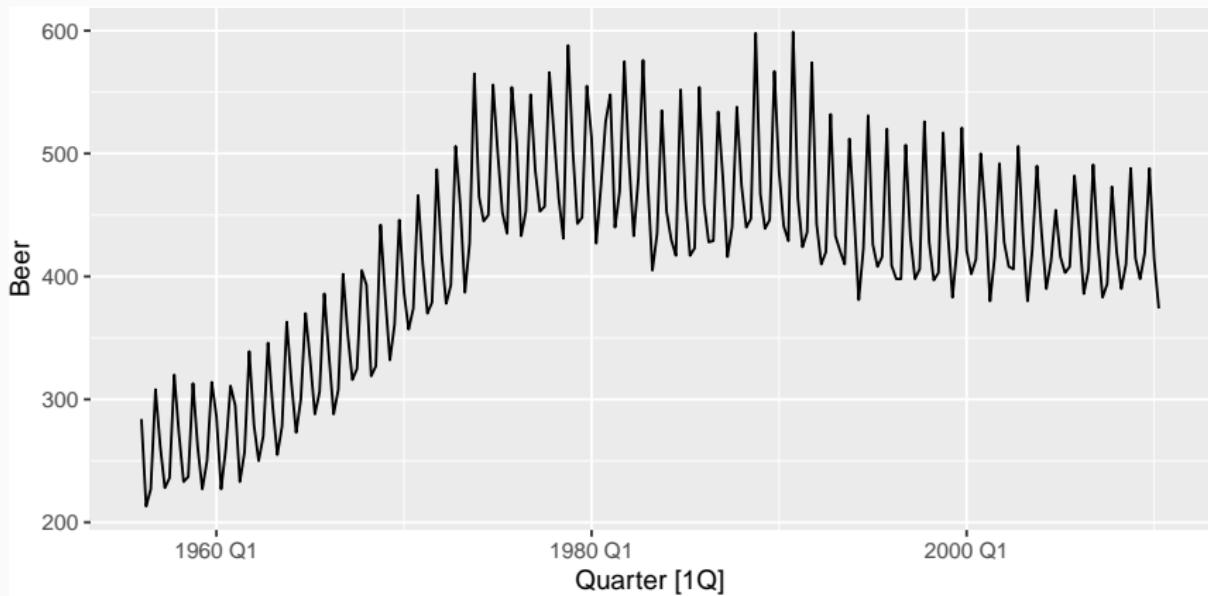
Subseries plot for diabetic

The blue horizontal lines indicate the means for each month.



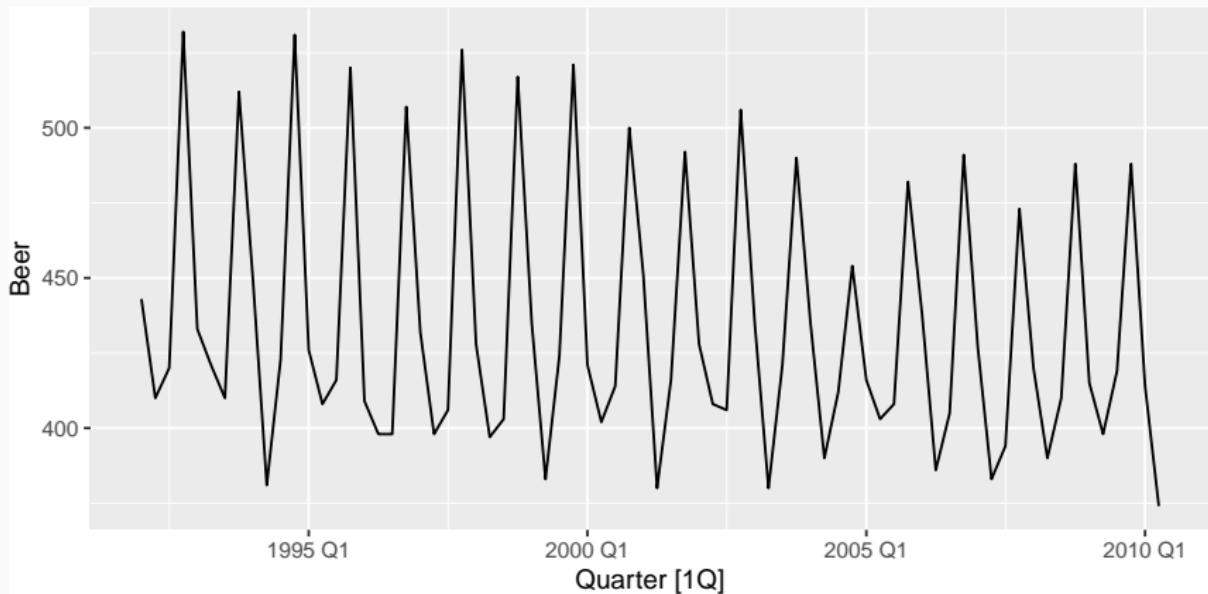
# Beer production in Australia

```
aus_production %>%
  select(Beer) %>%
  autoplot()
```



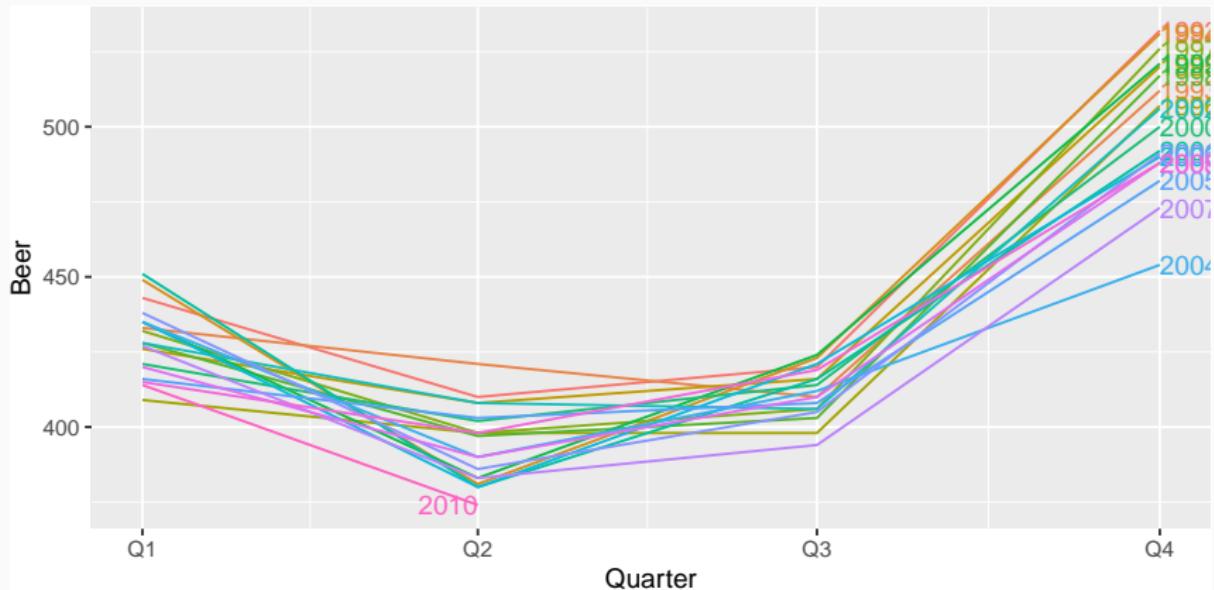
# Beer production in Australia

```
aus_production %>%
  select(Beer) %>%
  filter(year(Quarter) >= 1992) -> beer
beer %>%
  autoplot(Beer)
```



# Beer production in Australia

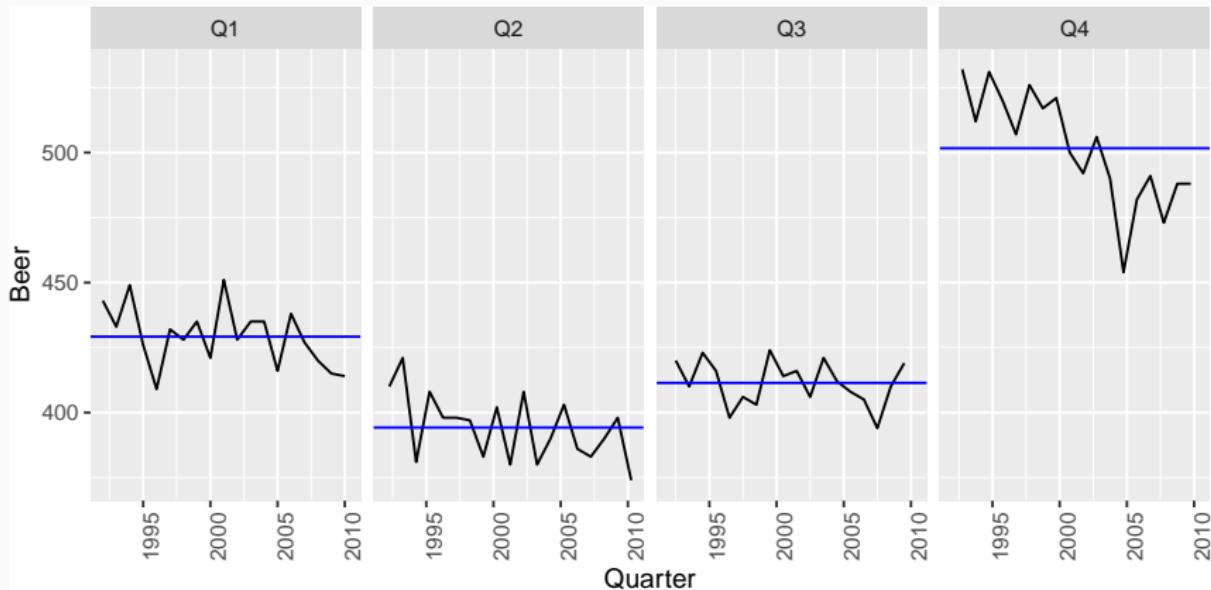
```
beer %>%  
  gg_season(Beer, labels = "right")
```



# Beer production in Australia

```
beer %>%
```

```
gg_subseries(Beer)
```



# Australian domestic tourism

```
holidays <- tourism %>%
  filter(Purpose == "Holiday") %>%
  group_by(State) %>%
  summarise(Trips = sum(Trips))
holidays %>% print(n = 7)
```

```
## # A tsibble: 640 x 3 [1Q]
## # Key:      State [8]
##   State Quarter Trips
##   <chr>   <qtr> <dbl>
## 1 ACT     1998 Q1    196.
## 2 ACT     1998 Q2    127.
## 3 ACT     1998 Q3    111.
## 4 ACT     1998 Q4    170.
## 5 ACT     1999 Q1    108.
## 6 ACT     1999 Q2    125.
## 7 ACT     1999 Q3    178.
## # ... with 633 more rows
```

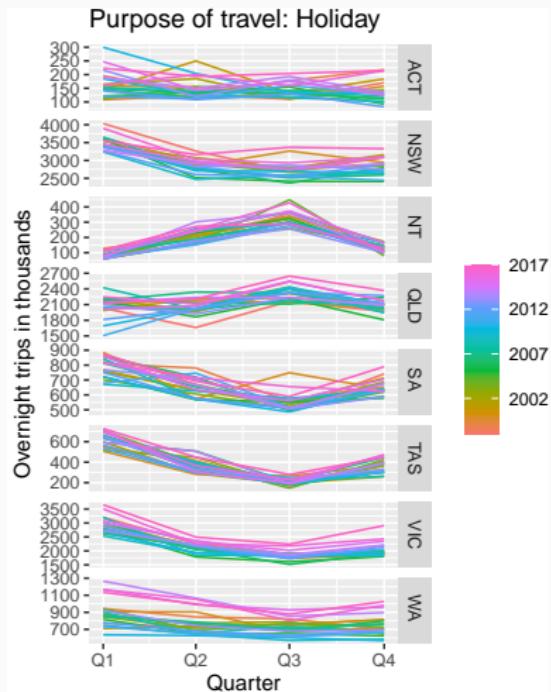
# Time plot

```
holidays %>%
  autoplot(Trips)
```



# Seasonal plot

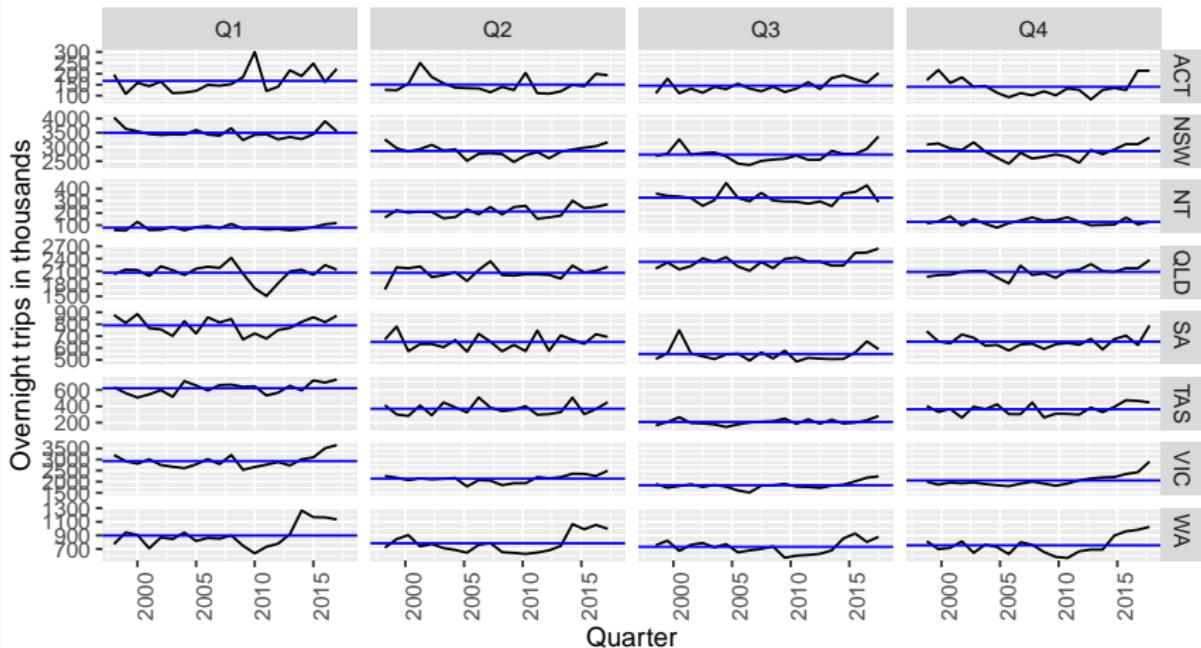
```
holidays %>% gg_season(Trips) +  
  labs(y = "Overnight trips in thousands",  
       title = "Purpose of travel: Holiday")
```



# Subseries plot

```
holidays %>% gg_subseries(Trips) +  
  labs(y = "Overnight trips in thousands",  
        title = "Purpose of travel: Holiday")
```

Purpose of travel: Holiday



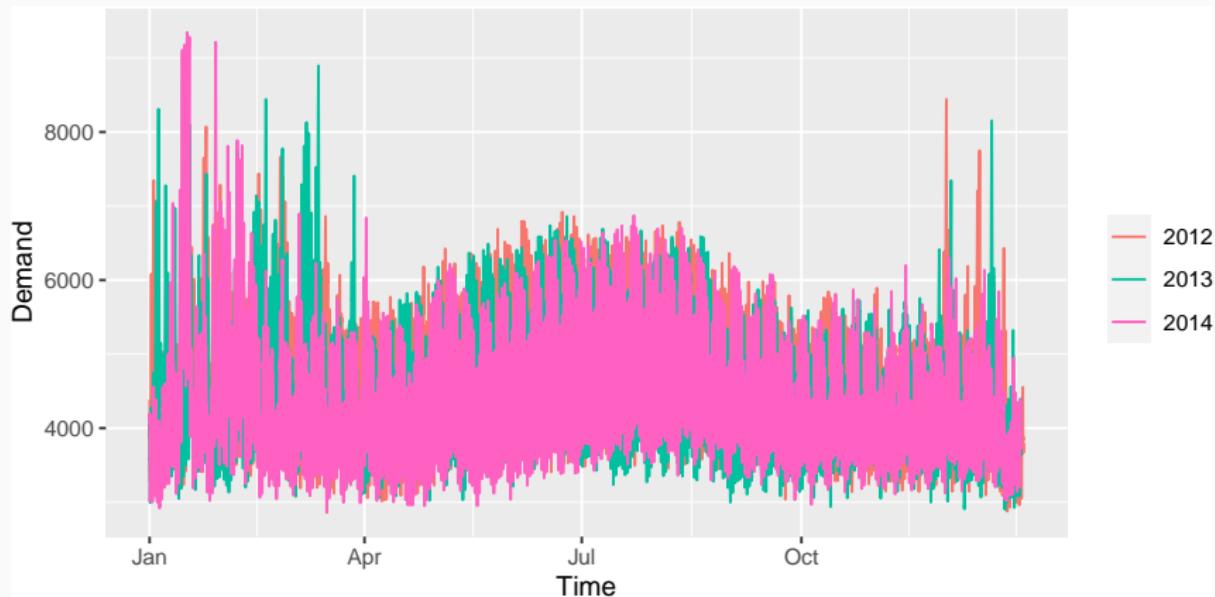
## Multiple seasonality

```
vic_elec
```

```
## # A tsibble: 52,608 x 5 [30m] <Australia/Melbourne>
##   Time           Demand Temperature Date     Holiday
##   <dttm>        <dbl>      <dbl> <date>    <lgl>
## 1 2012-01-01 00:00:00  4383.       21.4 2012-01-01 TRUE
## 2 2012-01-01 00:30:00  4263.       21.0 2012-01-01 TRUE
## 3 2012-01-01 01:00:00  4049.       20.7 2012-01-01 TRUE
## 4 2012-01-01 01:30:00  3878.       20.6 2012-01-01 TRUE
## 5 2012-01-01 02:00:00  4036.       20.4 2012-01-01 TRUE
## 6 2012-01-01 02:30:00  3866.       20.2 2012-01-01 TRUE
## 7 2012-01-01 03:00:00  3694.       20.1 2012-01-01 TRUE
## 8 2012-01-01 03:30:00  3562.       19.6 2012-01-01 TRUE
## 9 2012-01-01 04:00:00  3433.       19.1 2012-01-01 TRUE
## 10 2012-01-01 04:30:00  3359.       19.0 2012-01-01 TRUE
## # ... with 52,598 more rows
```

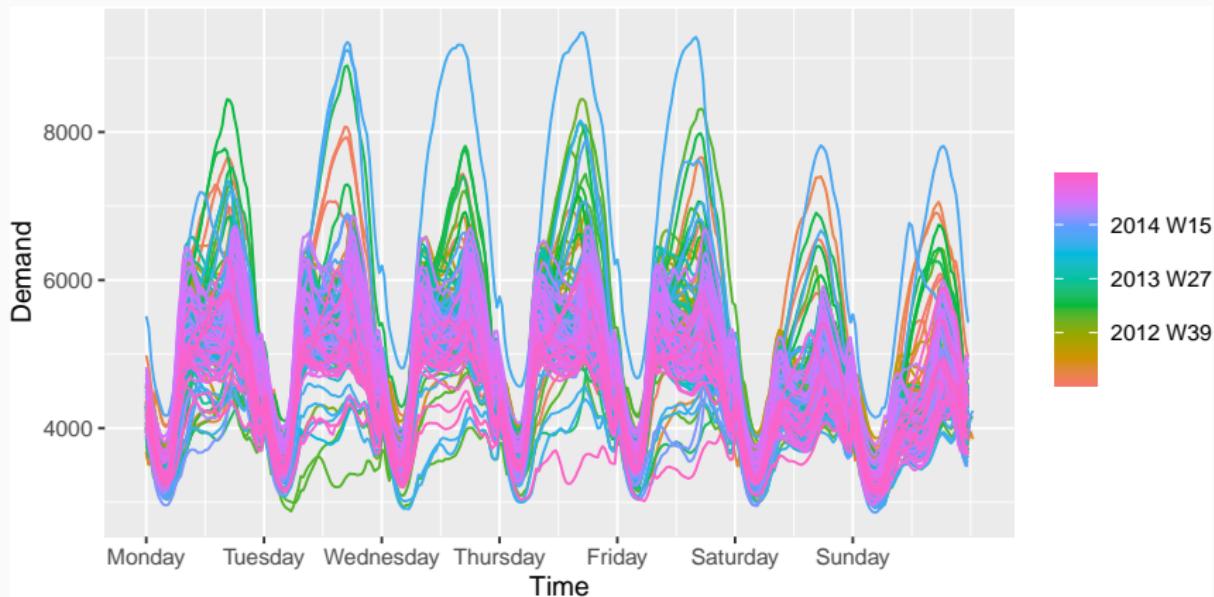
# Annual seasonality

```
vic_elec %>%  
  gg_season(Demand)
```



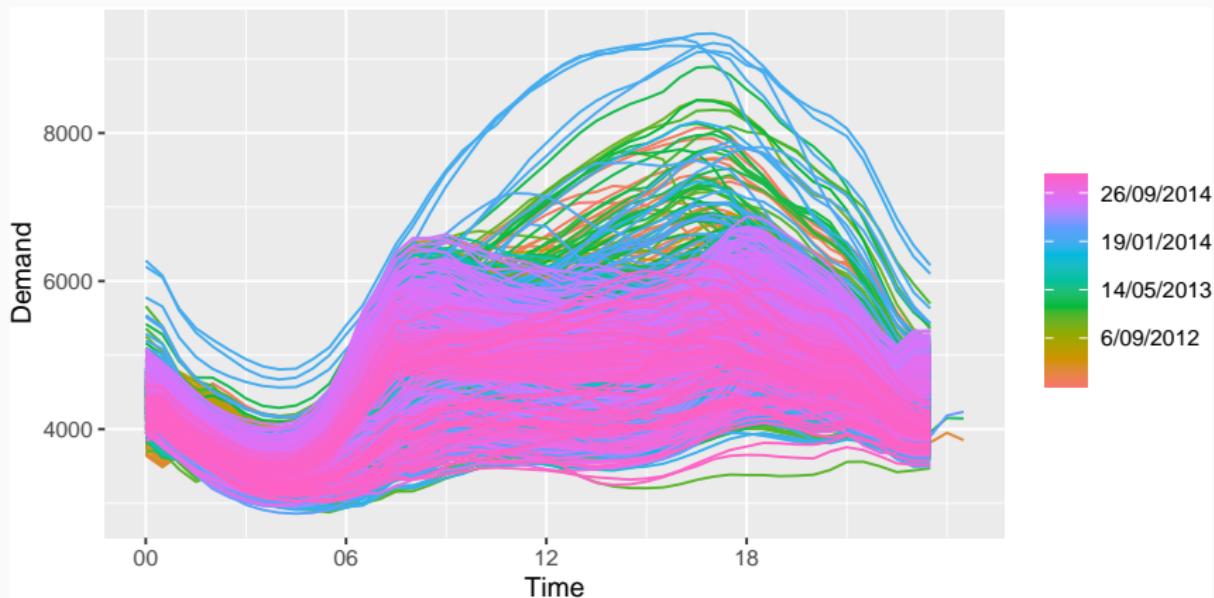
# Weekly seasonality

```
vic_elec %>%  
  gg_season(Demand, period = "week")
```



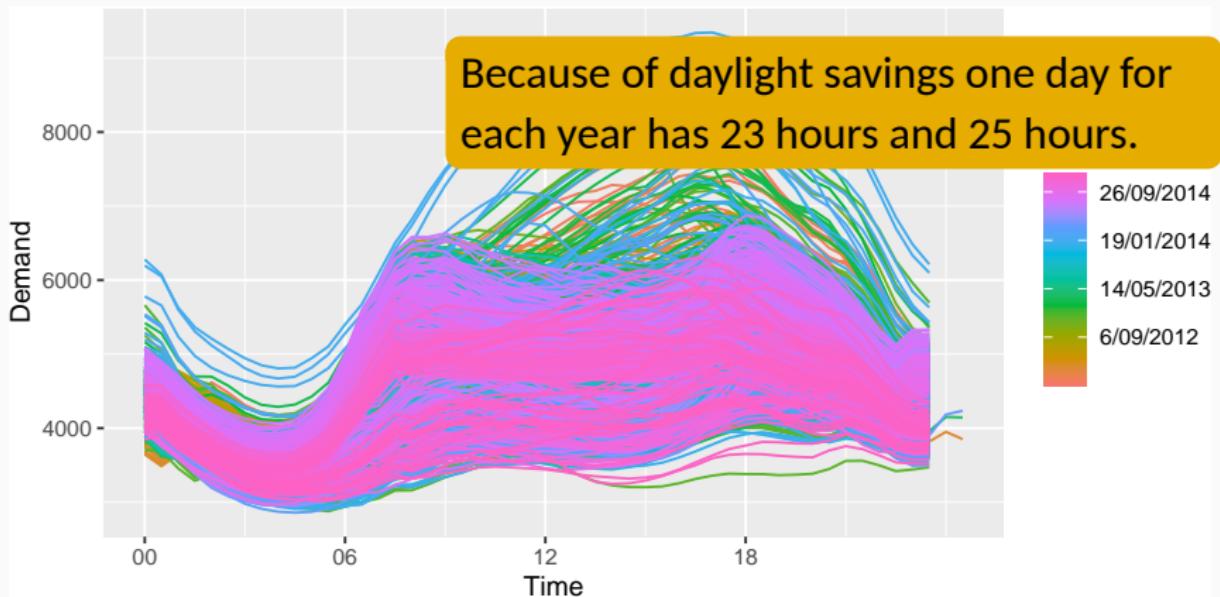
## Daily seasonality

```
vic_elec %>%  
  gg_season(Demand, period = "day")
```



## Daily seasonality

```
vic_elec %>%  
  gg_season(Demand, period = "day")
```



# Outline

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Time series patterns
- 5 Lag plots and autocorrelation
- 6 White noise

## Time series patterns

**Trend** exists when there is a long-term increase or decrease in the data.

It doesn't have to be linear.

**Seasonal** occurs when the series is affected by seasonal factors such as quarter of the year, month, day of the week.

**Cyclic** occurs when the data show rises and falls that are not of a fixed frequency.

duration of these cycles is usually at least 2 years.

## Seasonal vs cyclic

- **Seasonal** pattern has a **fixed length** and associated with some aspect of the calendar.
- **Cyclic** pattern has a **variable length**.
- Average length of cycles is longer than that of a seasonal pattern.
- Magnitudes of cycles tend to be more variable than that of seasonal patterns.

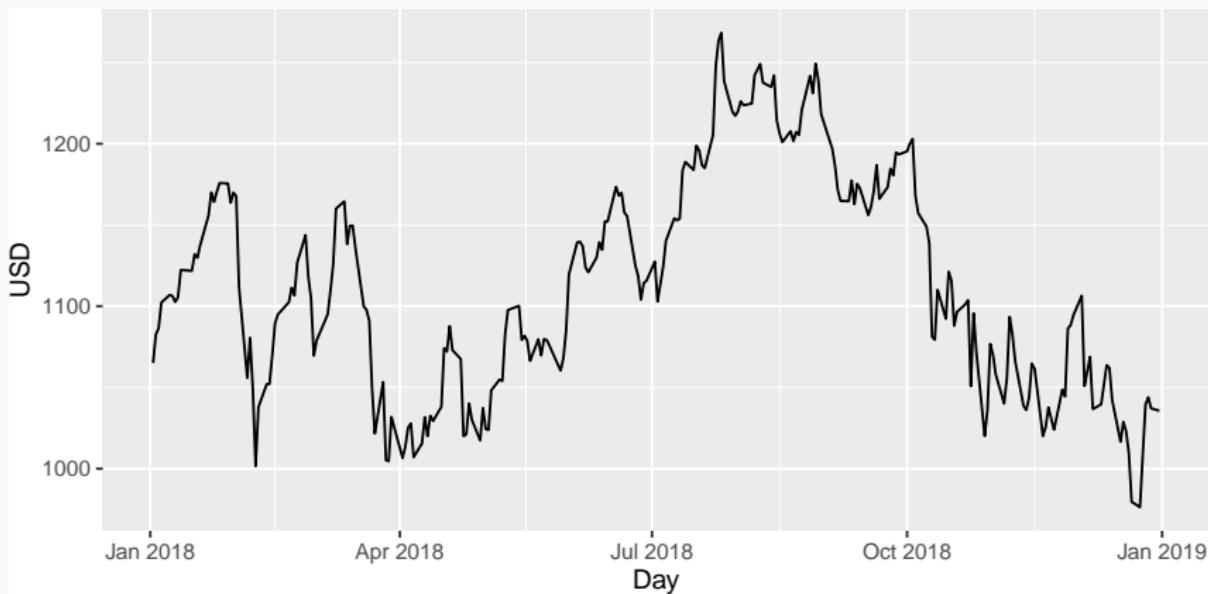
## Seasonal vs cyclic

- **Seasonal** pattern has a **fixed length** and associated with some aspect of the calendar.
- **Cyclic** pattern has a **variable length**.
- Average length of cycles is longer than that of a seasonal pattern.
- Magnitudes of cycles tend to be more variable than that of seasonal patterns.

The timing of *peaks and troughs* is *predictable* with **seasonal** data, but unpredictable in the long term with cyclic data.

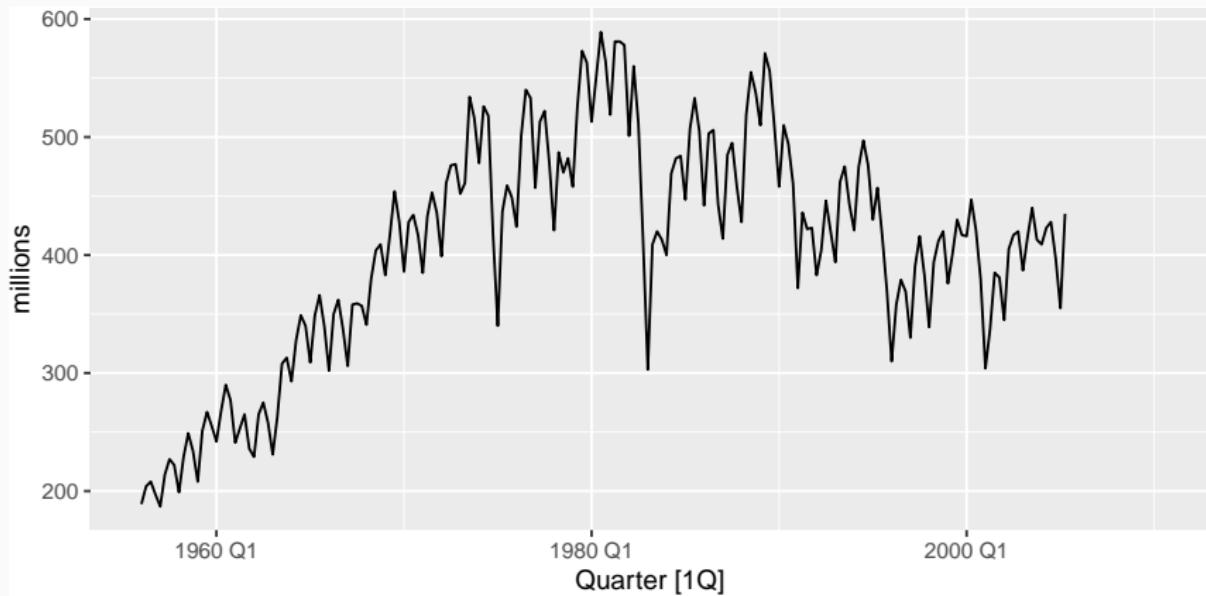
## Daily closing stock price for Google

```
gafa_stock %>%
  filter(Symbol == "GOOG", year(Date) >= 2018) %>%
  autoplot(Close) +
  labs(y = "USD", x = "Day")
```



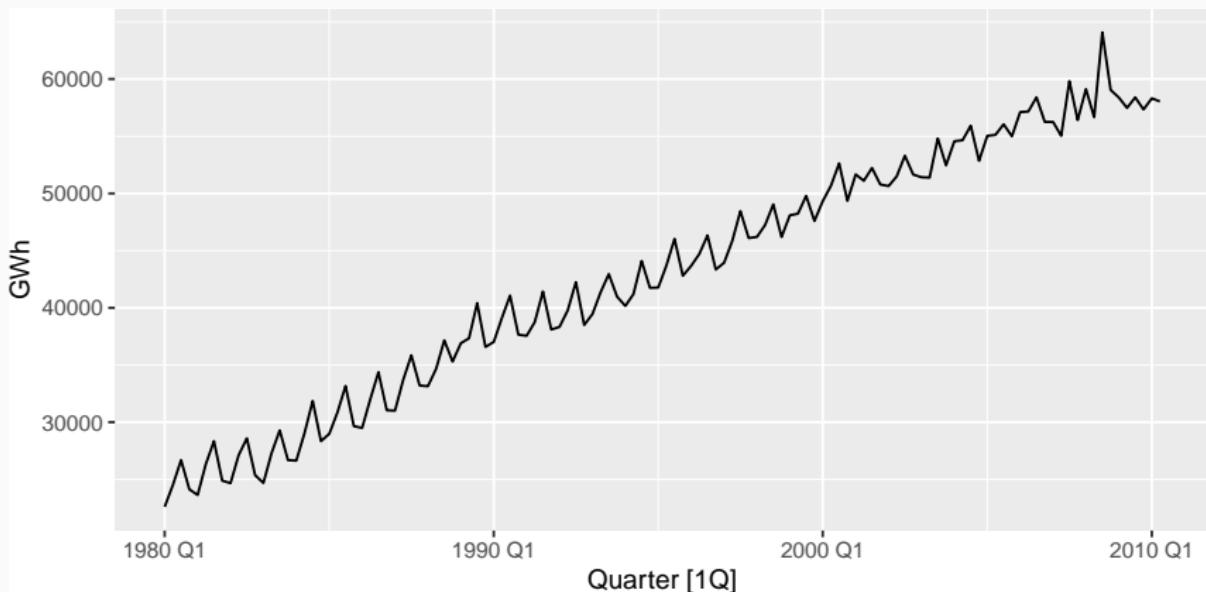
## Australian quarterly brick production

```
aus_production %>%
  autoplot(Bricks) +
  ylab("millions")
```



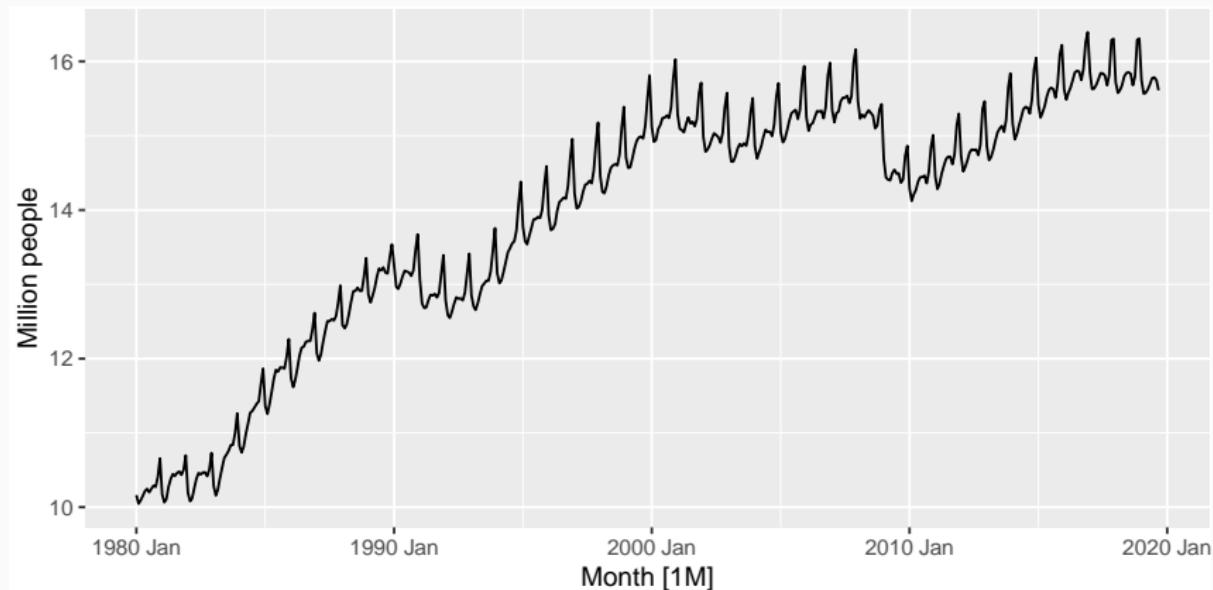
# Australian quarterly electricity production

```
aus_production %>%
  filter(year(Quarter) >= 1980) %>%
  autoplot(Electricity) +
  ylab("GWh")
```



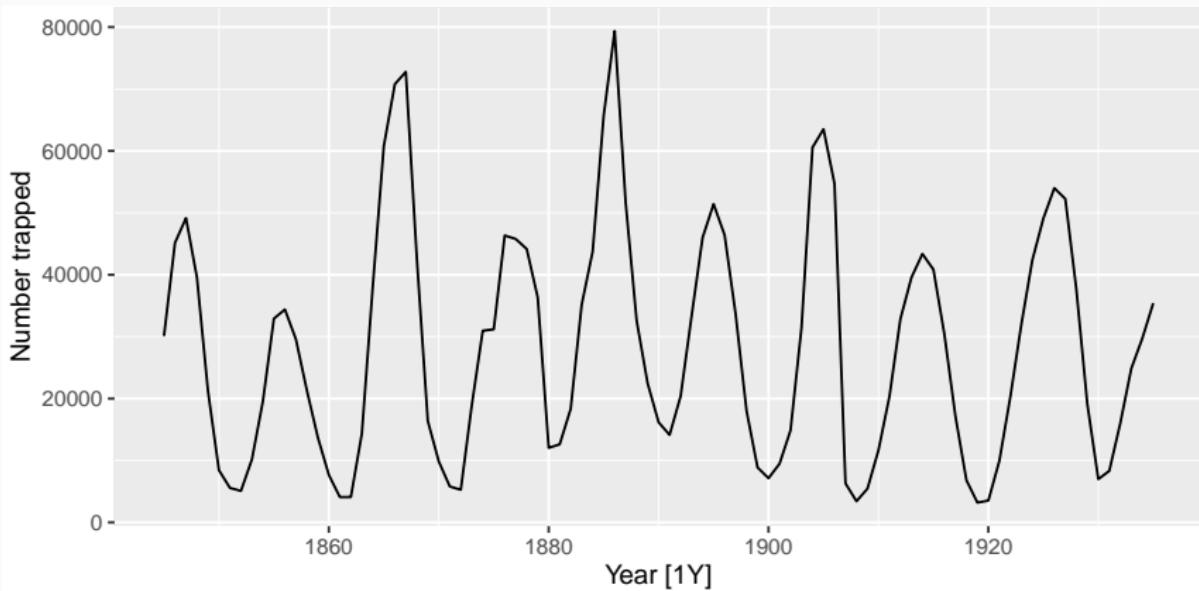
# Monthly employment in retail industry

```
us_employment %>%
  filter>Title == "Retail Trade", year(Month) >= 1980) %>%
  autoplot(Employed/1e3) +
  ylab("Million people")
```



# Annual Canadian lynx trapping

```
pelt %>%  
  autoplot(Lynx) +  
  ylab("Number trapped")
```



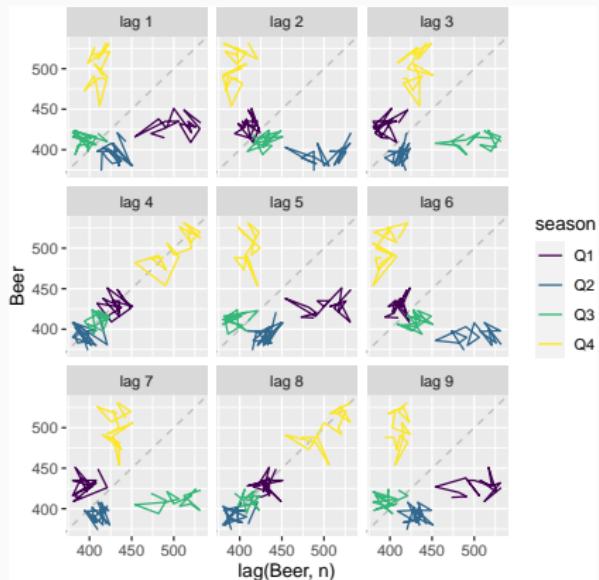
# Outline

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Time series patterns
- 5 Lag plots and autocorrelation
- 6 White noise

# Lag plots

- These graphs plot time series against lagged versions of themselves allowing to analyze the relationship between a series and its lags.

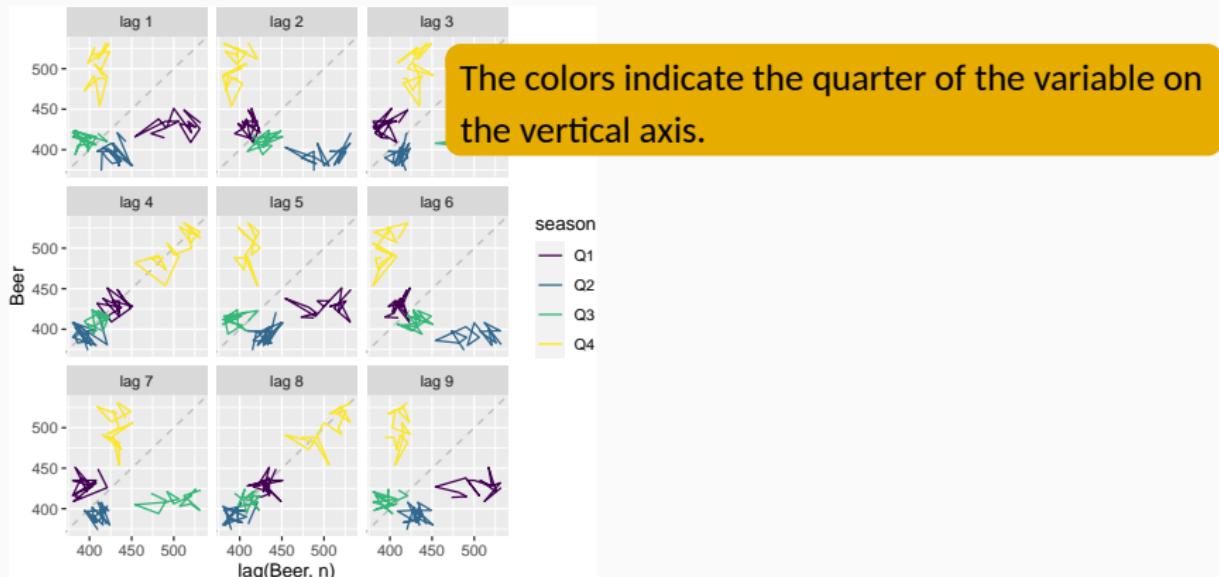
```
aus_production %>%  
  filter(year(Quarter) >= 1992) %>% gg_lag(Beer)
```



# Lag plots

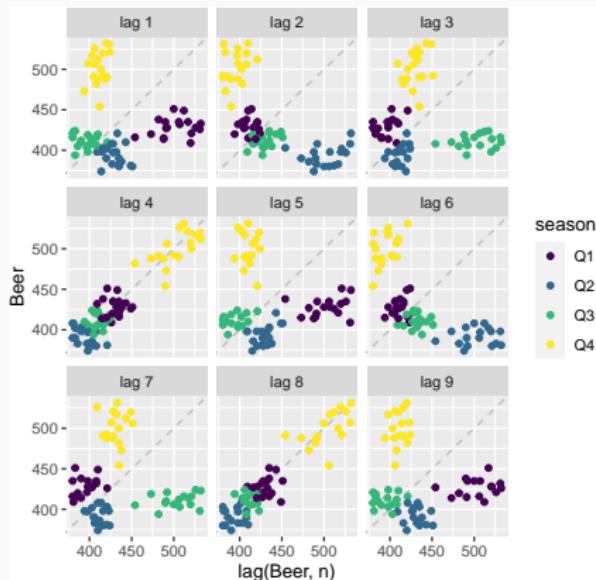
- These graphs plot time series against lagged versions of themselves allowing to analyze the relationship between a series and its lags.

```
aus_production %>%  
  filter(year(Quarter) >= 1992) %>% gg_lag(Beer)
```



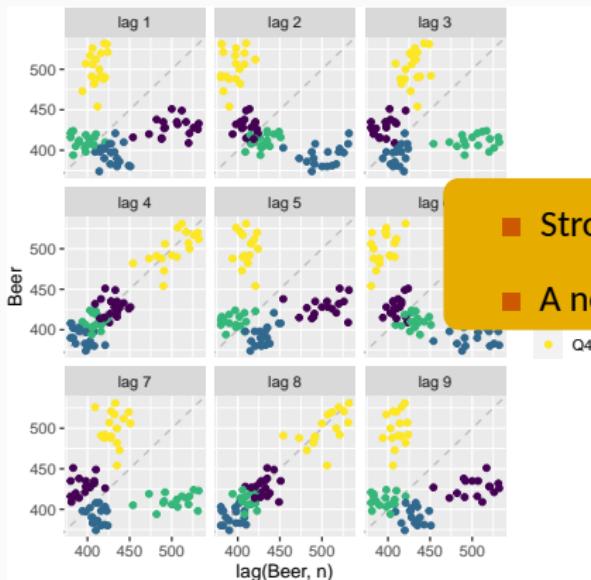
# Lag plots

```
aus_production %>%
  filter(year(Quarter) >= 1992) %>%
  gg_lag(Beer, geom = "point")
```



# Lag plots

```
aus_production %>%  
  filter(year(Quarter) >= 1992) %>%  
  gg_lag(Beer, geom = "point")
```



- Strongly positive relationship at lags 4 and 8.
- A negative relationship seen for lags 2 and 6.

## Lag plots

- In each of these plots we graph  $y_t$  against  $y_{t-k}$  for  $k = 1, 2, \dots$
- If we compute the correlation coefficients associated with each of these scatter plots we get **autocorrelations**.
- The autocorrelation coefficients make up the **autocorrelation function** (ACF). i.e.
  - ▶  $r_1 = \text{Correlation}(y_t, y_{t-1})$
  - ▶  $r_2 = \text{Correlation}(y_t, y_{t-2})$
  - ▶  $r_3 = \text{Correlation}(y_t, y_{t-3})$
  - ▶ so on.

## Autocorrelation

- Covariance and correlation measure the extent of linear relationship between two random variables (say  $y$  and  $x$ ).
- Autocovariance and autocorrelation measure the extent of linear relationship between lagged values of a time series  $y$ .
- Given a time series  $y_1, y_2, \dots, y_T$ , we denote the sample autocovariance at lag  $k$  by  $c_k$  and the sample autocorrelation at lag  $k$  by  $r_k$ .

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

$$r_k = c_k/c_0, \text{ and}$$

$$\bar{y} = \frac{1}{T} \sum_{t=1}^T y_t$$

## Autocorrelation

- $r_1$  indicates how consecutive observations are related to each other.
- $r_2$  indicates how  $y$  values which are two periods apart related to each other.
- $r_k$  indicates how  $y$  values which are  $k$  periods apart related to each other.

# Autocorrelation

First 9 sample autocorrelations for beer data:

```
recent_production <- aus_production %>%
  filter(year(Quarter) >= 1992)
recent_production %>% ACF(Beer, lag_max = 9)
```

```
## # A tsibble: 9 x 2 [1Q]
##   lag      acf
##   <dbl>    <dbl>
## 1 1Q -0.102
## 2 2Q -0.657
## 3 3Q -0.0603
## 4 4Q  0.869
## 5 5Q -0.0892
## 6 6Q -0.635
## 7 7Q -0.0542
## 8 8Q  0.832
## 9 9Q -0.108
```

# Autocorrelation

First 9 sample autocorrelations for beer data:

```
recent_production <- aus_production %>%
  filter(year(Quarter) >= 1992)
recent_production %>% ACF(Beer, lag_max = 9)
```

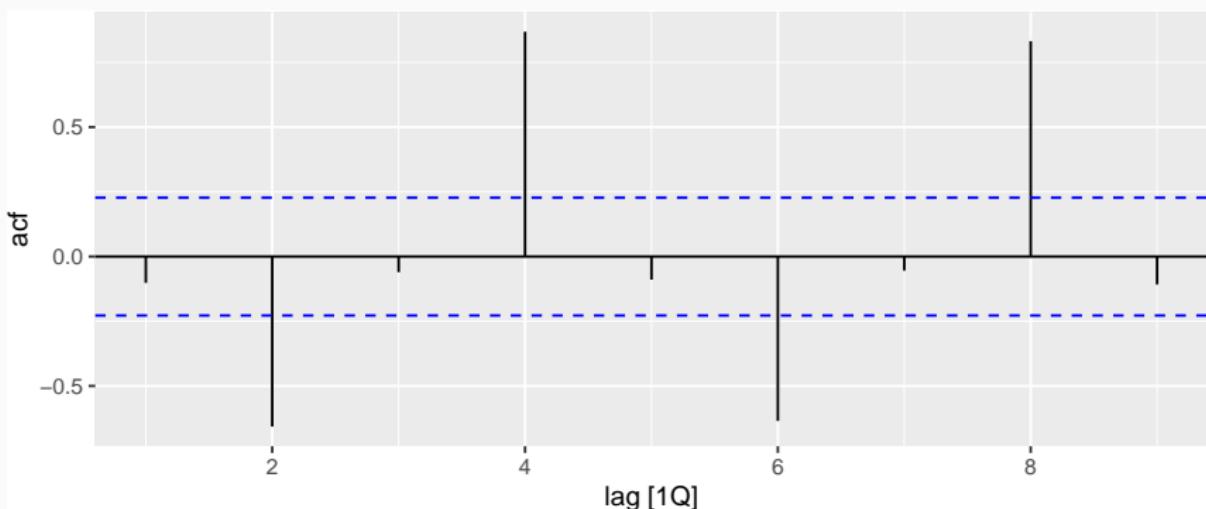
```
## # A tsibble: 9 x 2 [1Q]
##   lag      acf
##   <dbl>    <dbl>
## 1 1Q -0.102
## 2 2Q -0.657
## 3 3Q -0.0603
## 4 4Q  0.869
## 5 5Q -0.0892
## 6 6Q -0.635
## 7 7Q -0.0542
## 8 8Q  0.832
## 9 9Q -0.108
```

The acf column give  $r_1, \dots, r_9$ , corresponding to the nine scatter plots.

## Correlogram

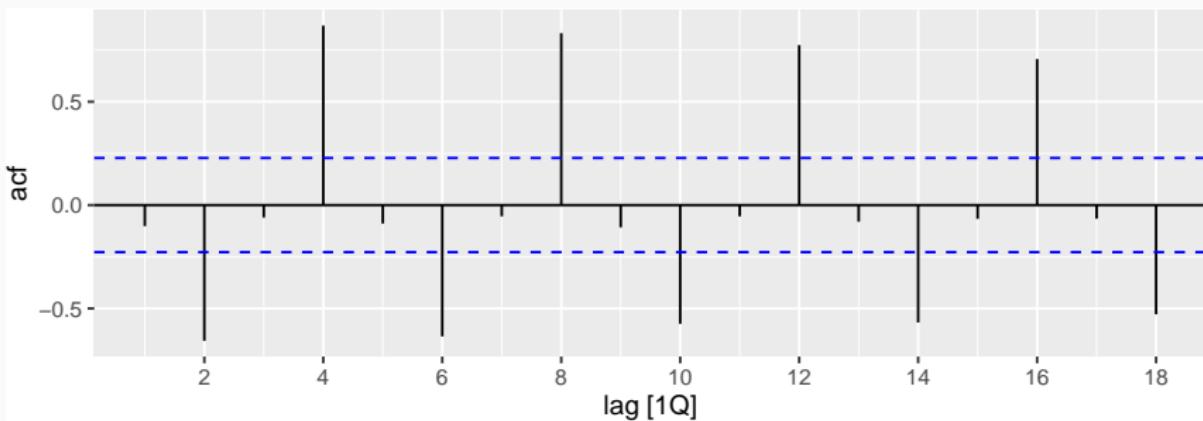
- We usually plot these autocorrelations to see how the correlations change with the lag  $k$ . We refer to this plot as **correlogram**.

```
recent_production %>%  
  ACF(Beer, lag_max = 9) %>%  
  autoplot()
```



## Correlogram

```
recent_production %>% ACF(Beer) %>% autoplot()
```



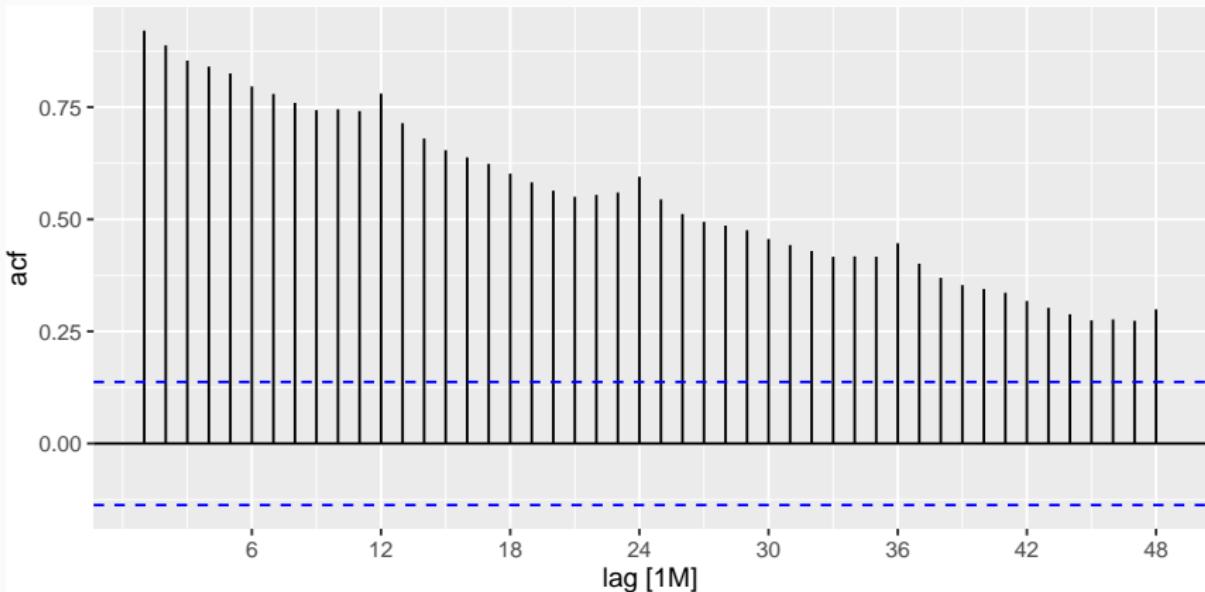
- $r_4$  is the highest. This is because of the seasonal pattern in the data: peaks tend to occur four quarters apart and troughs tend to occur four quarters apart.
- $r_2$  is more negative than other lags. This is because troughs tend to be two quarters behind peaks.

## Trend and seasonality in ACF plots

- If data show a trend, the autocorrelations for small lags tend to be large and positive.
- This is because observations nearby in time are also nearby in magnitude.
- ACF plot of trended time series tend to have positive values and decays slowly as the lags increases.
- If data are seasonal, the autocorrelations tend to be larger for the seasonal lags (i.e., at multiples of the seasonal frequency) than for other lags.
- If data are trended and seasonal, we can see a combination of these effects.

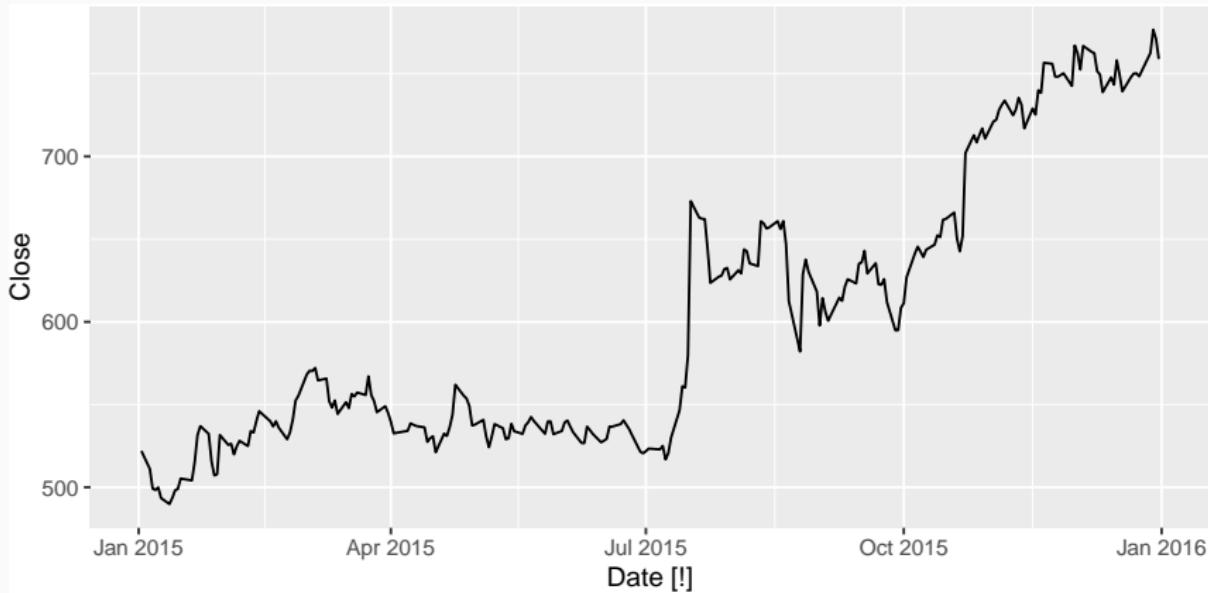
## Diabetes data set

```
a10 %>%  
  ACF(Total_cost, lag_max = 48) %>%  
  autoplot()
```



## Google closing stock price

```
google_2015 <- gafa_stock %>%
  filter(Symbol == "GOOG", year(Date) == 2015)
google_2015 %>% autoplot(Close)
```



## Sample ACF for Google stock prices

```
google_2015 %>%  
  ACF(Close, lag_max=100)  
# Error: Can't handle tsibble of irregular interval.
```

## Sample ACF for Google stock prices

```
google_2015
```

```
## # A tsibble: 252 x 8 [!]
## # Key:      Symbol [1]
## #   Symbol Date      Open  High   Low Close Adj_Close
## #   <chr>  <date>    <dbl> <dbl> <dbl> <dbl>    <dbl>
## # 1 GOOG  2015-01-02 526.  528.  521.  522.    522.
## # 2 GOOG  2015-01-05 520.  521.  510.  511.    511.
## # 3 GOOG  2015-01-06 512.  513.  498.  499.    499.
## # 4 GOOG  2015-01-07 504.  504.  497.  498.    498.
## # 5 GOOG  2015-01-08 495.  501.  488.  500.    500.
## # 6 GOOG  2015-01-09 502.  502.  492.  493.    493.
## # 7 GOOG  2015-01-12 492.  493.  485.  490.    490.
## # 8 GOOG  2015-01-13 496.  500.  490.  493.    493.
## # 9 GOOG  2015-01-14 492.  500.  490.  498.    498.
## # 10 GOOG 2015-01-15 503.  503.  495.  499.    499.
## # ... with 242 more rows, and 1 more variable: Volume <dbl>
```

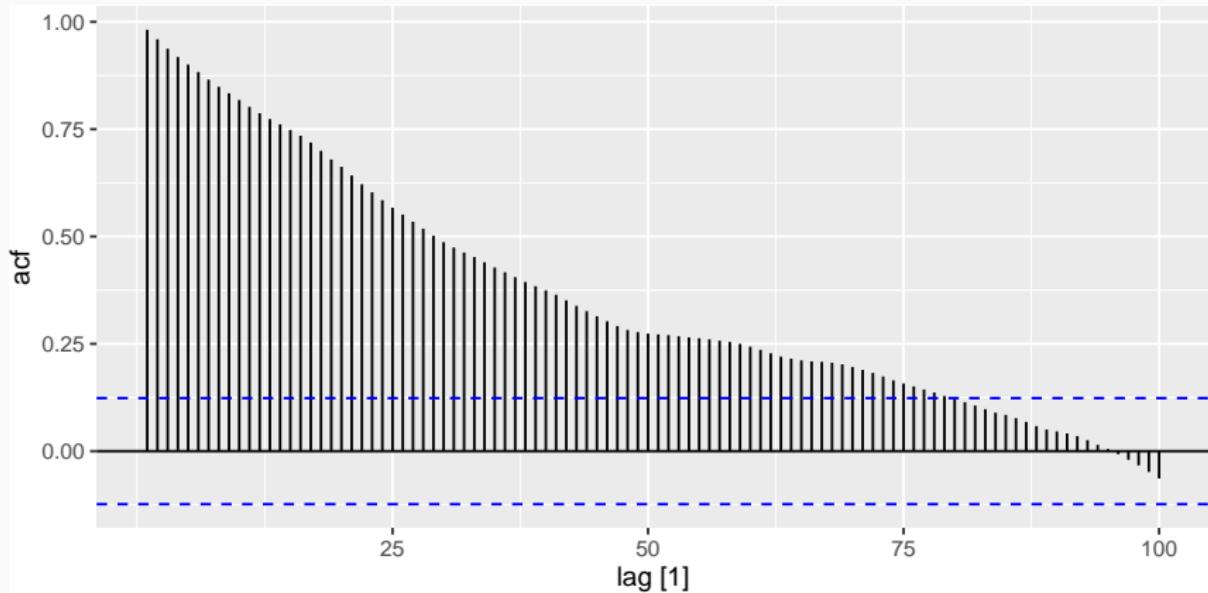
## Sample ACF for Google stock prices

```
google_2015 <- google_2015 %>%
  mutate(Trading_day = row_number()) %>%
  update_tsibble(index = Trading_day, regular = TRUE)
google_2015 %>% print(n = 7)

## # A tsibble: 252 x 9 [1]
## # Key:     Symbol [1]
##   Symbol Date      Open  High   Low Close Adj_Close Volume
##   <chr>  <date>    <dbl> <dbl> <dbl> <dbl>    <dbl>   <dbl>
## 1 GOOG  2015-01-02  526.  528.  521.  522.    522.  1.45e6
## 2 GOOG  2015-01-05  520.  521.  510.  511.    511.  2.06e6
## 3 GOOG  2015-01-06  512.  513.  498.  499.    499.  2.90e6
## 4 GOOG  2015-01-07  504.  504.  497.  498.    498.  2.07e6
## 5 GOOG  2015-01-08  495.  501.  488.  500.    500.  3.35e6
## 6 GOOG  2015-01-09  502.  502.  492.  493.    493.  2.07e6
## 7 GOOG  2015-01-12  492.  493.  485.  490.    490.  2.32e6
## # ... with 245 more rows, and 1 more variable:
## #   Trading_day <int>
```

## Sample ACF for Google stock prices

```
google_2015 %>%  
  ACF(Close, lag_max = 100) %>%  
  autoplot()
```



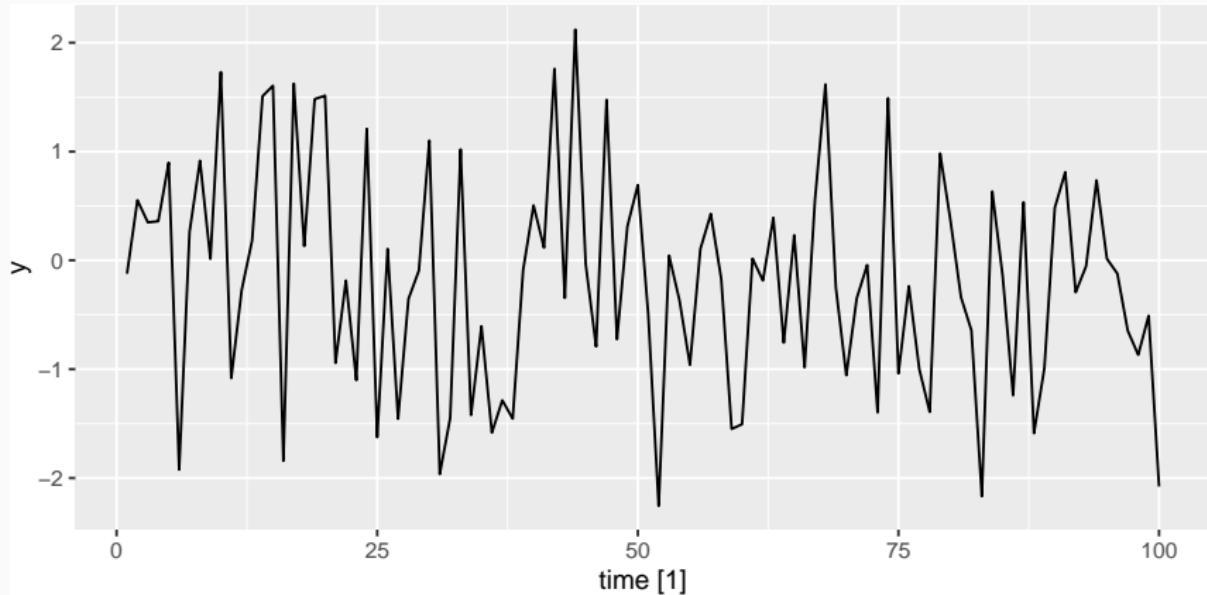
# Outline

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Time series patterns
- 5 Lag plots and autocorrelation
- 6 White noise

- White noise data are uncorrelated across time with zero mean and constant variance.
- In many models randomness is introduced by means of white noise.
- Hence when we model a series we should aim to remove all predictable components of the series leaving white noise residuals.

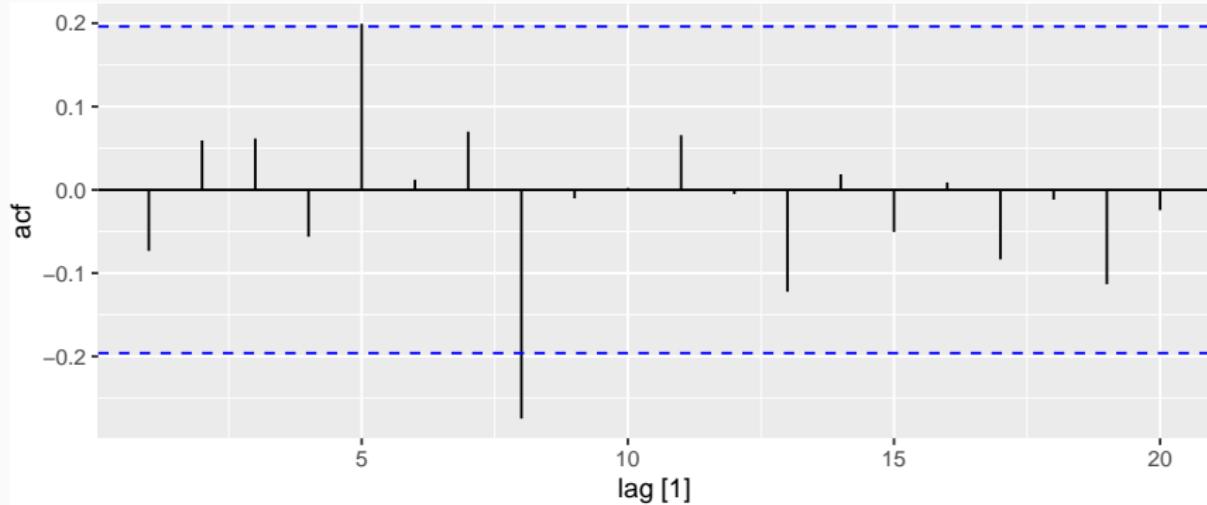
## Example

```
set.seed(2021)
wn <- tsibble(time = 1:100, y = rnorm(100), index = time)
wn %>% autoplot(y)
```



## Example

```
wn %>%  
  ACF(y) %>% autoplot()
```



- We expect each autocorrelation to be close to zero.
- The dashed blue lines can be used to check significance of the spikes in the ACF plot.

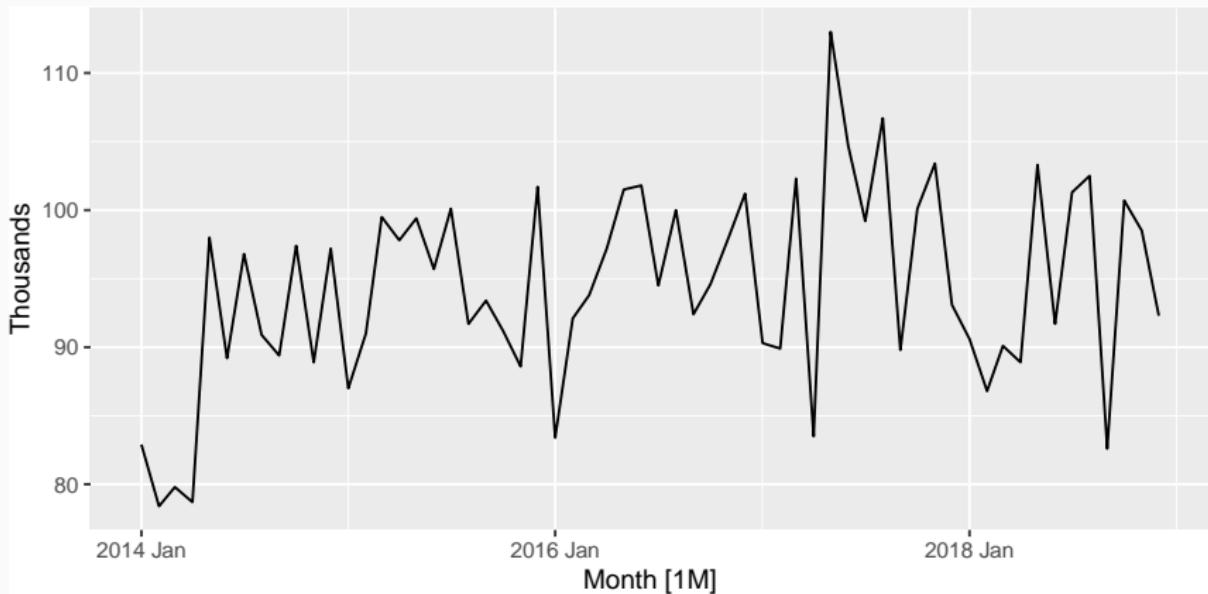
## Large sample distribution of the ACF

Sampling distribution of  $r_k$  for independent and identically distributed white noise series is asymptotically  $N(0, 1/T)$ .

- For a white noise series we would expect approximately 95% of  $r_k$  to lie within  $\pm 1.96/\sqrt{T}$ .
- If this is not the case, the series is probably not white noise.

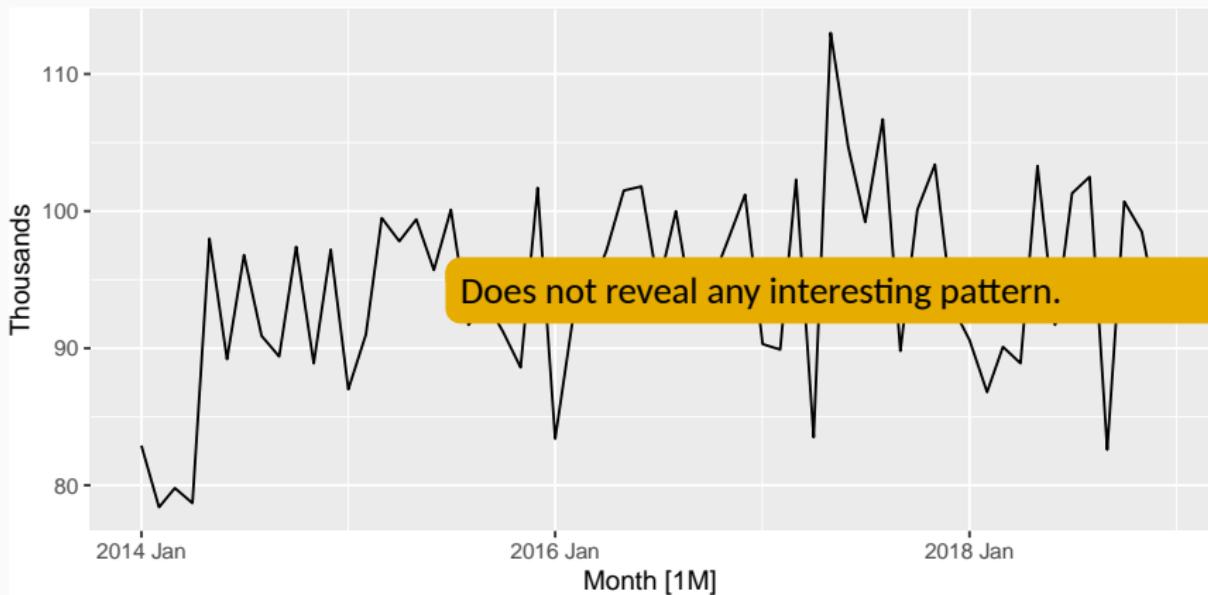
# Pigs slaughtered in Victoria

```
pigs <- aus_livestock %>%
  filter(State == "Victoria", Animal == "Pigs", year(Month) >= 2014)
pigs %>%
  autoplot(Count/1e3) +
  ylab("Thousands")
```



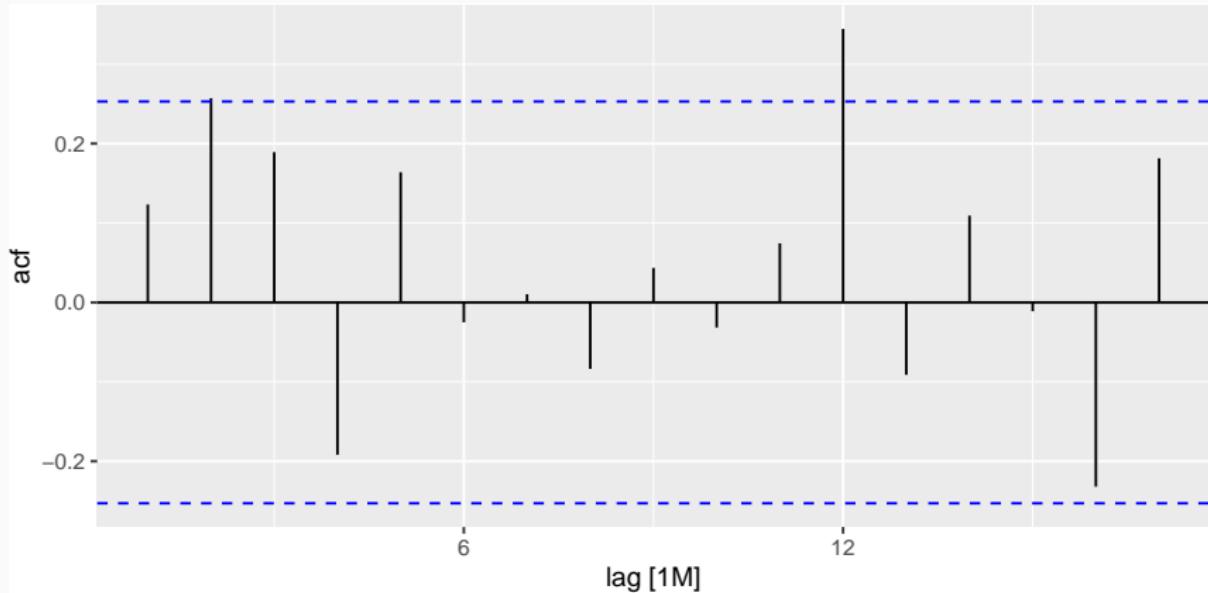
# Pigs slaughtered in Victoria

```
pigs <- aus_livestock %>%  
  filter(State == "Victoria", Animal == "Pigs", year(Month) >= 2014)  
pigs %>%  
  autoplot(Count/1e3) +  
  ylab("Thousands")
```



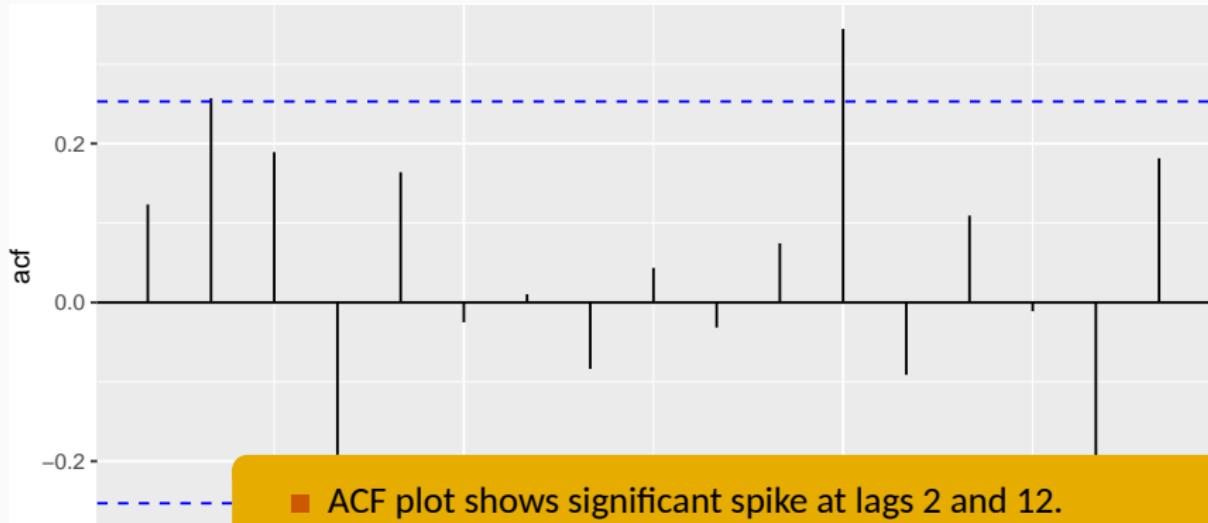
# Pigs slaughtered in Victoria

```
pigs %>%  
  ACF(Count) %>%  
  autoplot()
```



# Pigs slaughtered in Victoria

```
pigs %>%  
  ACF(Count) %>%  
  autoplot()
```



- ACF plot shows significant spike at lags 2 and 12.
- Slight seasonality is present in the data.
- The series is not white noise.

# Time series decomposition



THE UNIVERSITY OF  
**AUCKLAND**  
Te Whare Wānanga o Tāmaki Makaurau  
NEW ZEALAND

# Outline

- 1 Transformations and adjustments
- 2 Time series components
- 3 Moving averages
- 4 Classical decomposition
- 5 X-11 and SEATS decomposition
- 6 STL decomposition

# Outline

- 1 Transformations and adjustments
- 2 Time series components
- 3 Moving averages
- 4 Classical decomposition
- 5 X-11 and SEATS decomposition
- 6 STL decomposition

## Time series decomposition

- Time series data can show a variety of patterns: trend, seasonality and cycles.
- It is useful to split a time series into several components where each represents an underlying pattern category.
- When decomposing a time series into components, we usually combine the trend and cycle into a one **trend-cycle** component (simply **trend**).
- We can think that a time series is consist of three components: trend-cycle, seasonal and remainder components.
- We can expect to see more than one seasonal component, relating to different seasonal periods (eg. with those observed at least daily).
- Often time series decomposition is carried out to improve understanding of the time series, but can also be used to improve forecast accuracy.

## Transformations and adjustments

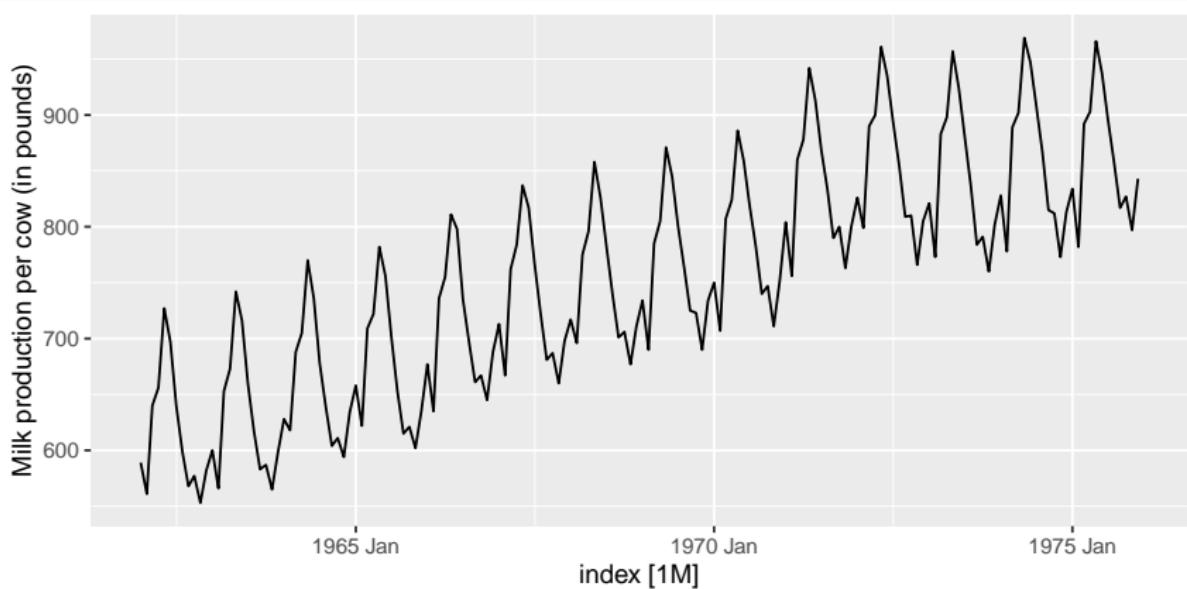
- It is sometimes helpful to transform or adjust the data first before decomposing.
- We discuss four kinds of adjustments:
  - ▶ Calendar adjustments
  - ▶ Population adjustments
  - ▶ Inflation adjustments
  - ▶ Mathematical transformations
- These adjustments and transformations simplify the patterns in data by removing known sources of variation or making the pattern consistent across the whole data set.
- Simpler patterns are usually easier to model and lead to more accurate forecasts.

## Calendar adjustments

- Some variation seen in seasonal data may be due to simple calendar effects.
- It is much easier to remove the variation before doing any further analysis.
- If we are studying monthly milk production on a farm, then there will be variation between the months due to different number of days in each month in addition to the seasonal variation across the year.
- We can remove this variation by computing average milk production per day in each month.
- A similar adjustment can be done for the total monthly sales in a retail store.

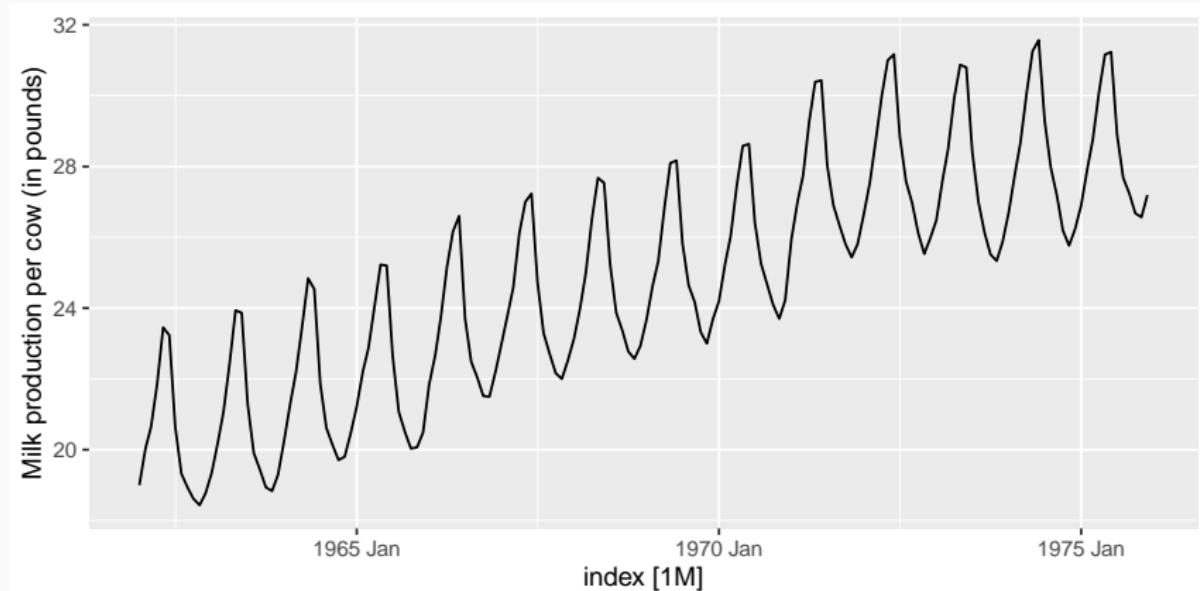
# Monthly milk production per cow

```
milk <- fma::milk %>% as_tsibble()
milk %>%
  autoplot() +
  ylab("Milk production per cow (in pounds)")
```



# Monthly milk production per cow

```
milk %>%
  mutate(days = days_in_month(index)) %>%
  mutate(value = value / days) %>%
  autoplot(value) +
  ylab("Milk production per cow (in pounds)")
```

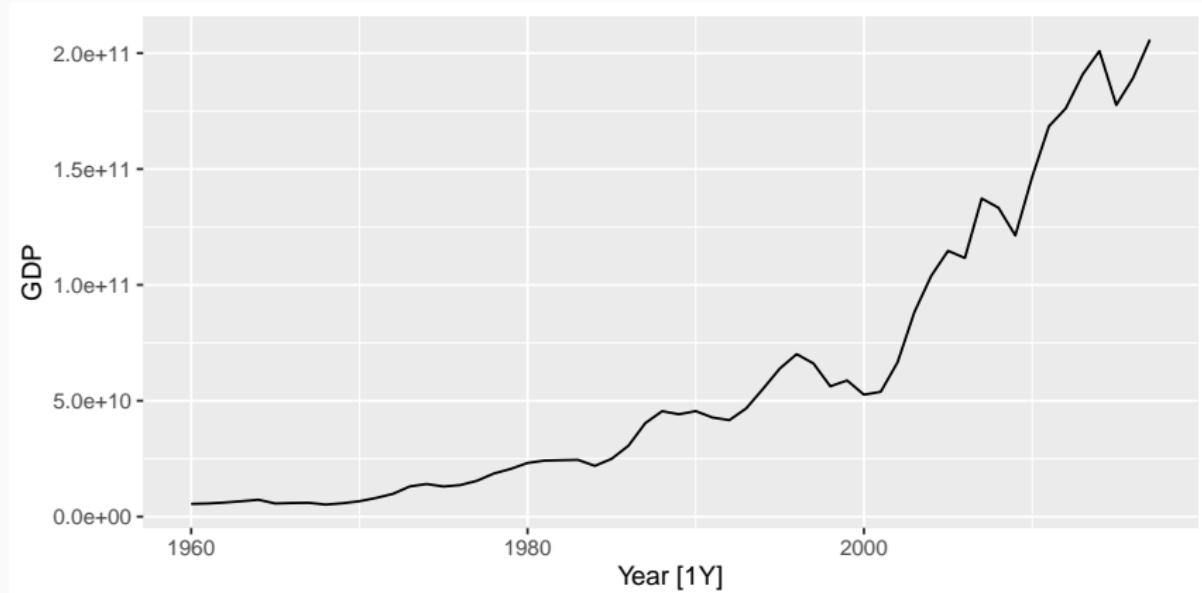


## Population adjustments

- Data that are affected by population changes can be adjusted to give per-capita data.
- We consider the data per person (or per thousand people or per million people) rather than the total.
- The number of hospital beds in a particular region over time can be affected by the population changes.
- We can remove this variation by considering the number of beds per thousand people.
- This will provide a better picture about whether there have been real increases in the number of beds, or whether the increases are due to population increases.

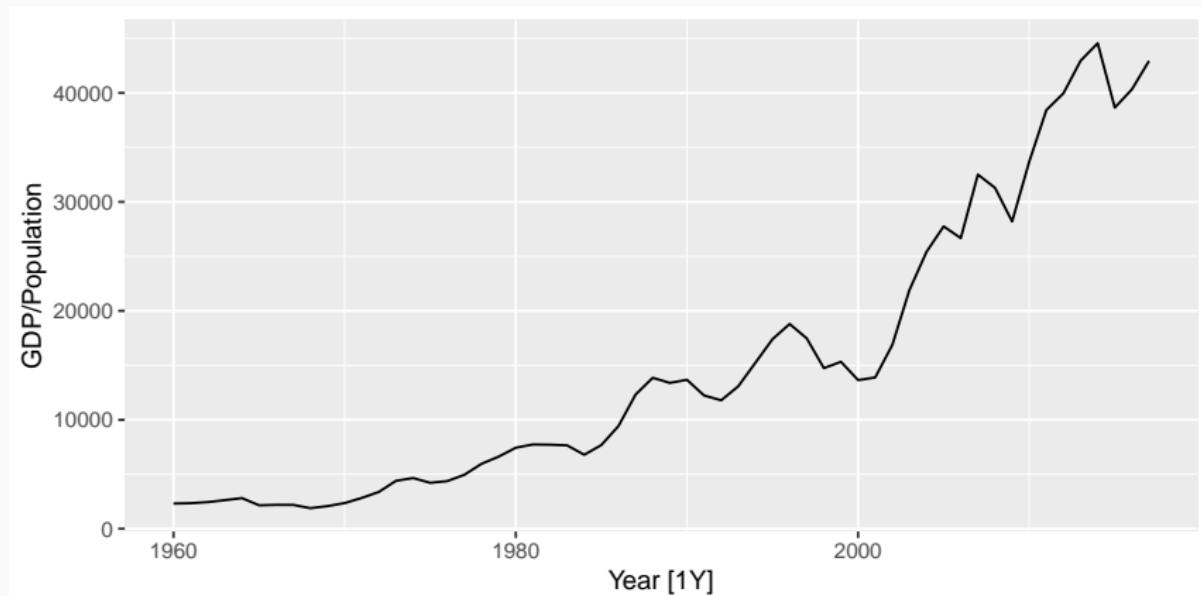
## Per capita adjustments

```
global_economy %>%  
  filter(Country == "New Zealand") %>%  
  autoplot(GDP)
```



## Per capita adjustments

```
global_economy %>%  
  filter(Country == "New Zealand") %>%  
  autoplot(GDP / Population)
```



## Inflation adjustments

- Data which are affected by the value of money are best adjusted before modelling.
- For example, the average cost of a new house in Auckland will have increased over time due to inflation.
- A house which worth \$300k this year is not the same as a \$300k house 20–30 years ago.
- Financial time series are often adjusted to represent them in dollar values from a particular year.
- The house price data may be stated in year 2000 dollar.
- The adjusted house price at year 2000 dollar value is given by

$$y_t / z_t * z_{2000},$$

$y_t$  is the original house price and  $z_t$  is the price index in year  $t$ .

- These price indexes are often constructed by government agencies.

# Inflation adjustments

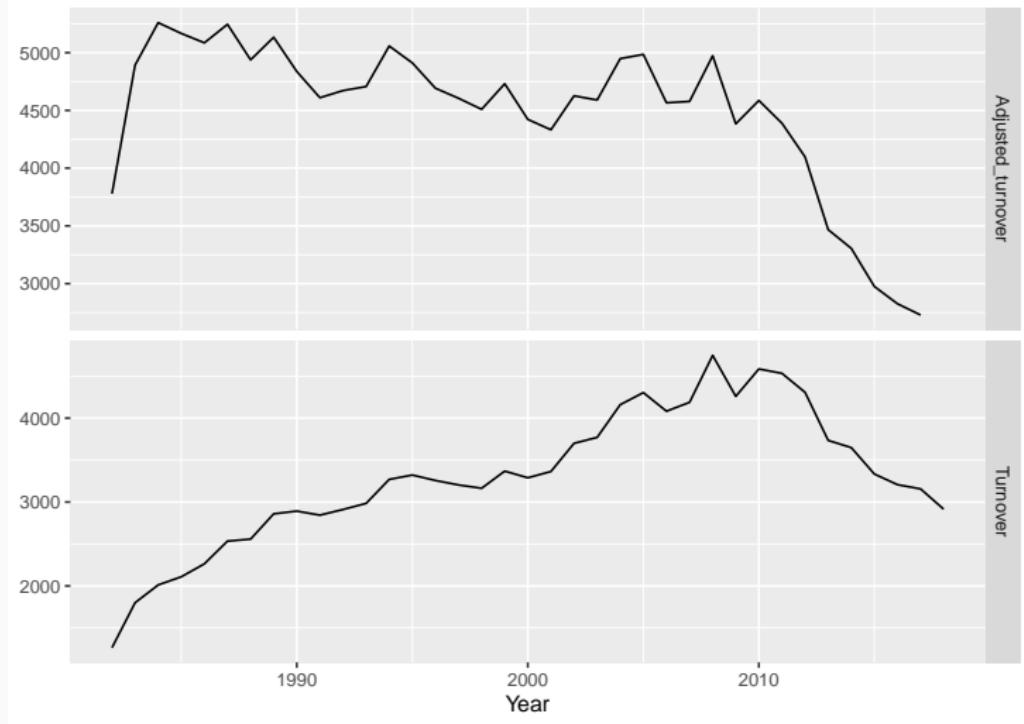
- For consumer goods, a common price index is consumer price index (CPI).

```
print_media <- aus_retail %>%
  filter(Industry == "Newspaper and book retailing") %>%
  index_by(Year = year(Month)) %>%
  summarise(Turnover = sum(Turnover))

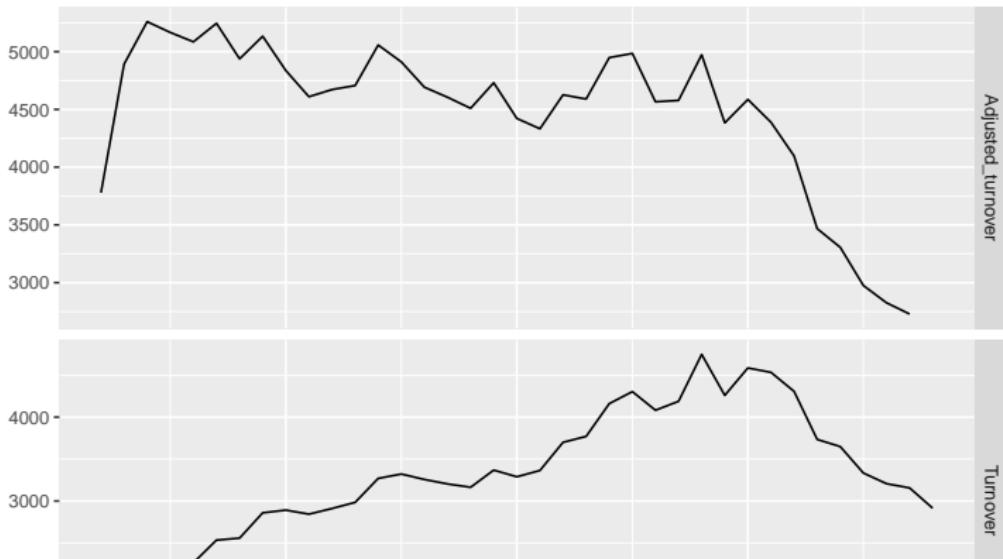
aus_economy <- global_economy %>%
  filter(Country == "Australia")

print_media %>%
  left_join(aus_economy, by = "Year") %>%
  mutate(Adjusted_turnover = Turnover / CPI * 100) %>%
  pivot_longer(c(Turnover, Adjusted_turnover),
               names_to = "Type", values_to = "Turnover") %>%
  ggplot(aes(x = Year, y = Turnover)) +
  geom_line() +
  facet_grid(Type ~ ., scales = "free_y") +
  labs(x = "Year", y = NULL)
```

# Inflation adjustments



## Inflation adjustments



- The adjusted turnover is in 2010 Australian dollars (CPI is 100 in 2010).
- Australia's newspaper and book retailing industry has started to decline much earlier than the original data suggests.

# Mathematical transformations

- A transformation can be useful if the data shows variation that increases or decreases with the level of the series.
- Denote the original observations by  $y_1, \dots, y_T$  and transformed observations by  $w_1, \dots, w_T$ .

## Mathematical transformations for stabilizing variation

Logarithm       $w_t = \log(y_t)$       ↓

Cube root       $w_t = \sqrt[3]{y_t}$       Decreasing

Square root       $w_t = \sqrt{y_t}$       strength

# Mathematical transformations

- A transformation can be useful if the data shows variation that increases or decreases with the level of the series.
- Denote the original observations by  $y_1, \dots, y_T$  and transformed observations by  $w_1, \dots, w_T$ .

## Mathematical transformations for stabilizing variation

Logarithm       $w_t = \log(y_t)$        $\downarrow$

Cube root       $w_t = \sqrt[3]{y_t}$

Square root       $w_t = \sqrt{y_t}$

These are called power

transformations as  $w_t = y_t^p$ .

## Log transformation

- Logarithm transformation is useful because it is interpretable: changes in a log value are relative (or percentage) changes on the original scale.

$$r_t = \frac{y_t - y_{t-1}}{y_{t-1}}$$

$$\frac{y_t}{y_{t-1}} = 1 + r_t$$

$$\log(y_t) - \log(y_{t-1}) = \log(1 + r_t) \approx r_t$$

### Taylor series expansion of $\log(1 + x)$

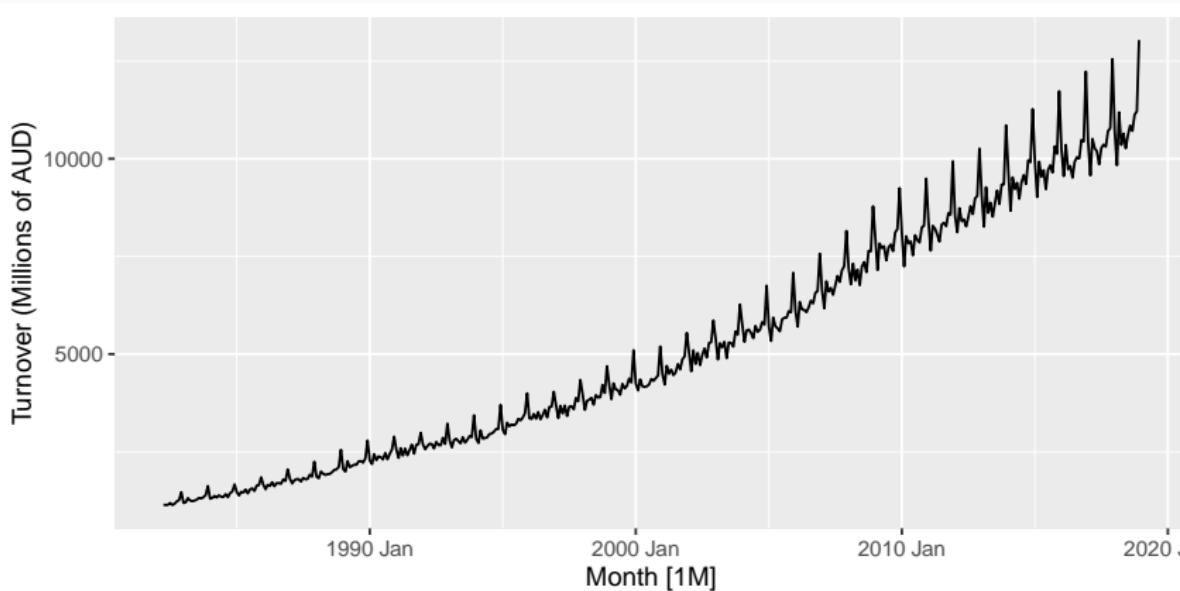
$$\log(1 + x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots, \quad \text{for } -1 < x < 1.$$

If  $x$  is near zero, then  $\log(1 + x) \approx x$ .

- It constrains the forecasts to be positive on the original scale.

# Mathematical transformation

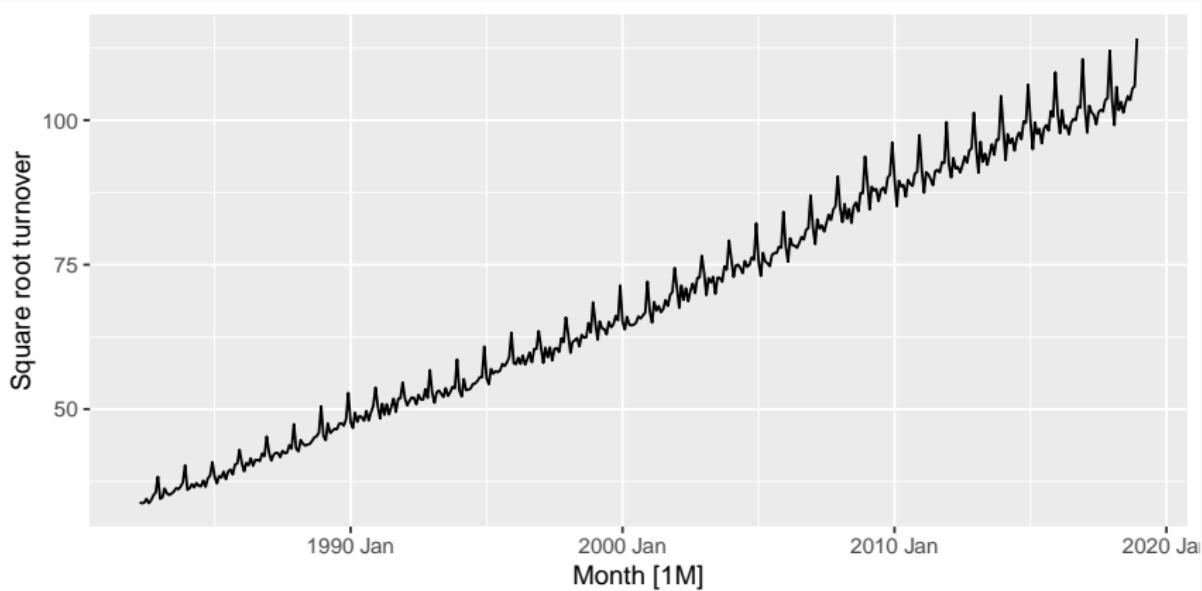
```
food <- aus_retail %>%
  filter(Industry == "Food retailing") %>%
  summarise(Turnover = sum(Turnover))
food %>%
  autoplot(Turnover) + ylab("Turnover (Millions of AUD)")
```



# Mathematical transformation

```
food %>%
```

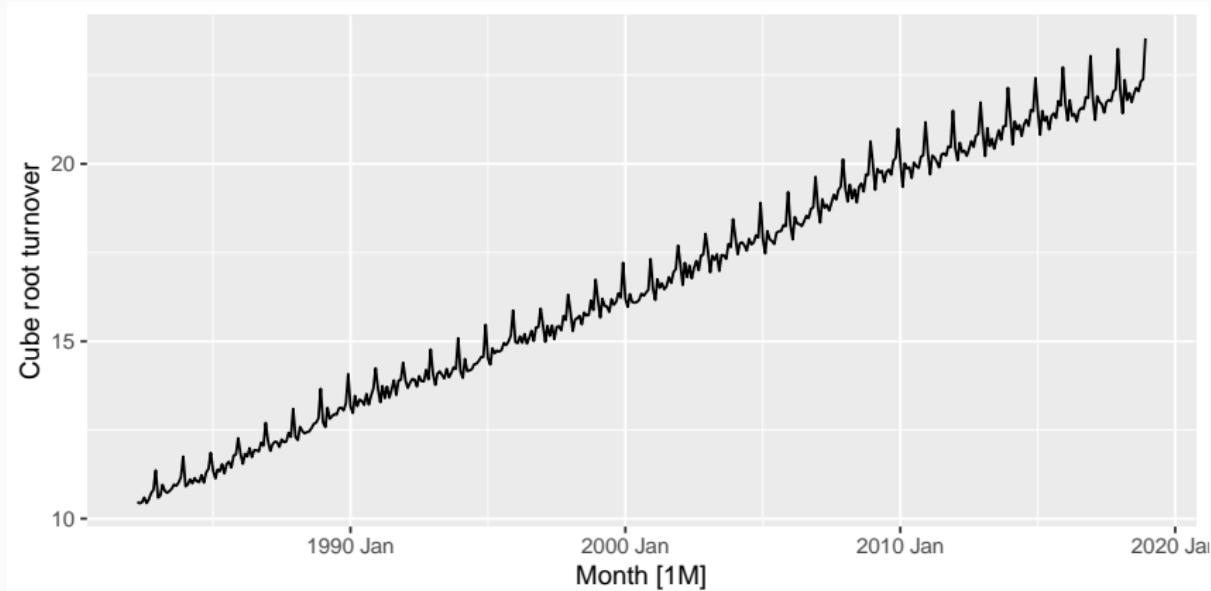
```
  autplot(sqrt(Turnover)) + ylab("Square root turnover")
```



# Mathematical transformation

```
food %>%
```

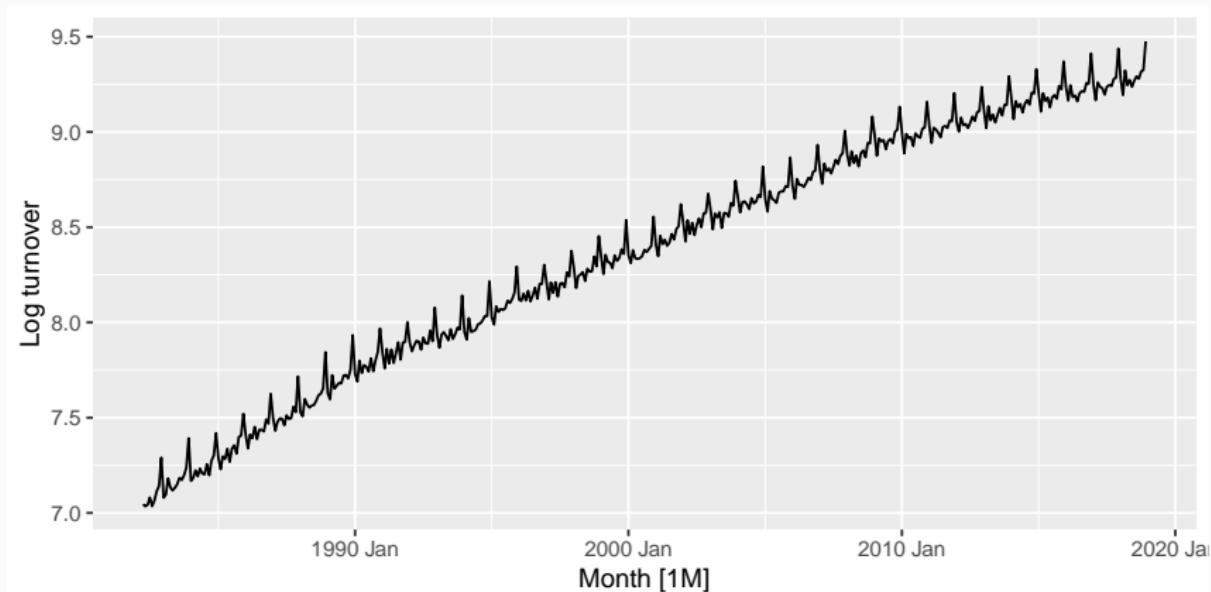
```
  autplot(Turnover^(1/3)) + ylab("Cube root turnover")
```



# Mathematical transformation

```
food %>%
```

```
  autplot(log(Turnover)) + ylab("Log turnover")
```



## Box-Cox transformations

- A useful family of transformations that includes logarithms and power transformations is the family of **Box-Cox transformations**:

### Box and Cox (1964)

For  $y_t > 0$ ,

$$w_t = \begin{cases} \log(y_t), & \lambda = 0, \\ \frac{y_t^\lambda - 1}{\lambda}, & \lambda \neq 0. \end{cases}$$

### Bickel and Doksum (1981)

For  $\lambda > 0, y_t \in \mathbb{R}$ ,

$$w_t = \frac{\text{sign}(y_t)|y_t|^\lambda - 1}{\lambda}.$$

- $\lambda = 1$ : Time series shifts vertically.
- $\lambda = \frac{1}{2}$ : Square root + linear transformation.
- $\lambda = 0$ : Natural logarithm.
- $\lambda = -1$ : Inverse + 1

## Box-Cox transformations

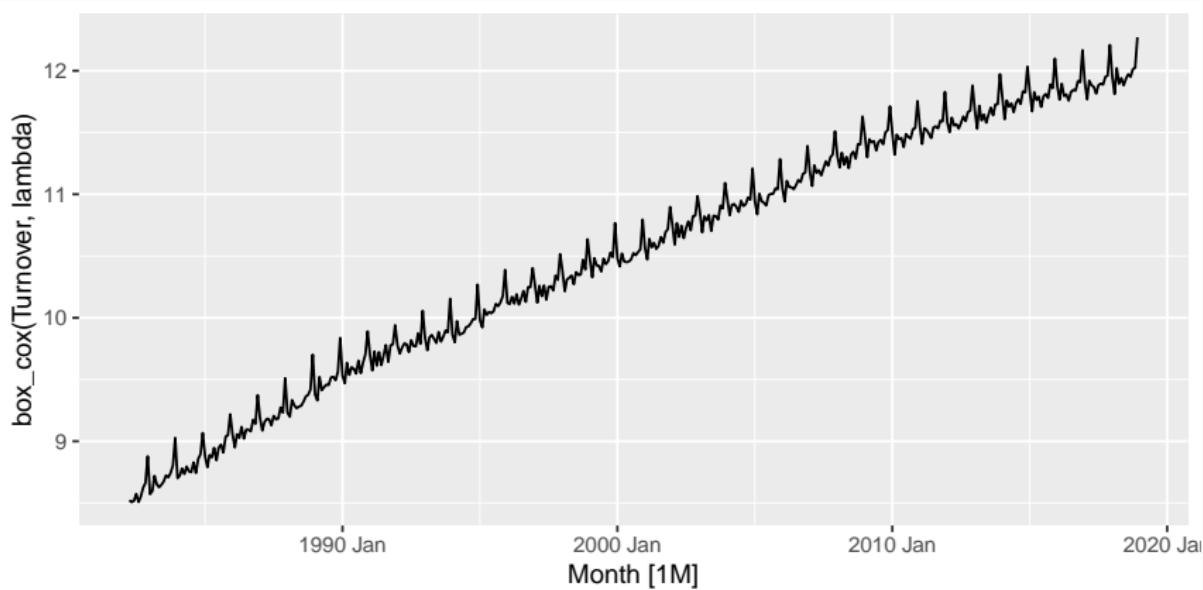
```
food %>%  
  features(Turnover, features = guerrero)
```

```
## # A tibble: 1 x 1  
##   lambda_guerrero  
##       <dbl>  
## 1      0.0524
```

- Box-Cox transformation parameter tries to balance the seasonal fluctuations and random variation across the series.

## Box-Cox transformations

```
lambda <- food %>%  
  features(Turnover, features = guerrero) %>%  
  pull(lambda_guerrero)  
food %>% autoplot(box_cox(Turnover, lambda))
```



## Transformations

- Often no transformation is needed.
- Simple transformations make explanation easier and often works well.
- If zeros are present then don't use the log transformation.
- `log1p()` can be used to handle data with zeros.
- Transformations must be reversed to produce forecasts on the original scale.
- Transformations can have large effects on prediction intervals:
  - ▶ Prediction intervals are no longer symmetric around back-transformed point forecast.
  - ▶ A low value of  $\lambda$  can give large prediction intervals.

# Outline

- 1 Transformations and adjustments
- 2 Time series components
- 3 Moving averages
- 4 Classical decomposition
- 5 X-11 and SEATS decomposition
- 6 STL decomposition

## Time series decomposition

$$y_t = f(S_t, T_t, R_t)$$

where  $y_t$  = data at period  $t$

$T_t$  = trend-cycle component at period  $t$

$S_t$  = seasonal component at period  $t$

$R_t$  = remainder component at period  $t$

**Additive decomposition:**  $y_t = S_t + T_t + R_t$ .

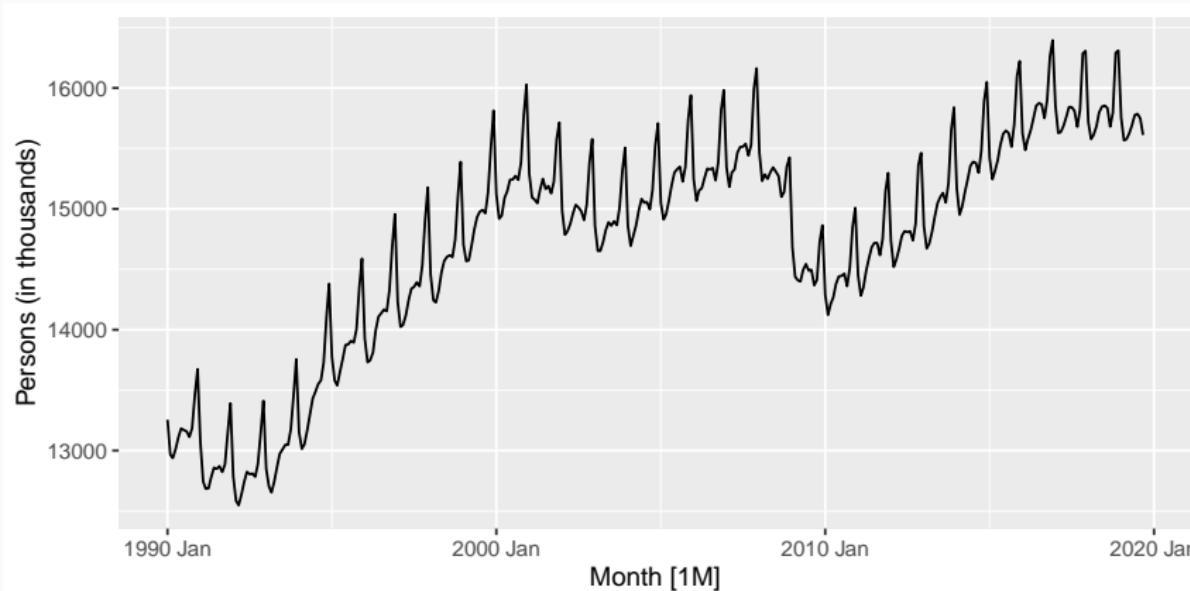
**Multiplicative decomposition:**  $y_t = S_t \times T_t \times R_t$ .

- Additive decomposition is suitable if the magnitude of the seasonal fluctuations, or the variation around the trend-cycle **does not vary** with the level.
- Multiplicative decomposition is suitable if the variation in the seasonal pattern, or the variation around the trend-cycle **does vary proportional** to the level of the series.
- Alternative to multiplicative decomposition: first transform the data until the variation in the series appears to be stable over time and then use an additive decomposition.
- Logs turn multiplicative relationship into an additive relationship.

$$y_t = S_t \times T_t \times R_t \implies \log(y_t) = \log(S_t) + \log(T_t) + \log(R_t).$$

# US retail employment

```
us_retail_employment <- us_employment %>%  
  filter(year(Month) >= 1990, Title == "Retail Trade") %>%  
  select(-Series_ID)  
us_retail_employment %>%  
  autoplot(Employed) + ylab("Persons (in thousands)")
```



## US retail employment

```
us_retail_employment %>%  
  model(stl = STL(Employed))
```

```
## # A mable: 1 x 1  
##       stl  
##     <model>  
## 1    <STL>
```

# US retail employment

```
dcmp <- us_retail_employment %>%  
  model(stl = STL(Employed))  
  components(dcmp) %>% print(n = 6)
```

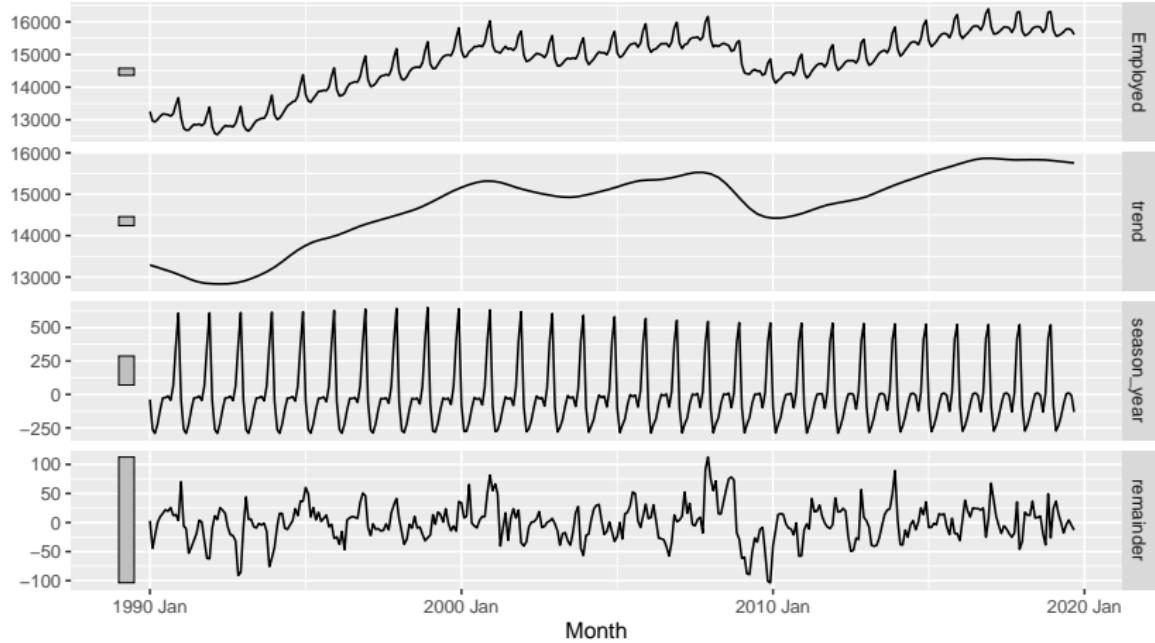
```
## # A dable:           357 x 7 [1M]  
## # Key:                 .model [1]  
## # STL Decomposition: Employed = trend + season_year +  
## #   remainder  
##   .model     Month Employed    trend season_year remainder  
##   <chr>      <mth>   <dbl>   <dbl>       <dbl>       <dbl>  
## 1 stl      1990 Jan    13256. 13291.      -38.1      3.08  
## 2 stl      1990 Feb    12966. 13272.     -261.     -44.2  
## 3 stl      1990 Mar    12938. 13252.     -291.     -23.0  
## 4 stl      1990 Apr    13012. 13233.     -221.     0.0892  
## 5 stl      1990 May    13108. 13213.     -115.      9.98  
## 6 stl      1990 Jun    13183. 13193.     -25.6     15.7  
## # ... with 351 more rows, and 1 more variable:  
## #   season_adjust <dbl>
```

# US retail employment

```
components(dcmp) %>% autoplot()
```

## STL decomposition

Employed = trend + season\_year + remainder

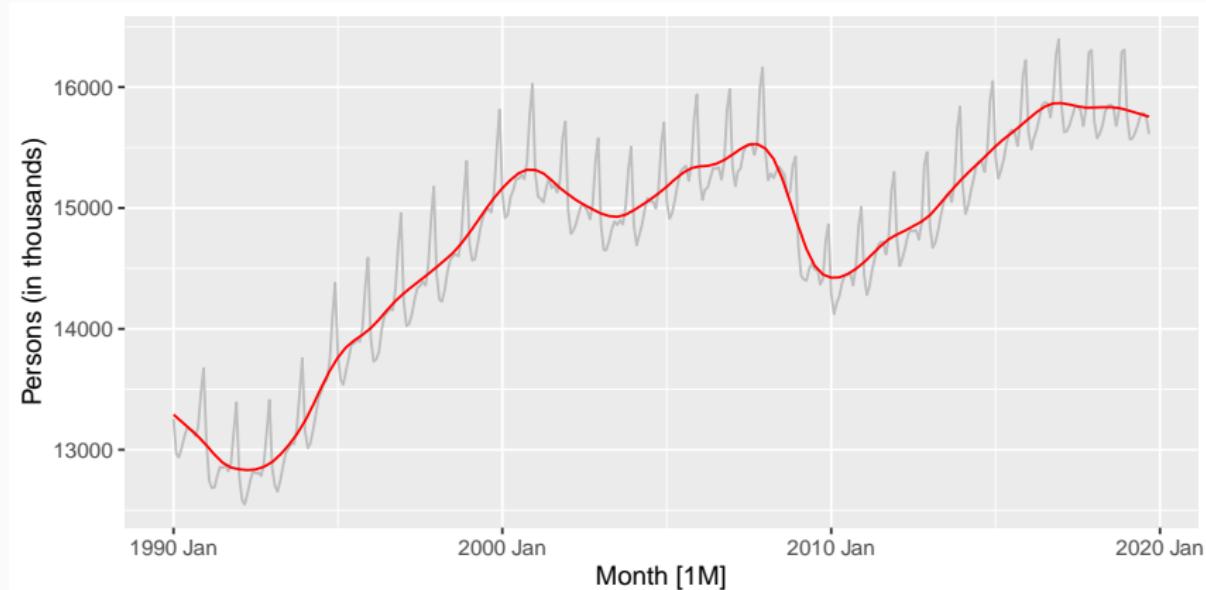


## US retail employment

- The plots show each component on different panels on its own scale.
- We need those gray bars to indicate relative scales of the components for comparison.
- Each gray bar represents the same length but because the plots are on different scales the size vary.
- The variation in the trend component is largest and remainder component is smallest.
- If we shrunk the bottom three panels until their bars became the same size as the first panel, then all the panels would be on the same scale.

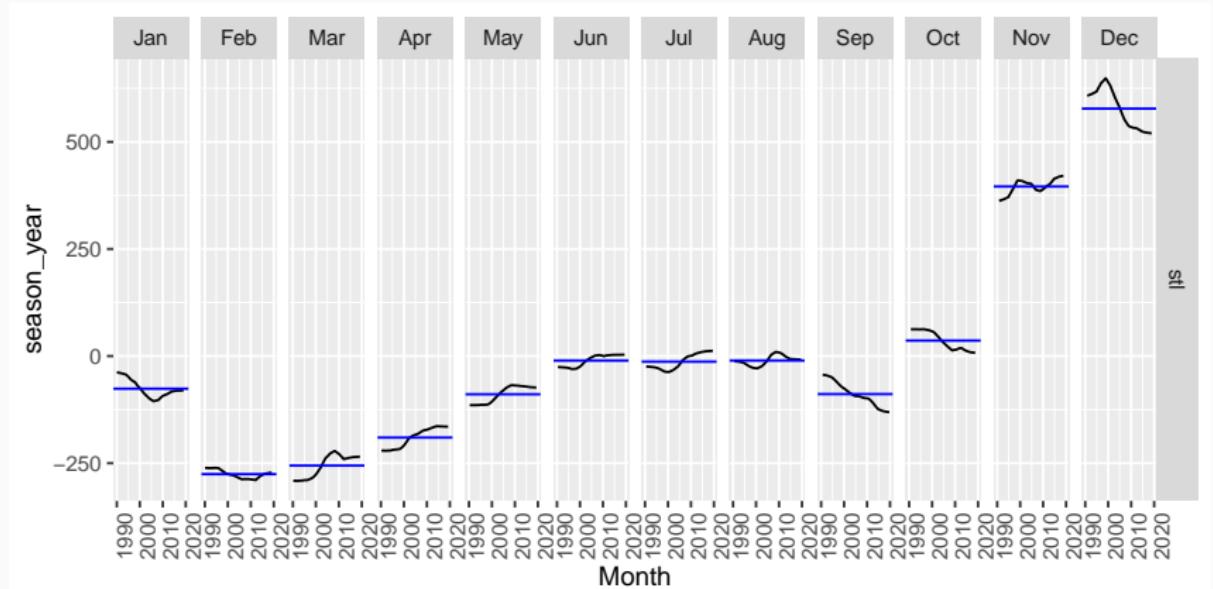
# US retail employment

```
us_retail_employment %>%
  autoplot(Employed, color = "gray") +
  autolayer(components(dcmp), trend, color = "red") +
  ylab("Persons (in thousands)")
```



# US retail employment

**components(dcmp) %>% gg\_subseries(season\_year)**



- If the variation due to seasonality is not the primary interest, we can remove the seasonal component giving **seasonally adjusted** data.
- For additive decomposition:

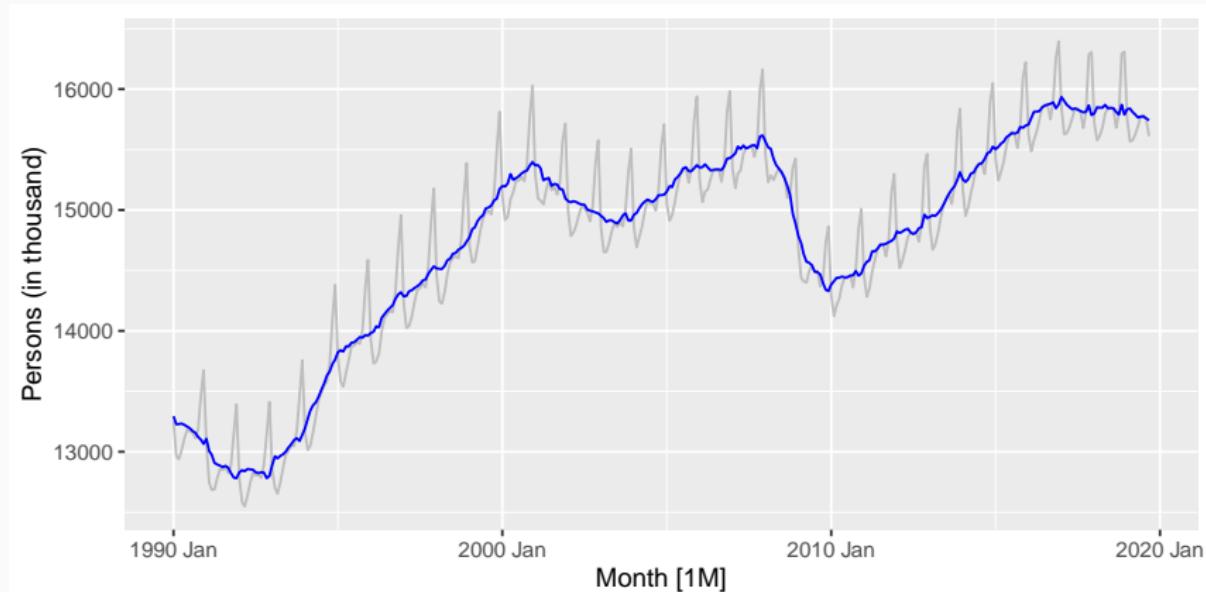
$$y_t - S_t = T_t + R_t$$

- For multiplicative decomposition:

$$y_t / S_t = T_t \times R_t$$

## Seasonal adjustment

```
us_retail_employment %>%
  autoplot(Employed, color = "gray") +
  autolayer(components(dcmp), season_adjust, color = "blue") +
  ylab("Persons (in thousand)")
```



- Seasonally adjusted data contain the trend-cycle and remainder components.
- They are **not smooth**, so upturns and downturns can be misleading.
- It is better to use the trend-cycle component to look for turning points.

# Outline

- 1 Transformations and adjustments
- 2 Time series components
- 3 Moving averages
- 4 Classical decomposition
- 5 X-11 and SEATS decomposition
- 6 STL decomposition

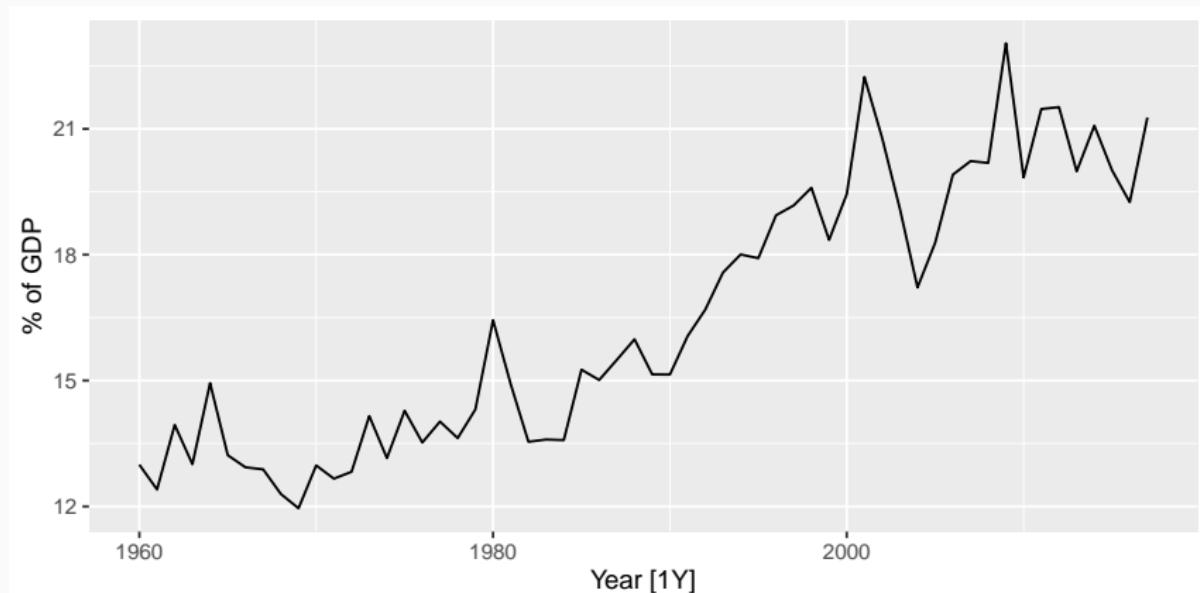
### A moving average of order $m$ ( $m$ -MA)

$$x_t = \frac{1}{m} \sum_{j=-k}^k y_{t+j}, \quad m = 2k + 1.$$

- It averages values which are within  $k$  periods from  $t$ .
- The average eliminates some of the randomness in the data.
- Useful to estimate the trend-cycle component.

## Moving averages

```
global_economy %>%  
  filter(Country == "Australia") %>%  
  autoplot(Exports) +  
  ylab("% of GDP")
```



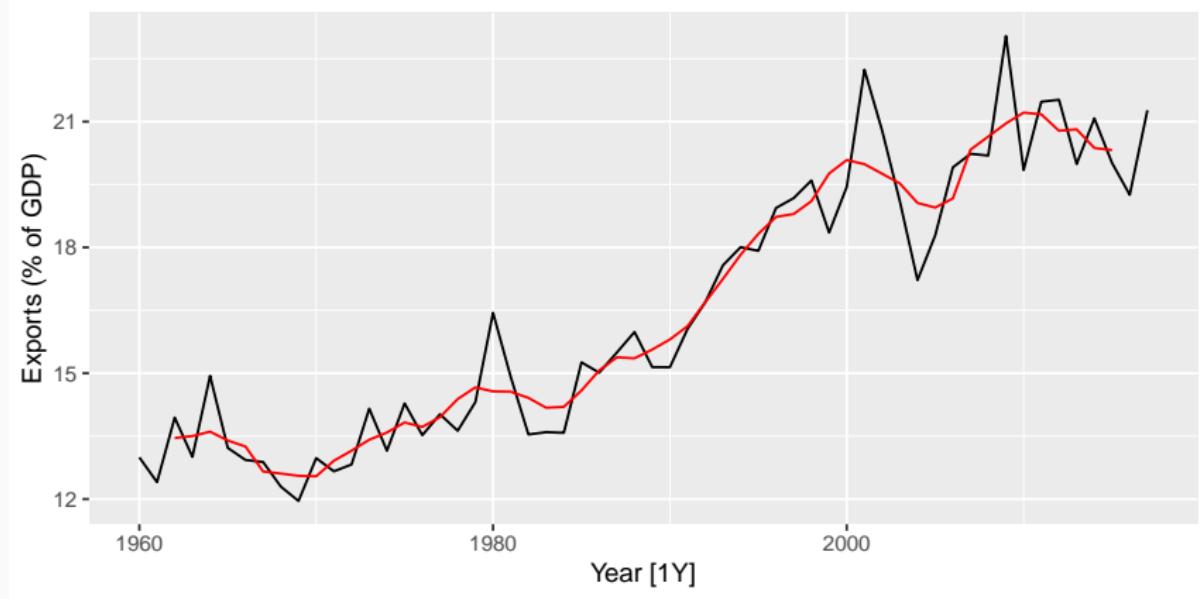
## Moving averages

Year	Exports	5-MA
1960	13.0	NA
1961	12.4	NA
1962	13.9	13.5
1963	13.0	13.5
1964	14.9	13.6
1965	13.2	13.4
2012	21.5	20.8
2013	20.0	20.8
2014	21.1	20.4
2015	20.0	20.3
2016	19.3	NA
2017	21.3	NA

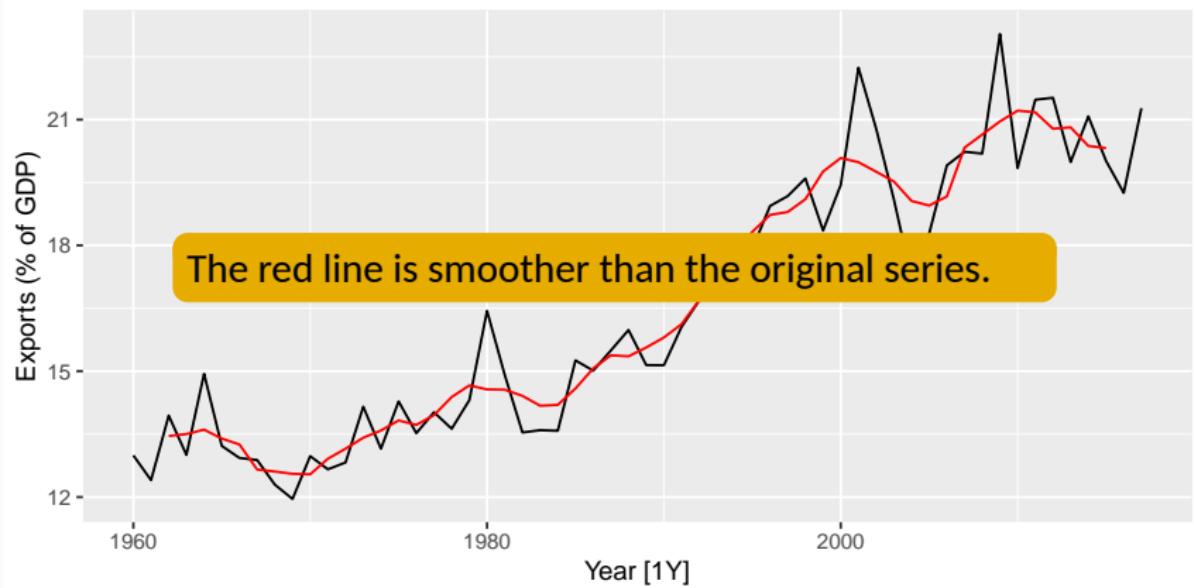
## Moving averages

```
aus_exports <- global_economy %>%
  filter(Country == "Australia") %>%
  mutate(
    `5-MA` = slider::slide dbl(Exports, mean,
      .before = 2, .after = 2, .complete = TRUE)
  )
aus_exports %>%
  autoplot(Exports) +
  autolayer(aus_exports, `5-MA`, color = "red") +
  ylab("Exports (% of GDP)")
```

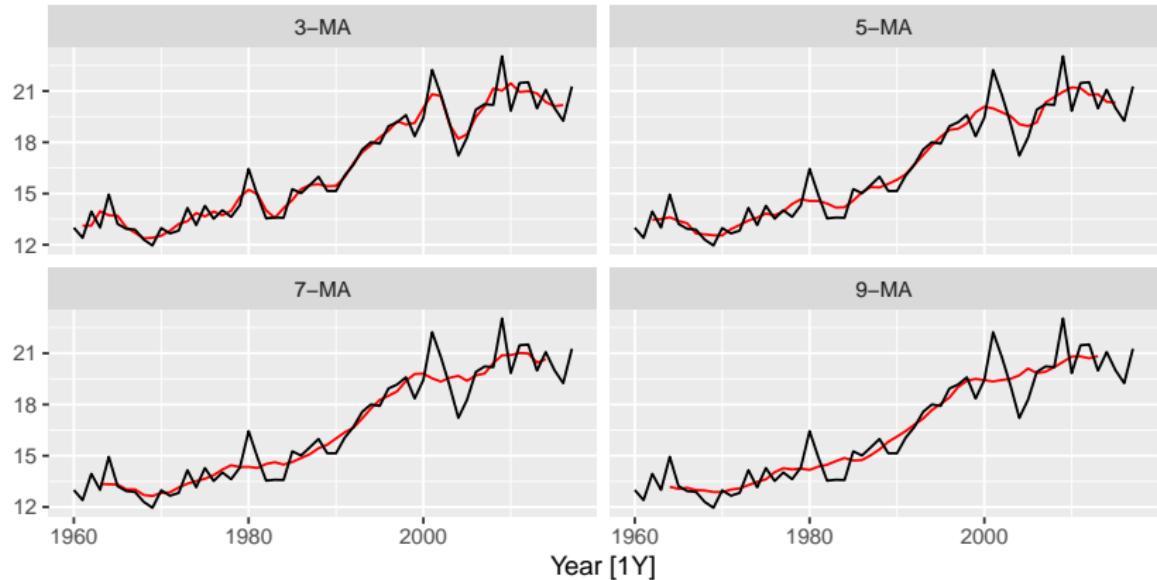
## Moving averages



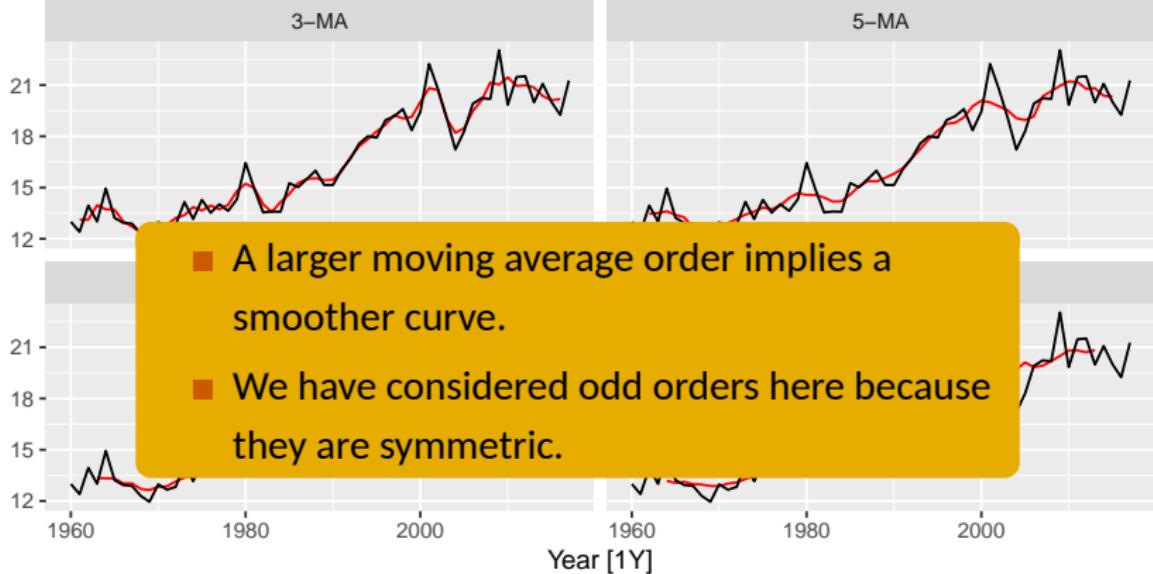
## Moving averages



## Moving averages



## Moving averages



## Moving averages of moving averages

- It is possible to apply a moving average to a moving average.
- We can use this to make an even-order moving average symmetric.
- eg. we can take a moving average of order 4, and then apply another moving average of order 2 to make it symmetric.
- This gives a weighted average of observations.

$$\begin{aligned}x_t &= \frac{1}{2} \left[ \frac{1}{4}(y_{t-2} + y_{t-1} + y_t + y_{t+1}) + \frac{1}{4}(y_{t-1} + y_t + y_{t+1} + y_{t+2}) \right] \\&= \frac{1}{8}y_{t-2} + \frac{1}{4}y_{t-1} + \frac{1}{4}y_t + \frac{1}{4}y_{t+1} + \frac{1}{8}y_{t+2}\end{aligned}$$

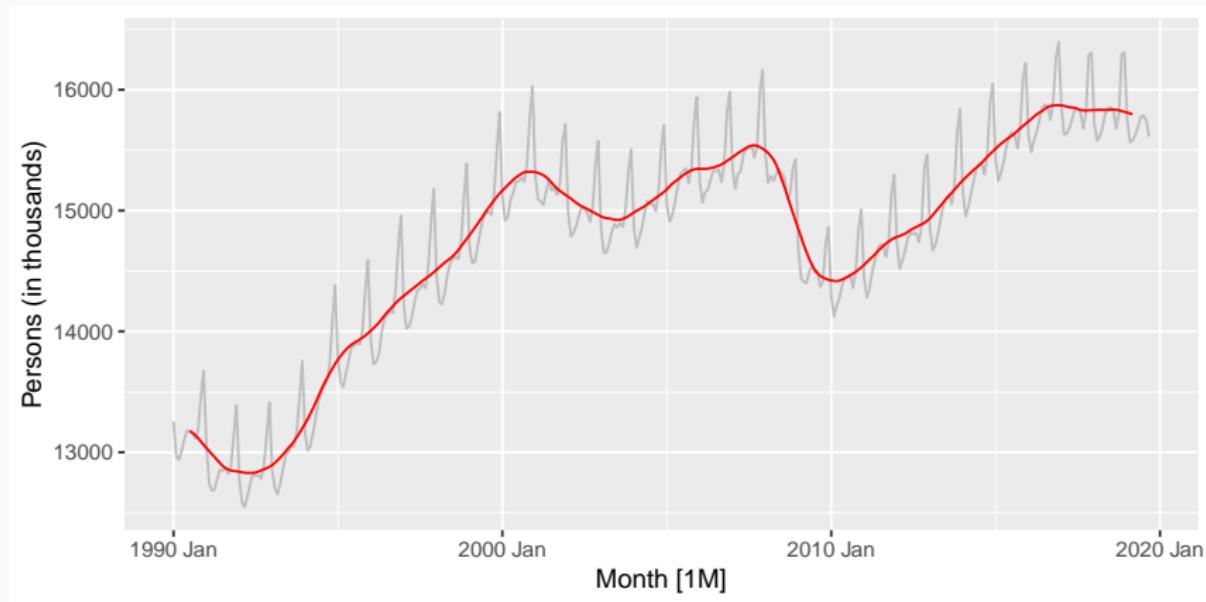
## Moving averages of moving averages

- Centred moving averages are useful to estimate the trend-cycle component from seasonal data.
- When we apply  $2 \times 4\text{-MA}$  to quarterly data, each quarter of the year receives an equal weight.
- As a result seasonal variation is averaged out.
- Same effect can be seen when using  $2 \times 8\text{-MA}$  or  $2 \times 12\text{-MA}$  to quarterly data.
- If the seasonal period is even and of order  $m$ , we use  $2 \times m\text{-MA}$ .
- If the seasonal period is odd and of order  $m$ , we use  $m\text{-MA}$ .
- eg: Monthly data ( $2 \times 12\text{-MA}$ ) and daily data (7-MA).

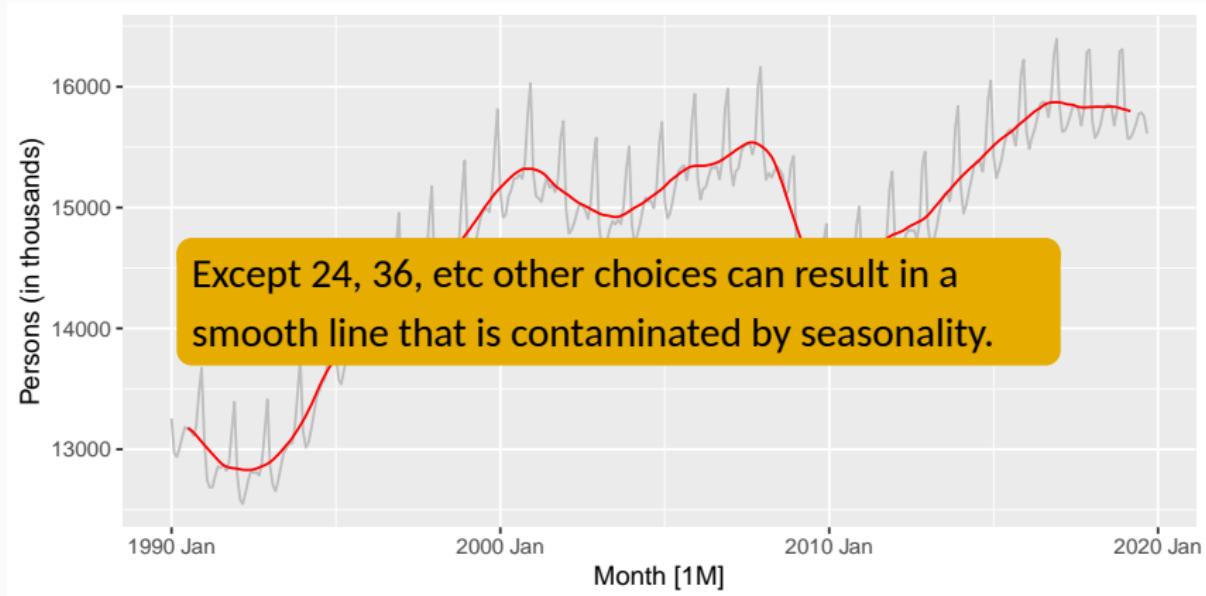
# US retail employment

```
us_retail_employment_ma <- us_retail_employment %>%
  mutate(
    `12-MA` = slider::slide dbl(Employed, mean,
                                 .before = 5, .after = 6, .complete = TRUE),
    `2x12-MA` = slider::slide dbl(`12-MA`, mean,
                                 .before = 1, .after = 0, .complete = TRUE)
  )
us_retail_employment_ma %>%
  autoplot(Employed, color = "gray") +
  autolayer(us_retail_employment_ma, `2x12-MA`, color = "red") +
  ylab("Persons (in thousands)")
```

# US retail employment



## US retail employment



# Outline

- 1 Transformations and adjustments
- 2 Time series components
- 3 Moving averages
- 4 Classical decomposition
- 5 X-11 and SEATS decomposition
- 6 STL decomposition

## Classical decomposition

- Originated in the 1920s.
- It is a relatively simple procedure.
- There are two forms: an additive decomposition and a multiplicative decomposition.
- We assume that the seasonal component doesn't vary from year to year.

# Classical decomposition

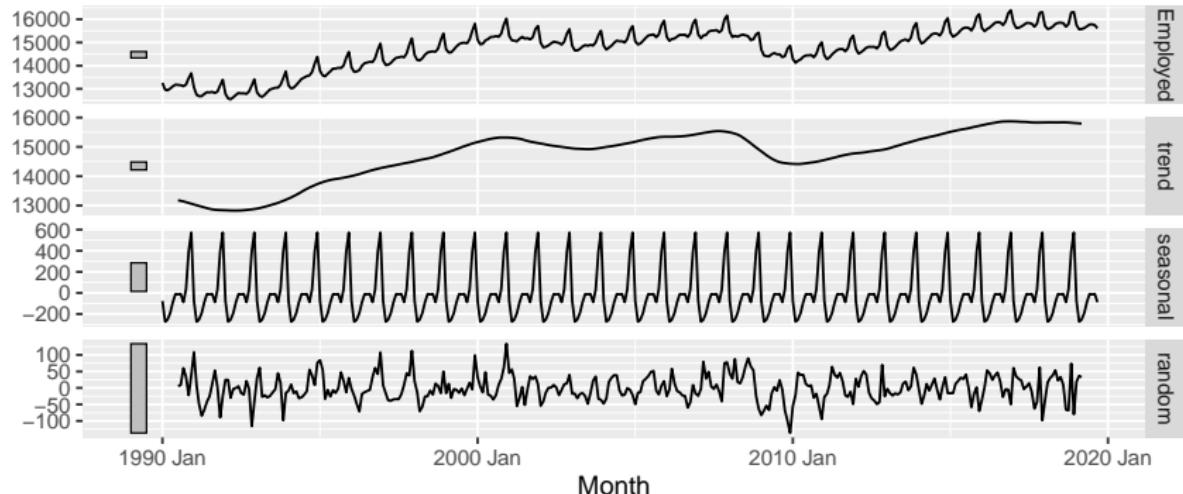
- 1 If  $m$  (seasonal period) is an even number, use  $2 \times m$ -MA to estimate the trend-cycle component ( $\hat{T}_t$ ). If  $m$  is an odd number, use  $m$ -MA.
- 2 Calculate the detrended series:
  - Additive:  $y_t - \hat{T}_t$
  - Multiplicative:  $y_t / \hat{T}_t$
- 3 Estimate the seasonal component for each season by averaging the detrended values for that season. The component values are adjusted to ensure they
  - Additive: sum to zero.
  - Multiplicative: sum to  $m$ .
- 4 Replicate the sequence for each year of data to get  $\hat{S}_t$ .
- 5 The remainder component is calculated by:
  - Additive:  $\hat{R}_t = y_t - \hat{T}_t - \hat{S}_t$ .
  - Multiplicative:  $\hat{R}_t = y_t / (\hat{T}_t \hat{S}_t)$ .

# US retail employment

```
us_retail_employment %>%  
  model(classical_decomposition(Employed, type = "additive")) %>%  
  components() %>%  
  autoplot()
```

## Classical decomposition

Employed = trend + seasonal + random



## Drawbacks of classical decomposition

- The estimate of trend-cycle is unavailable for the first few and last few observations.
- The trend-cycle estimate can over-smooth rapid changes in the data.
- This method assumes that the seasonal component is fixed and repeats from year to year.
- Not robust to unusual observations.

# Outline

- 1 Transformations and adjustments
- 2 Time series components
- 3 Moving averages
- 4 Classical decomposition
- 5 X-11 and SEATS decomposition
- 6 STL decomposition

## X-11 and X-11-ARIMA decomposition

- Original X-11 software was developed by US Census Bureau in 1960s.
- It overcomes the drawbacks of classical decomposition:
  - ▶ Trend-cycle estimates are available for all observations.
  - ▶ The seasonal component is allowed to vary with time.
  - ▶ Relatively robust to outliers.
- Poor seasonally adjusted data at the end of the series.
- Statistics Canada introduced X-11-ARIMA to address this drawback.
  - ▶ Model original series with an ARIMA model and extend the original series 1-3 years with forecasts from the ARIMA model
  - ▶ Each component is estimated with moving averages.

## X-12-ARIMA, TRAMO-SEATS and X-13-ARIMA-SEATS

- **X-12-ARIMA** can estimate deterministic components such as trading day variations, moving holiday effects, outliers and level shifts.
  - ▶ Uses regression model with ARIMA errors.
- **TRAMO-SEATS** developed at the Bank of Spain.
- TRAMO (Time series regression with ARIMA noise, missing observations and outliers)
- SEATS (Signal extraction in ARIMA time series)
- Firstly, TRAMO estimates the deterministic components via regression and removed from the original data.
- Secondly, SEATS estimates the trend-cycle and seasonal components from the ARIMA model fitted to the data from previous step.
- **X-13-ARIMA-SEATS** integrates an enhanced version of X-12-ARIMA with an enhanced version of SEATS.

- Only available for quarterly and monthly data.
- Available through seasonal package for R.
- A new function will be available soon via feasts package.

# Outline

- 1 Transformations and adjustments
- 2 Time series components
- 3 Moving averages
- 4 Classical decomposition
- 5 X-11 and SEATS decomposition
- 6 STL decomposition

## STL decomposition

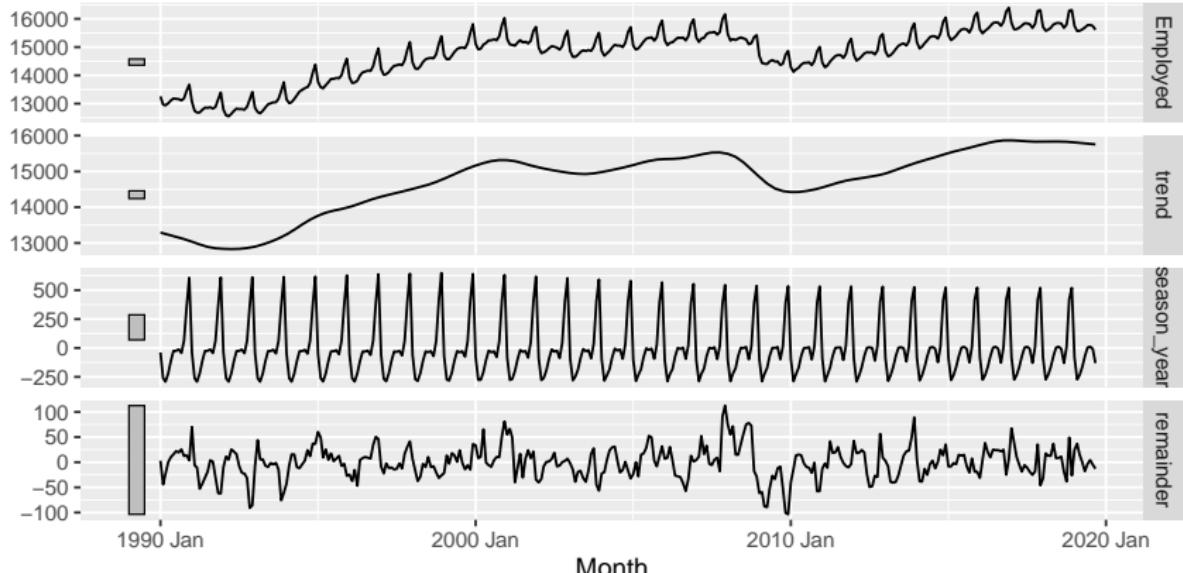
- STL: Seasonal and trend decomposition using loess.
- Very versatile and robust approach.
- Can handle any type of seasonality.
- Seasonal component can allowed to vary with time, and the rate of change can be controlled by the user.
- Smoothness of the trend-cycle can be controlled by the user.
- Robust to outliers so that seasonal and trend-cycle will not get affected. But these occasional unusual observations can affect the remainder component.
- STL does not allow for trading day or moving holiday effects.
- Decomposition is additive.
- Take logs to get multiplicative decomposition.
- Decompositions between additive and multiplicative can be obtained using a Box-Cox transformation of data where  $0 < \lambda < 1$ .

# US retail employment

```
us_retail_employment %>%  
  model(STL(Employed)) %>%  
  components() %>% autoplot()
```

## STL decomposition

Employed = trend + season\_year + remainder

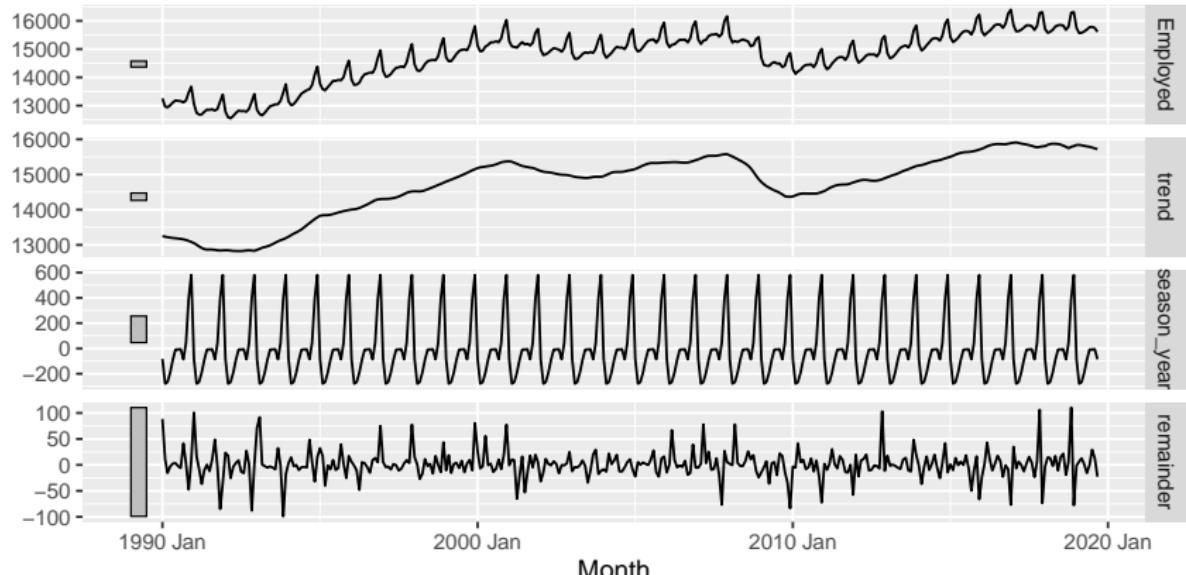


# US retail employment

```
us_retail_employment %>%  
  model(STL(Employed ~ trend(window = 7) + season(window = "periodic"),  
            robust = TRUE)) %>% components() %>% autoplot()
```

## STL decomposition

Employed = trend + season\_year + remainder



## STL decomposition

- Trend and seasonal windows should be odd numbers.
- `trend(window = ?)` controls wiggliness of the trend component.
- Trend window is the number of consecutive observations to be used for estimating the trend-cycle.
- `season(window = ?)` controls variation on the seasonal component.
- Season window is the number of consecutive years to be used for estimating each value in the seasonal component.
- `season(window = "periodic")` is equivalent to an infinite window.
- Smaller windows allow for more rapid changes.

## STL decomposition

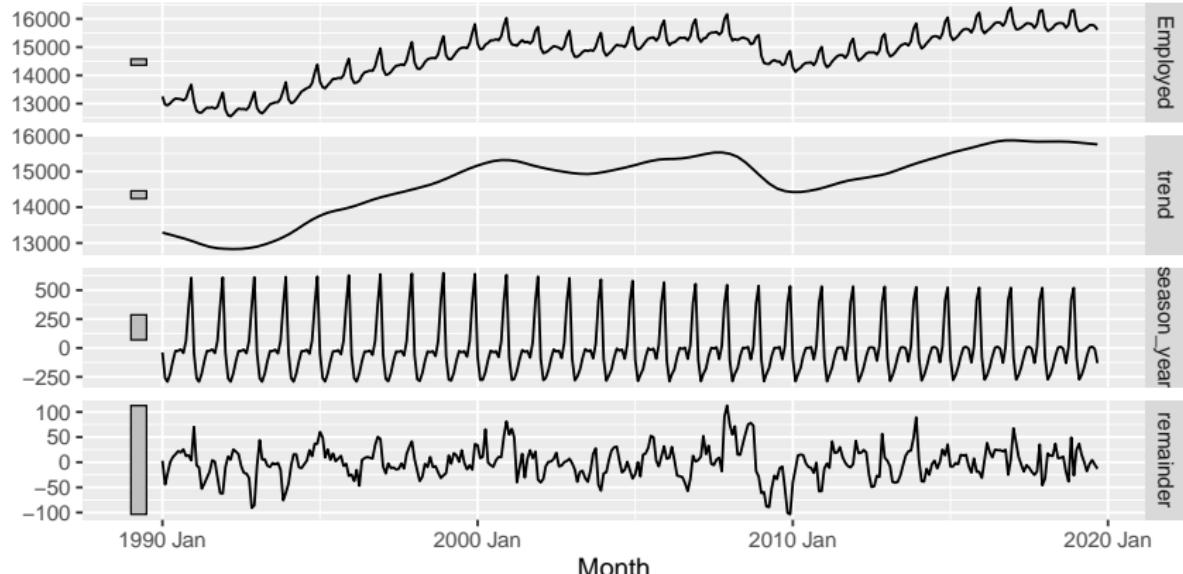
- STL algorithm updates trend and seasonal components iteratively.
- Starts with  $\hat{T}_t = 0$ .
- Uses a mixture of loess and moving averages to refine the trend and seasonal estimates.
- Trend window controls loess bandwidth applied to deseasonalized data.
- Seasonal window controls loess bandwidth applied to detrended subsamples.
- Robustness weights are based on remainder.
- Default season window = 13.
- Default trend window = `nextodd(ceiling((1.5 * period)/(1 - (1.5/s.window))))`.

# US retail employment

```
us_retail_employment %>%  
  model(STL(Employed)) %>%  
  components() %>% autoplot()
```

## STL decomposition

Employed = trend + season\_year + remainder

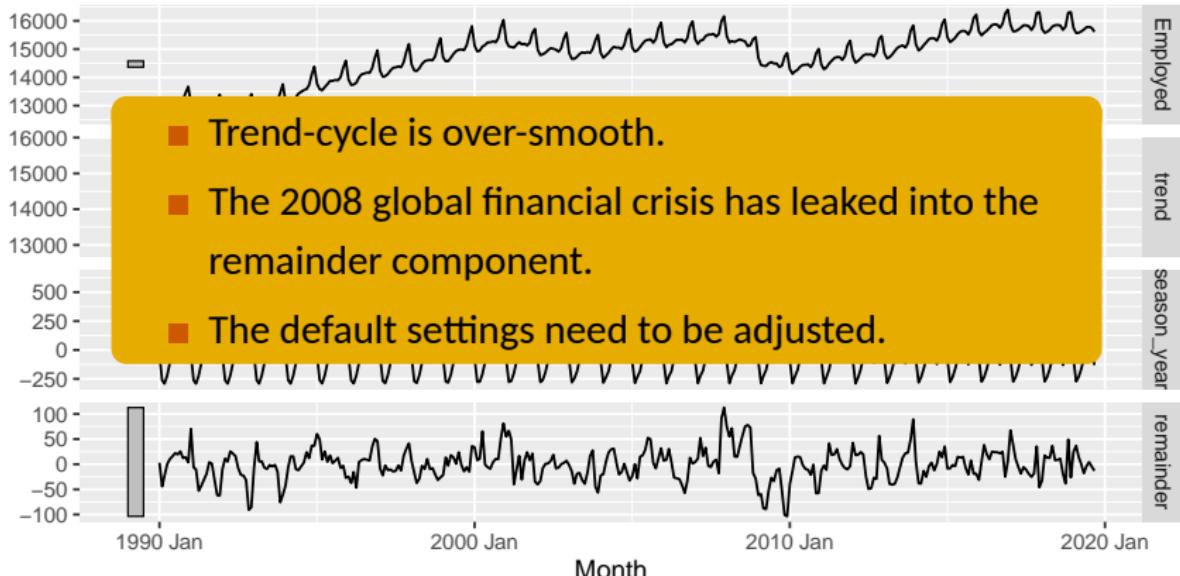


# US retail employment

```
us_retail_employment %>%  
  model(STL(Employed)) %>%  
  components() %>% autoplot()
```

## STL decomposition

Employed = trend + season\_year + remainder

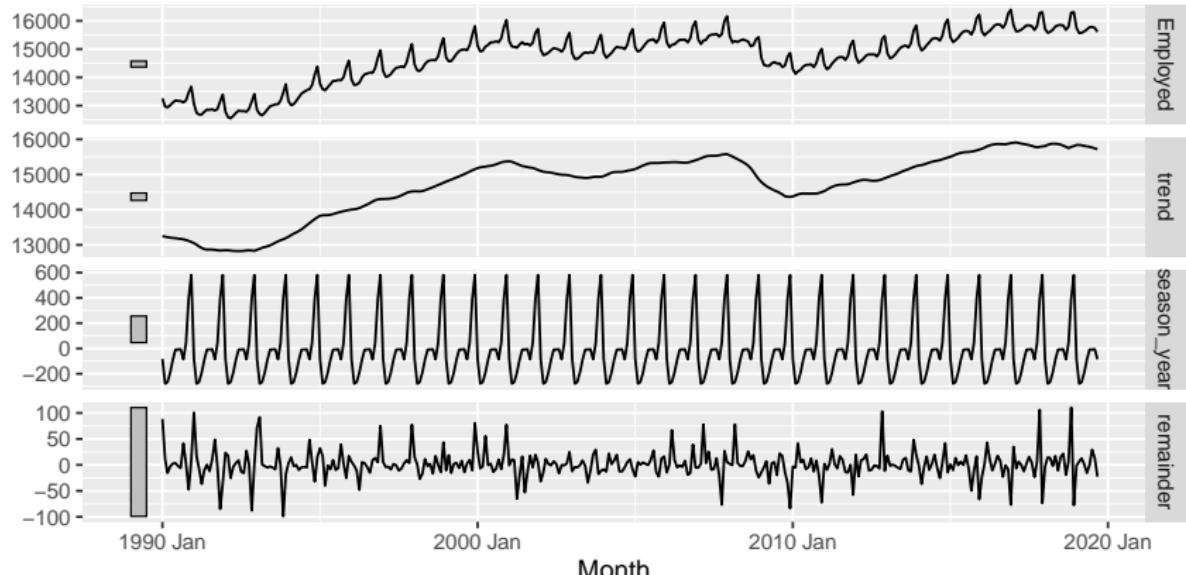


# US retail employment

```
us_retail_employment %>%  
  model(STL(Employed ~ trend(window = 7) + season(window = "periodic"),  
            robust = TRUE)) %>% components() %>% autoplot()
```

## STL decomposition

Employed = trend + season\_year + remainder





THE UNIVERSITY OF  
**AUCKLAND**  
Te Whare Wānanga o Tāmaki Makaurau  
NEW ZEALAND

# Forecaster's toolbox

# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

# Outline

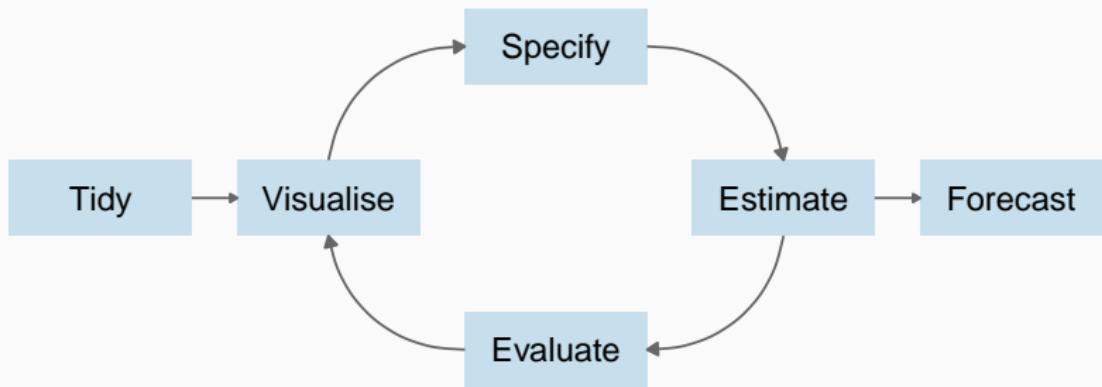
- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

# A tidy forecasting workflow

The steps to produce a set of forecasts for time series data:

- 1 Data preparation
- 2 Data visualization
- 3 Specify a model
- 4 Estimate the model
- 5 Accuracy & performance evaluation
- 6 Produce forecasts

# A tidy forecasting workflow



## Data preparation

- The process may involve loading the data, identifying missing values, outliers, filtering the time series, and other pre-processing.
- Different models have different data requirements: some require the series to be in time order, others require no missing values.
- These investigations are essential and always done prior to model fitting.
- `tsibble` and other packages in tidyverse can substantially simplify this step.

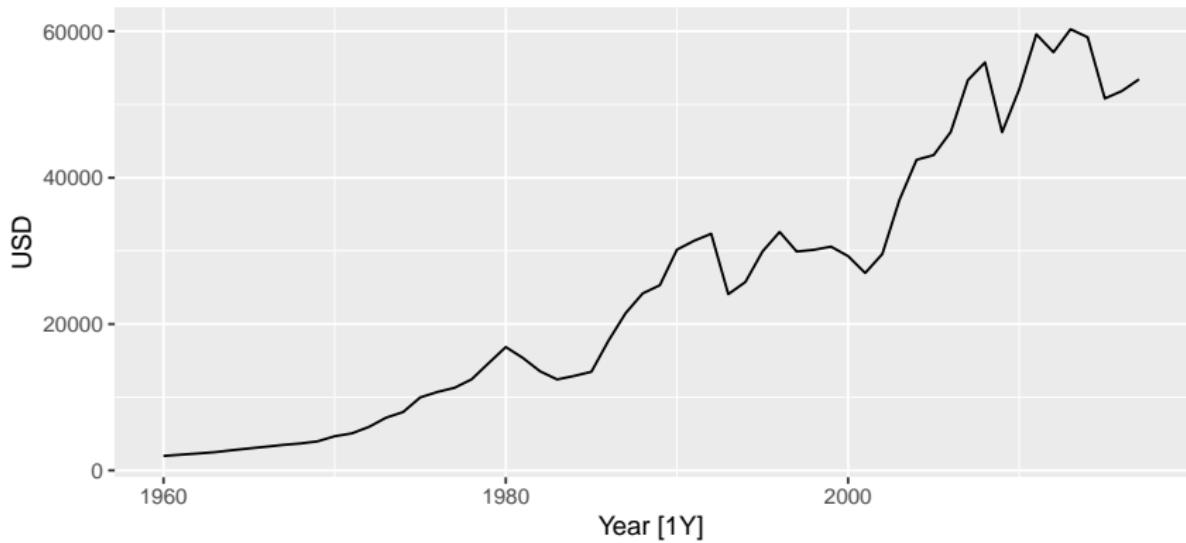
# Data visualization

- Allows you to identify time series patterns, outliers, relationship with other variables etc.
- Based on these features we can specify an appropriate model.

```
gdppc <- global_economy %>%
  mutate(GDP_per_capita = GDP / Population)
gdppc %>%
  filter(Country == "Sweden") %>%
  autoplot(GDP_per_capita) +
  labs(y = "USD", title = "GDP per capita for Sweden")
```

# Data visualization

GDP per capita for Sweden



## Specify a model

- Specifying an appropriate model for data is important for producing appropriate forecasts.
- Models in fable package are specified using model functions, where each use a formula ( $y \sim x$ ) interface.
  - ▶ LHS: the response variable
  - ▶ RHS: structure of the model
- Example: `TSLM(GDP_per_capita ~ trend())`
- The model function `TSLM` fits linear model.
- The structure is build using `trend()` (a “special” function specifying a linear trend when used with `TSLM`).
- The “special” functions used to define model’s structure can vary between models.
- The left hand side supports transformations to simplify the time series patterns.

## Estimate the model

```
fit <- gdppc %>%
  model(trend_model = TSLM(GDP_per_capita ~ trend()))
fit
```

```
## # A mable: 263 x 2
## # Key:      Country [263]
##   Country          trend_model
##   <fct>            <model>
## 1 Afghanistan      <TSLM>
## 2 Albania          <TSLM>
## 3 Algeria          <TSLM>
## 4 American Samoa   <TSLM>
## 5 Andorra          <TSLM>
## 6 Angola           <TSLM>
## 7 Antigua and Barbuda <TSLM>
## 8 Arab World        <TSLM>
## 9 Argentina         <TSLM>
## 10 Armenia          <TSLM>
## # ... with 253 more rows
```

## Estimate the model

```
fit <- gdppc %>%  
  model(trend_model = TSLM(GDP_per_capita ~ trend()))  
fit
```

```
## # A mable: 263 x 2  
## # Key:      Country [263]  
##   Country          trend_model  
##   <fct>            <model>  
## 1 Afghanistan      <TSLM>  
## 2 Albania          <TSLM>  
## 3 Algeria          <TSLM>  
## 4 American Samoa    <TSLM>  
## 5 Andorra          <TSLM>  
## 6 Austria          <TSLM>  
## 7 Azerbaijan       <TSLM>  
## 8 Bahrain          <TSLM>  
## 9 Argentina         <TSLM>  
## 10 Armenia          <TSLM>  
## # ... with 253 more rows
```

■ This is a model table (or mable).

■ trend\_model contains information about each fitted model.

## Evaluate the model

- It is important to check how well the model has performed on the data.
- Several diagnostic tools are available to assess the fit of the model.
- We can also compute accuracy measures to compare against different models.
- We will discuss these later.

## Produce forecasts

- When the fit of the model is adequate, we can produce forecasts using `forecast()`.
- You need to specify how many future observations to forecast.
- Example: `forecast(h = 10)` or `forecast(h = "10 years")`.

```
fit %>% forecast(h = 3)
```

```
## # A fable: 789 x 5 [1Y]
## # Key:     Country, .model [263]
##   Country      .model     Year GDP_per_capita .mean
##   <fct>        <chr>     <dbl>    <dist>     <dbl>
## 1 Afghanistan trend_model 2018     N(526, 9653) 526.
## 2 Afghanistan trend_model 2019     N(534, 9689) 534.
## 3 Afghanistan trend_model 2020     N(542, 9727) 542.
## 4 Albania      trend_model 2018     N(4716, 476419) 4716.
## 5 Albania      trend_model 2019     N(4867, 481086) 4867.
## 6 Albania      trend_model 2020     N(5018, 486012) 5018.
## 7 Algeria      trend_model 2018     N(4410, 643094) 4410.
```

## Produce forecasts

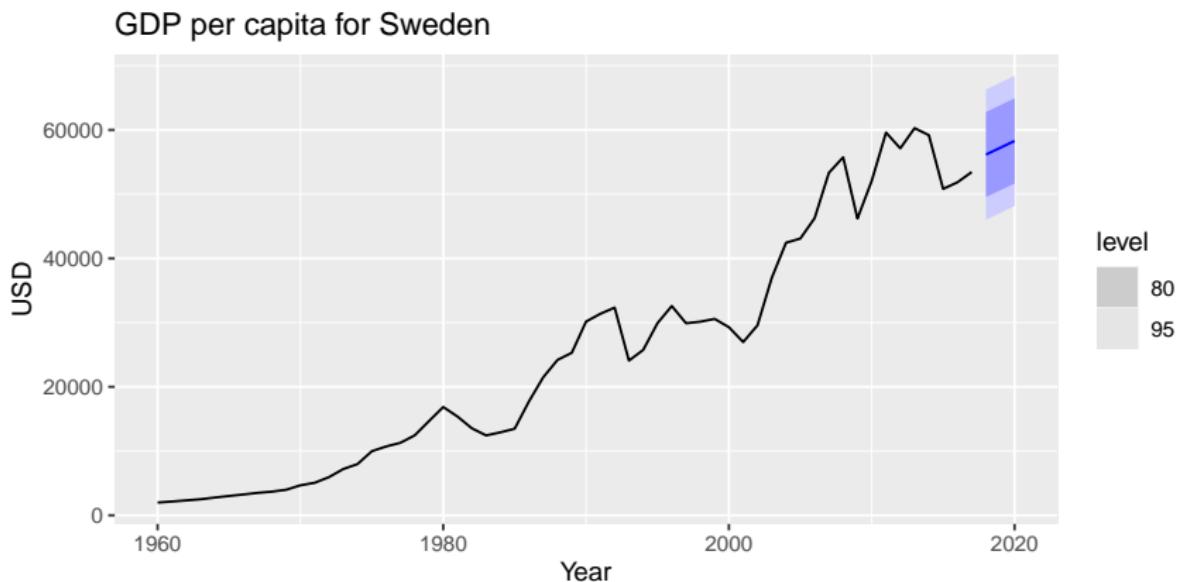
- When the fit of the model is adequate, we can produce forecasts using `forecast()`.
- You need to specify how many future observations to forecast.
- Example: `forecast(h = 10)` or `forecast(h = "10 years")`.

```
fit %>% forecast(h = 3)
```

```
## # A fable: 789 x 5 [1Y]
## # Key:     Country, .model [263]
##   Country      .model      Year    GDP_per_capita  .mean
##   <fct>        <chr>       <dbl>      <dist>    <dbl>
## 
##   ■ This is a forecast table (fable).
## 
##   ■ Each row corresponds to one forecast period for each country.
## 
##   ■ GDP_per_capita column contains the forecast distribution.
## 
##   ■ .mean column gives the point forecast (mean of the forecast
##     distribution).
```

# Visualizing forecasts

```
fit %>%
  forecast(h = "3 years") %>%
  filter(Country == "Sweden") %>%
  autoplot(gdppc) +
  labs(y = "USD", title = "GDP per capita for Sweden")
```



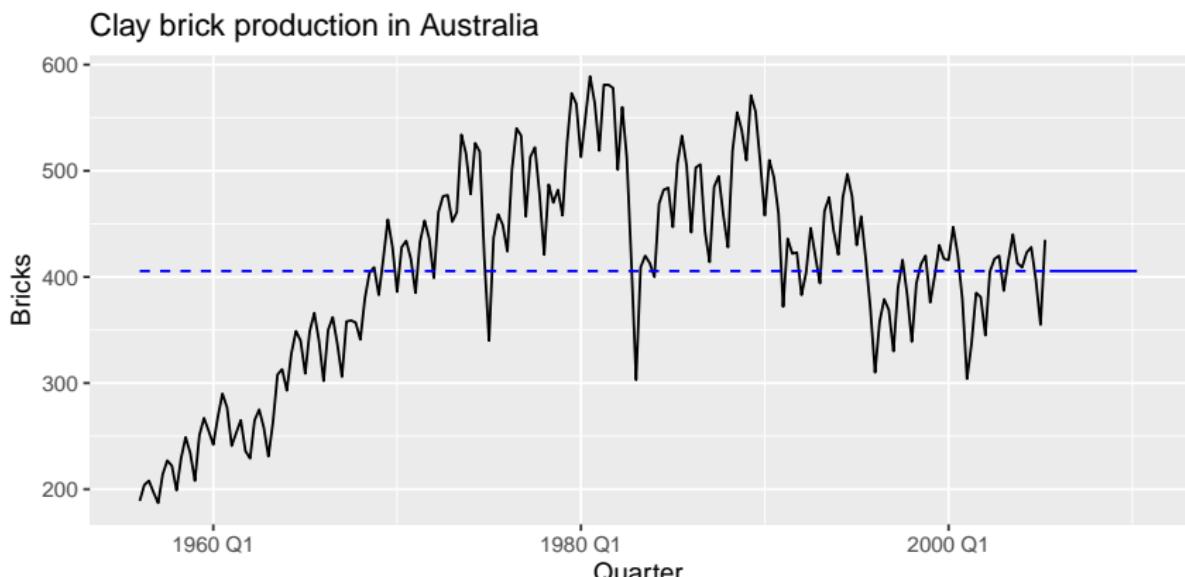
# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

# Simple forecasting methods

## MEAN( $y$ ): Average method

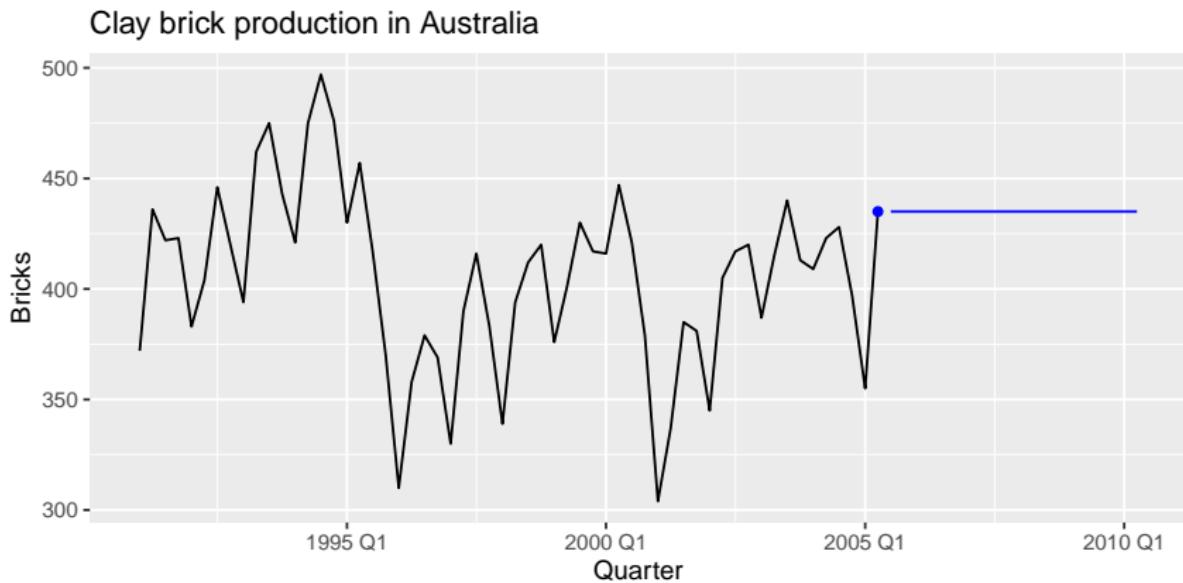
- Forecasts of all future values are equal to the mean of the historical data.
- $\hat{y}_{T+h|T} = \bar{y} = (y_1 + y_2 + \dots + y_T)/T.$



# Simple forecasting methods

## NAIVE( $y$ ): Naïve method

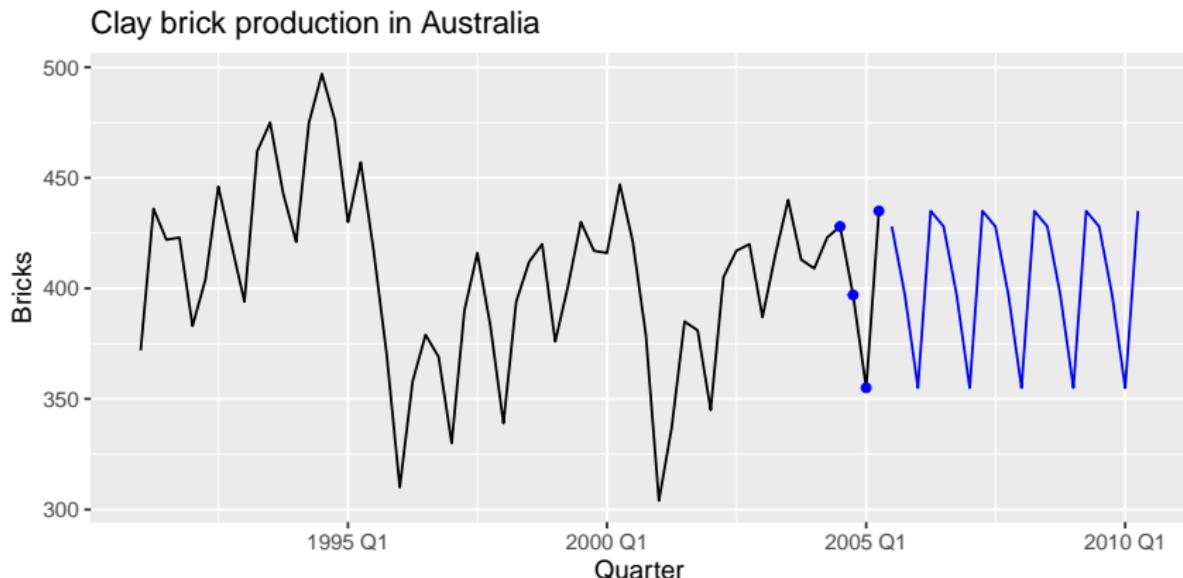
- Forecasts of all future values are equal to the last observation.
- $\hat{y}_{T+h|T} = y_T$



## Simple forecasting methods

### SNAIVE( $y \sim \text{lag}(m)$ ): Seasonal naïve method

- Forecasts equal to the last observed value from the same season.
- $\hat{y}_{T+h|T} = y_{T+h-m(k+1)}$ .
- $m$ : seasonal period and  $k$ : integer part of  $(h - 1)/m$



### RW(y ~ drift()): Random walk with a drift

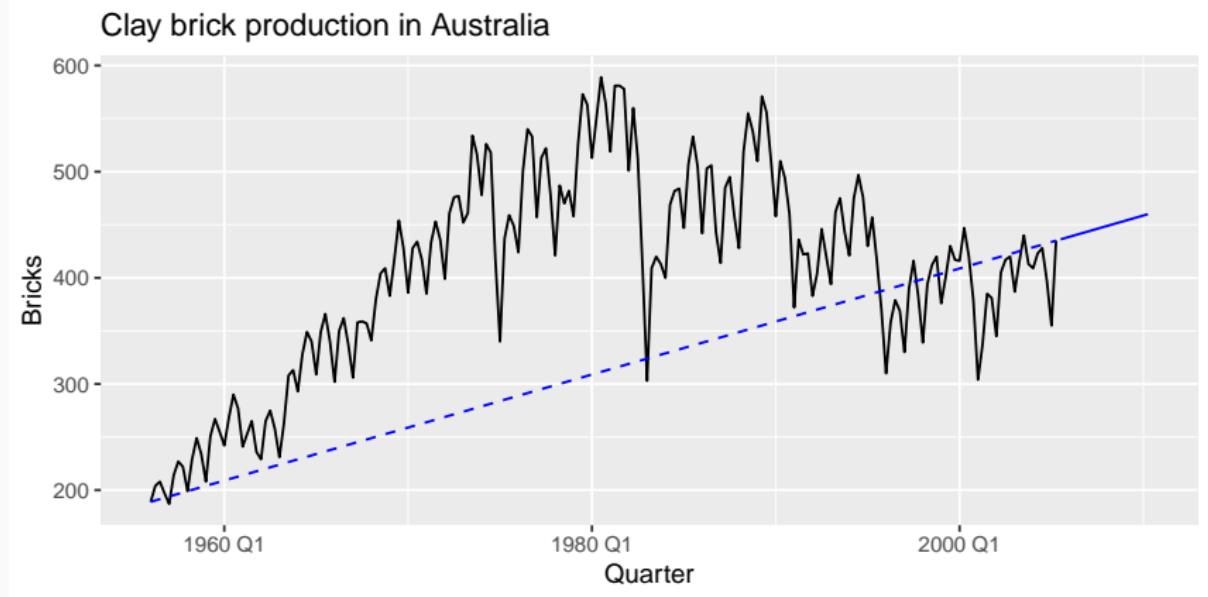
- Forecasts equal to the last value plus average change.



$$\begin{aligned}\hat{y}_{T+h|T} &= y_T + \frac{h}{T-1} \sum_{t=2}^T (y_t - y_{t-1}) \\ &= y_T + \frac{h}{T-1} (y_T - y_1)\end{aligned}$$

- Same as extrapolating a line drawn between the first and last observations.

## Simple forecasting methods



## Model fitting

- The `model()` function fits models to the data.

```
brick_fit <- aus_production %>%  
  filter(!is.na(Bricks)) %>%  
  model(mean = MEAN(Bricks),  
        naive = NAIVE(Bricks),  
        drift = RW(Bricks ~ drift()),  
        seasonal_naive = SNAIVE(Bricks))
```

```
## # A mable: 1 x 4  
##      mean    naive        drift seasonal_naive  
##      <model> <model>       <model>       <model>  
## 1  <MEAN> <NAIVE> <RW w/ drift> <SNAIVE>
```

## Model fitting

- The `model()` function fits models to the data.

```
brick_fit <- aus_production %>%  
  filter(!is.na(Bricks)) %>%  
  model(mean = MEAN(Bricks),  
        naive = NAIVE(Bricks),  
        drift = RW(Bricks ~ drift()),  
        seasonal_naive = SNAIVE(Bricks))
```

```
## # A mable: 1 x 4  
##      mean    naive      drift seasonal_naive  
##      <model> <model>      <model>      <model>  
## 1  <MEAN> <NAIVE> <RW w/ drift>      <SNAIVE>
```

A `mable` is a model table and each cell corresponds to a fitted model.

# Forecasting

```
brick_fc <- brick_fit %>%  
  forecast(h = "5 years")  
  
## # A fable: 80 x 4 [1Q]  
## # Key:      .model [4]  
##   .model Quarter       Bricks .mean  
##   <chr>    <qtr>       <dist> <dbl>  
## 1 mean    2005 Q3 N(405, 9294) 405.  
## 2 mean    2005 Q4 N(405, 9294) 405.  
## 3 mean    2006 Q1 N(405, 9294) 405.  
## 4 mean    2006 Q2 N(405, 9294) 405.  
## 5 mean    2006 Q3 N(405, 9294) 405.  
## 6 mean    2006 Q4 N(405, 9294) 405.  
## # ... with 74 more rows
```

# Forecasting

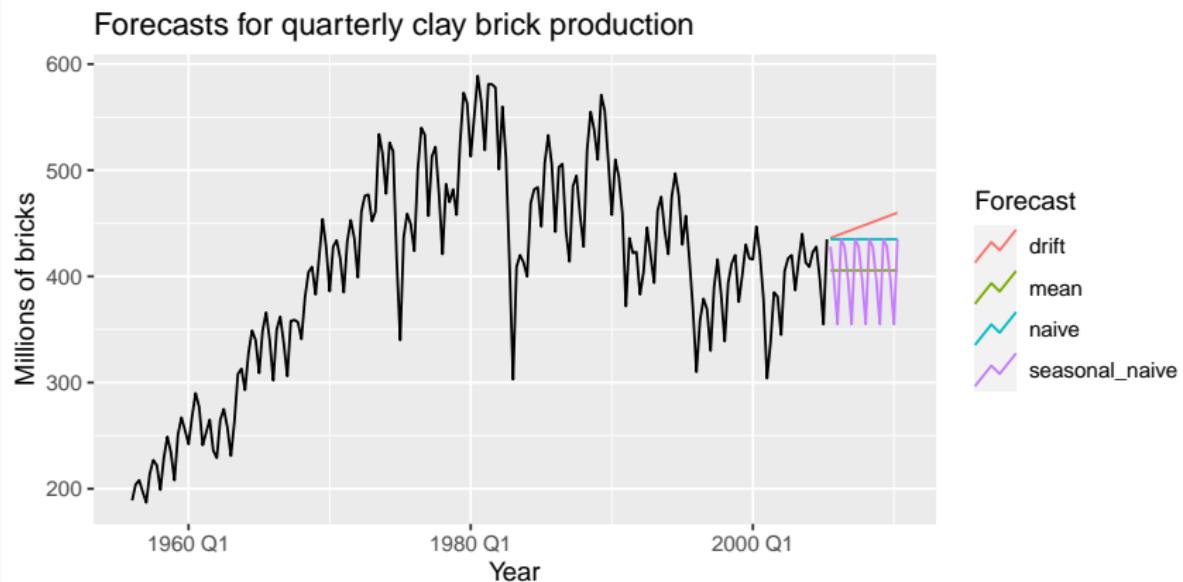
```
brick_fc <- brick_fit %>%  
  forecast(h = "5 years")
```

```
## # A fable: 80 x 4 [1Q]  
## # Key:      .model [4]  
##   .model Quarter       Bricks .mean  
##   <chr>    <qtr>       <dist> <dbl>  
## 1 mean    2005 Q3 N(405, 9294) 405.  
## 2 mean    2005 Q4 N(405, 9294) 405.  
## 3 mean    2006 Q1 N(405, 9294) 405.  
## 4 mean    2006 Q2 N(405, 9294) 405.  
## 5 mean    2006 Q3 N(405, 9294) 405.  
## 6 mean    2006 Q4 N(405, 9294) 405.
```

# A **fable** is a forecast table with point forecasts and distributional forecasts.

# Visualizing forecasts

```
brick_fc %>%
  autoplot(aus_production, level = NULL) +
  ggtitle("Forecasts for quarterly clay brick production") +
  xlab("Year") + ylab("Millions of bricks") +
  guides(colour = guide_legend(title = "Forecast"))
```



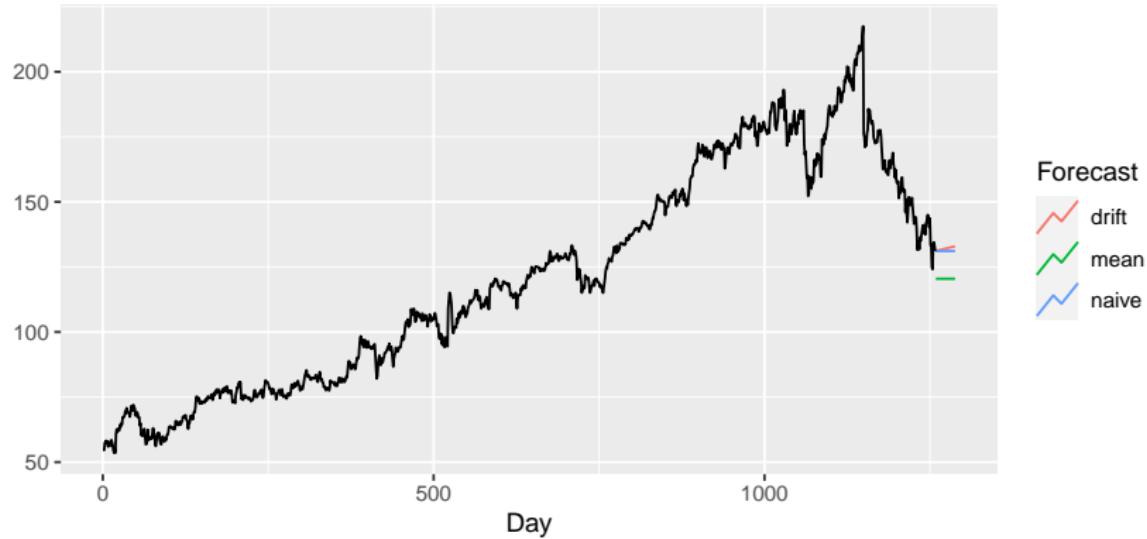
# Facebook daily closing stock price

```
# Re-index based on trading days
fb_stock <- gafa_stock %>%
  group_by(Symbol) %>%
  mutate(trading_day = row_number()) %>%
  update_tsibble(index = trading_day, regular = TRUE) %>%
  ungroup() %>%
  filter(Symbol == "FB")

# Specify, estimate and forecast
fb_stock %>%
  model(mean = MEAN(Close),
        naive = NAIVE(Close),
        drift = RW(Close ~ drift())) %>%
  forecast(h = 30) %>%
  autoplot(fb_stock, level = NULL) +
  xlab("Day") + ylab("") +
  ggtitle("Facebook closing stock price") +
  guides(colour=guide_legend(title = "Forecast"))
```

# Facebook daily closing stock price

Facebook closing stock price



# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

## Fitted values

- $\hat{y}_{t|t-1}$ : forecast of  $y_t$  based on observations  $y_1, y_2, \dots, y_{t-1}$  for  $t = 1, 2, \dots, T$ .
  - We call these as “fitted values”.
  - Sometimes we drop the subscript:  $\hat{y}_t \equiv \hat{y}_{t|t-1}$ .
  - Fitted values are often not true forecasts because parameters in the forecasting method are estimated using all observations.
- $\hat{y}_t = \bar{y}$  for the mean method.
  - $\hat{y}_t = y_{t-1} + (y_T - y_1)/(T - 1)$  for the drift method.
- Naïve and seasonal naïve forecasts do not have parameters, so fitted values are true forecasts.

## Residuals

- Residuals in a time series model refers to what is left over after fitting a model.

$$e_t = y_t - \hat{y}_t.$$

- If a transformation has been used in the model, we often look at residuals on the transformed scale.
- We refer to these as **innovation residuals**.
- Example: Suppose we modelled  $w_t = \log(y_t)$ . The innovation residuals are given by  $w_t - \hat{w}_t$ .
- If no transformation is used innovation and regular residuals are the same.

# Properties of innovation residuals

## Assumptions

- 1  $\{e_t\}$  are uncorrelated. If they aren't then there is information left in the residuals that should be used in computing forecasts.
- 2  $\{e_t\}$  have zero mean. If they don't then the forecasts are biased.

## Useful properties (for distributions & prediction intervals)

- 3  $\{e_t\}$  have constant variance.
- 4  $\{e_t\}$  are normally distributed.

# Facebook daily closing stock price

```
fb_stock %>%  
  autoplot(Close)
```



## Facebook daily closing stock price

```
fit <- fb_stock %>% model(NAIVE(Close))
fit %>% augment()

## # A tsibble: 1,258 x 7 [1]
## # Key:      Symbol, .model [1]
##   Symbol .model     trading_day Close .fitted .resid .innov
##   <chr>  <chr>        <int>  <dbl>    <dbl>    <dbl>    <dbl>
## 1 FB    NAIve(Clo~         1  54.7     NA     NA     NA
## 2 FB    NAIve(Clo~         2  54.6  54.7 -0.150 -0.150
## 3 FB    NAIve(Clo~         3  57.2  54.6  2.64  2.64
## 4 FB    NAIve(Clo~         4  57.9  57.2  0.720 0.720
## 5 FB    NAIve(Clo~         5  58.2  57.9  0.310 0.310
## 6 FB    NAIve(Clo~         6  57.2  58.2 -1.01 -1.01
## 7 FB    NAIve(Clo~         7  57.9  57.2  0.720 0.720
## 8 FB    NAIve(Clo~         8  55.9  57.9 -2.03 -2.03
## 9 FB    NAIve(Clo~         9  57.7  55.9  1.83  1.83
## 10 FB   NAIve(Clo~        10 57.6  57.7 -0.140 -0.140
## # ... with 1,248 more rows
```

# Facebook daily closing stock price

```
fit <- fb_stock %>% model(NAIVE(Close))  
fit %>% augment()
```

```
## # A tsibble: 1,258 x 7 [1]  
## # Key:     Symbol, .model [1]       $\hat{y}_{t|t-1}$   $e_t$   
##   Symbol .model    trading_day Close .fitted .resid .innov  
##   <chr>  <chr>        <int> <dbl>  <dbl>  <dbl>  <dbl>  
## 1 FB    NAIve(Clo~         1  54.7    NA    NA    NA  
## 2 FB    NAIve(Clo~         2  54.6  54.7 -0.150 -0.150  
## 3 FB    NAIve(Clo~         3  57.2  54.6  2.64  2.64  
## 4 FB    NAIve(Clo~         4  57.9  57.2  0.720 0.720  
## 5 FB    NAIve(Clo~         5  58.2  57.9  0.310 0.310  
## 6 FB    NAIve(Clo~         6  57.2  58.2 -1.01 -1.01
```

## Naïve forecasts

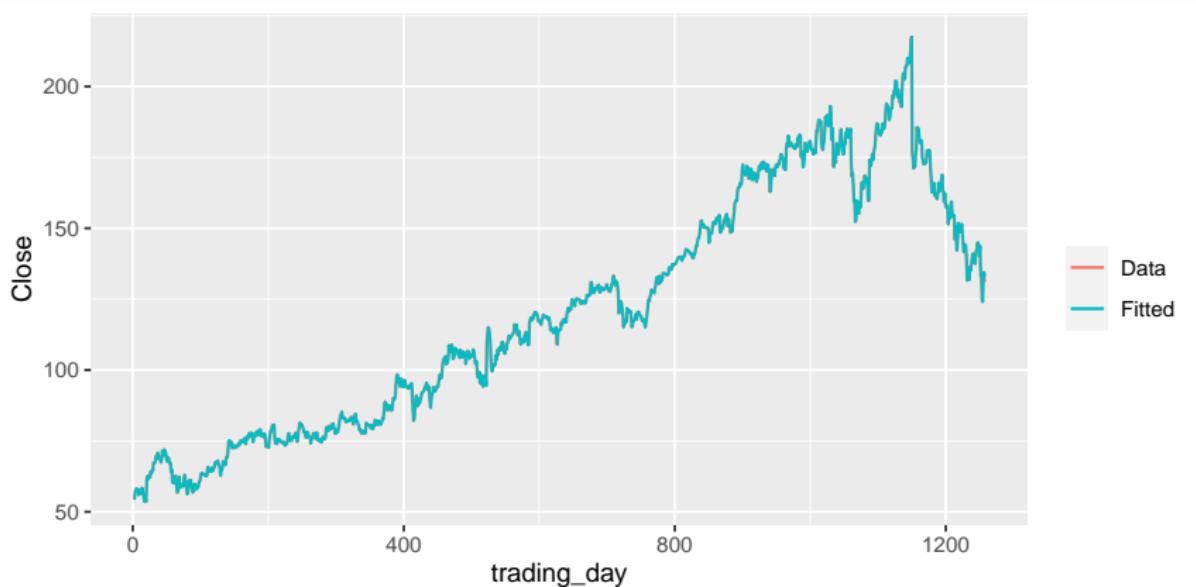
$$\hat{y}_{t|t-1} = y_{t-1}$$

$$e_t = y_t - \hat{y}_{t|t-1} = y_t - y_{t-1}$$

57.9	57.2	0.720	0.720
55.9	57.9	-2.03	-2.03
57.7	55.9	1.83	1.83
57.6	57.7	-0.140	-0.140

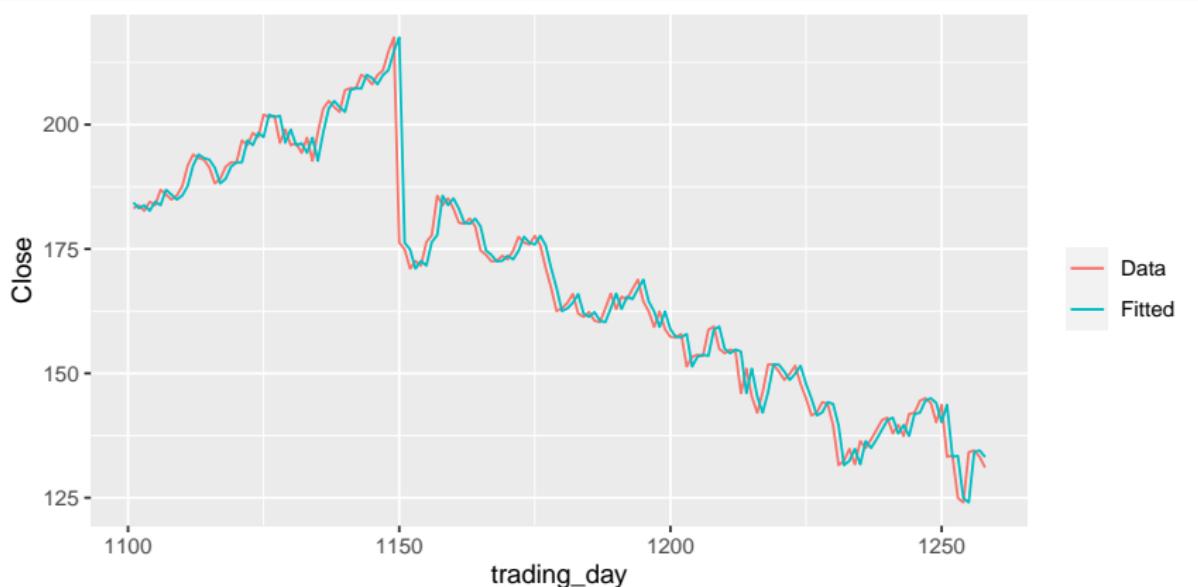
# Facebook daily closing stock price

```
augment(fit) %>%
  ggplot(aes(x = trading_day)) +
  geom_line(aes(y = Close, colour = "Data")) +
  geom_line(aes(y = .fitted, colour = "Fitted")) +
  guides(colour = guide_legend(title = ""))
```



# Facebook daily closing stock price

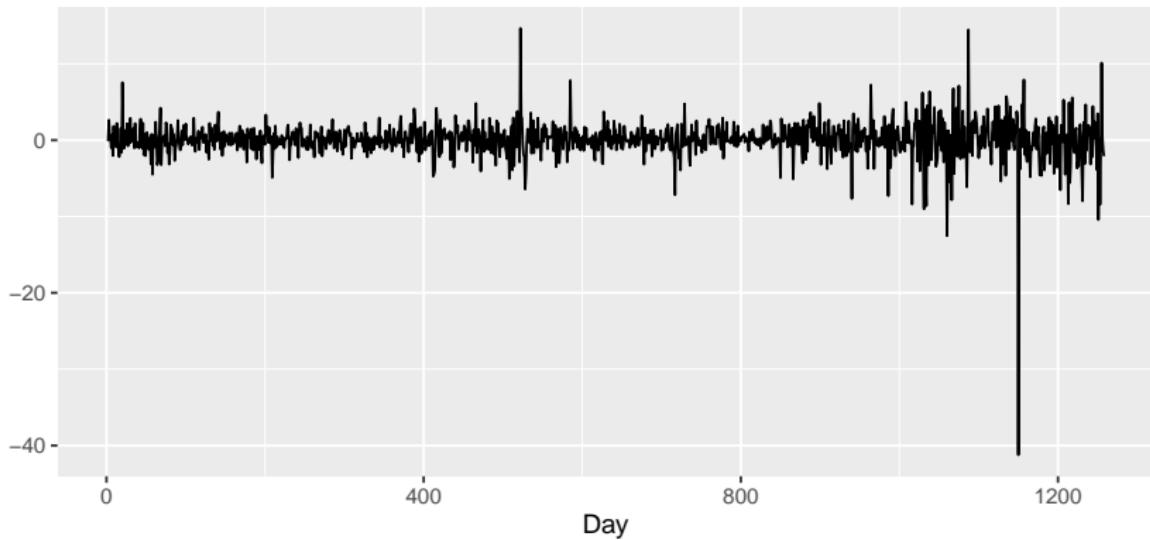
```
augment(fit) %>% filter(trading_day > 1100) %>%  
  ggplot(aes(x = trading_day)) +  
  geom_line(aes(y = Close, colour = "Data")) +  
  geom_line(aes(y = .fitted, colour = "Fitted")) +  
  guides(colour = guide_legend(title = ""))
```



# Facebook daily closing stock price

```
augment(fit) %>%
  autoplot(.resid) +
  labs(x = "Day", y = "", title = "Residuals from naïve method")
```

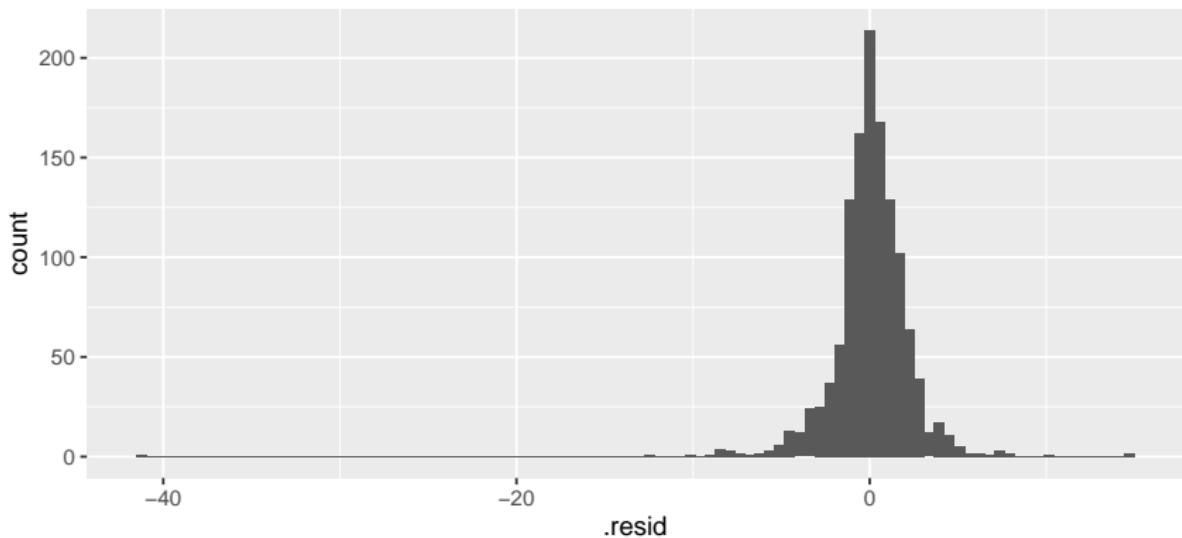
Residuals from naïve method



# Facebook daily closing stock price

```
augment(fit) %>%
  ggplot(aes(x = .resid)) +
  geom_histogram(bins = 100) +
  ggtitle("Histogram of residuals")
```

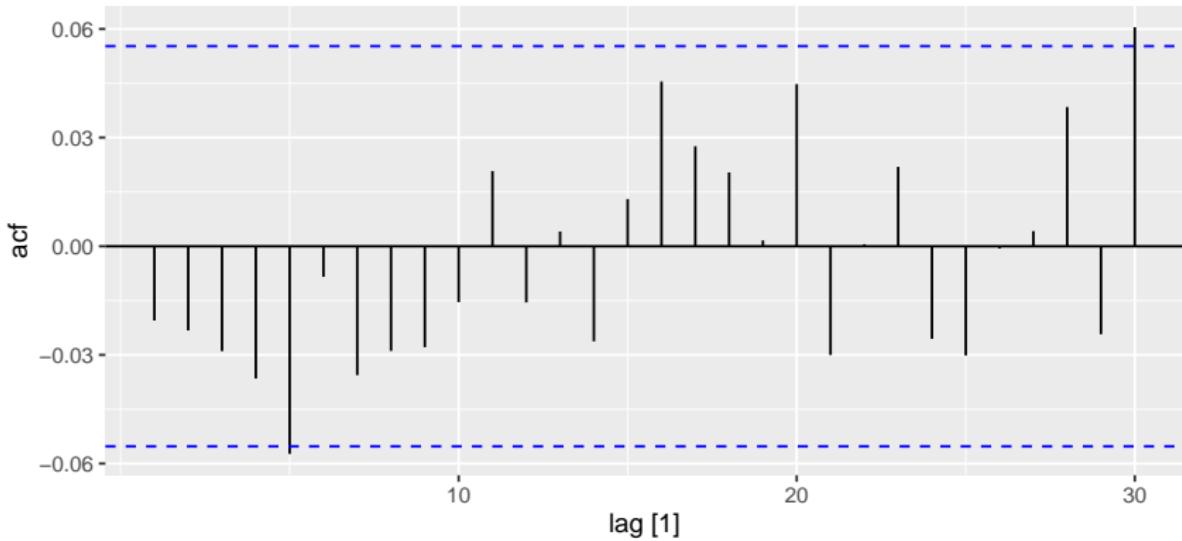
Histogram of residuals



# Facebook daily closing stock price

```
augment(fit) %>%
  ACF(.resid) %>%
  autoplot() + ggtitle("ACF of residuals")
```

ACF of residuals



- The residuals should resemble a white noise series (uncorrelated, mean-zero, constant variance).
- If they aren't, then there is information left in the residuals that we should take into account.
- A standard residual diagnostic is to check the ACF of the residuals of a forecasting method.
- We expect these to look like white noise.

## Portmanteau tests

- We can perform a general test that takes into account the magnitude of  $r_k$  as a group.
- For example, it may be that, individually, each  $r_k$  is small in magnitude, say, each one is slightly less than  $2/\sqrt{T}$  in magnitude. But collectively the values are large.

### Box-Pierce test

$$Q = T \sum_{k=1}^h r_k^2$$

where  $h$  is the maximum lag being considered and  $T$  is the number of observations.

- If each  $r_k$  is close to zero,  $Q$  will be small.
- If some  $r_k$  values are large in magnitude,  $Q$  will be large.

### Ljung-Box test

$$Q^* = T(T + 2) \sum_{k=1}^h \frac{r_k^2}{T - k}$$

where  $h$  is the maximum lag being considered and  $T$  is the number of observations.

- Usually we set  $h = 10$  for non-seasonal data and  $h = 2m$  for seasonal data.
- Large values of  $Q^*$  suggest that the autocorrelations do not come from a white noise series.

## Portmanteau tests

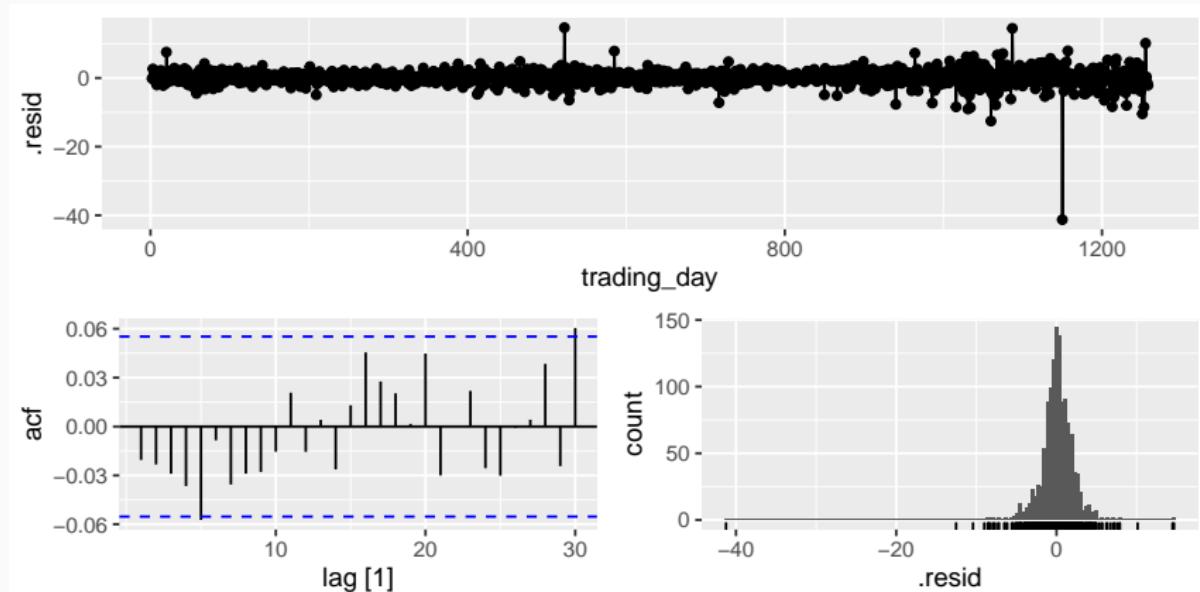
- If the data are IID white noise,  $Q^*$  has a  $\chi^2$ -distribution with  $(h - K)$  degrees of freedom, where  $K$ : no. of parameters in the model.
- When applied to raw data (rather than to residuals from a model) then  $K = 0$ .

```
augment(fit) %>%
  features(.resid, features = ljung_box, lag = 10, dof = 0)

## # A tibble: 1 x 4
##   Symbol .model      lb_stat lb_pvalue
##   <chr>  <chr>       <dbl>     <dbl>
## 1 FB    NAIVE(Close)  12.1     0.276
```

## gg\_tsresiduals function

```
fit %>% gg_tsresiduals()
```



# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

- A point forecast  $\hat{y}_{T+h|T}$  is (usually) the mean of the conditional distribution  $y_{T+h}|y_1, y_2, \dots, y_T$ .
- We can represent the uncertainty in the forecasts using a probability distribution.
- It describes the probability of observing possible future values using the fitted model.
- Most time series models provide normally distributed forecasts.

## Forecast distributions

- Let  $\hat{\sigma} = \sqrt{\sum_{t=1}^T e_t^2 / (T - K)}$ , where  $K$ : no. of parameters estimated in the forecasting method.
- Assume that the residuals are normally distributed:

**Mean:**  $y_{T+h|T} \sim N(\bar{y}, (1 + 1/T)\hat{\sigma}^2)$

**Naïve:**  $y_{T+h|T} \sim N(y_T, h\hat{\sigma}^2)$

**Seasonal naïve:**  $y_{T+h|T} \sim N(y_{T+h-m(k+1)}, (k + 1)\hat{\sigma}^2)$

**Drift:**  $y_{T+h|T} \sim N(y_T + \frac{h}{T-1}(y_T - y_1), h(1 + h/T)\hat{\sigma}^2)$

- $k$ : integer part of  $(h - 1)/m$ .
- When  $h = 1$  and  $T$  is large, all these methods give the same approximate forecast variance  $\hat{\sigma}^2$ .

## Prediction intervals

- A prediction interval gives an interval within which we expect  $y_t$  to lie with a specified probability.
- Assuming that future observations are normally distributed, 95% prediction interval for  $h$ -step forecast

$$\hat{y}_{T+h|T} \pm 1.96\hat{\sigma}_h$$

- $\hat{\sigma}_h$  is an estimate of SD of  $h$ -step forecast distribution.
- Point forecasts often carry little information without a measure of uncertainty such as prediction intervals.
- Usually too narrow due to unaccounted uncertainty.

## Prediction intervals

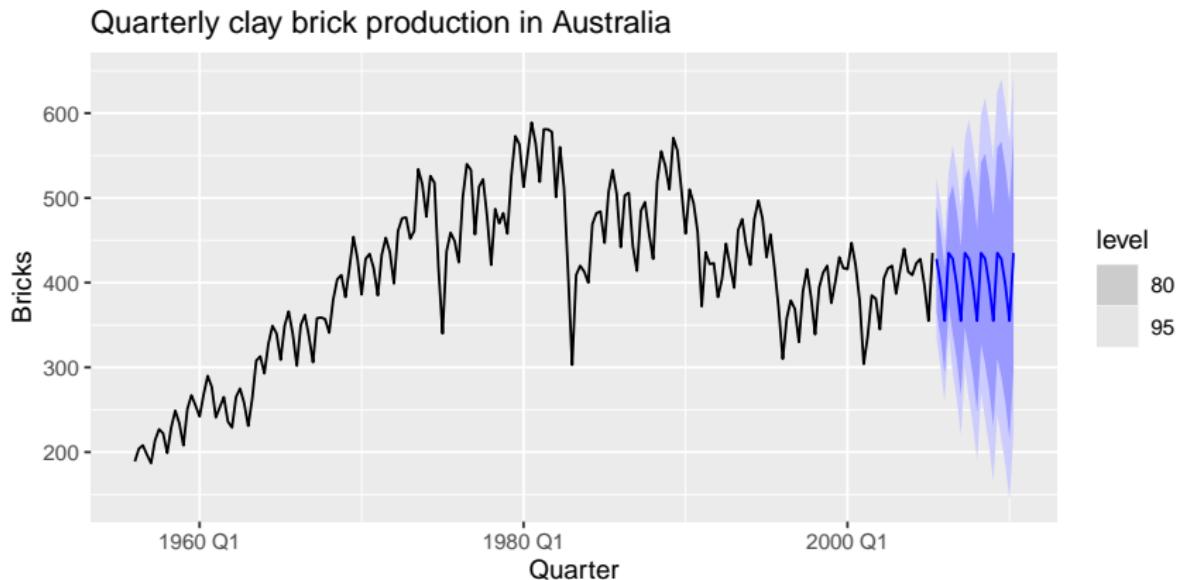
- Automatically computes from the forecast distribution.
- Use `level` argument to control coverage probability.

```
brick_fc %>% hilo(level = 95) %>% print(n = 5)
```

```
## # A tsibble: 80 x 5 [1Q]
## # Key:      .model [4]
##   .model Quarter     Bricks .mean      `95%
##   <chr>    <qtr>     <dist> <dbl>      <hilo>
## 1 mean    2005 Q3 N(405, 9294)  405. [217, 594]95
## 2 mean    2005 Q4 N(405, 9294)  405. [217, 594]95
## 3 mean    2006 Q1 N(405, 9294)  405. [217, 594]95
## 4 mean    2006 Q2 N(405, 9294)  405. [217, 594]95
## 5 mean    2006 Q3 N(405, 9294)  405. [217, 594]95
## # ... with 75 more rows
```

# Prediction intervals

```
bricks %>%  
  model(SNAIVE(Bricks ~ lag("year"))) %>%  
  forecast(h = "5 years") %>%  
  autoplot(bricks) +  
  ggtitle("Quarterly clay brick production in Australia")
```



# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

# Modelling with transformations

- We can specify the transformation to be used in the LHS of the formula.
- It will be automatically back-transformed.

```
food <- aus_retail %>%  
  filter(Industry == "Food retailing") %>%  
  summarise(Turnover = sum(Turnover))  
  
# Modelling  
fit_food <- food %>%  
  model(SNAIVE(log(Turnover)))
```

## Forecasting with transformations

```
food_fc <- fit_food %>%  
  forecast(h = "3 years")
```

```
## # A fable: 36 x 4 [1M]  
## # Key:     .model [1]  
##   .model                 Month        Turnover  .mean  
##   <chr>                  <mth>       <dist>  <dbl>  
## 1 SNAIVE(log(Turnover)) 2019 Jan t(N(9.3, 0.0047)) 10738.  
## 2 SNAIVE(log(Turnover)) 2019 Feb t(N(9.2, 0.0047))  9856.  
## 3 SNAIVE(log(Turnover)) 2019 Mar t(N(9.3, 0.0047)) 11214.  
## 4 SNAIVE(log(Turnover)) 2019 Apr t(N(9.2, 0.0047)) 10378.  
## 5 SNAIVE(log(Turnover)) 2019 May t(N(9.3, 0.0047)) 10670.  
## 6 SNAIVE(log(Turnover)) 2019 Jun t(N(9.2, 0.0047)) 10292.  
## # ... with 30 more rows
```

## Forecasting with transformations

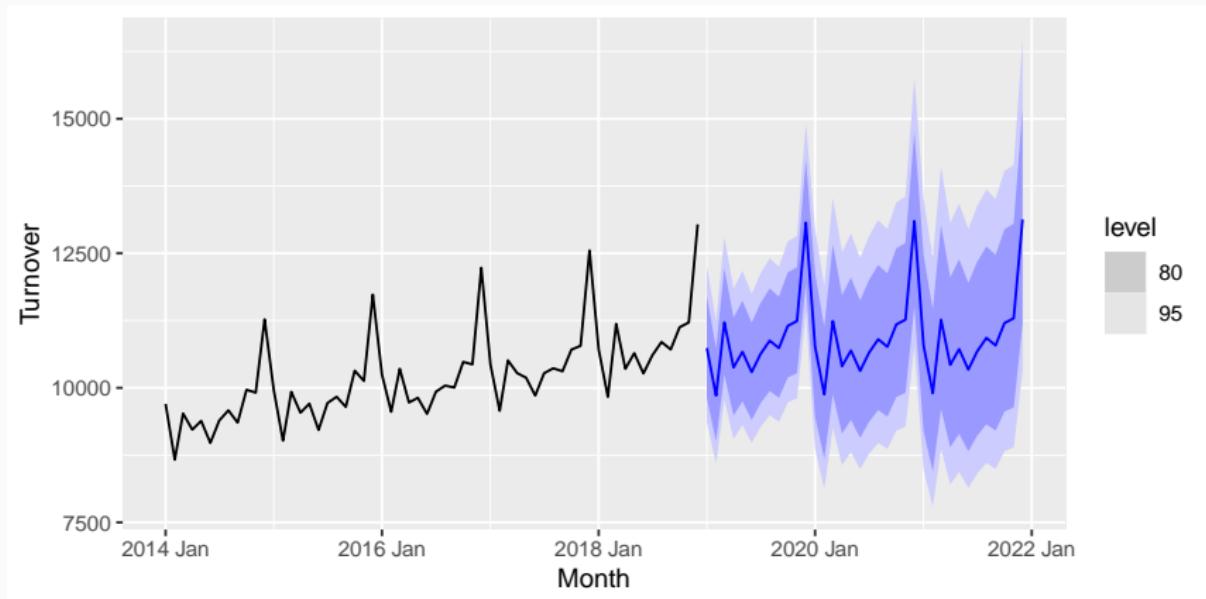
```
food_fc <- fit_food %>%  
  forecast(h = "3 years")
```

```
## # A fable: 36 x 4 [1M]  
## # Key:     .model [1]  
##   .model                      Month      Turnover  .mean  
##   <chr>                      <mth>      <dist>  <dbl>  
## 1 SNAIVE(log(Turnover)) 2019 Jan t(N(9.3, 0.0047)) 10738.  
## 2 SNAIVE(log(Turnover)) 2019 Feb t(N(9.2, 0.0047))  9856.  
## 3 SNAIVE(log(Turnover)) 2019 Mar t(N(9.3, 0.0047)) 11214.  
## 4 SNAIVE(log(Turnover)) 2019 Apr t(N(9.2, 0.0047)) 10378.  
## 5 SNAIVE(log(Turnover)) 2019 May t(N(9.3, 0.0047)) 10670.  
## 6 SNAIVE(log(Turnover)) 2019 Jun t(N(9.2, 0.0047)) 10292.  
## # ... The distributions are given as transformed normal.
```

# Forecasting with transformations

```
food_fc %>%
```

```
autoplot(food %>% filter(year(Month) > 2013))
```



## Back-transformation

### Box and Cox (1964)

For  $y_t > 0$ ,

$$w_t = \begin{cases} \log(y_t), & \lambda = 0, \\ \frac{y_t^\lambda - 1}{\lambda}, & \lambda \neq 0. \end{cases}$$

### Bickel and Doksum (1981)

For  $\lambda > 0, y_t \in \mathbb{R}$ ,

$$w_t = \frac{\text{sign}(y_t)|y_t|^\lambda - 1}{\lambda}.$$

### Back-transformation

$$y_t = \begin{cases} \exp(w_t), & \lambda = 0, \\ \text{sign}(\lambda w_t + 1) |\lambda w_t + 1|^{\frac{1}{\lambda}}, & \lambda > 0, \\ (\lambda w_t + 1)^{\frac{1}{\lambda}}, & \lambda < 0, \lambda w_t + 1 > 0, \\ \text{NA}, & \lambda < 0, \lambda w_t + 1 < 0. \end{cases}$$

- We obtain back-transformed point forecasts by replacing  $y_t$  and  $w_t$  by  $\hat{y}_{T+h|T}$  and  $\hat{w}_{T+h|T}$ , respectively.
- Back-transformed point forecasts are medians and PI have the correct coverage.

## Bias adjustment

Let  $X$  have mean  $\mu$  and variance  $\sigma^2$  and  $f(X)$  be back-transformation function, and  $Y = f(X)$ .

From Taylor series expansion about  $\mu$ :

$$f(X) \approx f(\mu) + (X - \mu)f'(\mu) + \frac{1}{2}(X - \mu)^2f''(\mu)$$

$$E[Y] = E[f(X)] \approx f(\mu) + \frac{1}{2}\sigma^2f''(\mu)$$

## Bias adjusted back-transformed mean

$$E[Y] \approx \begin{cases} \exp(\mu) \left[ 1 + \frac{\sigma^2}{2} \right], & \lambda = 0, \\ \text{sign}(\lambda\mu + 1) |\lambda\mu + 1|^{\frac{1}{\lambda}} \left[ 1 + \frac{\sigma^2(1-\lambda)}{2(\lambda\mu+1)^2} \right], & \lambda > 0, \\ (\lambda\mu + 1)^{\frac{1}{\lambda}} \left[ 1 + \frac{\sigma^2(1-\lambda)}{2(\lambda\mu+1)^2} \right], & \lambda < 0, \lambda\mu + 1 > 0, \\ \text{NA}, & \lambda < 0, \lambda\mu + 1 < 0. \end{cases}$$

## Bias adjustment

Let  $X$  have mean  $\mu$  and variance  $\sigma^2$  and  $f(X)$  be back-transformation function, and  $Y = f(X)$ .

From Taylor series expansion about  $\mu$ :

$$f(X) \approx f(\mu) + (X - \mu)f'(\mu) + \frac{1}{2}(X - \mu)^2f''(\mu)$$

$$E[Y] = E[f(X)] \approx f(\mu) + \frac{1}{2}\sigma^2f''(\mu)$$

### Bias adjusted back-transformed mean

$$E[Y] \approx \begin{cases} \exp(\mu) \left[ 1 + \frac{\sigma^2}{2} \right], & \lambda = 0, \\ \text{sign}(\lambda\mu + 1) |\lambda\mu + 1|^{\frac{1}{\lambda}} \left[ 1 + \frac{\sigma^2(1-\lambda)}{2(\lambda\mu+1)^2} \right], & \lambda > 0, \\ (\lambda\mu + 1)^{\frac{1}{\lambda}} \left[ 1 + \frac{\sigma^2(1-\lambda)}{2(\lambda\mu+1)^2} \right], & \lambda < 0, \lambda\mu + 1 > 0, \\ \dots & \dots \end{cases}$$

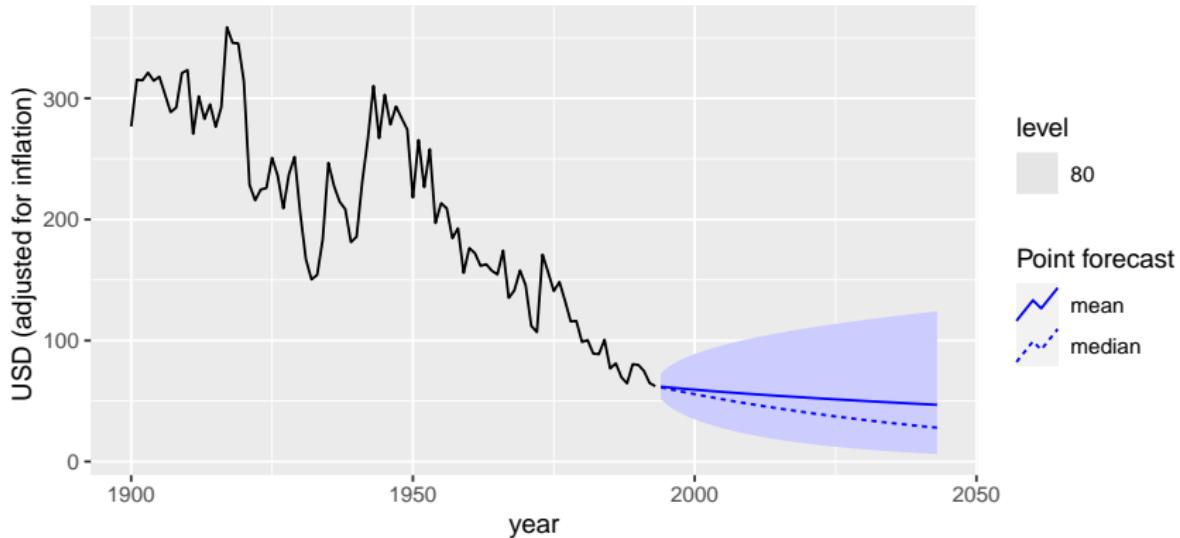
Larger the variance, bigger the difference between the mean and median.

# Bias adjustment

```
prices %>%  
  filter(!is.na(eggs)) %>%  
  model(RW(log(eggs) ~ drift())) %>%  
  forecast(h = 50) %>%  
  autoplot(prices %>% filter(!is.na(eggs)),  
           level = 80, point_forecast = lst(mean, median)  
  ) +  
  labs(title = "Annual egg prices",  
       y="USD (adjusted for inflation) ")
```

# Bias adjustment

Annual egg prices



# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

## Forecasting with decomposition

Let  $\hat{A}$  is the seasonally adjusted component.

- **Additive decomposition:**  $y_t = \hat{S}_t + \hat{A}_t$  where  $\hat{A}_t = \hat{T}_t + \hat{R}_t$ .
- **Multiplicative decomposition:**  $y_t = \hat{S}_t \hat{A}_t$  where  $\hat{A}_t = \hat{T}_t \hat{R}_t$ .
- Forecast the seasonal component by repeating the last year.
- Forecast the seasonally adjusted component using non-seasonal forecasting methods such as drift, Holt's method, non-seasonal ARIMA.
- Combine these two forecasts to get forecasts of original data.

## US retail employment

```
us_retail_employment <- us_employment %>%  
  filter(year(Month) >= 1990, Title == "Retail Trade") %>%  
  select(-Series_ID)
```

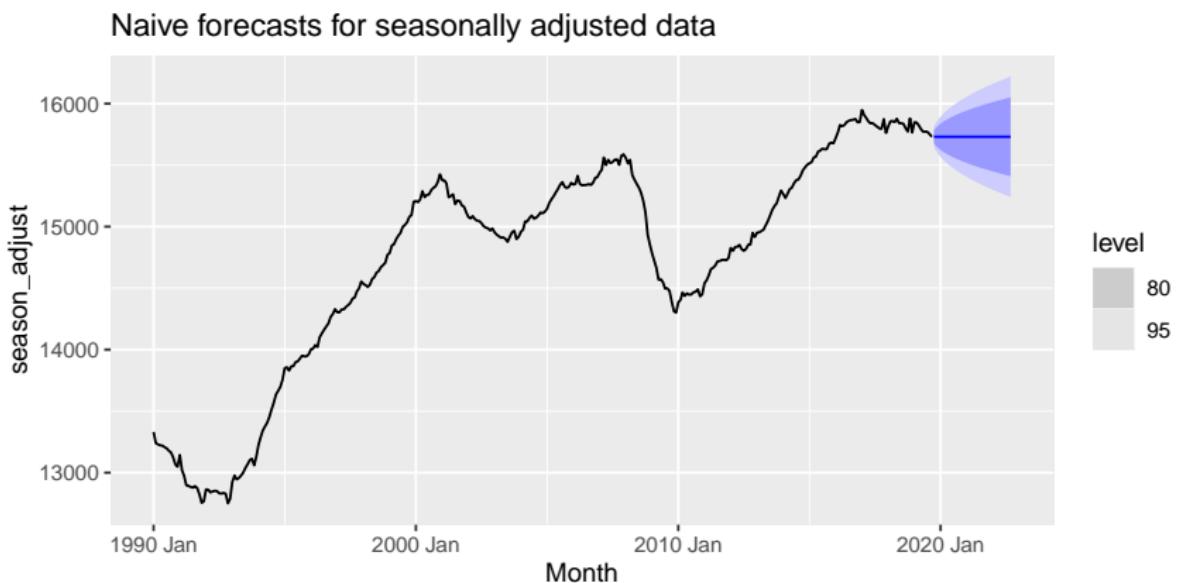
```
## # A tsibble: 357 x 3 [1M]  
##   Month Title     Employed  
##   <mth> <chr>     <dbl>  
## 1 1990  Jan Retail Trade 13256.  
## 2 1990  Feb Retail Trade 12966.  
## 3 1990  Mar Retail Trade 12938.  
## 4 1990  Apr Retail Trade 13012.  
## 5 1990  May Retail Trade 13108.  
## 6 1990  Jun Retail Trade 13183.  
## 7 1990  Jul Retail Trade 13170.  
## # ... with 350 more rows
```

# US retail employment

```
dcmp <- us_retail_employment %>%  
  model(STL(Employed ~ trend(window = 7), robust = TRUE)) %>%  
  components() %>% select(-.model)  
  
## # A tsibble: 357 x 6 [1M]  
##       Month Employed   trend season_year remainder  
##     <mth>    <dbl>  <dbl>      <dbl>      <dbl>  
## 1 1990 Jan  13256. 13246.     -74.9     84.5  
## 2 1990 Feb  12966. 13239.    -274.      1.30  
## 3 1990 Mar  12938. 13231.    -292.     -1.57  
## 4 1990 Apr  13012. 13224.    -210.     -1.23  
## 5 1990 May  13108. 13216.    -112.      3.63  
## 6 1990 Jun  13183. 13207.    -24.7     0.664  
## # ... with 351 more rows, and 1 more variable:  
## #   season_adjust <dbl>
```

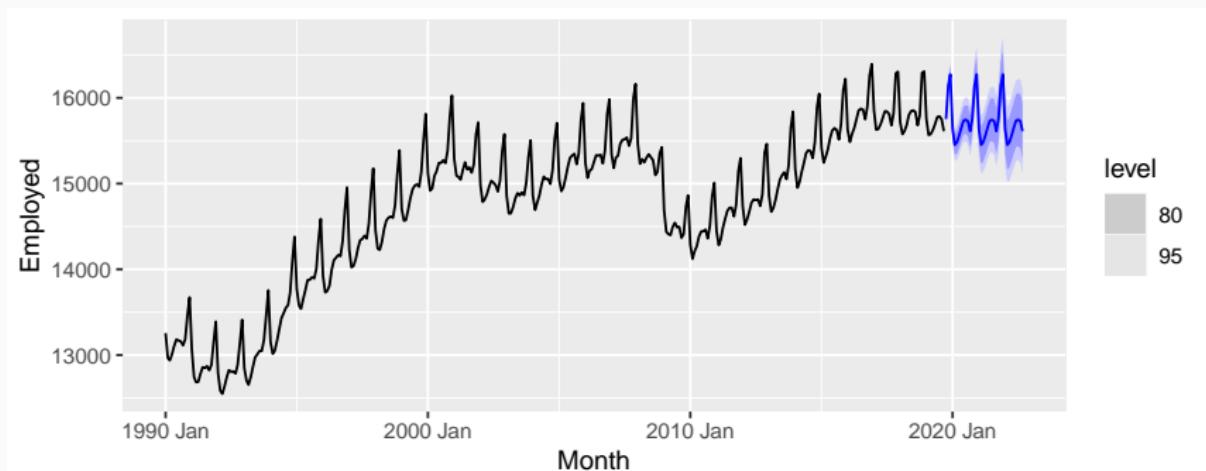
# US retail employment

```
dcmp %>%
  model(NAIVE(season_adjust)) %>%
  forecast(h = "3 years") %>%
  autoplot(dcmp) +
  ggtitle("Naive forecasts for seasonally adjusted data")
```



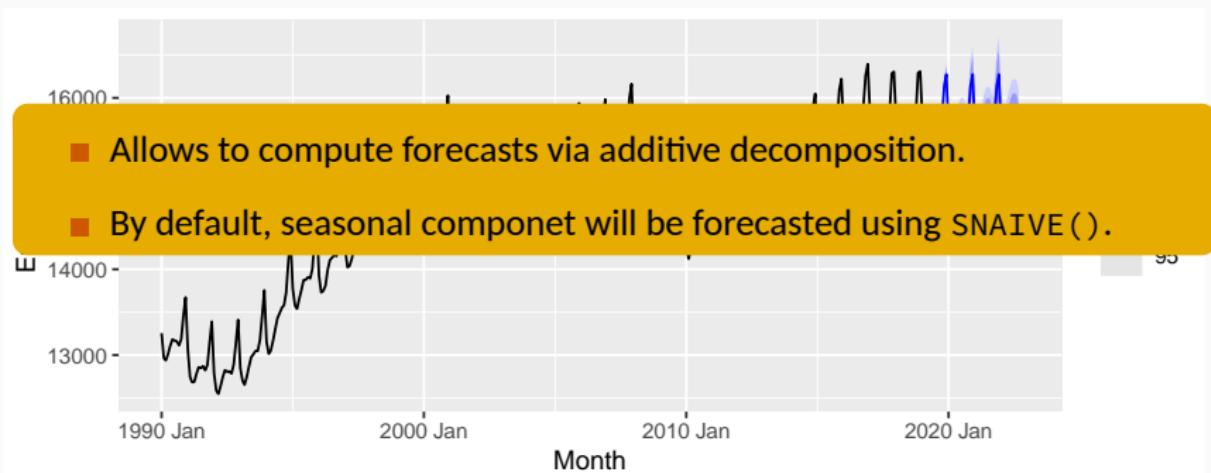
# US retail employment

```
fit_dcmp <- us_retail_employment %>%
  model(stlf = decomposition_model(
    STL(Employed ~ trend(window = 7), robust = TRUE),
    NAIVE(season_adjust)))
fit_dcmp %>% forecast(h = "3 years") %>%
  autoplot(us_retail_employment)
```



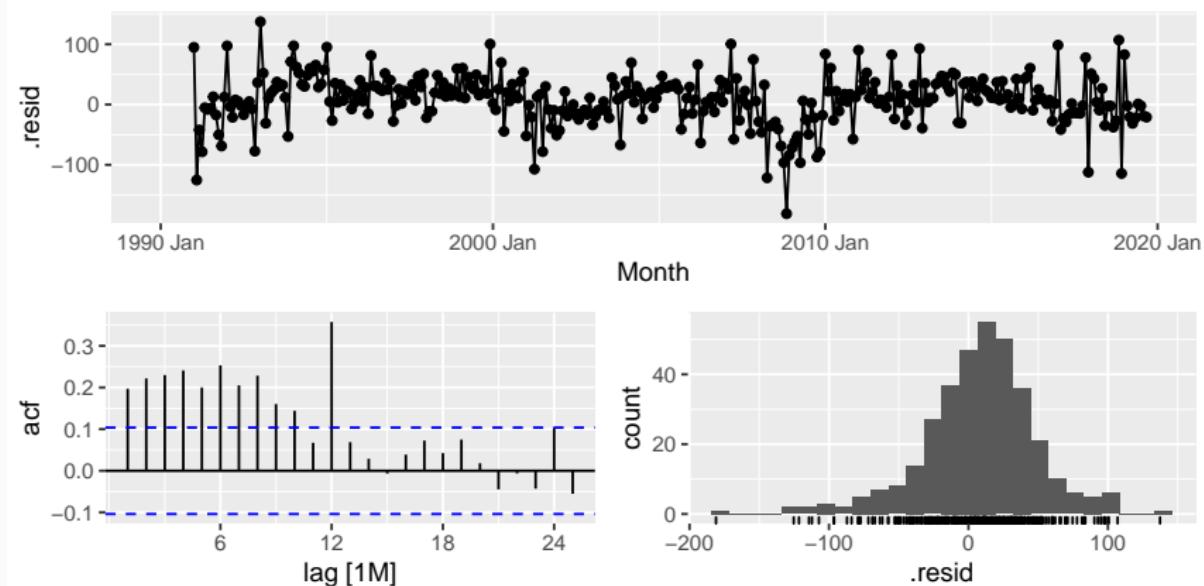
# US retail employment

```
fit_dcmp <- us_retail_employment %>%
  model(stlf = decomposition_model(
    STL(Employed ~ trend(window = 7), robust = TRUE),
    NAIVE(season_adjust)))
fit_dcmp %>% forecast(h = "3 years") %>%
  autoplot(us_retail_employment)
```



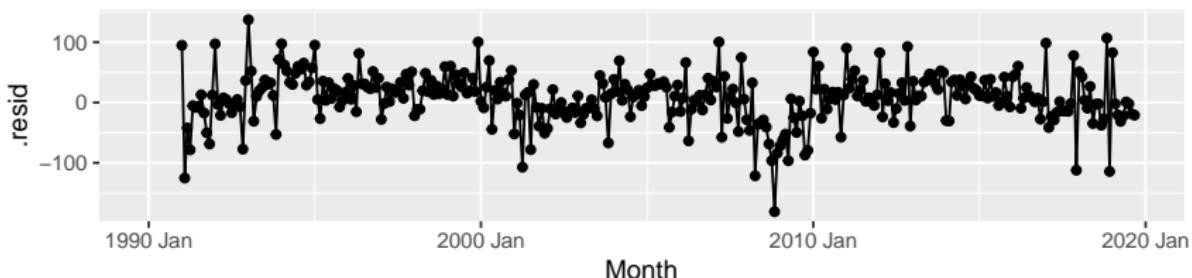
# US retail employment

```
fit_dcmp %>%  
  gg_tsresiduals()
```

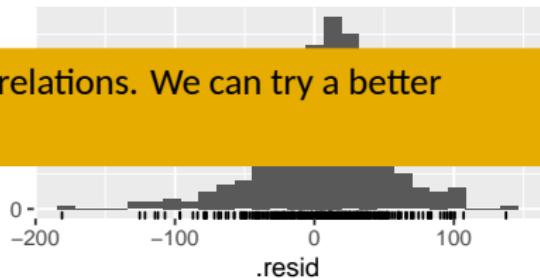
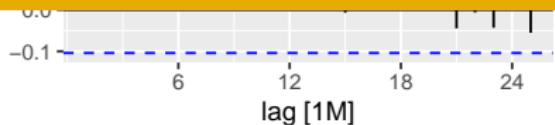


# US retail employment

```
fit_dcmp %>%  
  gg_tsresiduals()
```



ACF of residuals show significant autocorrelations. We can try a better forecasting method than naïve method.



# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

## Training and test sets



- The size of the residuals does not give a clear idea about how large true forecast errors can be.
- A perfect fit can be achieved using a model with enough parameters.
- A model which fits the training data well will not necessarily forecast well.
- The test set must not be used for model estimation or calculation of forecasts.
- This is used only to evaluate the accuracy of the fitted model.

- Forecast error is the difference between an observed value and its forecast.

$$e_{T+h} = y_{T+h} - \hat{y}_{T+h|T}$$

where the training set is given by  $\{y_1, y_2, \dots, y_T\}$ .

- Residuals are calculated on the training set.
- Forecast errors are calculated on the test set.
- Residuals are based on **1-step forecasts** whereas forecast errors can involve **multi-step forecasts**.

## Measures of accuracy

$y_{T+h}$  =  $(T + h)$ th observation,  $h = 1, \dots, H$

$\hat{y}_{T+h|T}$  = its forecast based on data up to time  $T$

$e_{T+h} = y_{T+h} - \hat{y}_{T+h|T}$

Mean Absolute Error (MAE)

$$\text{mean}(|e_{T+h}|)$$

Mean Squared Error (MSE)

$$\text{mean}(e_{T+h}^2)$$

Root Mean Squared Error (RMSE)

$$\sqrt{\text{mean}(e_{T+h}^2)}$$

Mean Absolute Percentage Error (MAPE)

$$100\text{mean}(|e_{T+h}|/|y_{T+h}|)$$

- MAE, MSE and RMSE are scale dependent measures.
- MAPE is scale independent, is only sensible with strictly positive data and when  $y$  has a meaningful zero.

# Measures of accuracy

## Mean Absolute Scaled Error (MASE)

$$\text{MASE} = \text{mean}(|e_{T+h}| / Q)$$

where  $Q$  is a stable measure of the scale of the time series  $\{y_t\}$ .

### Non-seasonal time series

$$Q = \frac{\sum_{t=2}^T |y_t - y_{t-1}|}{T - 1}$$

MASE is equivalent to MAE relative to naïve method.

### Seasonal time series

$$Q = \frac{\sum_{t=m+1}^T |y_t - y_{t-m}|}{T - m}$$

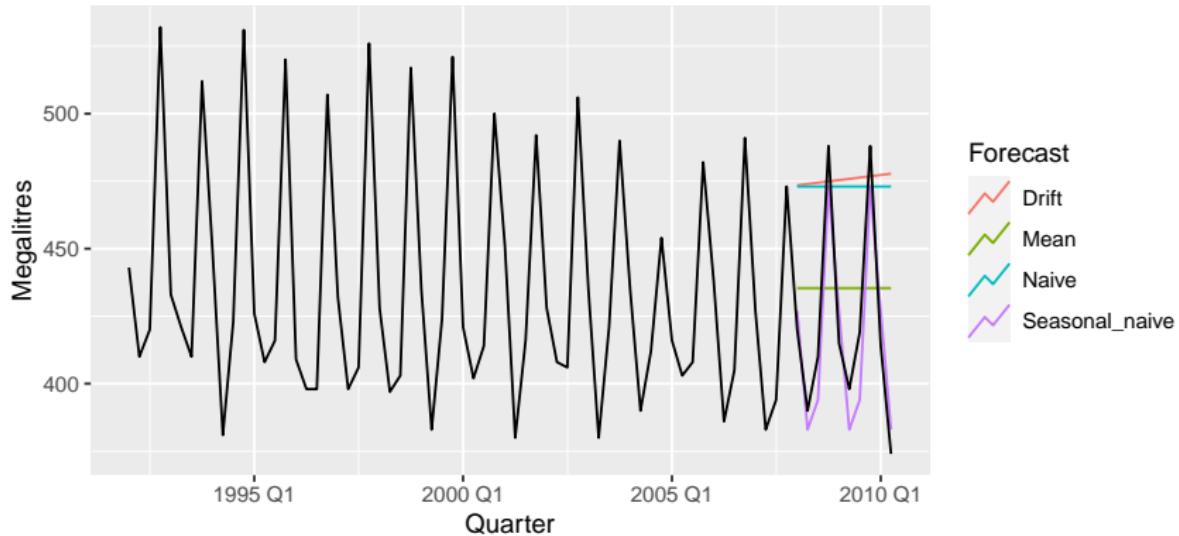
MASE is equivalent to MAE relative to seasonal naïve method.

## Measures of accuracy

```
recent_production <- aus_production %>%  
  filter(year(Quarter) >= 1992)  
beer_train <- recent_production %>%  
  filter(year(Quarter) <= 2007)  
beer_fit <- beer_train %>%  
  model(  
    Mean = MEAN(Beer),  
    Naive = NAIVE(Beer),  
    Seasonal_naive = SNAIVE(Beer),  
    Drift = RW(Beer ~ drift())  
  )  
beer_fc <- beer_fit %>%  
  forecast(h = 10)  
beer_fc %>%  
  autoplot(recent_production, level = NULL) +  
  guides(colour = guide_legend(title = "Forecast")) +  
  labs(y = "Megalitres",  
    title = "Forecasts for quarterly beer production")
```

# Measures of accuracy

Forecasts for quarterly beer production



## Measures of accuracy

```
accuracy(beer_fit)
```

```
## # A tibble: 4 x 6
##   .model      .type    RMSE    MAE    MAPE    MASE
##   <chr>      <chr>    <dbl>   <dbl>   <dbl>   <dbl>
## 1 Drift      Training  65.3   54.8   12.2   3.83
## 2 Mean       Training  43.6   35.2   7.89   2.46
## 3 Naive      Training  65.3   54.7   12.2   3.83
## 4 Seasonal_naive Training 16.8   14.3   3.31   1
```

```
accuracy(beer_fc, recent_production)
```

```
## # A tibble: 4 x 6
##   .model      .type    RMSE    MAE    MAPE    MASE
##   <chr>      <chr>    <dbl>   <dbl>   <dbl>   <dbl>
## 1 Drift      Test     64.9   58.9   14.6   4.12
## 2 Mean       Test     38.4   34.8   8.28   2.44
## 3 Naive      Test     62.7   57.4   14.2   4.01
## 4 Seasonal_naive Test    14.3   13.4   3.17   0.937
```

# Outline

- 1 Tidy forecasting workflow
- 2 Simple forecasting methods
- 3 Residual diagnostics
- 4 Distributional forecasts and prediction intervals
- 5 Forecasting using transformations
- 6 Forecasting with decomposition
- 7 Evaluating point forecast accuracy
- 8 Time series cross-validation

# Time series cross-validation

## Traditional evaluation

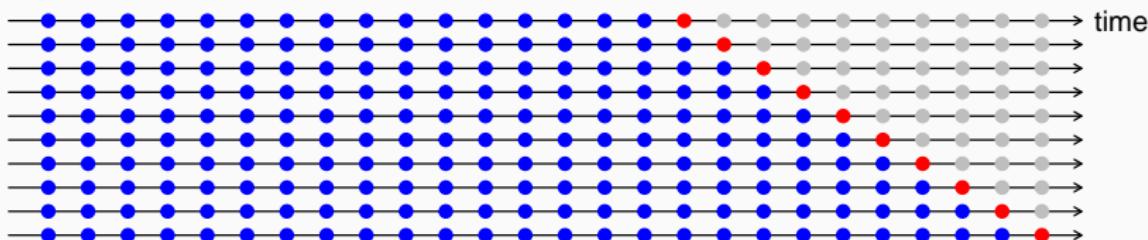


# Time series cross-validation

## Traditional evaluation



## Time series cross-validation



- Forecast accuracy is computed by averaging over the test sets.
- The procedure is known as “evaluation on a rolling forecasting origin”.
- This can be modified to produce multi-step forecasts.

## Creating the rolling training training sets

There are three main rolling types which can be used.

- Stretch: extends a growing length window with new data.
- Slide: shifts a fixed length window through the data.
- Tile: moves a fixed length window without overlap.

we can use `stretch_tsibble()`, `slide_tsibble()` and `tile_tsibble()`.

## Graphical illustration

## Time series cross-validation

```
fb_stretch <- fb_stock %>%  
  stretch_tsibble(.init = 3, .step = 1)
```

```
## # A tsibble: 791,908 x 4 [1]  
## # Key:      .id [1,256]  
##   Date       Close trading_day   .id  
##   <date>     <dbl>        <int> <int>  
## 1 2014-01-02  54.7          1     1  
## 2 2014-01-03  54.6          2     1  
## 3 2014-01-06  57.2          3     1  
## 4 2014-01-02  54.7          1     2  
## 5 2014-01-03  54.6          2     2  
## 6 2014-01-06  57.2          3     2  
## 7 2014-01-07  57.9          4     2  
## # ... with 791,901 more rows
```

## Time series cross-validation

```
fb_stretch <- fb_stock %>%  
  stretch_tsibble(.init = 3, .step = 1)  
  
.init: starting training set length and .step: successive increments of  
# training sets.  
  
## # Key:          .id [1,256]  
##   Date        Close trading_day   .id  
##   <date>      <dbl>       <int> <int>  
## 1 2014-01-02  54.7           1     1  
## 2 2014-01-03  54.6           2     1  
## 3 2014-01-06  57.2           3     1  
## 4 2014-01-02  54.7           1     2  
## 5 2014-01-03  54.6           2     2  
## 6 2014-01-06  57.2           3     2  
## 7 2014-01-07  57.9           4     2  
## # ... with 791,901 more rows
```

## Time series cross-validation

Estimate a random walk with a drift model to each training set.

```
fit_cv <- fb_stretch %>%
  model(RW(Close ~ drift()))  
  
## # A mable: 1,256 x 3  
## # Key:      .id, Symbol [1,256]  
##      .id Symbol `RW(Close ~ drift())`  
##      <int> <chr>                <model>  
## 1      1 FB                  <RW w/ drift>  
## 2      2 FB                  <RW w/ drift>  
## 3      3 FB                  <RW w/ drift>  
## 4      4 FB                  <RW w/ drift>  
## 5      5 FB                  <RW w/ drift>  
## # ... with 1,251 more rows
```

## Time series cross-validation

Compute 1-step-ahead forecasts from all fitted models.

```
fc_cv <- fit_cv %>%  
  forecast(h = 1)  
  
## # A fable: 1,256 x 5 [1]  
## # Key:     .id, Symbol [1,256]  
##     .id Symbol trading_day     Close .mean  
##     <int> <chr>      <dbl>     <dist> <dbl>  
## 1     1 FB          4 N(58, 3.9) 58.4  
## 2     2 FB          5 N(59, 2)   59.0  
## 3     3 FB          6 N(59, 1.5) 59.1  
## 4     4 FB          7 N(58, 1.8) 57.7  
## 5     5 FB          8 N(58, 1.5) 58.5  
## # ... with 1,251 more rows
```

## Time series cross-validation

```
# accuracy of cross-validated sets  
fc_cv %>%  
  accuracy(fb_stock)  
  
# accuracy of training sets  
fb_stock %>% model(RW(Close ~ drift())) %>% accuracy()
```

Type	RMSE	MAE	MAPE
Cross-validation	2.418	1.469	1.266
Training	2.414	1.465	1.261

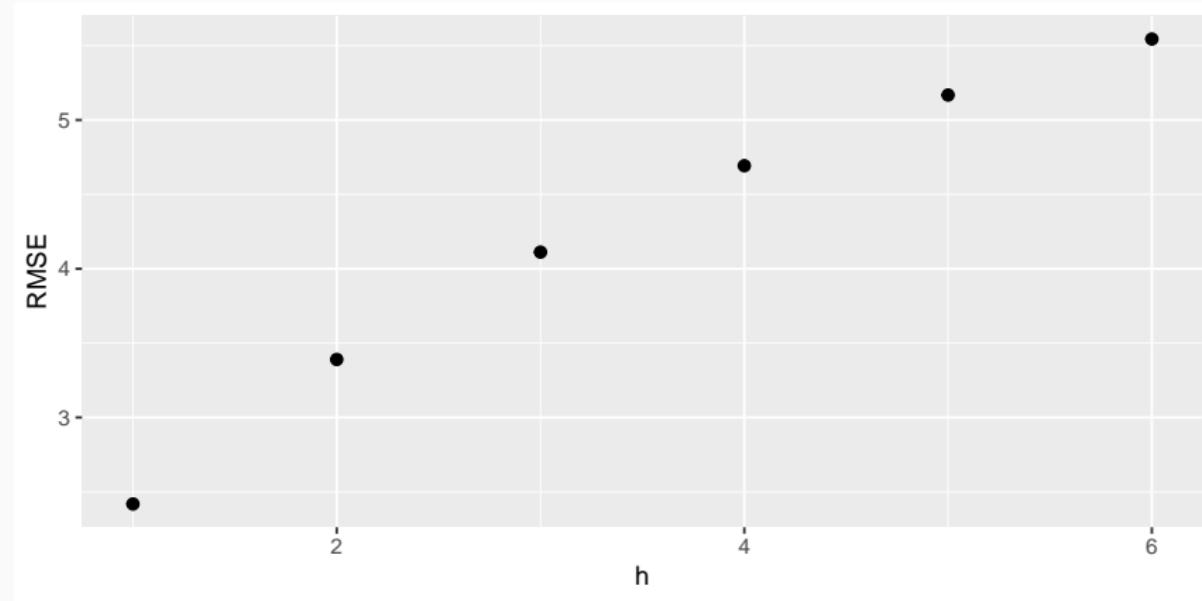
- A better choice to find the best forecasting model is to find the model with the smallest RMSE computed using time series cross-validation.

## Time series cross-validation

```
fc <- fb_stretch %>%
  model(RW(Close ~ drift())) %>%
  forecast(h = 6) %>%
  group_by(.id) %>%
  mutate(h = row_number()) %>%
  ungroup()

fc %>%
  accuracy(fb_stock, by = c("h", ".model")) %>%
  ggplot(aes(x = h, y = RMSE)) +
  geom_point(size = 2)
```

## Time series cross-validation



# Time series regression models



THE UNIVERSITY OF  
**AUCKLAND**  
Te Whare Wānanga o Tāmaki Makaurau  
NEW ZEALAND

# Outline

- 1 Linear models for time series
- 2 Useful predictors for time series regression
- 3 Predictor selection
- 4 Forecasting with regression
- 5 Matrix algebra
- 6 Multicollinearity

# Outline

- 1 Linear models for time series
- 2 Useful predictors for time series regression
- 3 Predictor selection
- 4 Forecasting with regression
- 5 Matrix algebra
- 6 Multicollinearity

## Multiple linear regression

$$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \cdots + \beta_k x_{K,t} + \varepsilon_t$$

- $y_t$ : the variable to be forecast (the **response** variable).
- $x_1, x_2, \dots, x_K$ : numerical “predictor” variables.
- We usually assume that the values of the predictor variables are known for past and future times.
- $\beta_1, \beta_2, \dots, \beta_K$ : measure the effect of each predictor after accounting for all the other predictors in the model. They measure the marginal effects of the **predictor variables**.
- $\varepsilon_t$ : white noise error term.
- When  $K = 1$ , we call this model as simple linear regression.
- The intercept should always be included unless we want otherwise.

## Assumptions

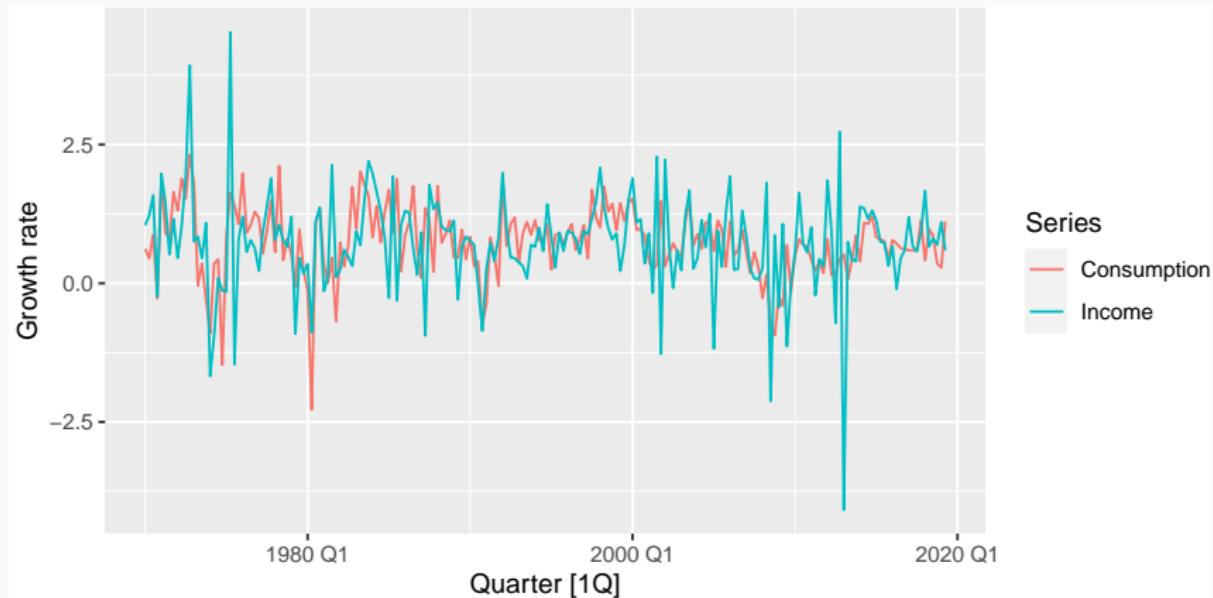
- The relationship between the forecast variable and the predictor variables is linear.
- We assume that the errors,  $\varepsilon_1, \varepsilon_2, \dots, \varepsilon_T$ :
  - ▶ have zero mean; otherwise the forecasts are biased.
  - ▶ are not autocorrelated; otherwise the forecasts are inefficient.
  - ▶ are unrelated to the predictor variables; otherwise there would be more information that should be included in the systematic part of the model.

### Useful (for producing prediction intervals and statistical tests)

- Errors are normally distributed with a constant variance  $\sigma^2$ .

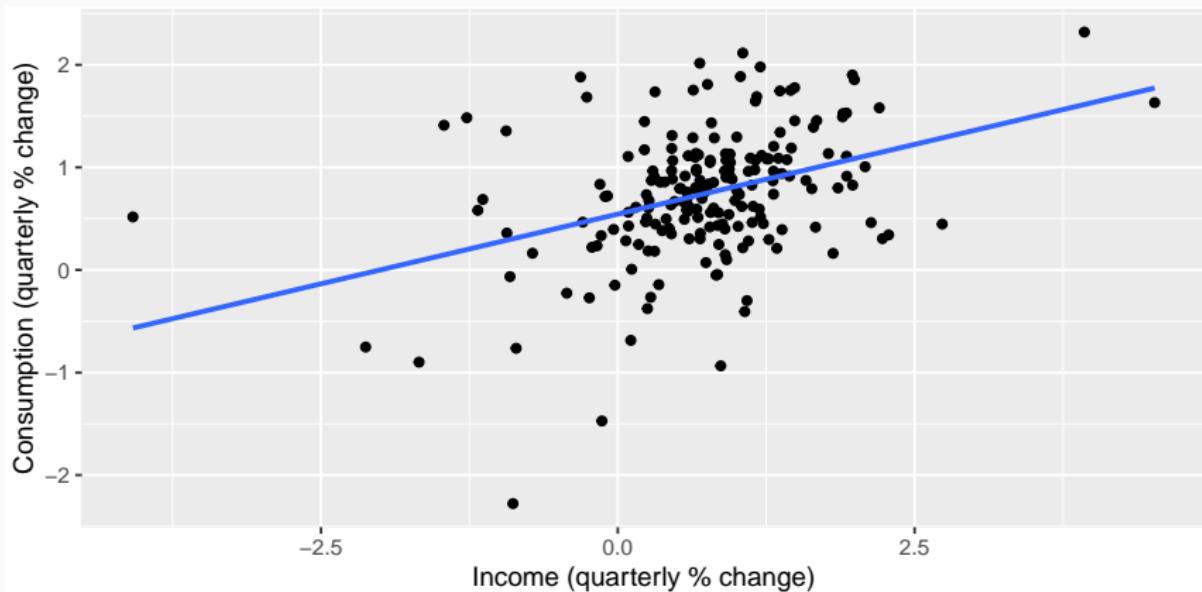
# US consumption expenditure

```
us_change %>%  
  pivot_longer(c(Consumption, Income), names_to = "Series") %>%  
  autoplot(value) +  
  labs(y = "Growth rate")
```



# US consumption expenditure

```
us_change %>%
  ggplot(aes(x = Income, y = Consumption)) +
  geom_point() + geom_smooth(method = "lm", se = FALSE) +
  labs(y = "Consumption (quarterly % change)",
       x = "Income (quarterly % change)")
```

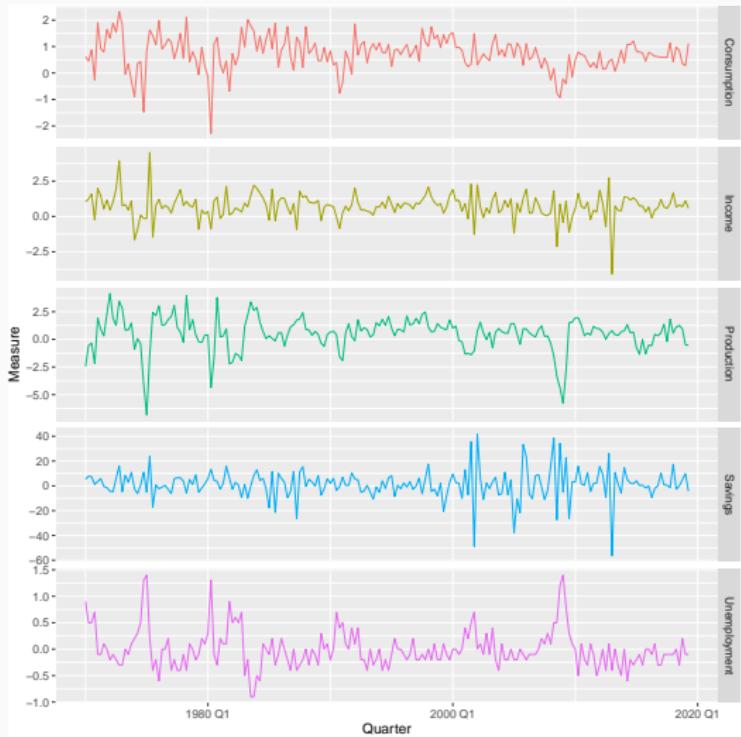


# US consumption expenditure

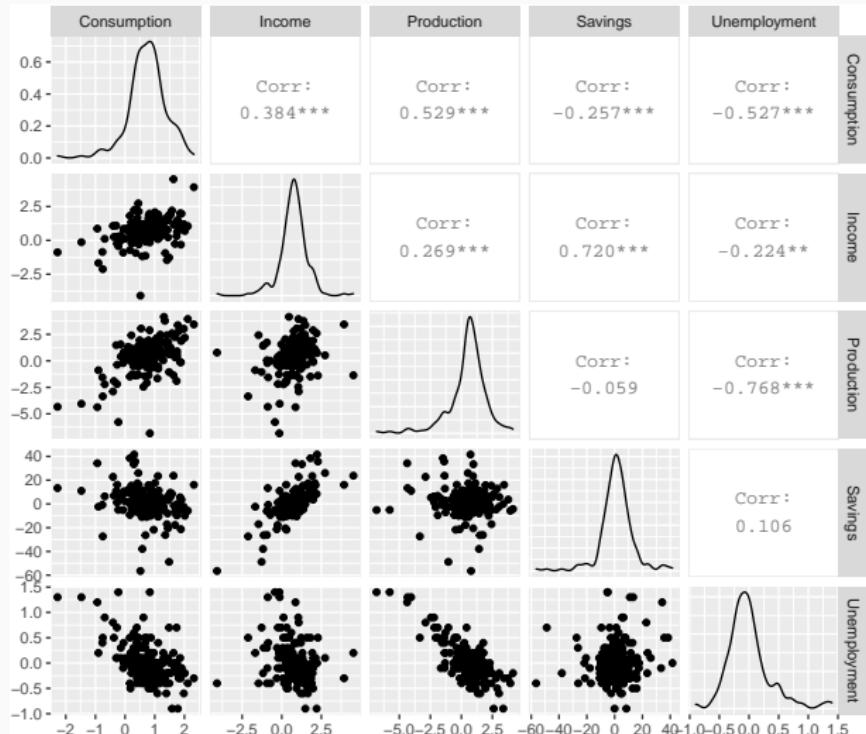
```
us_change %>%
  model(TSLM(Consumption ~ Income)) %>%
  report()

## Series: Consumption
## Model: TSLM
##
## Residuals:
##      Min     1Q Median     3Q    Max
## -2.582 -0.278  0.019  0.323  1.422
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.5445    0.0540   10.08 < 2e-16 ***
## Income       0.2718    0.0467    5.82  2.4e-08 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.591 on 196 degrees of freedom
## Multiple R-squared: 0.147, Adjusted R-squared: 0.143
## F-statistic: 33.8 on 1 and 196 DF, p-value: 2e-08
```

# US consumption expenditure



# US consumption expenditure



# US consumption expenditure

```
fit <- us_change %>%
  model(TSLM(Consumption ~ Income + Production + Savings + Unemployment))
report(fit)

## Series: Consumption
## Model: TSLM
##
## Residuals:
##      Min    1Q Median    3Q   Max
## -0.906 -0.158 -0.036  0.136  1.155
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.25311   0.03447    7.34  5.7e-12 ***
## Income      0.74058   0.04012   18.46  < 2e-16 ***
## Production  0.04717   0.02314    2.04   0.043 *
## Savings     -0.05289   0.00292  -18.09  < 2e-16 ***
## Unemployment -0.17469   0.09551   -1.83   0.069 .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.31 on 193 degrees of freedom
## Multiple R-squared: 0.768, Adjusted R-squared: 0.763
## F-statistic: 160 on 4 and 193 DF, p-value: <2e-16
```

# US consumption expenditure

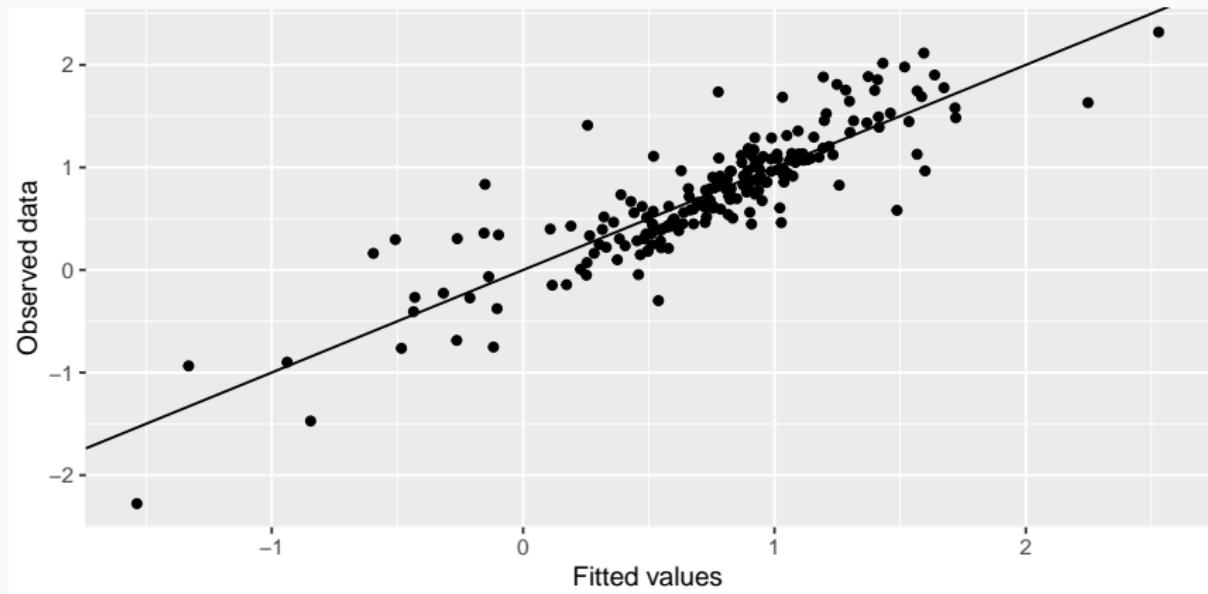
**Fitted values:** these are not genuine forecasts.

$$\hat{C}_t = 0.253 + 0.741I_t + 0.047P_t - 0.053S_t - 0.175U_t$$

for  $t = 1, 2, \dots, T$ .



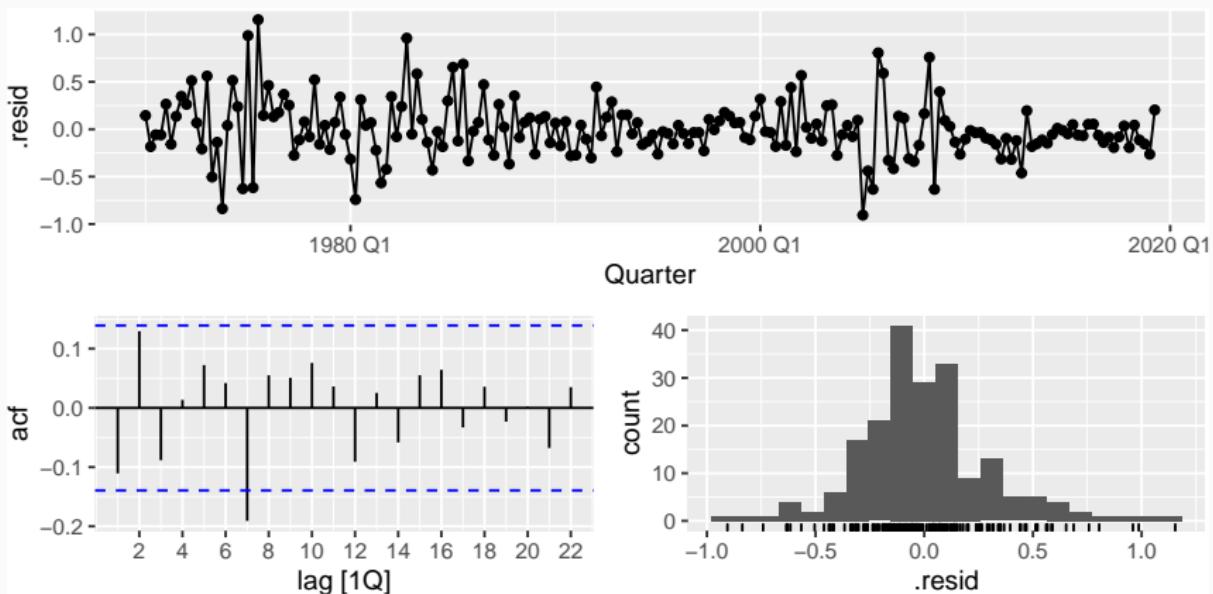
# US consumption expenditure



# US consumption expenditure

```
fit %>%
```

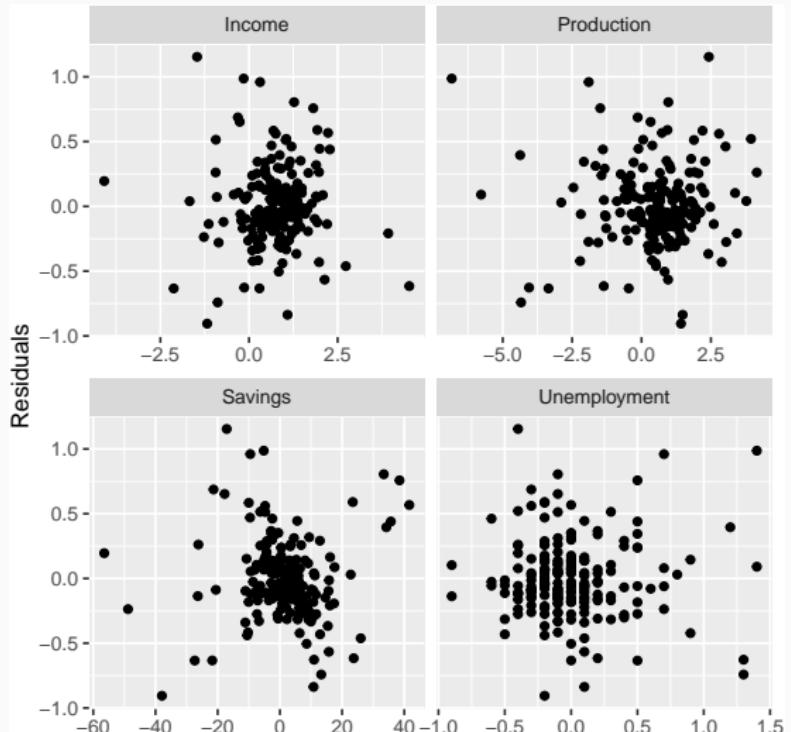
```
gg_tsresiduals()
```



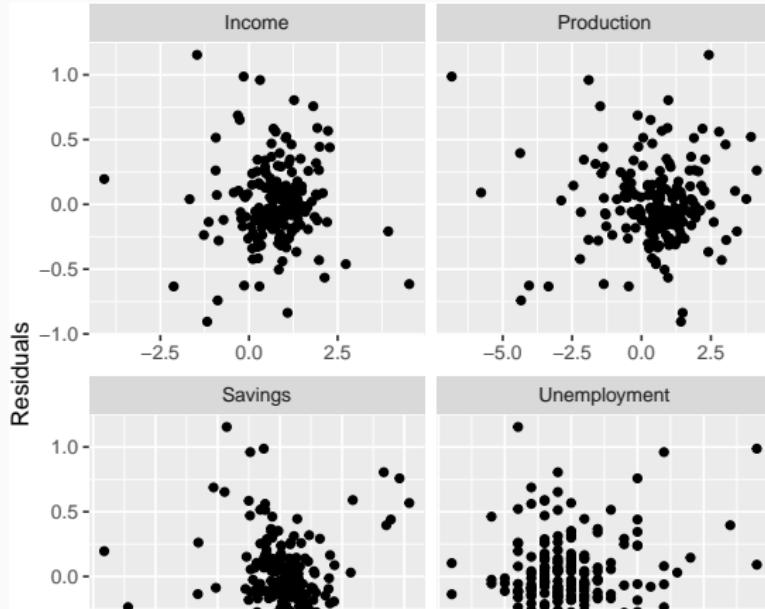
## US consumption expenditure

```
augment(fit) %>%  
  features(.resid, ljung_box, lag = 10, dof = 5)  
  
## # A tibble: 1 x 3  
##   .model                      lb_stat lb_pvalue  
##   <chr>                         <dbl>     <dbl>  
## 1 TSLM(Consumption ~ Income + Production ~      18.9    0.00204
```

# US consumption expenditure

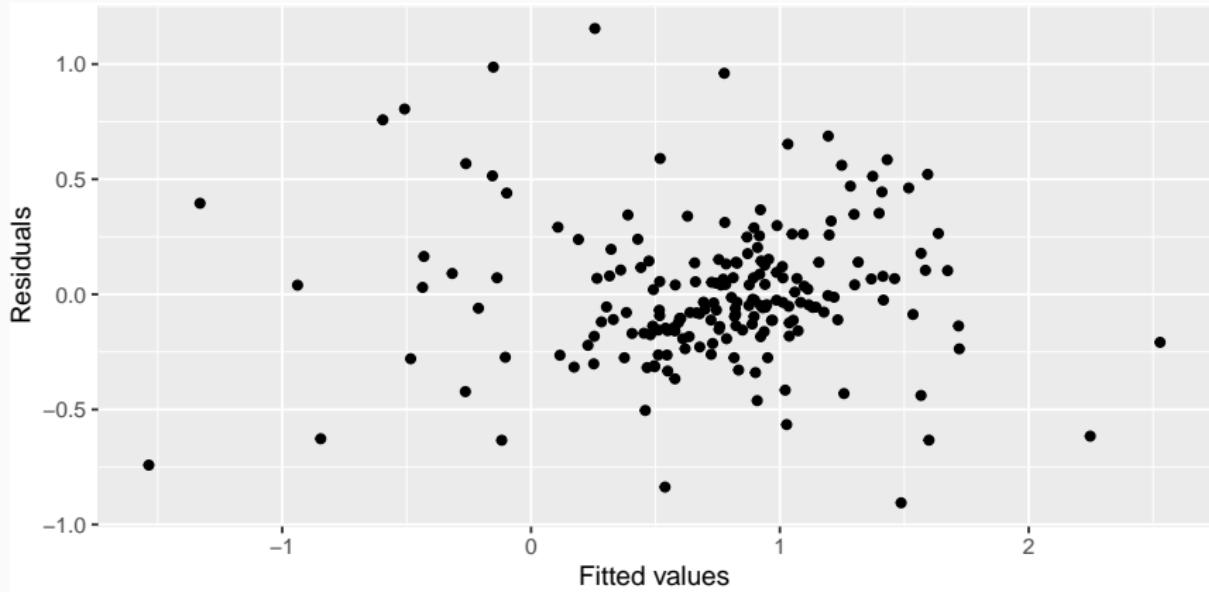


# US consumption expenditure

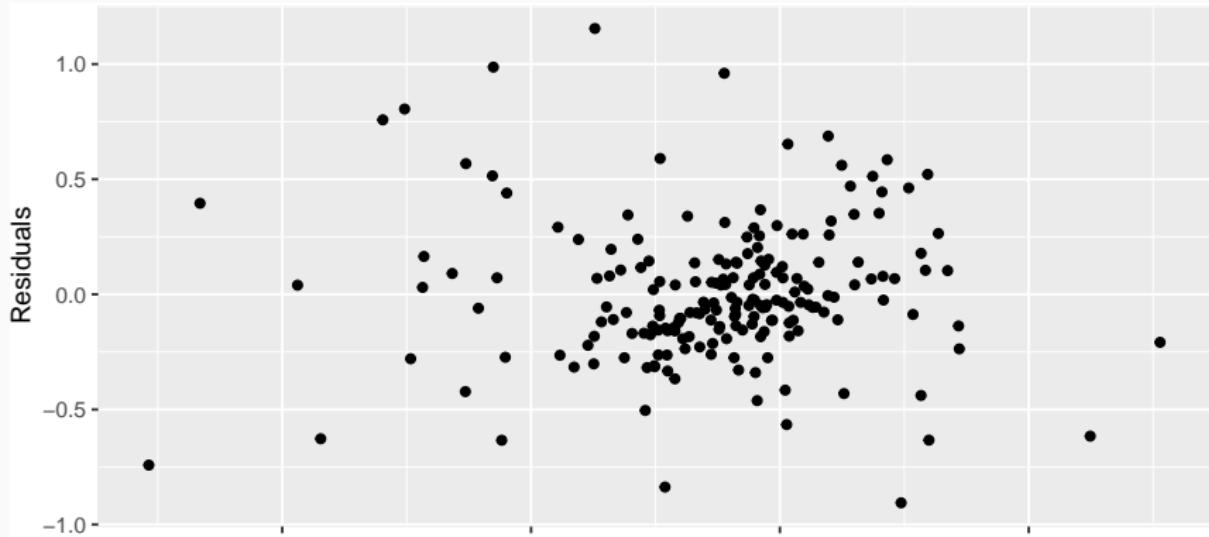


- If a pattern is visible, then the relationship is nonlinear.
- If a plot of residuals vs any predictor (not in the model) shows a pattern, then the predictor should be included in the model.

## US consumption expenditure



## US consumption expenditure



If a pattern is visible, then there is heteroscedasticity in the errors. We can try a transformation.

# Outline

- 1 Linear models for time series
- 2 Useful predictors for time series regression
- 3 Predictor selection
- 4 Forecasting with regression
- 5 Matrix algebra
- 6 Multicollinearity

- It is common for time series data to be trending.
- A linear trend can be modelled:

$$y_t = \beta_0 + \beta_1 t + \varepsilon_t, \quad \text{for } t = 1, 2, \dots, T.$$

- This assumes that the trend will continue into the future.
- A trend variable can be specified in the `TSLM()` function using the `trend()` special.

## Dummy variables

- It is possible to have categorical predictor variables.
- When a categorical variable takes two possible outcomes (say 'yes' or 'no'), we can construct a numeric variable by assigning 1 if yes and 0 if no. This is called a **dummy variable**.
- If there are more than two possible outcomes (say  $k$ ) then the number of dummy variables required is  $k - 1$ .
- For example: day of the week
  - ▶  $D_1 = 1$ , if Monday;  $D_1 = 0$ , otherwise.
  - ▶  $D_2 = 1$ , if Tuesday;  $D_2 = 0$ , otherwise.
  - ▶  $D_3 = 1$ , if Wednesday;  $D_3 = 0$ , otherwise.
  - ▶ so on.
- TSLM() will automatically construct these dummy variables if the predictor is specified as a factor variable.

## Avoid dummy variable trap

- When constructing dummy variables, a common mistake is to define too many variables.
- If a categorical variable takes  $k$  possible outcomes, we may consider constructing  $k$  dummy variables.
- Either omit the intercept, or omit the dummy variable for one outcome.
- Otherwise the regression will be singular.
- The coefficients of the dummy variables are interpreted relative to the omitted category.

# Uses of dummy variables

## Seasonal dummies

- For monthly data: 11 dummies
- For quarterly data: 3 dummies
- For daily data: 6 dummies

These are automatically handled by `TSLM()` if `season()` special is used.

## For weekly data?

## Outliers

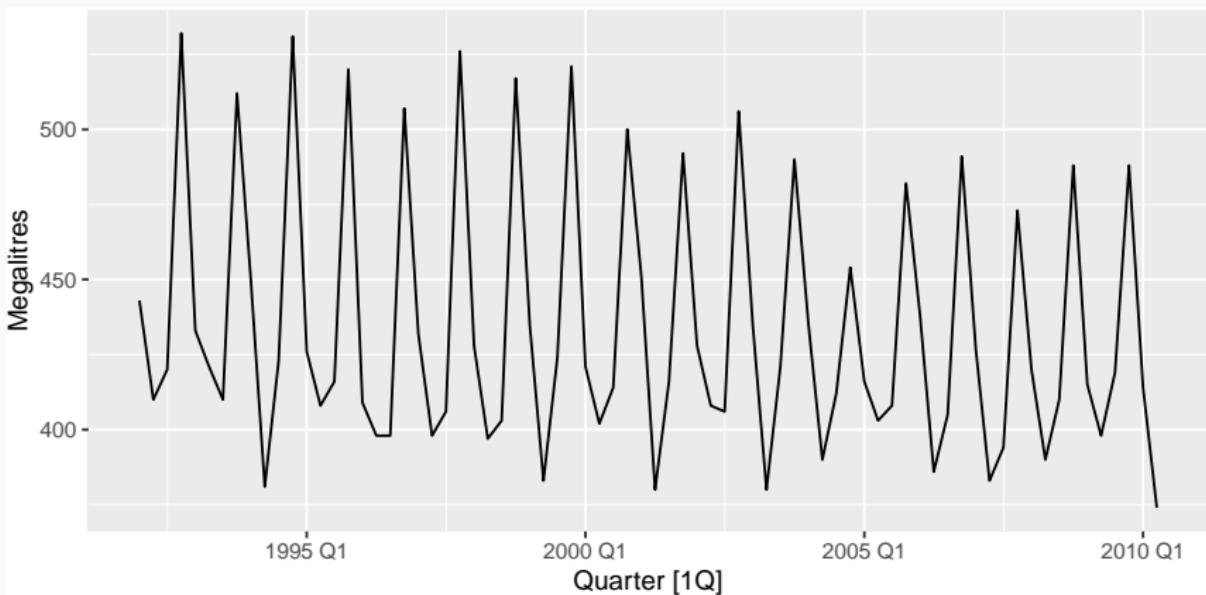
- Rather than omitting the outlier, we can use a dummy variable to remove its effect.
- The dummy variable takes value 1 for that observation and 0 everywhere else.

## Public holiday

- For daily data, the effect of public holidays can be accounted for by creating a dummy taking value 1 on public holidays and 0 elsewhere.

# Australian quarterly beer production

```
recent_production <- aus_production %>%  
  filter(year(Quarter) >= 1992)  
recent_production %>%  
  autoplot(Beer) +  
  labs(y = "Megalitres")
```



# Australian quarterly beer production

```
fit <- recent_production %>%
  model(TSLM(Beer ~ trend() + season()))
report(fit)

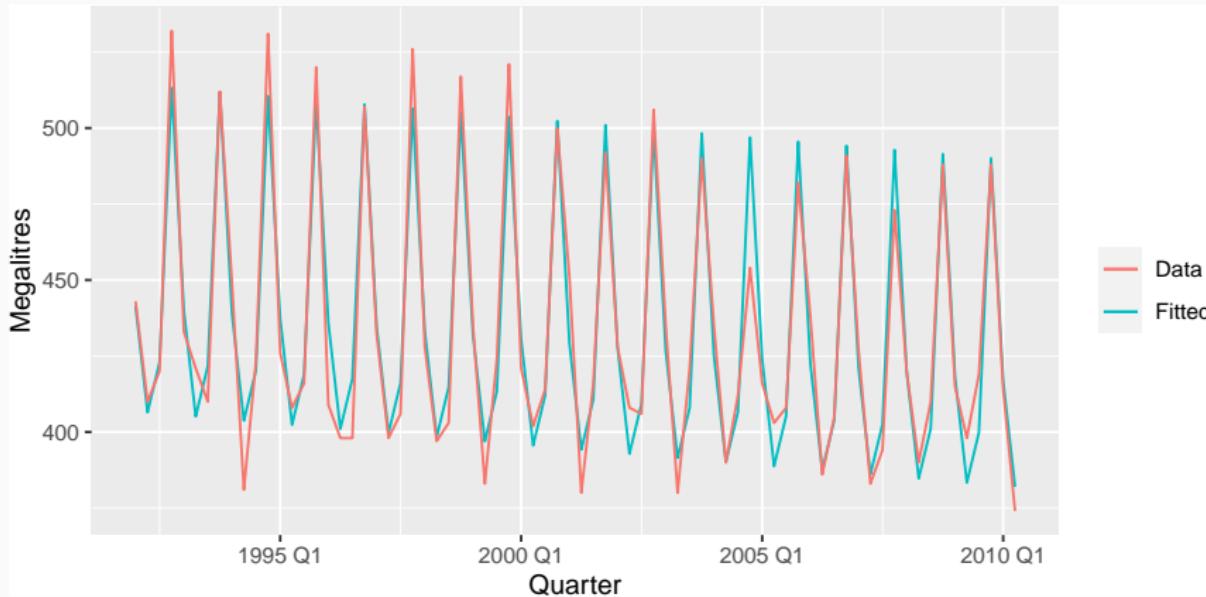
## Series: Beer
## Model: TSLM
##
## Residuals:
##      Min    1Q Median    3Q   Max
## -42.9  -7.6  -0.5   8.0  21.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 441.8004   3.7335 118.33 < 2e-16 ***
## trend()     -0.3403   0.0666  -5.11 2.7e-06 ***
## season()year2 -34.6597  3.9683  -8.73 9.1e-13 ***
## season()year3 -17.8216  4.0225  -4.43 3.4e-05 ***
## season()year4  72.7964  4.0230  18.09 < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.2 on 69 degrees of freedom
## Multiple R-squared: 0.924, Adjusted R-squared: 0.92
## F-statistic: 211 on 4 and 69 DF, p-value: <2e-16
```

# Australian quarterly beer production

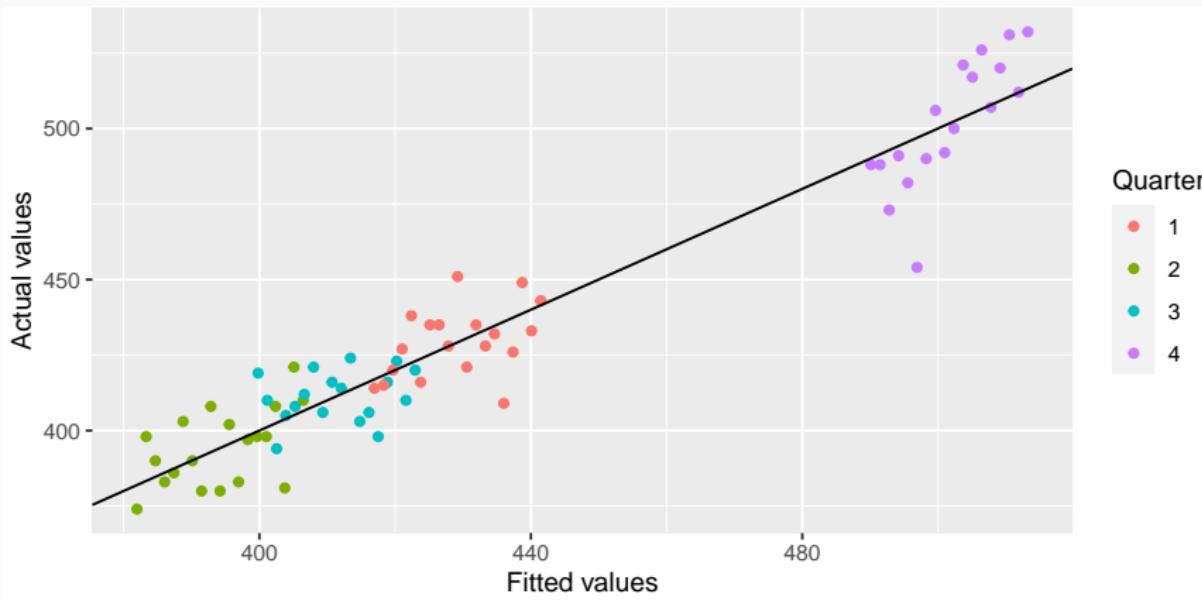
```
fit <- recent_production %>%
  model(TSLM(Beer ~ trend() + season()))
report(fit)

## Series: Beer
## Model: TSLM
##
## Residuals:
##      Min    1Q Median    3Q   Max
## -42.9  -7.6  -0.5   8.0  21.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 441.8004   3.7335 118.33 < 2e-16 ***
## trend()     -0.3403   0.0666  -5.11  2.7e-06 ***
## season()year2 -34.6597  3.9683  -8.73  9.1e-13 ***
## season()year3 -17.8216  4.0225  -4.43  3.4e-05 ***
## season()year4  72.7964  4.0230  18.09 < 2e-16 ***
## ---
## Signif. codes:
##  ■ First quarter variable has been omitted.
##  ■ Coefficients associated with the other quarters are interpreted relative to this.
## Multiple R-squared: 0.924,   Adjusted R-squared: 0.92
## F-statistic: 211 on 4 and 69 DF, p-value: <2e-16
```

# Australian quarterly beer production

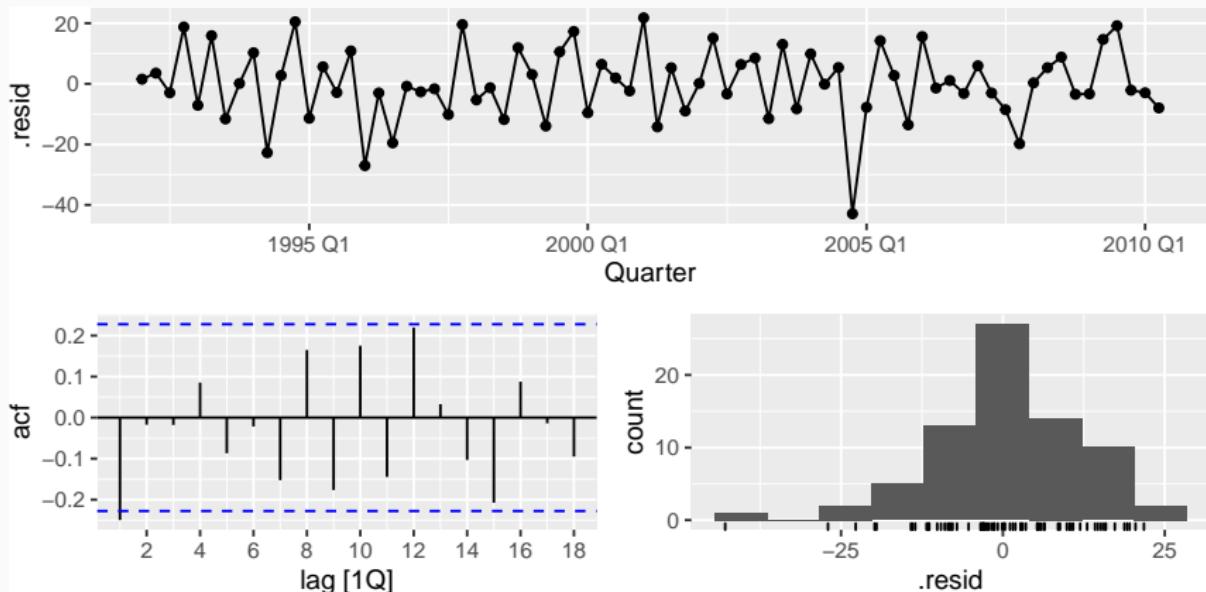


# Australian quarterly beer production



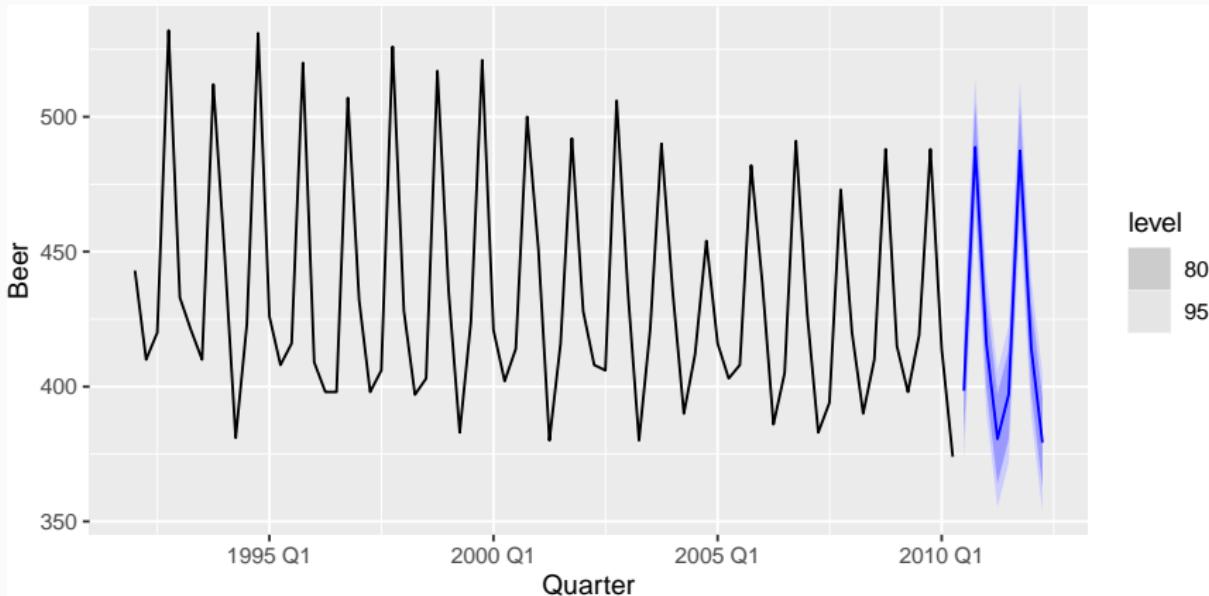
# Australian quarterly beer production

```
fit %>% gg_tsresiduals()
```



# Australian quarterly beer production

```
fit %>%
  forecast() %>%
  autoplot(recent_production)
```



## Fourier series

- An alternative to seasonal dummies, especially suitable for long seasonal periods.
- Every periodic function can be approximated by a sum of sine and cosine terms for sufficiently large  $K$ :

$$y_t = a + bt + \sum_{k=1}^K \alpha_k s_k(t) + \beta_k c_k(t) + \varepsilon_t,$$

$$s_k(t) = \sin\left(\frac{2\pi k t}{m}\right) \quad c_k(t) = \cos\left(\frac{2\pi k t}{m}\right),$$

where  $m$  is the seasonal period.

- Maximum allowed  $K = m/2$ . We choose  $K$  by minimizing AICc.
- A regression model with Fourier terms is sometimes called as **harmonic regression**.

- For monthly seasonality: if we use 6 Fourier pairs as predictor variables, then we get exactly the same forecasts as using 11 dummy variables.
- We often need fewer Fourier terms than dummy variables, especially when  $m$  is large.
- This is useful for weekly data:  $m \approx 52$ .

```
TSLM(y ~ trend() + fourier(K))
```

# Harmonic regression: beer production

```
recent_production %>%
  model(TSLM(Beer ~ trend() + fourier(K = 2))) %>%
  report()

## Series: Beer
## Model: TSLM
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -42.9    -7.6    -0.5     8.0    21.8
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)             446.8792    2.8732 155.53 < 2e-16 ***
## trend()                  -0.3403    0.0666  -5.11  2.7e-06 ***
## fourier(K = 2)C1_4      8.9108    2.0112   4.43  3.4e-05 ***
## fourier(K = 2)S1_4   -53.7281    2.0112  -26.71 < 2e-16 ***
## fourier(K = 2)C2_4   -13.9896    1.4226  -9.83  9.3e-15 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 12.2 on 69 degrees of freedom
## Multiple R-squared: 0.924, Adjusted R-squared: 0.92
## F-statistic: 211 on 4 and 69 DF, p-value: <2e-16
```

## Intervention variables

- Often we need to model interventions that may have affected the variable to be forecast.
- For example: competitor activity, change to a procedure or law etc.

### Spikes

- Intervention effect lasts only for one period.
- This can be handled by a variable that takes value 1 in the period of the intervention and 0 elsewhere.

### Steps

- Intervention causes a level shift (sudden and permanent shift).
- Step variable takes value 0 before the intervention and 1 from the time of intervention.

### Change of slope

- The intervention can be handled by creating a variable that takes zero before the intervention and then takes values 1, 2, ... afterwards.

## For monthly data

- Easter: use a dummy variable  $d_t = 1$  if any part of Easter is in that month,  $d_t = 0$ , otherwise.

If Easter falls in March then dummy variable takes value 1 in March and if it falls in April the dummy variable takes value 1 in April.

If Easter starts in March and ends in April, then the dummy variable takes 1 for both months.

- Ramadan and Chinese new year is handled similarly.
- Christmas: always in December so part of monthly seasonal effect.

## Trading days

With monthly data, if the measures vary depending on how many different types of days in the month, then trading day predictor can be useful:

$x_1 = \#\text{Mondays in month};$

$x_2 = \#\text{Tuesdays in month};$

$\vdots$

$x_7 = \#\text{Sundays in month}.$

## Distributed lags

Lagged values of a predictor can also be useful:

Example: Effect of advertising expenditure can last beyond the actual campaign:

$x_1$  = advertising expenditure for last month;

$x_2$  = advertising expenditure two months before;

⋮

$x_m$  = advertising expenditure  $m$  months before.

## Non-linear trend

- The simplest way of fitting non-linear trends is to use quadratic or higher order trends.

$$x_{1,t} = t, x_{2,t} = t^2, \dots$$

**Not recommended**

- We can use a piecewise linear trend which bends at some time.
- If the trend bends at time  $\tau$ , then the predictors are

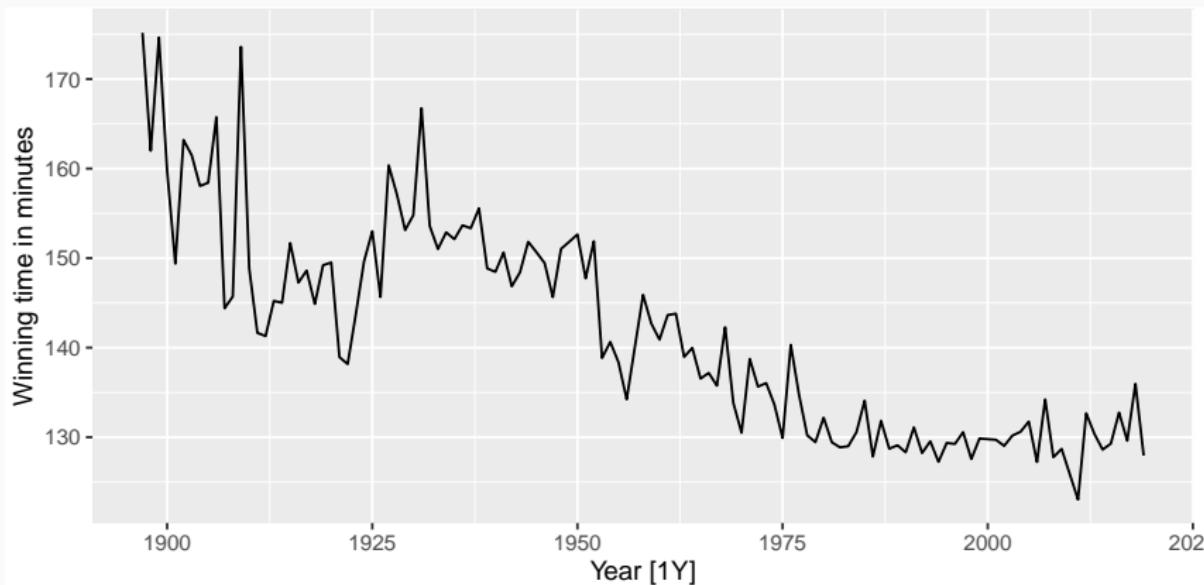
$$x_{1,t} = t,$$

$$x_{2,t} = (t - \tau)_+ = \begin{cases} 0, & t < \tau, \\ t - \tau, & t \geq \tau. \end{cases}$$

- Additional bends can be included by adding further variables of the form  $(t - \tau)_+$ , where  $\tau$  is the point at which the line should bend.

# Boston marathon winning times

```
marathon <- boston_marathon %>%  
  filter(Event == "Men's open division") %>% select(-Event) %>%  
  mutate(Minutes = as.numeric(Time)/60)  
marathon %>%  
  autoplot(Minutes) + ylab("Winning time in minutes")
```



# Boston marathon winning times

```
fit_trends <- marathon %>%
  model(linear = TSLM(Minutes ~ trend()),
        exponential = TSLM(log(Minutes) ~ trend()),
        piecewise = TSLM(Minutes ~ trend(knots = c(1940, 1980))))
fit_trends
```

```
## # A mable: 1 x 3
##      linear exponential piecewise
##      <model>      <model>    <model>
## 1  <TSLM>      <TSLM>    <TSLM>
```

# Boston marathon winning times

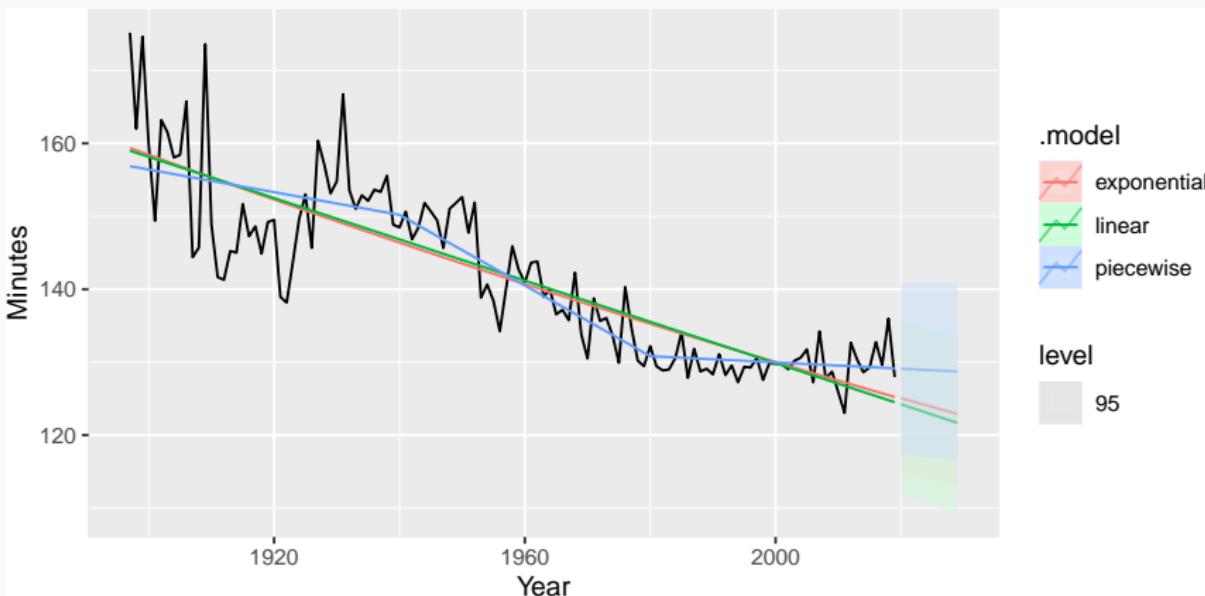
```
fit_trends <- marathon %>%  
  model(linear = TSLM(Minutes ~ trend()),  
        exponential = TSLM(log(Minutes) ~ trend()),  
        piecewise = TSLM(Minutes ~ trend(knots = c(1940, 1980))))  
fit_trends
```

```
## # A mable: 1 x 3  
##   linear exponential piecewise  
##   <model>      <model>    <model>  
## 1  <TSLM>      <TSLM>    <TSLM>
```

- Subjective identification of knots can lead to over-fitting.
- This can affect the forecast performance of a model.
- Hence should be performed with caution.

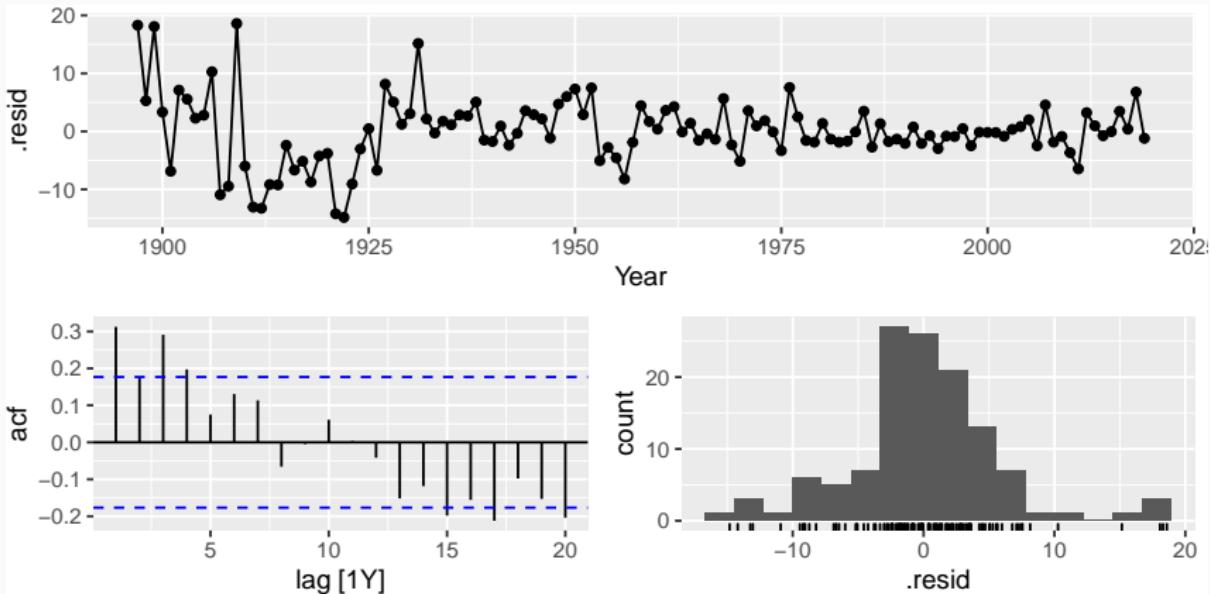
# Boston marathon winning times

```
fc_trends <- fit_trends %>% forecast(h = 10)
fc_trends %>%
  autoplot(marathon, level = 95, alpha = 0.5) +
  geom_line(data = fitted(fit_trends),
             aes(y = .fitted, colour = .model))
```



# Boston marathon winning times

```
fit_trends %>%
  select(piecewise) %>%
  gg_tsresiduals()
```



# Outline

- 1 Linear models for time series
- 2 Useful predictors for time series regression
- 3 Predictor selection
- 4 Forecasting with regression
- 5 Matrix algebra
- 6 Multicollinearity

## $R^2$ : Coefficient of determination

- Software outputs for regression will always give the value of  $R^2$ .
- It measures how well the model fits the historical data, but doesn't indicate how well the model will forecast future data.
- It is computed by

$$R^2 = \frac{\sum_{t=1}^T (\hat{y}_t - \bar{y})^2}{\sum_{t=1}^T (y_t - \bar{y})^2} = \frac{RSS}{TSS} = 1 - \frac{ESS}{TSS},$$

where  $TSS$  is the total sum of squares,  $RSS$  is the regression sum of squares,  $ESS$  is the error sum of squares and  $TSS = RSS + ESS$

- It explains the proportion of variability accounted by the predictors used in the model.
- It is equal to the square of the correlation between  $y_t$  and  $\hat{y}_t$ .
- Maximizing  $R^2$  is equivalent to minimizing  $ESS$ .

## Adjusted $R^2$

- $R^2$  doesn't allow for "degrees of freedom".
- Adding more variables tends to increase the value of  $R^2$ , even if they are irrelevant.
- Forecasters should not use  $R^2$  to determine whether a model will give good predictions or not.
- To overcome this problem, we can use **adjusted  $R^2$** :

$$\bar{R}^2 = 1 - \frac{ESS/(T - K - 1)}{TSS/(T - 1)} = 1 - (1 - R^2) \frac{T - 1}{T - K - 1},$$

where  $K$  is the no. of predictors and  $T$  is the no. of observations.

- Maximizing  $\bar{R}^2$  is equivalent to minimizing

$$\hat{\sigma}^2 = \frac{1}{T - K - 1} \sum_{t=1}^T e_t^2.$$

- Maximizing  $\bar{R}^2$  works quite well, but it tends to select too many predictors.

## Cross-validation

- We learned about time series cross-validation for determining the predictive ability of a model.
  - For regression, we can also use traditional leave-one-out cross-validation to select predictors.
- 1 Remove observation  $t$  from the data set, and fit the model using remaining data.
  - 2 Compute  $e_t^* = y_t - \hat{y}_t$  for the omitted observation.
  - 3 Repeat step 1 for  $t = 1, 2, \dots, T$ .
  - 4 Compute MSE from  $e_1^*, e_2^*, \dots, e_T^*$ .
- Re-estimating the model  $T$  times is time-consuming.
  - There is a formula which we can use to compute this measure.

## Akaike's Information Criterion

$$AIC = -2\log(L) + 2(K + 2),$$

where  $L$  is the likelihood function and  $K$  is the no. of predictors in the model.

- The second term accounts for  $(K + 2)$  parameters in the model:  $K$  predictors, intercept and the variance of the residuals.
- For Gaussian likelihood

$$AIC = T \log \left( \frac{ESS}{T} \right) + 2(K + 2).$$

- The model with the lowest AIC is often the best model for forecasting.
- For any linear regression, minimizing AIC is asymptotically equivalent to minimizing MSE via leave-one-out cross-validation.

## Corrected AIC and BIC

- AIC tends to select too many predictors for small values of  $T$ .
- A bias-corrected version of AIC has been proposed:

### Corrected AIC

$$AIC_c = AIC + \frac{2(K+2)(K+3)}{T-K-3}$$

### Schwarz's Bayesian Information Criterion

$$BIC = -2\log(L) + (K+2)\log(T)$$

- Minimizing the BIC tends to give the best model.
- BIC penalizes more than AIC, hence either the same model as AIC or one with fewer terms will be chosen.
- For any linear regression, minimizing BIC is asymptotically equivalent to minimizing MSE via leave- $\nu$ -out cross-validation where  
$$\nu = T[1 - 1/(\log(T) - 1)].$$

- $\bar{R}^2$  tends to select too many predictors, can be less useful for forecasting.
- If there is a true underlying model, BIC will select that model for sufficiently large  $T$ .
- In reality, it is rare that such true model exists and finding the true model will not necessarily give the best forecasts (parameters may be inaccurate).
- We recommend to use one of AIC, AICc or CV as each has forecasting as their objective.
- For large  $T$ , they will lead to the same model.

# Comparing regression models

```
glance(fit_trends) %>%
  select(.model, r_squared, adj_r_squared, AICc, CV)

## # A tibble: 3 x 5
##   .model      r_squared adj_r_squared    AICc        CV
##   <chr>        <dbl>        <dbl> <dbl> <dbl>
## 1 linear       0.728       0.726  452.  39.1
## 2 exponential  0.744       0.742 -779.  0.00176
## 3 piecewise    0.767       0.761  438.  34.8
```

# Comparing regression models

```
glance(fit_trends) %>%  
  select(.model, r_squared, adj_r_squared, AICc, CV)
```

```
## # A tibble: 3 x 5  
##   .model      r_squared adj_r_squared    AICc       CV  
##   <chr>        <dbl>        <dbl> <dbl> <dbl>  
## 1 linear       0.728       0.726  452.  39.1  
## 2 exponential  0.744       0.742 -779.  0.00176  
## 3 piecewise    0.767       0.761  438.  34.8
```

Be careful when comparing models with different transformations!

## Best subset regression

- Fit all possible regression models using one or more predictors.
- Pick the best model using one of the measures of predictive ability (CV/AIC/AICc)
- If there are larger number of predictors, this is impossible (no. of models =  $2^K$ ).
- If there are 20 variables we need to fit 1 million models!

## Model selection

I <sup>1</sup>	P <sup>1</sup>	S <sup>1</sup>	U <sup>1</sup>	AdjR2	CV	AIC	AICc	BIC
✓	✓	✓	✓	0.763	0.104	-456.6	-456.1	-436.9
✓	✓		✓	0.761	0.105	-455.2	-454.9	-438.7
✓		✓	✓	0.760	0.104	-454.4	-454.0	-437.9
✓			✓	0.735	0.114	-435.7	-435.5	-422.6
✓	✓	✓		0.366	0.271	-262.3	-262.0	-245.8
	✓	✓	✓	0.349	0.279	-257.1	-256.8	-240.7
✓		✓		0.345	0.276	-256.9	-256.6	-243.7
✓	✓			0.336	0.282	-254.2	-254.0	-241.0
	✓		✓	0.324	0.287	-250.7	-250.5	-237.5
		✓	✓	0.311	0.291	-246.9	-246.7	-233.7
✓	✓			0.308	0.293	-246.1	-245.9	-232.9
✓				0.276	0.304	-238.1	-238.0	-228.2
			✓	0.274	0.303	-237.4	-237.3	-227.5
✓				0.143	0.356	-204.6	-204.5	-194.7
			✓	0.061	0.388	-186.5	-186.4	-176.7
				0.000	0.409	-175.1	-175.0	-168.5

<sup>1</sup>I: Income, P: Production, S: Savings, and U: Unemployment.

## Backwards stepwise regression

- Start with the model including all predictor variables.
- Remove one variable at a time. Keep the model if it improves the measure of predictive accuracy.
- Iterate until there is no further improvement.
- It is not suitable if the no. of predictors is too large.
- We can use **forward stepwise regression**.
- We can also use a hybrid approach.
- Stepwise regression does not guarantee to give the best possible model, but often leads to a good model.

## Backwards stepwise regression

- Start with the model including all predictor variables.
- Remove one variable at a time. Keep the model if it improves the measure of predictive accuracy.
- Iterate until there is no further improvement.
- It is not suitable if the no. of predictors is too large.
- We can use **forward stepwise regression**.
- We can also use a hybrid approach.
- Stepwise regression does not guarantee to give the best possible model, but often leads to a good model.

- The selected model is not helpful for studying the effect of any predictor on the forecast variable.
- These procedures are helpful when the model is used for forecasting.

# Outline

- 1 Linear models for time series
- 2 Useful predictors for time series regression
- 3 Predictor selection
- 4 Forecasting with regression
- 5 Matrix algebra
- 6 Multicollinearity

## Forecasting with regression

There are two types of forecasts that we can produce depending on the information we have when the forecasts are computed.

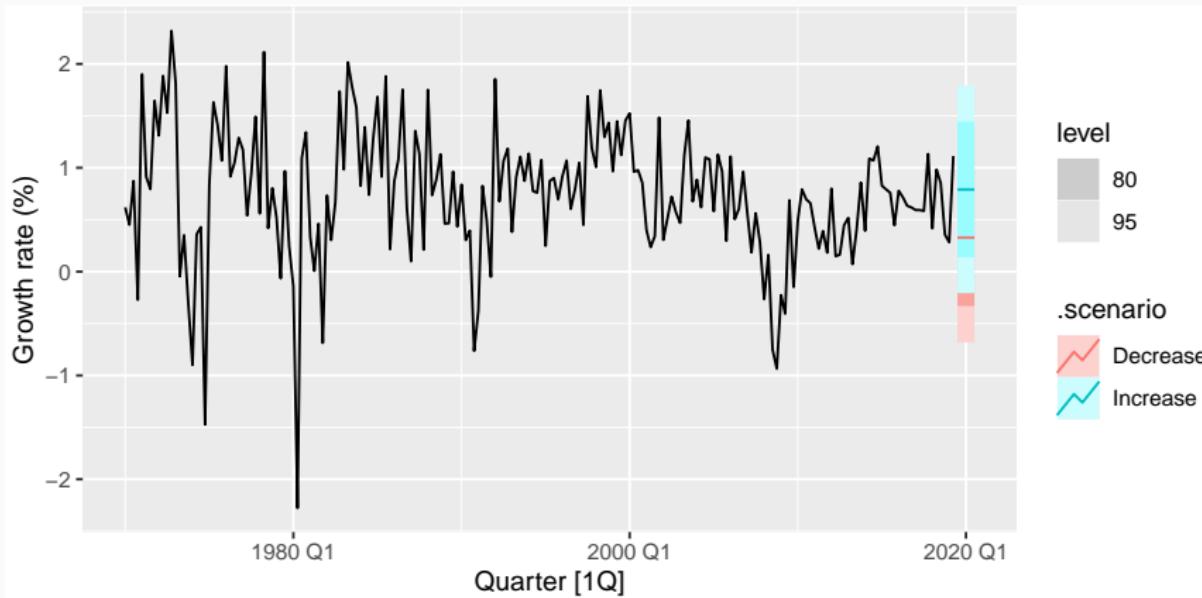
- **Ex-ante forecasts** are made using only the information that is available in advance.
  - ▶ Might require forecasts of predictors.
  - ▶ We can use one of the benchmarks methods we discussed or some advanced time series approach.
- **Ex-post forecasts** are made using later information on the predictors.
  - ▶ These are not genuine forecasts, but useful for studying the behavior of forecasting models.
- For trend, seasonal and other calendar variables, the values are known in advance, so need not to be forecast.

## Scenario based forecasting

- Forecaster assumes possible scenarios for the predictor variables.
- Prediction intervals do not include the uncertainty associated with the future values of the predictor variables.

```
fit_best <- us_change %>%
  model(lm = TSLM(Consumption ~ Income + Production + Unemployment))
future_scenarios <- scenarios(
  Increase = new_data(us_change, 5) %>%
    mutate(Income = 1, Production = 0.5, Unemployment = 0),
  Decrease = new_data(us_change, 5) %>%
    mutate(Income = -1, Production = -0.5, Unemployment = 0))
fc <- forecast(fit_best, new_data = future_scenarios)
us_change %>%
  autoplot(Consumption) +
  autolayer(fc) +
  ylab("Growth rate (%)")
```

## Scenario based forecasting



## Building predictive regression models

- A challenge with regression models is about generating ex-ante forecasts.
- Sometimes, obtaining forecasts of the predictors can be challenging.
- It can be worse than forecasting directly the forecast variable without any predictors.
- An alternative:

$$y_t = \beta_0 + \beta_1 x_{1,t-h} + \beta_2 x_{2,t-h} + \cdots + \beta_K x_{K,t-h} + \varepsilon_t, \quad h = 1, 2, \dots$$

- A different model for each  $h$ .
- This is useful when changes in one of the predictor variables does not have an instantaneous effect on the variable to be forecast.

# Outline

- 1 Linear models for time series
- 2 Useful predictors for time series regression
- 3 Predictor selection
- 4 Forecasting with regression
- 5 Matrix algebra
- 6 Multicollinearity

## Matrix formulation

$y_t = \beta_0 + \beta_1 x_{1,t} + \beta_2 x_{2,t} + \cdots + \beta_k x_{k,t} + \varepsilon_t,$   
for  $t = 1, 2, \dots, T$  and  $\varepsilon_t$  has mean zero and variance  $\sigma^2$ .

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix} = \begin{bmatrix} 1 & x_{1,1} & x_{2,1} & \dots & x_{K,1} \\ 1 & x_{1,2} & x_{2,2} & \dots & x_{K,2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{1,T} & x_{2,T} & \dots & x_{K,T} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_K \end{bmatrix} + \begin{bmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_T \end{bmatrix}$$
$$\mathbf{y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where  $\boldsymbol{\varepsilon}$  has mean  $\mathbf{0}$  and variance  $\sigma^2 I_T$

## Least squares estimation

Minimize:  $\varepsilon^\top \varepsilon = (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$

Differentiate w.r.t  $\boldsymbol{\beta}$  and equating to zero gives:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$$

- $\mathbf{X}$  should be of full column rank for  $\mathbf{X}^\top \mathbf{X}$  to be invertible.
- If we fall into dummy variable trap then  $\mathbf{X}$  is not of full column rank.

## Fitted values and residuals

Fitted values:  $\hat{\mathbf{y}} = \mathbf{X}\hat{\boldsymbol{\beta}}$

Residuals:  $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}}$

Residual variance is given by

$$\hat{\sigma}^2 = \frac{1}{T - K - 1} (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})^\top (\mathbf{y} - \mathbf{X}\hat{\boldsymbol{\beta}})$$

Let  $\mathbf{H} = \mathbf{X}(\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top$  (we refer as hat-matrix).

$$CV = \frac{1}{T} \sum_{t=1}^T \left( \frac{e_t}{1 - h_{tt}} \right)^2,$$

where  $e_t$ :  $t$ -th component of  $\mathbf{e}$  and  $h_{tt}$ :  $t$ -th diagonal element of  $\mathbf{H}$ .

## Likelihood function

If the errors are IID and normally distributed, then

$$\mathbf{y} \sim N(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}_T).$$

The likelihood function is

$$L = \frac{1}{(2\pi)^T/2 \sigma^T} \exp \left( -\frac{1}{2\sigma^2} (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) \right).$$

- $L$  is maximized when  $(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top (\mathbf{y} - \mathbf{X}\boldsymbol{\beta})$  is minimized.

$$\text{MLE}(\boldsymbol{\beta}) = \text{OLS}(\boldsymbol{\beta})$$

## Forecasts from regression model

Given  $\mathbf{x}^*$ , a row vector containing the future values of the predictors:

### Forecast

$$\hat{y}^* = E[y^* | \mathbf{X}, \mathbf{x}^*] = \mathbf{x}^* \hat{\beta} = \mathbf{x}^* (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

### Forecast variance

$$\text{Var}(y^* | \mathbf{X}, \mathbf{x}^*) = \sigma^2 [1 + \mathbf{x}^* (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}^{*\top}]$$

- The forecast variance doesn't account for the uncertainty in  $\mathbf{x}^*$ .
- Assuming that the forecast errors are normally distributed, 95% prediction interval

$$\hat{y}^* \pm 1.96 \hat{\sigma} \sqrt{[1 + \mathbf{x}^* (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{x}^{*\top}]}.$$

- If the future values of predictors are uncertain the prediction intervals can be narrower.

# Outline

- 1 Linear models for time series
- 2 Useful predictors for time series regression
- 3 Predictor selection
- 4 Forecasting with regression
- 5 Matrix algebra
- 6 Multicollinearity

Multicollinearity occurs when

- Two predictors are highly correlated:
  - ▶ knowing the value of one variable provides a lot of information about the other variable.
  - ▶ provides similar information.
- A linear combination of some of the predictors is highly correlated with another predictor.
  - ▶ For quarterly data, we have 4 dummies:  $d_1, d_2, d_3$ , and  $d_4$ .
  - ▶ We can write  $d_4 = 1 - d_1 - d_2 - d_3$ , so  $d_4$  and  $d_1 + d_2 + d_3$  are perfectly negatively correlated.

When multicollinearity is present

- numerical estimates of coefficients may not be reliable.
- uncertainty of the coefficients will be large.
- statistical tests are unreliable. So don't rely on p-values to determine the significance.
- we can't interpret the contribution of each separate predictor to the forecast.
- there is no problem with predictions provided that the values of the predictors used for forecasting are within the range used for fitting.
- can be problematic with scenario forecasting, as they should take account of the relationship between predictors.
- omitting and combining variables can be helpful.

## Outliers and influential observations

- **Outliers:** observations that take extreme values compared to the majority of the data.
- **Influential observations:** observations that have a large influence on the estimated coefficients of a regression model.
- Influential observations are also outliers that are extreme in the  $x$  direction.
- Outliers shouldn't be removed without a good explanation of why they are different.

# Exponential smoothing



THE UNIVERSITY OF  
**AUCKLAND**  
Te Whare Wānanga o Tāmaki Makaurau  
NEW ZEALAND

# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

## History

- Exponential smoothing developed during 1950–1960 as methods (algorithms) to produce point forecasts.
- Forecasts produced are weighted averages of past observations, where the weights decay exponentially as the observations get older.
- Equivalent ETS state space models developed during 1990–2000.
- They provide statistical models that underlie exponential smoothing methods.
- Generate identical point forecasts to equivalent exponential smoothing methods.
- Also generate prediction intervals.

## Simple forecasting methods

Given a time series:  $y_1, y_2, \dots, y_T$  (with no trend or seasonality)

### Naïve (random walk) forecasts

$$\hat{y}_{T+h|T} = y_T, \quad \text{for } h = 1, 2, \dots$$

### Average method

$$\hat{y}_{T+h|T} = \frac{1}{T} \sum_{t=1}^T y_t, \quad \text{for } h = 1, 2, \dots$$

- Naïve method assigns all the weight to the last observation.
- Average method assigns equal weight to all observations.

# Simple forecasting methods

Given a time series:  $y_1, y_2, \dots, y_T$  (with no trend or seasonality)

## Naïve (random walk) forecasts

$$\hat{y}_{T+h|T} = y_T, \quad \text{for } h = 1, 2, \dots$$

## Average method

$$\hat{y}_{T+h|T} = \frac{1}{T} \sum_{t=1}^T y_t, \quad \text{for } h = 1, 2, \dots$$

- Naïve method assigns all the weight to the last observation.
- Average method assigns equal weight to all observations.

How about something in between these two extremes?

# Simple exponential smoothing

## Forecast equation

$$\hat{y}_{T+1|T} = \alpha y_T + \alpha(1 - \alpha)y_{T-1} + \alpha(1 - \alpha)^2y_{T-2} + \dots$$

where  $0 \leq \alpha \leq 1$ .

Observation	Weights assigned to observations for:			
	$\alpha = 0.2$	$\alpha = 0.4$	$\alpha = 0.6$	$\alpha = 0.8$
$y_T$	0.2	0.4	0.6	0.8
$y_{T-1}$	0.16	0.24	0.24	0.16
$y_{T-2}$	0.128	0.144	0.096	0.032
$y_{T-3}$	0.1024	0.0864	0.0384	0.0064
$y_{T-4}$	$(0.2)(0.8)^4$	$(0.4)(0.6)^4$	$(0.6)(0.4)^4$	$(0.8)(0.2)^4$
$y_{T-5}$	$(0.2)(0.8)^5$	$(0.4)(0.6)^5$	$(0.6)(0.4)^5$	$(0.8)(0.2)^5$

- $\alpha = 1$ : naïve forecasts
- $\alpha = 0$ : mean forecasts

## Weighted average form

Forecast at time  $t + 1$  is a weighted average between  $y_t$  and  $\hat{y}_{t|t-1}$ :

$$\hat{y}_{t+1|t} = \alpha y_t + (1 - \alpha) \hat{y}_{t|t-1}$$

for  $t = 1, 2, \dots, T$  and  $0 \leq \alpha \leq 1$ .

## Weighted average form

Forecast at time  $t + 1$  is a weighted average between  $y_t$  and  $\hat{y}_{t|t-1}$ :

$$\hat{y}_{t+1|t} = \alpha y_t + (1 - \alpha) \hat{y}_{t|t-1}$$

for  $t = 1, 2, \dots, T$  and  $0 \leq \alpha \leq 1$ .

Let  $l_0$  be the forecast for  $y_1$ .

$$\hat{y}_{2|1} = \alpha y_1 + (1 - \alpha) l_0$$

$$\hat{y}_{3|2} = \alpha y_2 + (1 - \alpha) \hat{y}_{2|1} = \alpha y_2 + \alpha(1 - \alpha)y_1 + (1 - \alpha)^2 l_0$$

$$\hat{y}_{4|3} = \alpha y_3 + (1 - \alpha) \hat{y}_{3|2} = \alpha y_3 + \alpha(1 - \alpha)y_2 + \alpha(1 - \alpha)^2 y_1 + (1 - \alpha)^3 l_0$$

⋮

$$\hat{y}_{T+1|T} = \sum_{j=0}^{T-1} \alpha(1 - \alpha)^j y_{T-j} + (1 - \alpha)^T l_0.$$

For large  $T$ , the last term becomes negligible.

## Component form

Forecast equation

$$\hat{y}_{t+1|t} = l_t$$

Smoothing/level equation

$$l_t = \alpha y_t + (1 - \alpha)l_{t-1}$$

where  $l_t$  is the level (or smoothed value) of the series at time  $t$ .

- If we replace  $l_t$  by  $\hat{y}_{t+1|t}$  and  $l_{t-1}$  by  $\hat{y}_{t|t-1}$  in the smoothing equation, we get weighted average form.
- Forecast at time  $t + 1$  is the estimated level at time  $t$ .

## Multi-horizon forecasts

Simple exponential smoothing has a flat forecast function:

$$\hat{y}_{T+h|T} = \hat{y}_{T+1|T} = l_T, \quad h = 1, 2, \dots$$

## Multi-horizon forecasts

Simple exponential smoothing has a flat forecast function:

$$\hat{y}_{T+h|T} = \hat{y}_{T+1|T} = l_T, \quad h = 1, 2, \dots$$

Suitable if the time series has no trend or seasonal component.

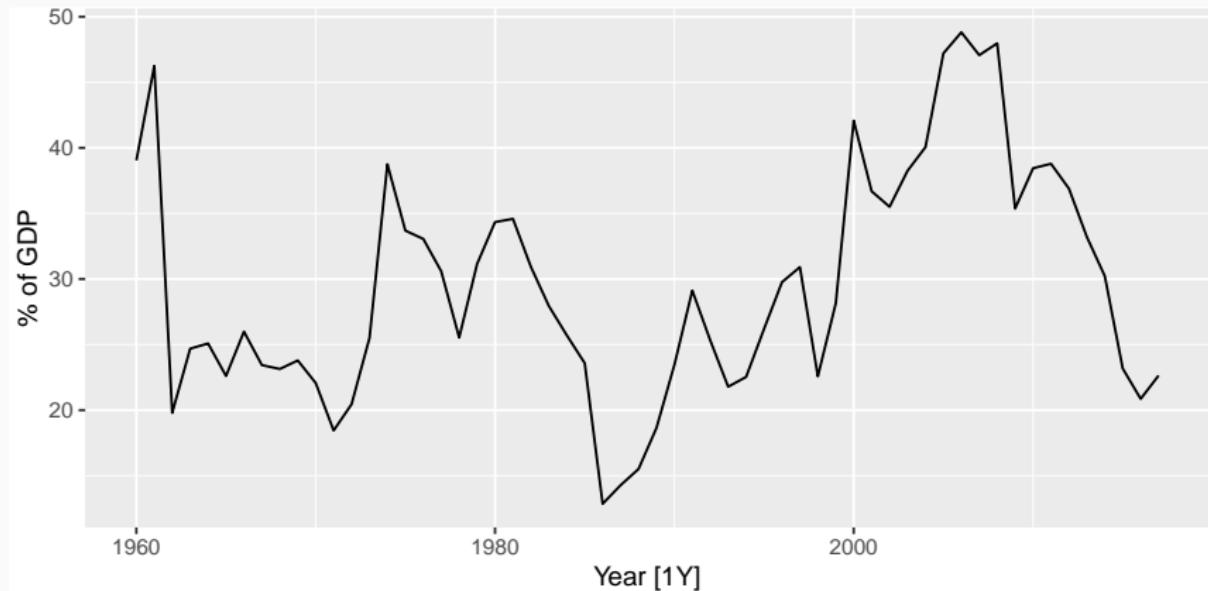
- Need to select best values for  $\alpha$  and  $l_0$ .
- We choose optimal parameters by minimising error sum of squares:

$$\text{ESS} = \sum_{t=1}^T (y_t - \hat{y}_{t|t-1})^2.$$

- We have used a similar idea for estimating regression coefficients.
- This is a non-linear minimization problem, and we need to use an optimization method to solve it.

# Algerian exports

```
algeria_economy <- global_economy %>%  
  filter(Country == "Algeria")  
algeria_economy %>%  
  autoplot(Exports) +  
  ylab("% of GDP")
```



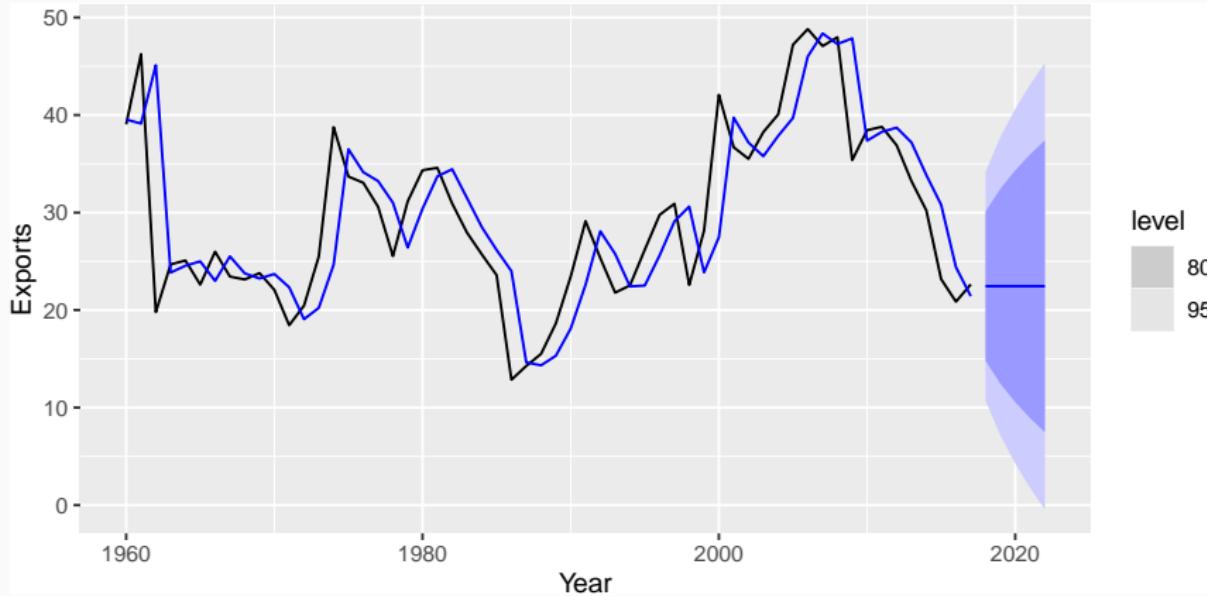
# Algerian exports

```
fit <- algeria_economy %>%
  model(ETS(Exports ~ error("A") + trend("N") + season("N")))
fc <- fit %>%
  forecast(h = 5)
fc %>%
  autoplot(algeria_economy) +
  geom_line(data = augment(fit), aes(y = .fitted), colour = "blue")
```

- $\alpha$  can be specified via `trend()` special.

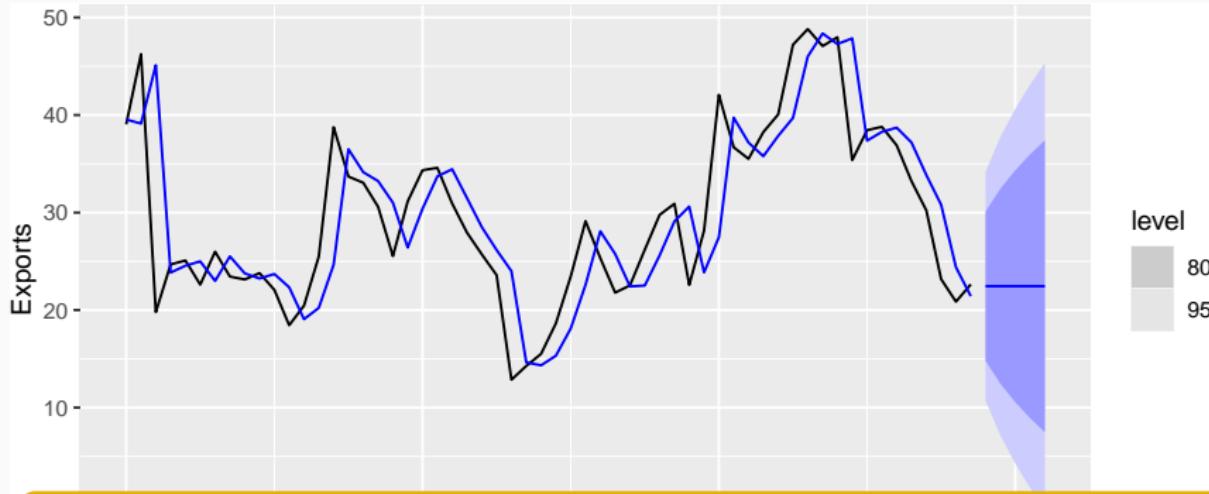
```
trend("N", alpha = 0.5)
trend("N", alpha_range = c(0.2, 0.8))
```

## Algerian exports



$$\hat{\alpha} = 0.84 \text{ and } \hat{l}_0 = 39.54.$$

# Algerian exports

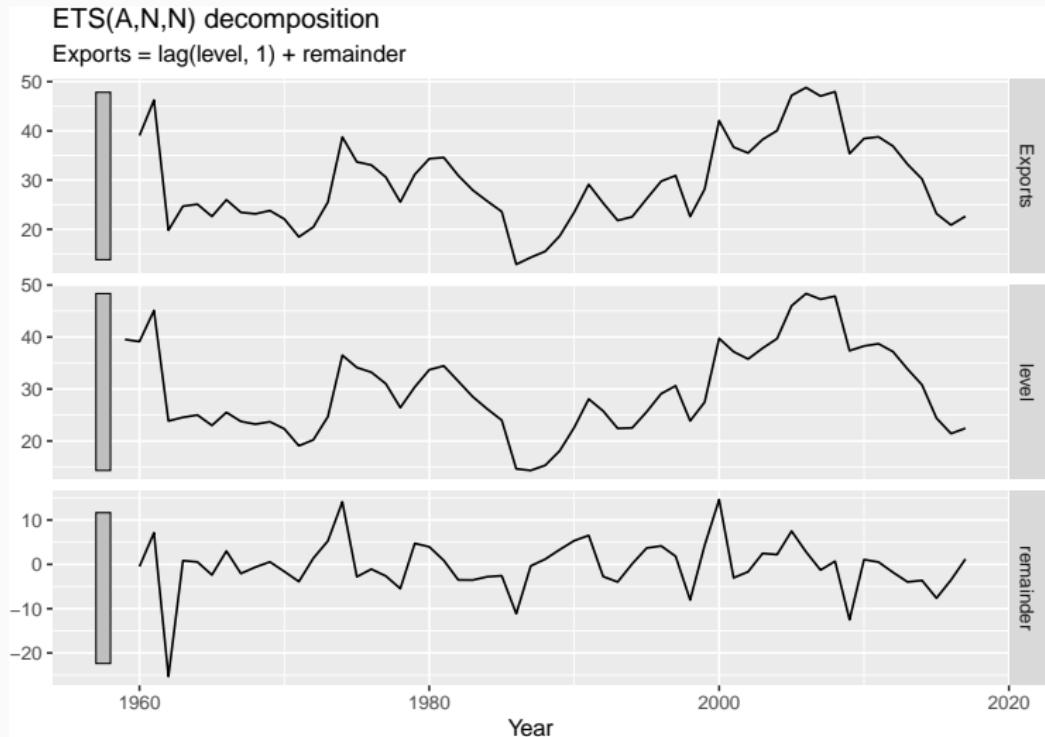


- As  $\alpha$  is large the adjustment that takes place in the next forecast in the direction of the previous data point is large.
- A smaller value of  $\alpha$  leads to less adjustment, so the fitted values are much smoother.

# Algerian exports

```
components(fit) %>%
```

```
autoplot()
```



# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

## Holt's linear trend method

Allows to forecast data with a trend.

### Component form

Forecast equation       $\hat{y}_{t+h|t} = l_t + hb_t$

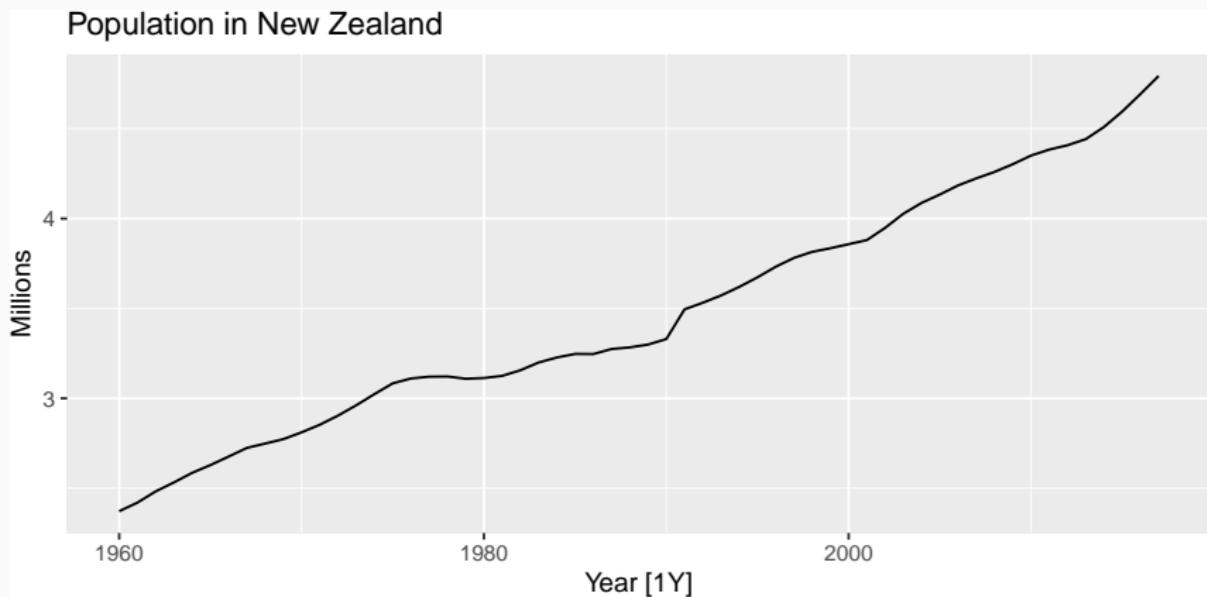
Level equation       $l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + b_{t-1})$

Trend/slope equation       $b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1}$

- Two smoothing parameters  $\alpha$  and  $\beta^*$  such that  $0 \leq \alpha, \beta^* \leq 1$ .
- $l_t$  is a weighted average of  $y_t$  and  $\hat{y}_{t|t-1} (= l_{t-1} + b_{t-1})$ .
- $b_t$  is a weighted average of estimated trend at time  $t$  ( $l_t - l_{t-1}$ ) and previous estimate of the trend ( $b_{t-1}$ ).
- Forecasts are a linear function of  $h$ .
- $\beta^* = 0$ : slope doesn't change with time.
- $\beta^* = 1$ : slope will have no memory of past slopes.

# New Zealand's population

```
nz_economy <- global_economy %>%  
  filter(Country == "New Zealand") %>%  
  mutate(Pop = Population / 1e6)  
  autoplot(nz_economy, Pop) +  
  labs(y = "Millions", title = "Population in New Zealand")
```



# New Zealand's population

```
fit <- nz_economy %>%
  model(AAN = ETS(Pop ~ error("A") + trend("A") + season("N")))
report(fit)
```

```
## Series: Pop
## Model: ETS(A,A,N)
##   Smoothing parameters:
##     alpha = 0.957
##     beta  = 0.508
##
##   Initial states:
##     l[0]    b[0]
##     2.32  0.0463
##
##     sigma^2:  8e-04
##
##   AIC AICc  BIC
## -175 -174 -165
```

# New Zealand's population

```
fit <- nz_economy %>%
  model(AAN = ETS(Pop ~ error("A") + trend("A") + season("N")))
report(fit)
```

```
## Series: Pop
## Model: ETS(A,A,N)
##   Smoothing parameters:
##     alpha = 0.957
##     beta  = 0.508
## 
##   Initial states:
##     l[0]    b[0]
##     2.32  0.0463
## 
##   sigma^2:  8e-04
## 
##   AIC AICc  BIC
## -175 -174 -165
```

$$\hat{\alpha} = 0.96 \quad \hat{l}_0 = 2.32$$
$$\hat{\beta}^* = \hat{\beta}/\hat{\alpha} = 0.5311 \quad \hat{b}_0 = 0.0463$$

# New Zealand's population

```
fit <- nz_economy %>%
  model(AAN = ETS(Pop ~ error("A") + trend("A") + season("N")))
report(fit)
```

```
## Series: Pop
## Model: ETS(A,A,N)
##   Smoothing parameters:
##     alpha = 0.957
##     beta  = 0.508
##
##   Initial states:
```

- # ■  $\hat{\alpha}$  is large: level changes rapidly to capture the trend.
- # ■  $\hat{\beta}^*$  is relatively large: slope also changes often (changes are slight).

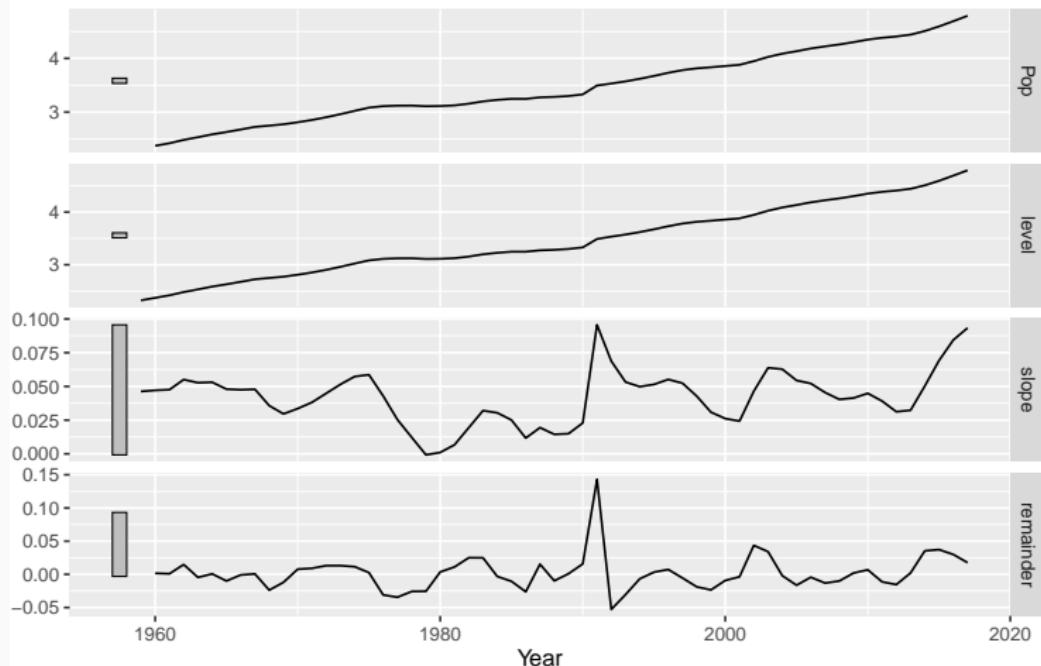
```
##   sigma^2:  8e-04
##
##   AIC AICc  BIC
## -175 -174 -165
```

# New Zealand's population

```
components(fit) %>% autoplot()
```

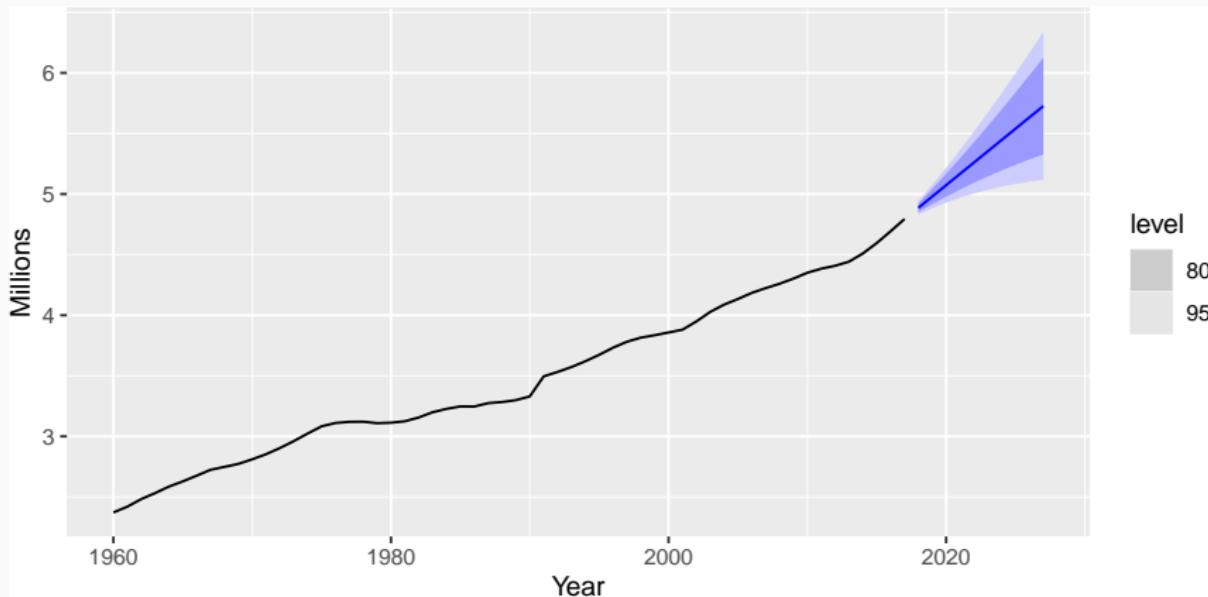
ETS(A,A,N) decomposition

Pop = lag(level, 1) + lag(slope, 1) + remainder



# New Zealand's population

```
forecast(fit, h = 10) %>% autoplot(nz_economy) +  
  ylab("Millions")
```



## Damped trend method

- Forecasts from Holt's linear trend method has a constant trend indefinitely into the future.
- Empirical findings suggested these methods tend to over-forecast, especially for longer horizons.
- An extra parameter introduced that "dampens" the trend to a flat line some time in the future.

### Component form

Forecast equation  $\hat{y}_{t+h|t} = l_t + (\phi + \phi^2 + \cdots + \phi^h)b_t$

Level equation  $l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + \phi b_{t-1})$

Slope equation  $b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)\phi b_{t-1}$

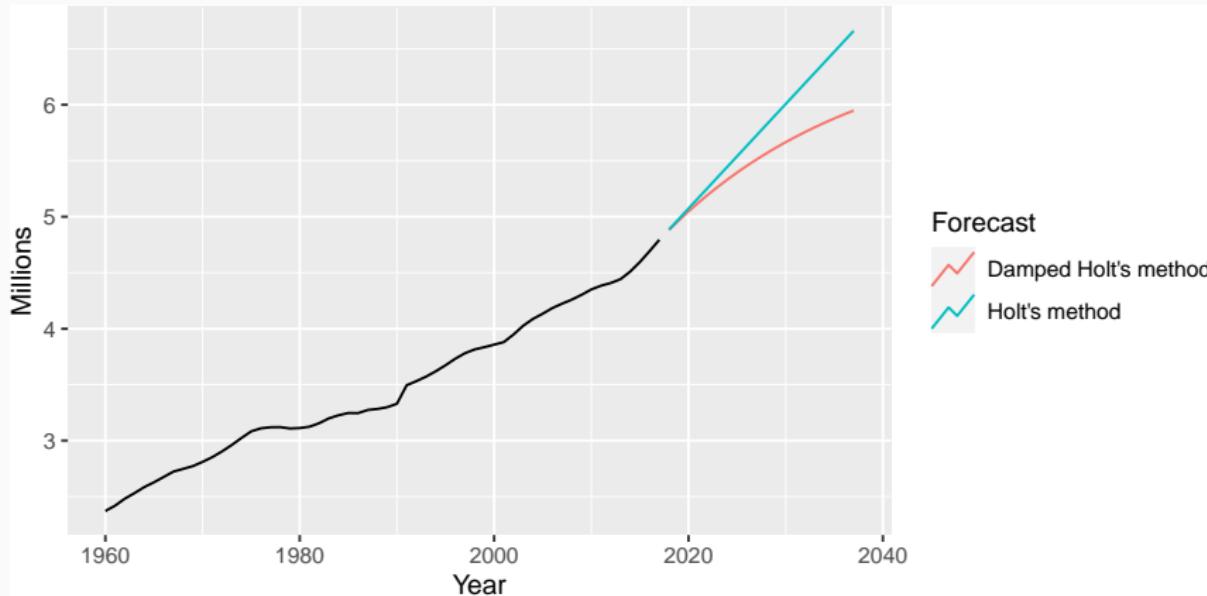
where  $0 < \phi < 1$ .

- $\phi = 1$ : Holt's linear trend method.
- As  $h \rightarrow \infty$ ,  $\hat{y}_{T+h|T} = l_T + \phi b_T / (1 - \phi)$ .
- Short-run forecasts are trended and long-run forecasts are constant.

# New Zealand's population

```
nz_economy %>%
  model(
    `Holt's method` = ETS(Pop ~ error("A") +
                           trend("A") + season("N")),
    `Damped Holt's method` = ETS(Pop ~ error("A") +
                                   trend("Ad") + season("N"))
  ) %>%
  forecast(h = 20) %>%
  autoplot(nz_economy, level = NULL) +
  ylab("Millions") +
  guides(colour = guide_legend(title = "Forecast"))
```

# New Zealand's population



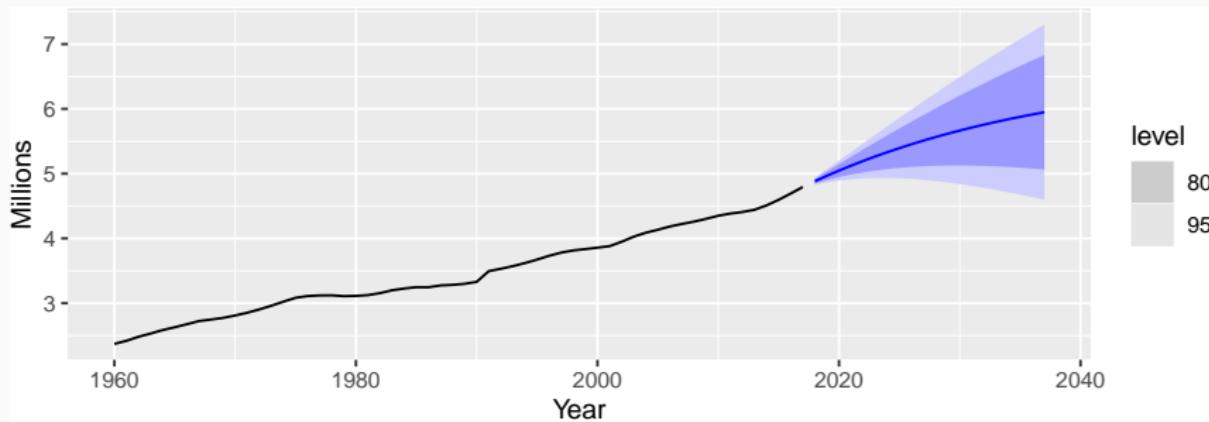
# New Zealand's population

```
nz_economy %>%
  stretch_tsibble(.init = 20) %>%
  model(
    SES = ETS(Pop ~ error("A") + trend("N") + season("N")),
    Holt = ETS(Pop ~ error("A") + trend("A") + season("N")),
    Damped = ETS(Pop ~ error("A") + trend("Ad") + season("N"))
  ) %>%
  forecast(h = 1) %>%
  accuracy(nz_economy) %>%
  select(.model, RMSE, MAE, MAPE, MASE)
```

```
## # A tibble: 3 x 5
##   .model     RMSE     MAE     MAPE     MASE
##   <chr>     <dbl>   <dbl>   <dbl>   <dbl>
## 1 Damped  0.0289  0.0178  0.468  0.414
## 2 Holt    0.0356  0.0214  0.571  0.498
## 3 SES     0.0540  0.0444  1.13   1.03
```

# New Zealand's population

```
nz_economy %>%
  model(ETS(Pop ~ error("A") + trend("Ad") + season("N"))) %>%
  forecast(h = 20) %>%
  autoplot(nz_economy) + ylab("Millions")
```



$$\hat{\alpha} = 1, \hat{\beta}^* = 0.6, \hat{\phi} = 0.95$$

$$\hat{l}_0 = 0.06, \hat{b}_0 = 0.06$$

# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

## Holt-Winters' additive method

- Holt and Winters extended Holt's method to capture seasonality.
- This method is preferred when the seasonal variations are roughly constant through the series.

### Component form

Forecast equation  $\hat{y}_{t+h|t} = l_t + hb_t + s_{t+h-m(k+1)}$

Level equation  $l_t = \alpha(y_t - s_{t-m}) + (1 - \alpha)(l_{t-1} + b_{t-1})$

Slope equation  $b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1}$

Seasonal equation  $s_t = \gamma(y_t - l_{t-1} - b_{t-1}) + (1 - \gamma)s_{t-m}$

where  $k$  is the integer part of  $(h - 1)/m$  (ensures seasonal indices from the final year are used) and  $m$  is the seasonal period.

- Within each year,  $\sum_i s_i \approx 0$ .

## Holt-Winters' additive method

- Level equation for time  $t$  shows a weighted average between seasonally adjusted observation ( $y_t - s_{t-m}$ ) and non-seasonal forecast ( $l_{t-1} + b_{t-1}$ ).
- Slope equation is identical to Holt's linear method.
- Seasonal equation for time  $t$  shows a weighted average between the current seasonal index ( $y_t - l_{t-1} - b_{t-1}$ ) and seasonal index of the same season last year.
- Some textbooks write the seasonal equation as

$$s_t = \gamma^*(y_t - l_t) + (1 - \gamma^*)s_{t-m}.$$

- If we substitute  $l_t$ :

$$s_t = \gamma^*(1 - \alpha)(y_t - l_{t-1} - b_{t-1}) + [1 - \gamma^*(1 - \alpha)]s_{t-m}.$$

- We set  $\gamma = \gamma^*(1 - \alpha)$ .
- Usual parameter restriction is  $0 \leq \gamma^* \leq 1$ , giving  $0 \leq \gamma \leq 1 - \alpha$ .

## Holt-Winters' multiplicative method

- This method is preferred when the seasonal variations are changing proportional to the level of the series.

### Component form

Forecast equation

$$\hat{y}_{t+h|t} = (l_t + hb_t)s_{t+h-m(k+1)}$$

Level equation

$$l_t = \alpha \frac{y_t}{s_{t-m}} + (1 - \alpha)(l_{t-1} + b_{t-1})$$

Slope equation

$$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1}$$

Seasonal equation

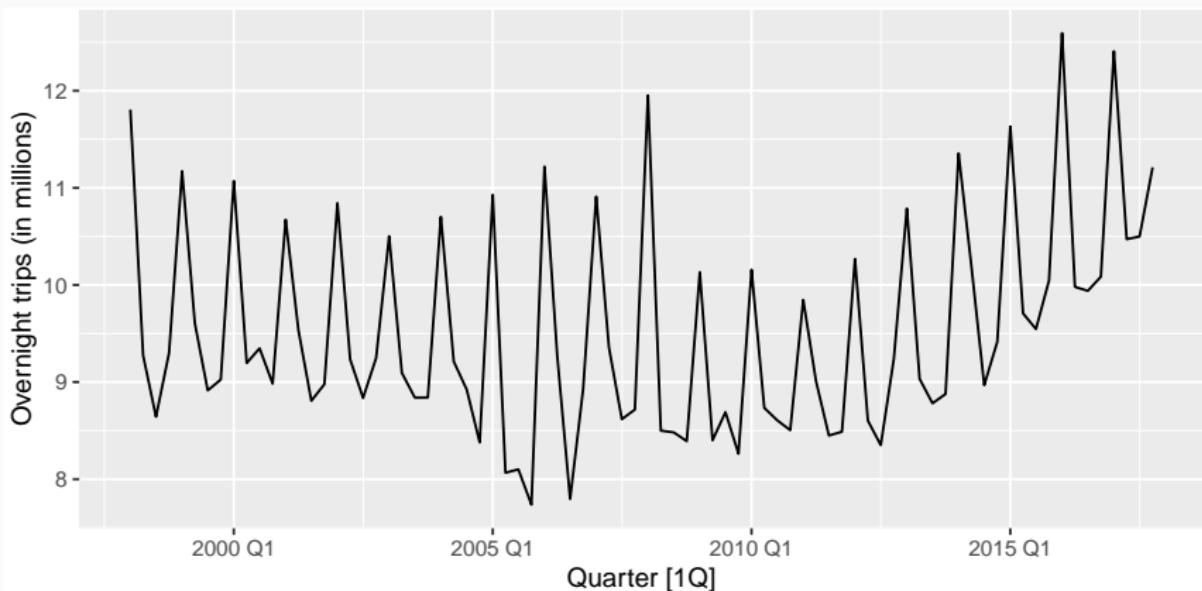
$$s_t = \gamma \frac{y_t}{l_{t-1} + b_{t-1}} + (1 - \gamma)s_{t-m}$$

where  $k$  is the integer part of  $(h - 1)/m$  (ensures seasonal indices from the final year are used) and  $m$  is the seasonal period.

- Within each year,  $\sum_i s_i \approx m$ .

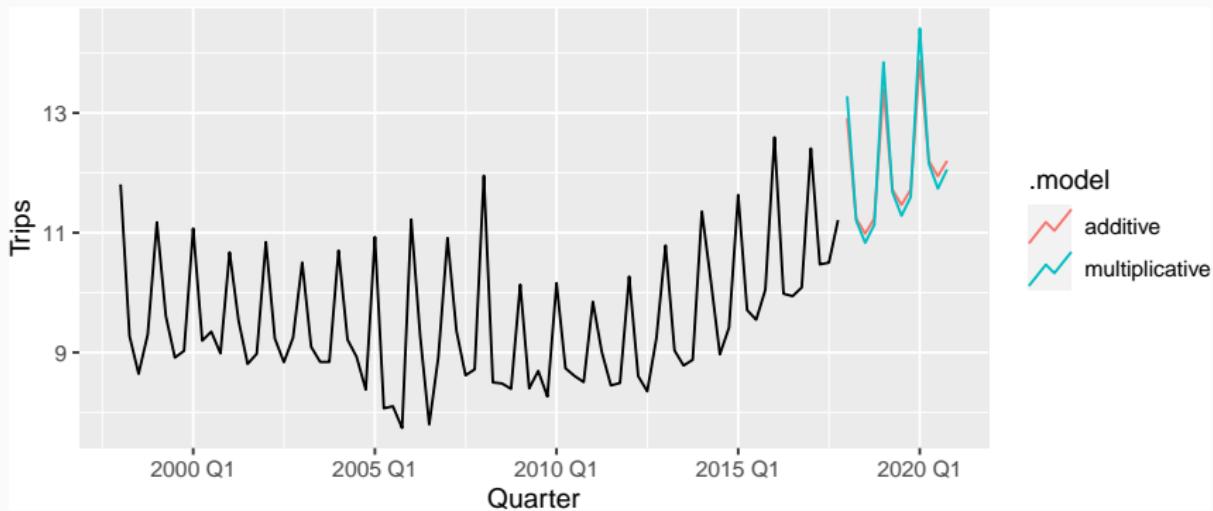
# Domestic overnight trips in Australia

```
aus_holidays <- tourism %>%  
  filter(Purpose == "Holiday") %>%  
  summarise(Trips = sum(Trips)/1e3)  
aus_holidays %>% autoplot(Trips) +  
  ylab("Overnight trips (in millions)")
```

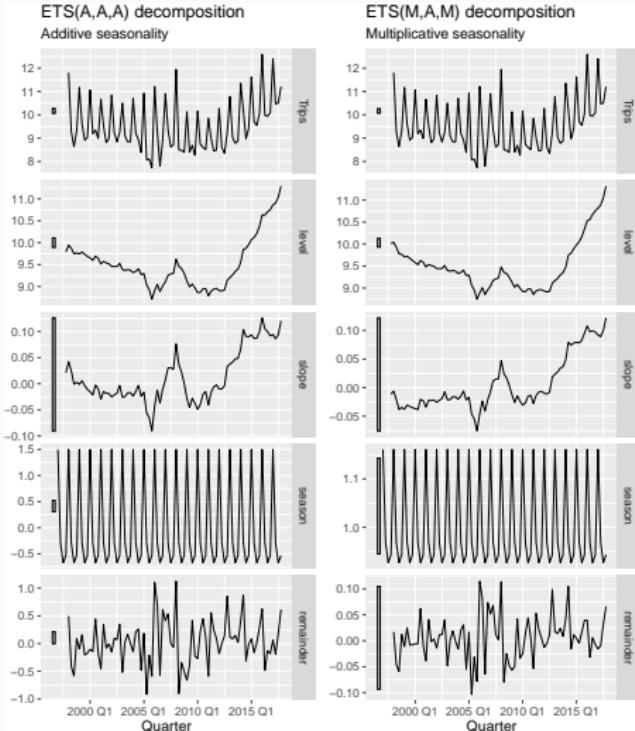


# Domestic overnight trips in Australia

```
fit <- aus_holidays %>%  
  model(additive = ETS(Trips ~ error("A") +  
                        trend("A") + season("A")),  
         multiplicative = ETS(Trips ~ error("M") +  
                               trend("A") + season("M")))  
  
fc <- fit %>% forecast(h = "3 years")  
fc %>% autoplot(aus_holidays, level = NULL)
```

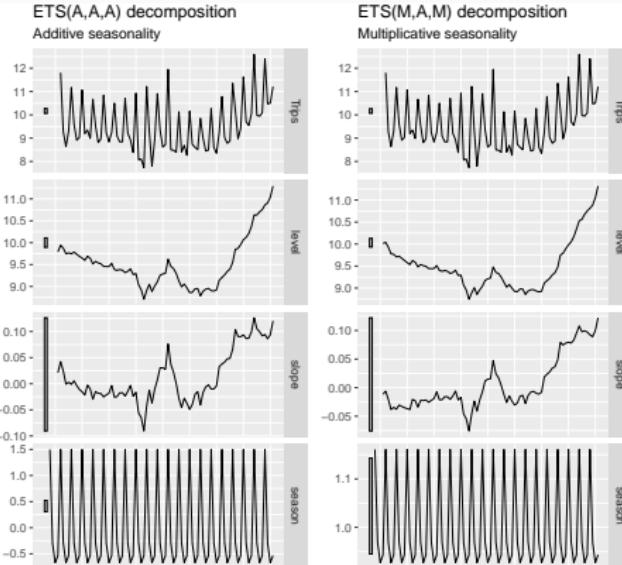


# Domestic overnight trips in Australia



- Additive seasonality:  $\hat{\alpha} = 0.262$ ,  $\hat{\beta}^* = 0.165$ ,  $\hat{\gamma} = 0.0001$ .
- Multiplicative seasonality:  $\hat{\alpha} = 1.498$ ,  $\hat{\beta}^* = 0.149$ ,  $\hat{\gamma} = 0.0304$ .

# Domestic overnight trips in Australia



- $\hat{\gamma}$  is small: seasonal component changes slowly with time.
- $\hat{\beta}^*$  is small: slope changes slowly over time.

- Additive seasonality:  $\hat{\alpha} = 0.262$ ,  $\hat{\beta}^* = 0.165$ ,  $\hat{\gamma} = 0.0001$ .
- Multiplicative seasonality:  $\hat{\alpha} = 1.498$ ,  $\hat{\beta}^* = 0.149$ ,  $\hat{\gamma} = 0.0304$ .

## Holt-Winters' damped method

Often the single most accurate forecasting method for seasonal data.

### Component form

Forecast equation  $\hat{y}_{t+h|t} = [l_t + (\phi + \phi^2 + \cdots + \phi^h)b_t]s_{t+h-m(k+1)}$

Level equation  $l_t = \alpha \frac{y_t}{s_{t-m}} + (1 - \alpha)(l_{t-1} + \phi b_{t-1})$

Slope equation  $b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)\phi b_{t-1}$

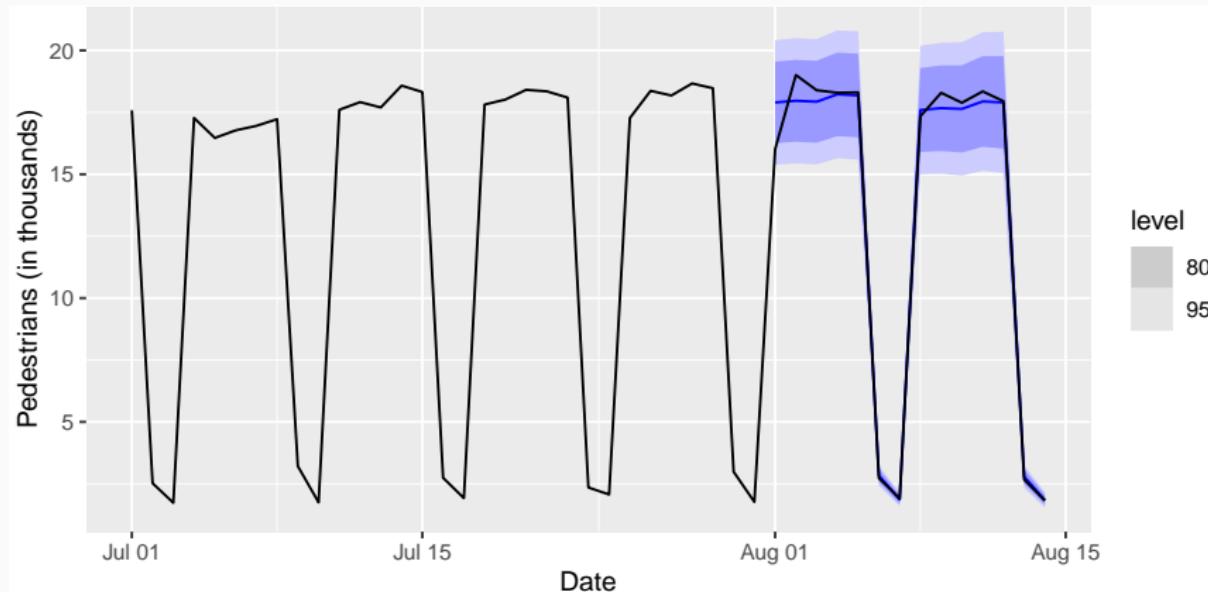
Seasonal equation  $s_t = \gamma \frac{y_t}{l_{t-1} + \phi b_{t-1}} + (1 - \gamma)s_{t-m}$

- Damping is possible with both additive and multiplicative Holt-Winters' methods.

# Pedestrian traffic at Southern Cross Station

```
sth_cross_ped <- pedestrian %>%
  filter(Date >= "2016-07-01",
         Sensor == "Southern Cross Station") %>%
  index_by(Date) %>%
  summarise(Count = sum(Count)/1000)
sth_cross_ped %>%
  filter(Date <= "2016-07-31") %>%
  model(
    hw = ETS(Count ~ error("M") + trend("Ad") + season("M"))
  ) %>%
  forecast(h = "2 weeks") %>%
  autoplot(sth_cross_ped %>% filter(Date <= "2016-08-14")) +
  ylab("Pedestrians (in thousands)")
```

# Pedestrian traffic at Southern Cross Station



# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

## Classification of exponential smoothing methods

		Seasonal Component		
Trend Component		N (None)	A (Additive)	M (Multiplicative)
N	(None)	(N,N)	(N,A)	(N,M)
A	(Additive)	(A,N)	(A,A)	(A,M)
A <sub>d</sub>	(Additive damped)	(A <sub>d</sub> ,N)	(A <sub>d</sub> ,A)	(A <sub>d</sub> ,M)

- We can also consider multiplicative trend methods, but they tend to produce poor forecasts.

## Classification of exponential smoothing methods

		Seasonal Component		
Trend Component		N (None)	A (Additive)	M (Multiplicative)
N	(None)	(N,N)	(N,A)	(N,M)
A	(Additive)	(A,N)	(A,A)	(A,M)
A <sub>d</sub>	(Additive damped)	(A <sub>d</sub> ,N)	(A <sub>d</sub> ,A)	(A <sub>d</sub> ,M)

- We can also consider multiplicative trend methods, but they tend to produce poor forecasts.

- (N,N): Simple exponential smoothing
- (A,N): Holt's linear method
- (A<sub>d</sub>,N): Additive damped trend method
- (A,A): Additive Holt-Winters' method
- (A,M): Multiplicative Holt-Winters' method
- (A<sub>d</sub>,M): Damped multiplicative Holt-Winters' method

# Classification of exponential smoothing methods

Trend		Seasonal		
	N	A	M	
<b>N</b>	$\hat{y}_{t+h t} = l_t$	$\hat{y}_{t+h t} = l_t + s_{t+h-m(k+1)}$	$\hat{y}_{t+h t} = l_t s_{t+h-m(k+1)}$	
	$l_t = \alpha y_t + (1 - \alpha)l_{t-1}$	$l_t = \alpha(y_t - s_{t-m}) + (1 - \alpha)l_{t-1}$	$l_t = \alpha(y_t/s_{t-m}) + (1 - \alpha)l_{t-1}$	
		$s_t = \gamma(y_t - l_{t-1}) + (1 - \gamma)s_{t-m}$	$s_t = \gamma(y_t/l_{t-1}) + (1 - \gamma)s_{t-m}$	
<b>A</b>	$\hat{y}_{t+h t} = l_t + hb_t$	$\hat{y}_{t+h t} = l_t + hb_t + s_{t+h-m(k+1)}$	$\hat{y}_{t+h t} = (l_t + hb_t)s_{t+h-m(k+1)}$	
	$l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + b_{t-1})$	$l_t = \alpha(y_t - s_{t-m}) + (1 - \alpha)(l_{t-1} + b_{t-1})$	$l_t = \alpha(y_t/s_{t-m}) + (1 - \alpha)(l_{t-1} + b_{t-1})$	
	$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1}$	$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1}$	$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1}$	
<b>A<sub>d</sub></b>	$\hat{y}_{t+h t} = l_t + \phi_h b_t$	$\hat{y}_{t+h t} = l_t + \phi_h b_t + s_{t+h-m(k+1)}$	$\hat{y}_{t+h t} = (l_t + \phi_h b_t)s_{t+h-m(k+1)}$	
	$l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + \phi b_{t-1})$	$l_t = \alpha(y_t - s_{t-m}) + (1 - \alpha)(l_{t-1} + \phi b_{t-1})$	$l_t = \alpha(y_t/s_{t-m}) + (1 - \alpha)(l_{t-1} + \phi b_{t-1})$	
	$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)\phi b_{t-1}$	$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)\phi b_{t-1}$	$b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)\phi b_{t-1}$	
$\phi_h = \phi + \phi^2 + \cdots + \phi^h.$				

# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

## State space models for exponential smoothing

- Exponential smoothing methods that we discussed so far are algorithms.
- They can generate point forecasts.
- We define statistical models here to generate both point forecasts and prediction intervals.
- Each model contains:
  - ▶ a measurement equation describing the observed data.
  - ▶ state equations describing how unobserved components (level, slope, seasonal) evolve over time.
- We refer to these as “state space models”.

General notation

E T S : ExponenTial Smoothing  
↑ ↑ ↗  
Error Trend Seasonal

**Error:** Additive ("A") or multiplicative ("M")

**Trend:** None ("N"), additive ("A"), multiplicative ("M"), or damped ("Ad" or "Md").

**Seasonal:** None ("N"), additive ("A") or multiplicative ("M")

- The point forecasts produced by additive and multiplicative errors are identical if the same smoothing parameters are used.
- However, prediction intervals are different.
- In total, we can define 30 models.

## ETS(A,N,N): A model for simple exponential smoothing (SES)

### Component form

Forecast equation  $\hat{y}_{t+1|t} = l_t$

Smoothing/level equation  $l_t = \alpha y_t + (1 - \alpha)l_{t-1}$

The residual at time  $t$ :  $e_t = y_t - \hat{y}_{t|t-1} = y_t - l_{t-1}$ .

### Error correction form

$$y_t = l_{t-1} + e_t$$

$$\begin{aligned} l_t &= l_{t-1} + \alpha(y_t - l_{t-1}) \\ &= l_{t-1} + \alpha e_t. \end{aligned}$$

We assume that  $e_t = \varepsilon_t \sim NID(0, \sigma^2)$ .

Measurement equation

$$y_t = l_{t-1} + \varepsilon_t$$

State equation

$$l_t = l_{t-1} + \alpha \varepsilon_t,$$

where  $\varepsilon_t \sim NID(0, \sigma^2)$ .

- Measurement equation shows the relationship between the observations and the unobserved states ( $y_t$  and  $l_{t-1}$  are linear).
- State equation shows the evolution of the state over time.

## ETS(M,N,N): SES with multiplicative errors

- This model uses relative errors:  $\varepsilon_t = \frac{y_t - \hat{y}_{t|t-1}}{\hat{y}_{t|t-1}} \sim NID(0, \sigma^2)$ .
- Substituting  $\hat{y}_{t|t-1} = l_{t-1}$ , we get

$$y_t = l_{t-1} + l_{t-1}\varepsilon_t \quad \text{and} \quad e_t = y_t - \hat{y}_{t|t-1} = l_{t-1}\varepsilon_t.$$

Measurement equation

$$y_t = l_{t-1}(1 + \varepsilon_t)$$

State equation

$$l_t = l_{t-1}(1 + \alpha\varepsilon_t).$$

## ETS(A,A,N): A model for Holt's linear trend method

### Component form

Forecast equation  $\hat{y}_{t+h|t} = l_t + hb_t$

Level equation  $l_t = \alpha y_t + (1 - \alpha)(l_{t-1} + b_{t-1})$

Trend/slope equation  $b_t = \beta^*(l_t - l_{t-1}) + (1 - \beta^*)b_{t-1}$

We assume that  $\varepsilon_t = y_t - l_{t-1} - b_{t-1}$ .

$$y_t = l_{t-1} + b_{t-1} + \varepsilon_t$$

$$l_t = l_{t-1} + b_{t-1} + \alpha \varepsilon_t$$

$$b_t = b_{t-1} + \beta \varepsilon_t,$$

where we set  $\beta = \alpha\beta^*$ .

## ETS(M,A,N): Holt's linear trend method with multiplicative errors

This model uses relative errors:  $\varepsilon_t = \frac{y_t - l_{t-1} - b_{t-1}}{l_{t-1} + b_{t-1}} \sim NID(0, \sigma^2)$ .

Following a similar approach as other models:

$$y_t = (l_{t-1} + b_{t-1})(1 + \varepsilon_t)$$

$$l_t = (l_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$$

$$b_t = b_{t-1} + \beta(l_{t-1} + b_{t-1})\varepsilon_t,$$

where  $\beta = \alpha\beta^*$ .

## ETS models

### Additive Error

#### Trend Component

N (None)

A (Additive)

$A_d$  (Additive damped)

### Seasonal Component

	N (None)	A (Additive)	M (Multiplicative)
--	-------------	-----------------	-----------------------

A,N,N A,N,A A,N,M

A,A,N A,A,A A,A,M

A, $A_d$ ,N A, $A_d$ ,A A, $A_d$ ,M

### Multiplicative Error

#### Trend Component

N (None)

A (Additive)

$A_d$  (Additive damped)

### Seasonal Component

	N (None)	A (Additive)	M (Multiplicative)
--	-------------	-----------------	-----------------------

M,N,N M,N,A M,N,M

M,A,N M,A,A M,A,M

M, $A_d$ ,N M, $A_d$ ,A M, $A_d$ ,M

# ETS: Additive models

Trend		Seasonal		
	N	A	M	
<b>N</b>	$y_t = l_{t-1} + \varepsilon_t$	$y_t = l_{t-1} + s_{t-m} + \varepsilon_t$	$y_t = l_{t-1}s_{t-m} + \varepsilon_t$	
	$l_t = l_{t-1} + \alpha\varepsilon_t$	$l_t = l_{t-1} + \alpha\varepsilon_t$	$l_t = l_{t-1} + \alpha\varepsilon_t / s_{t-m}$	
		$s_t = s_{t-m} + \gamma\varepsilon_t$	$s_t = s_{t-m} + \gamma\varepsilon_t / l_{t-1}$	
<b>A</b>	$y_t = l_{t-1} + b_{t-1} + \varepsilon_t$	$y_t = l_{t-1} + b_{t-1} + s_{t-m} + \varepsilon_t$	$y_t = (l_{t-1} + b_{t-1})s_{t-m} + \varepsilon_t$	
	$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t$	$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t$	$l_t = l_{t-1} + b_{t-1} + \alpha\varepsilon_t / s_{t-m}$	
	$b_t = b_{t-1} + \beta\varepsilon_t$	$b_t = b_{t-1} + \beta\varepsilon_t$	$b_t = b_{t-1} + \beta\varepsilon_t / s_{t-m}$	
		$s_t = s_{t-m} + \gamma\varepsilon_t$	$s_t = s_{t-m} + \gamma\varepsilon_t / (l_{t-1} + b_{t-1})$	
<b>A<sub>d</sub></b>	$y_t = l_{t-1} + \phi b_{t-1} + \varepsilon_t$	$y_t = l_{t-1} + \phi b_{t-1} + s_{t-m} + \varepsilon_t$	$y_t = (l_{t-1} + \phi b_{t-1})s_{t-m} + \varepsilon_t$	
	$l_t = l_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$	$l_t = l_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t$	$l_t = l_{t-1} + \phi b_{t-1} + \alpha\varepsilon_t / s_{t-m}$	
	$b_t = \phi b_{t-1} + \beta\varepsilon_t$	$b_t = \phi b_{t-1} + \beta\varepsilon_t$	$b_t = \phi b_{t-1} + \beta\varepsilon_t / s_{t-m}$	
		$s_t = s_{t-m} + \gamma\varepsilon_t$	$s_t = s_{t-m} + \gamma\varepsilon_t / (l_{t-1} + \phi b_{t-1})$	

# ETS: Multiplicative models

Trend		Seasonal		
	N	A	M	
<b>N</b>	$y_t = l_{t-1}(1 + \varepsilon_t)$ $l_t = l_{t-1}(1 + \alpha\varepsilon_t)$	$y_t = (l_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $l_t = l_{t-1} + \alpha(l_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(l_{t-1} + s_{t-m})\varepsilon_t$	$y_t = l_{t-1}s_{t-m}(1 + \varepsilon_t)$ $l_t = l_{t-1}(1 + \alpha\varepsilon_t)$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$	
<b>A</b>	$y_t = (l_{t-1} + b_{t-1})(1 + \varepsilon_t)$ $l_t = (l_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(l_{t-1} + b_{t-1})\varepsilon_t$	$y_t = (l_{t-1} + b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $l_t = l_{t-1} + b_{t-1} + \alpha(l_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = b_{t-1} + \beta(l_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(l_{t-1} + b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (l_{t-1} + b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $l_t = (l_{t-1} + b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = b_{t-1} + \beta(l_{t-1} + b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$	
<b>A<sub>d</sub></b>	$y_t = (l_{t-1} + \phi b_{t-1})(1 + \varepsilon_t)$ $l_t = (l_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(l_{t-1} + \phi b_{t-1})\varepsilon_t$	$y_t = (l_{t-1} + \phi b_{t-1} + s_{t-m})(1 + \varepsilon_t)$ $l_t = l_{t-1} + \phi b_{t-1} + \alpha(l_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $b_t = \phi b_{t-1} + \beta(l_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$ $s_t = s_{t-m} + \gamma(l_{t-1} + \phi b_{t-1} + s_{t-m})\varepsilon_t$	$y_t = (l_{t-1} + \phi b_{t-1})s_{t-m}(1 + \varepsilon_t)$ $l_t = (l_{t-1} + \phi b_{t-1})(1 + \alpha\varepsilon_t)$ $b_t = \phi b_{t-1} + \beta(l_{t-1} + \phi b_{t-1})\varepsilon_t$ $s_t = s_{t-m}(1 + \gamma\varepsilon_t)$	

# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

- We need to estimate:
  - ▶ smoothing parameters:  $\alpha, \beta, \gamma, \phi$
  - ▶ initial states:  $l_0, b_0, s_0, s_{-1}, \dots, s_{-m+1}$ .
- We use maximum likelihood estimation (MLE).
- i.e., we find these unknowns such that the probability of observing the data arising from the specified model is maximized.
- For additive errors: MLE = min SSE.
- For multiplicative errors: MLE  $\neq$  min SSE.

## Innovations state space models

Let  $\mathbf{x}_t = (l_t, b_t, s_t, s_{t-1}, \dots, s_{t-m+1})^\top$  and  $\varepsilon_t \sim NID(0, \sigma^2)$ .

$$y_t = \underbrace{w(\mathbf{x}_{t-1})}_{\mu_t = E(y_t | \mathbf{x}_{t-1})} + \underbrace{r(\mathbf{x}_{t-1})\varepsilon_t}_{e_t}$$

$$\mathbf{x}_t = f(\mathbf{x}_{t-1}) + g(\mathbf{x}_{t-1})\varepsilon_t$$

### Additive errors

$$r(\mathbf{x}_{t-1}) = 1 \quad y_t = \mu_t + \varepsilon_t$$

### Multiplicative errors

$$r(\mathbf{x}_{t-1}) = \mu_t \quad y_t = \mu_t(1 + \varepsilon_t) \quad \varepsilon_t = \frac{y_t - \mu_t}{\mu_t} \quad (\text{relative error})$$

- We can obtain multiplicative error models by replacing  $\varepsilon_t$  in additive error models by  $\mu_t\varepsilon_t$ .

## Maximum likelihood estimation

Let  $\theta = (\alpha, \beta, \gamma, \phi)$ . The Gaussian likelihood:

$$L(\theta, \mathbf{x}_0, \sigma^2 | y_1, y_2, \dots, y_T) = p(y_1, y_2, \dots, y_T | \theta, \mathbf{x}_0, \sigma^2)$$

$$= \prod_{t=1}^T p(y_t | y_{t-1}, \dots, y_1, \mathbf{x}_0, \theta, \sigma^2)$$

$$= \prod_{t=1}^T p(y_t | \mathbf{x}_{t-1}, \theta, \sigma^2)$$

$$= \prod_{t=1}^T p(\varepsilon_t) / |r(\mathbf{x}_{t-1})|$$

$$= (2\pi\sigma^2)^{-T/2} \left| \prod_{t=1}^T r(\mathbf{x}_{t-1}) \right|^{-1} \exp \left( -\frac{1}{2\sigma^2} \sum_{t=1}^T \varepsilon_t^2 \right)$$

and the log-likelihood is

$$\log L = -\frac{T}{2} \log(2\pi\sigma^2) - \sum_{t=1}^T \log|r(\mathbf{x}_{t-1})| - \frac{1}{2\sigma^2} \sum_{t=1}^T \varepsilon_t^2.$$

## Maximum likelihood estimation

Substitute  $\hat{\sigma}^2$  in the likelihood function to get concentrated likelihood:

$$L(\theta, \mathbf{x}_0 | y_1, y_2, \dots, y_T) = (2\pi e \hat{\sigma}^2)^{-T/2} \left| \prod_{t=1}^T r(\mathbf{x}_{t-1}) \right|^{-1}.$$

Twice the negative log-likelihood is:

$$\begin{aligned} -2 \log L(\theta, \mathbf{x}_0 | y_1, y_2, \dots, y_T) &= T \log(2\pi e \hat{\sigma}^2) + 2 \sum_{t=1}^T \log |r(\mathbf{x}_{t-1})| \\ &= c_T + T \log \left( \sum_{t=1}^T \varepsilon_t^2 \right) + 2 \sum_{t=1}^T \log |r(\mathbf{x}_{t-1})|, \end{aligned}$$

where  $c_T$  is a constant which depends on  $T$  but not on  $\theta$  and  $\mathbf{x}_0$ .

Therefore, maximum likelihood estimates for  $\theta$  and  $\sigma^2$  are obtained by minimizing:

$$L^*(\theta, \mathbf{x}_0) = T \log \left( \sum_{t=1}^T \varepsilon_t^2 \right) + 2 \sum_{t=1}^T \log |r(\mathbf{x}_{t-1})|.$$

### Traditional (usual) region

- We take  $0 < \alpha, \beta^*, \gamma^*, \phi < 1$  so that equations can be interpreted as weighted averages.
- For some models we have set  $\beta = \alpha\beta^*$  and  $\gamma = (1 - \alpha)\gamma^*$ .
- Hence  $0 < \alpha < 1$ ,  $0 < \beta < \alpha$  and  $0 < \gamma < 1 - \alpha$ .
- In practice, we set  $0.8 < \phi < 0.98$  to prevent numerical difficulties.

### Admissible region

- These are defined considering the mathematical properties of the state space models.
- To prevent observations in the distant past having a continuing effect on current forecasts.
- Usually (but not always) less restrictive than the traditional region.
- For example: ETS(A,N,N):  $0 < \alpha < 2$  and ETS(A,A,N):  $0 < \alpha < 2$  and  $0 < \beta < 4 - 2\alpha$ .

## Model selection

### Akaike's Information Criterion

$$AIC = -2\log L + 2k$$

where  $L$  is the likelihood of the model and  $k$  is the no. of parameters and initial states estimated (including the residual variance)

### Corrected AIC

$$AICc = AIC + \frac{2k(k+1)}{T-k-1}$$

### Bayesian Information Criterion

$$BIC = AIC + k[\log(T) - 2]$$

- For Gaussian residuals, minimizing AIC is asymptotically equivalent to minimizing one-step time series cross-validation MSE.

## Automatic forecasting algorithm

From Hyndman et al. (2002):

- 1 Apply each model that is appropriate to the data. The parameters and initial states are estimated using MLE (or some other criterion).
  - 2 The best model is selected using AICc.
- This algorithm performs well in M3-competition.

# Domestic overnight trips in Australia

```
fit <- aus_holidays %>%
  model(ETS(Trips))
report(fit)

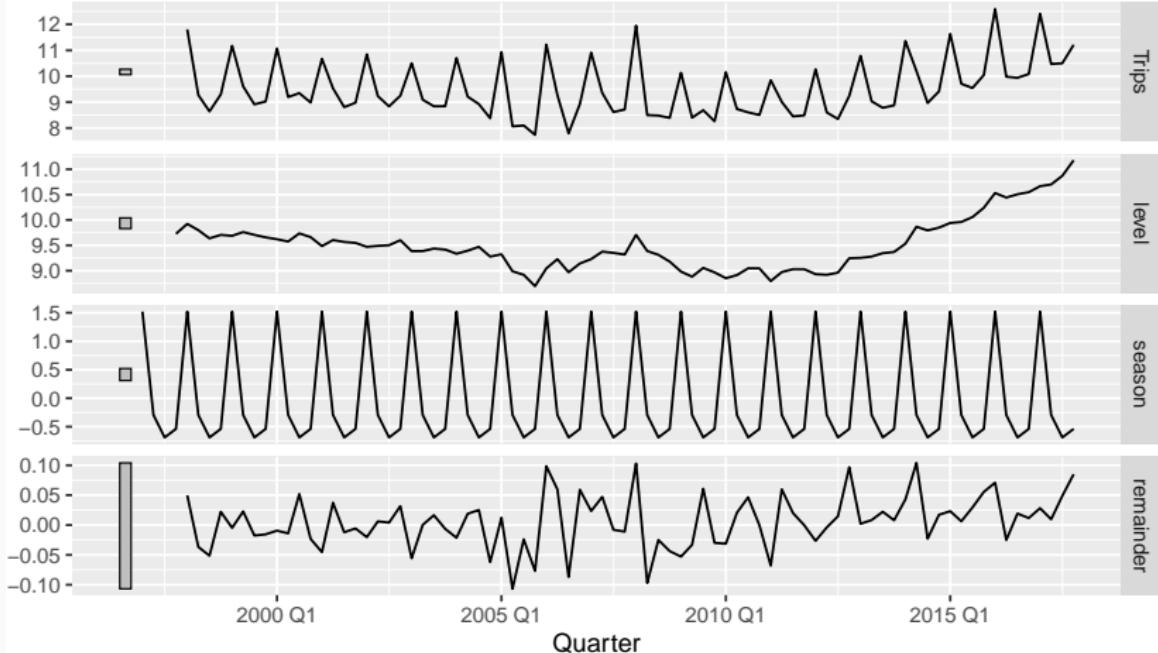
## Series: Trips
## Model: ETS(M,N,A)
##   Smoothing parameters:
##     alpha = 0.348
##     gamma = 1e-04
##
##   Initial states:
##     l[0]    s[0]    s[-1]   s[-2]  s[-3]
##     9.73 -0.538 -0.688 -0.293  1.52
##
##   sigma^2:  0.0022
##
##   AIC AICc  BIC
##   226  228  243
```

# Domestic overnight trips in Australia

```
fit %>%  
  components() %>% autoplot()
```

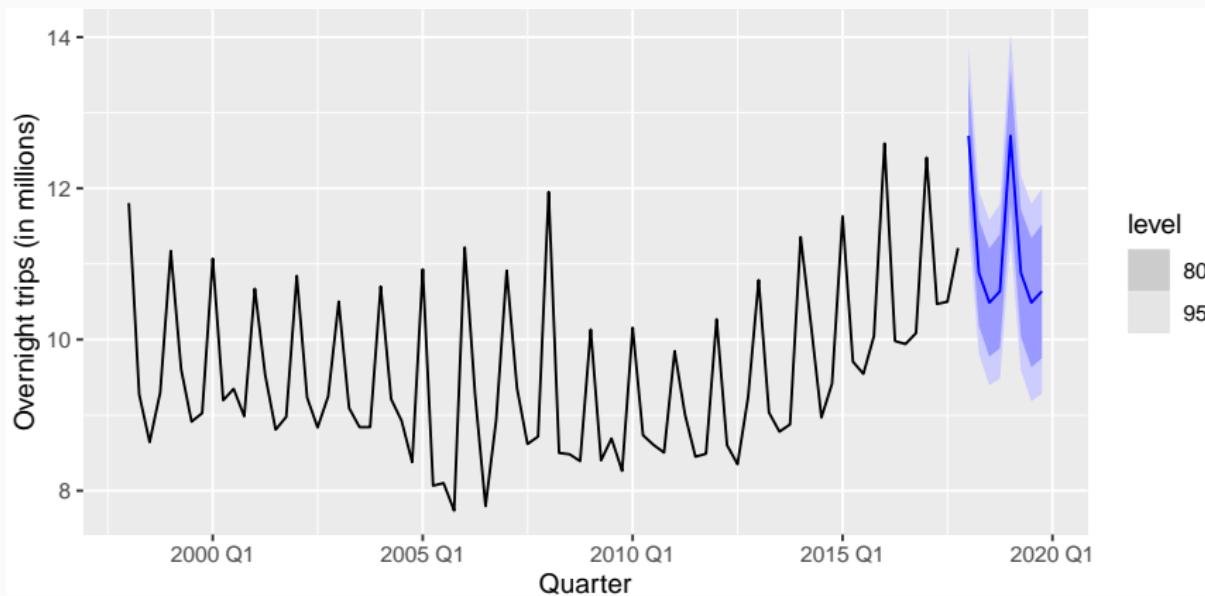
## ETS(M,N,A) decomposition

$$\text{Trips} = (\text{lag(level, 1)} + \text{lag(season, 4)}) * (1 + \text{remainder})$$



# Domestic overnight trips in Australia

```
fit %>%
  forecast() %>%
  autoplot(aus_holidays) +
  ylab("Overnight trips (in millions)")
```



# Types of residuals

## Response residuals

$$\hat{e}_t = y_t - \hat{y}_{t|t-1}$$

## Innovation residuals

### Additive error models

$$\hat{\varepsilon}_t = y_t - \hat{y}_{t|t-1}$$

### Multiplicative error models

$$\hat{\varepsilon}_t = \frac{y_t - \hat{y}_{t|t-1}}{\hat{y}_{t|t-1}}$$

for  $t = 1, 2, \dots, T$ .

- We can obtain these using the `augment()` function.

## ETS models with numerical difficulties

Additive Error		Seasonal Component		
	Trend Component	N (None)	A (Additive)	M (Multiplicative)
N	(None)	A,N,N	A,N,A	A,N,M
A	(Additive)	A,A,N	A,A,A	A,A,M
$A_d$	(Additive damped)	A, $A_d$ ,N	A, $A_d$ ,A	A, $A_d$ ,M

Due to division by values potentially close to zero in the state equations.

Multiplicative Error		Seasonal Component		
	Trend Component	N (None)	A (Additive)	M (Multiplicative)
N	(None)	M,N,N	M,N,A	M,N,M
A	(Additive)	M,A,N	M,A,A	M,A,M
$A_d$	(Additive damped)	M, $A_d$ ,N	M, $A_d$ ,A	M, $A_d$ ,M

# ETS models with numerical difficulties

Additive Error	Seasonal Component		
Trend Component	N (None)	A (Additive)	M (Multiplicative)

- Useful when the data are strictly positive.
- Can be numerically not stable when the data contains zeros or negative values.
- Six fully additive models will be considered.

Multiplicative Error	Seasonal Component		
Trend Component	N (None)	A (Additive)	M (Multiplicative)
N (None)	M,N,N	M,N,A	M,N,M
A (Additive)	M,A,N	M,A,A	M,A,M
A <sub>d</sub> (Additive damped)	M,A <sub>d</sub> ,N	M,A <sub>d</sub> ,A	M,A <sub>d</sub> ,M

# Outline

- 1 Exponential smoothing
- 2 Methods with trend
- 3 Methods with seasonality
- 4 Taxonomy of exponential smoothing methods
- 5 Innovation state space models
- 6 Estimation and model selection
- 7 Forecasting with ETS models

**Traditional point forecasts** are obtained from the models by iterating the equations for  $t = T + 1, T + 2, \dots, T + h$  and setting all  $\varepsilon_t = 0$  for  $t > T$ .

- These are not the same as  $E(y_{t+h|t})$  unless trend and seasonal components are both additive.
- fable package implements  $E(y_{t+h|t})$ .
- ETS(A, \*, \*) and ETS(M, \*, \*) produce the same point forecasts if both use the same parameter values.

## Examples

### ETS(A,A,N)

$$y_{T+1} = l_T + b_T + \varepsilon_{T+1}$$

$$\hat{y}_{T+1|T} = l_T + b_T.$$

$$y_{T+2} = l_{T+1} + b_{T+1} + \varepsilon_{T+2}$$

$$= (l_T + b_T + \alpha \varepsilon_{T+1}) + (b_T + \beta \varepsilon_{T+1}) + \varepsilon_{T+2}$$

$$\hat{y}_{T+2|T} = l_T + 2b_T. \text{ etc.}$$

### ETS(M,A,N)

$$y_{T+1} = (l_T + b_T)(1 + \varepsilon_{T+1})$$

$$\hat{y}_{T+1|T} = l_T + b_T.$$

$$y_{T+2} = (l_{T+1} + b_{T+1})(1 + \varepsilon_{T+2})$$

$$= [(l_T + b_T)(1 + \alpha \varepsilon_{T+1}) + b_T + \beta(l_T + b_T)\varepsilon_{T+1}](1 + \varepsilon_{T+2})$$

$$\hat{y}_{T+2|T} = l_T + 2b_T. \text{ etc.}$$

## Prediction intervals (PI)

- An advantage of ETS models is that prediction intervals can be constructed.
- Prediction intervals will differ between additive and multiplicative error models.
- For ETS(A,N/A/A<sub>d</sub>,N/A) models prediction distributions are Gaussian.
- For ETS(M,N/A/A<sub>d</sub>,N/A/M) models prediction distributions are non-Gaussian because of nonlinearity of the state space equations.
  - ▶ Gaussian approximation usually give reasonably accurate results.
  - ▶ If this approximation is not reasonable, we can generate future sample paths conditional on the last estimate of the states, and then to obtain prediction intervals from the percentiles of these simulated future paths.

## Prediction intervals

100(1 -  $\alpha$ )% PI (for most ETS models):  $\hat{y}_{T+h|T} \pm z_{\alpha/2} \hat{\sigma}_h$ , where  $z_q$  depends the q-th quantile of a standard Gaussian distribution and  $\hat{\sigma}_h$  is an estimate of the forecast standard deviation.

---

$$(A, N, N) \quad \sigma_h^2 = \sigma^2 \left[ 1 + \alpha^2(h-1) \right]$$

$$(A, A, N) \quad \sigma_h^2 = \sigma^2 \left[ 1 + (h-1) \left\{ \alpha^2 + \alpha\beta h + \frac{1}{6}\beta^2 h(2h-1) \right\} \right]$$

$$(A, A_d, N) \quad \sigma_h^2 = \sigma^2 \left[ 1 + \alpha^2(h-1) + \frac{\beta\phi h}{(1-\phi)^2} \left\{ 2\alpha(1-\phi) + \beta\phi \right\} \right. \\ \left. - \frac{\beta\phi(1-\phi^h)}{(1-\phi)^2(1-\phi^2)} \left\{ 2\alpha(1-\phi^2) + \beta\phi(1+2\phi-\phi^h) \right\} \right]$$

$$(A, N, A) \quad \sigma_h^2 = \sigma^2 \left[ 1 + \alpha^2(h-1) + \gamma k(2\alpha + \gamma) \right]$$

$$(A, A, A) \quad \sigma_h^2 = \sigma^2 \left[ 1 + (h-1) \left\{ \alpha^2 + \alpha\beta h + \frac{1}{6}\beta^2 h(2h-1) \right\} + \gamma k \left\{ 2\alpha + \gamma + \beta m(k+1) \right\} \right]$$

$$(A, A_d, A) \quad \sigma_h^2 = \sigma^2 \left[ 1 + \alpha^2(h-1) + \frac{\beta\phi h}{(1-\phi)^2} \left\{ 2\alpha(1-\phi) + \beta\phi \right\} \right. \\ \left. - \frac{\beta\phi(1-\phi^h)}{(1-\phi)^2(1-\phi^2)} \left\{ 2\alpha(1-\phi^2) + \beta\phi(1+2\phi-\phi^h) \right\} \right. \\ \left. + \gamma k(2\alpha + \gamma) + \frac{2\beta\gamma\phi}{(1-\phi)(1-\phi^m)} \left\{ k(1-\phi^m) - \phi^m(1-\phi^{mk}) \right\} \right]$$

---

$m$ : seasonal period,  $k$ : integer part of  $(h-1)/m$ ,  $\hat{\sigma}^2 = \sum_{t=1}^T \hat{\varepsilon}_t^2 / (T-K)$  and  $K$ : no. of parameters estimated in the model.

## References

Hyndman, R. J., Koehler, A. B., Snyder, R. D. and Grose, S. (2002). A state space framework for automatic forecasting using exponential smoothing methods. *International Journal of Forecasting* 18, 439–454.



THE UNIVERSITY OF  
**AUCKLAND**  
Te Whare Wānanga o Tāmaki Makaurau  
NEW ZEALAND

# ARIMA models

# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

**ARIMA:** AutoRegressive Integrated Moving Average.

**AR:** autoregressive (lagged observations of the response)

**I:** integrated (differencing to ensure stationarity)

**MA:** moving average (lagged errors)

- ARIMA models describe the autocorrelations present in the data.
- They are less interpretable in terms of the time series patterns (such as trend and seasonality) present in the data.
- They can account for a range of time series patterns.

# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

## Stationarity

### Definition: Strictly stationary

A time series  $\{y_t\}$  is a strictly stationary time series if the joint distribution of

$$\{y_1, y_2, \dots, y_t\}$$

is the same as that of

$$\{y_{1+s}, y_{2+s}, \dots, y_{t+s}\}$$

for all integers  $s$  and  $t \geq 1$ .

### Definition: (Weakly) stationary

A time series  $\{y_t\}$  is said to be weakly stationary if

- The mean is constant.
- $\text{Cov}(y_t, y_{t+s})$  is independent of time but depends only on the lag  $s$ .
  - ▶ The variance is constant.

A strictly stationary time series is weakly stationary but not vice-versa.

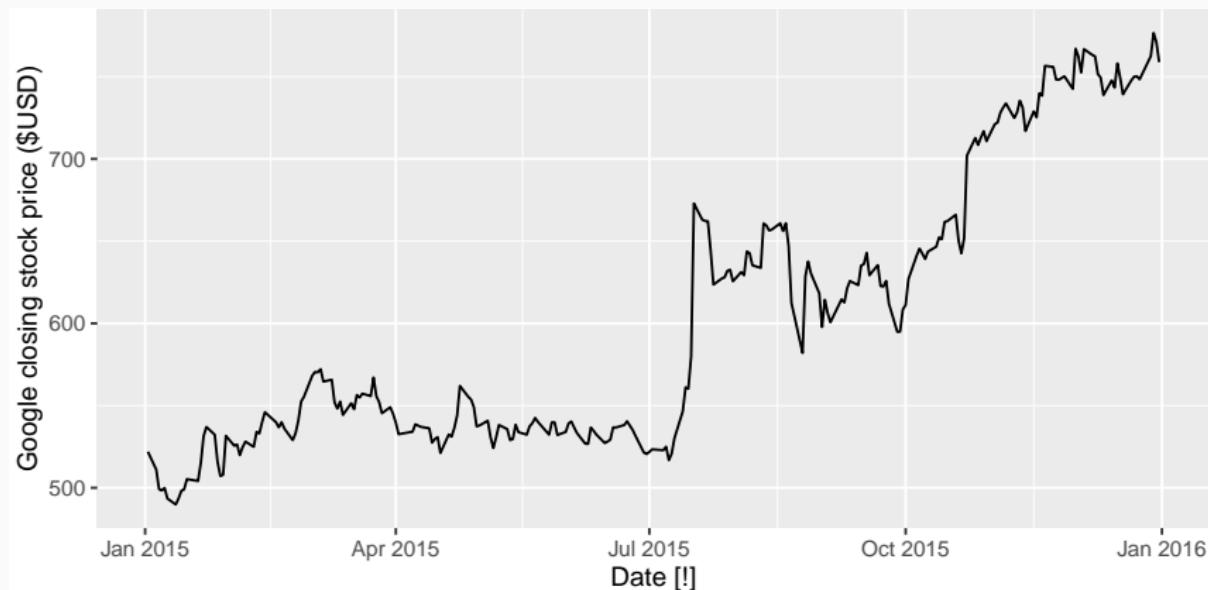
## Stationarity

In layman's terms a stationary time series is one whose properties do not depend on the time at which the series is observed.

- Time series with trends or with seasonality are not stationary.
- A time series with cyclic behaviour (with no trend or seasonality) is stationary.
  - ▶ The cycles do not have a fixed length, so we cannot predict the peaks and troughs of the cycles in advance.
- A white noise series is weakly stationary.

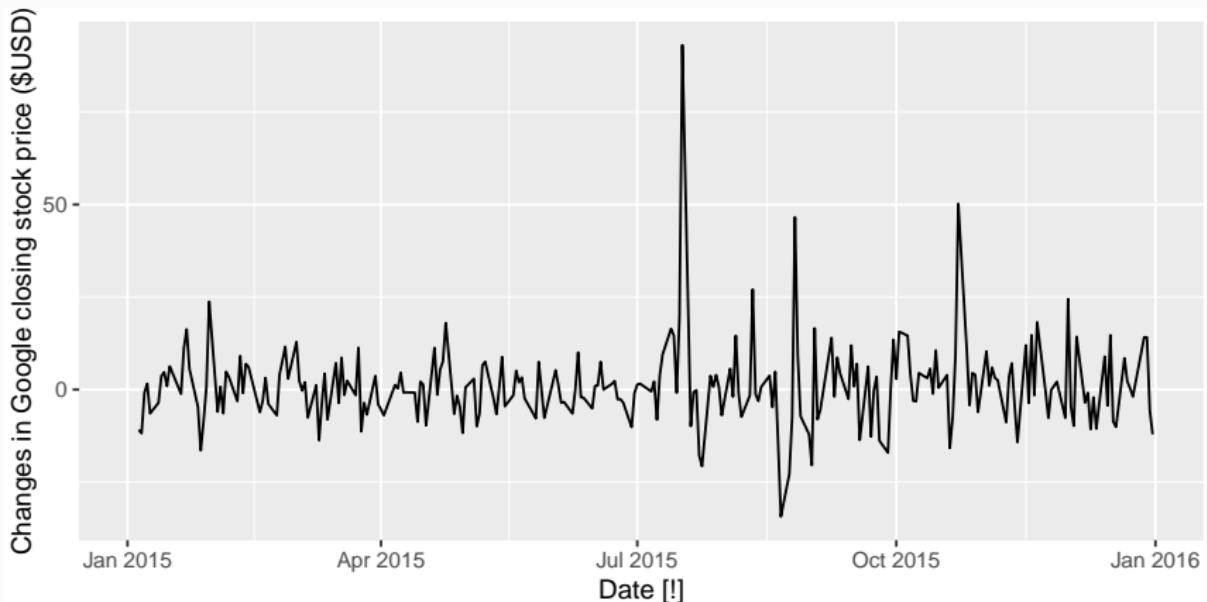
# Google closing stock price

```
gafa_stock %>%
  filter(Symbol == "GOOG", year(Date) == 2015) %>%
  autoplot(Close) +
  ylab("Google closing stock price ($USD)")
```



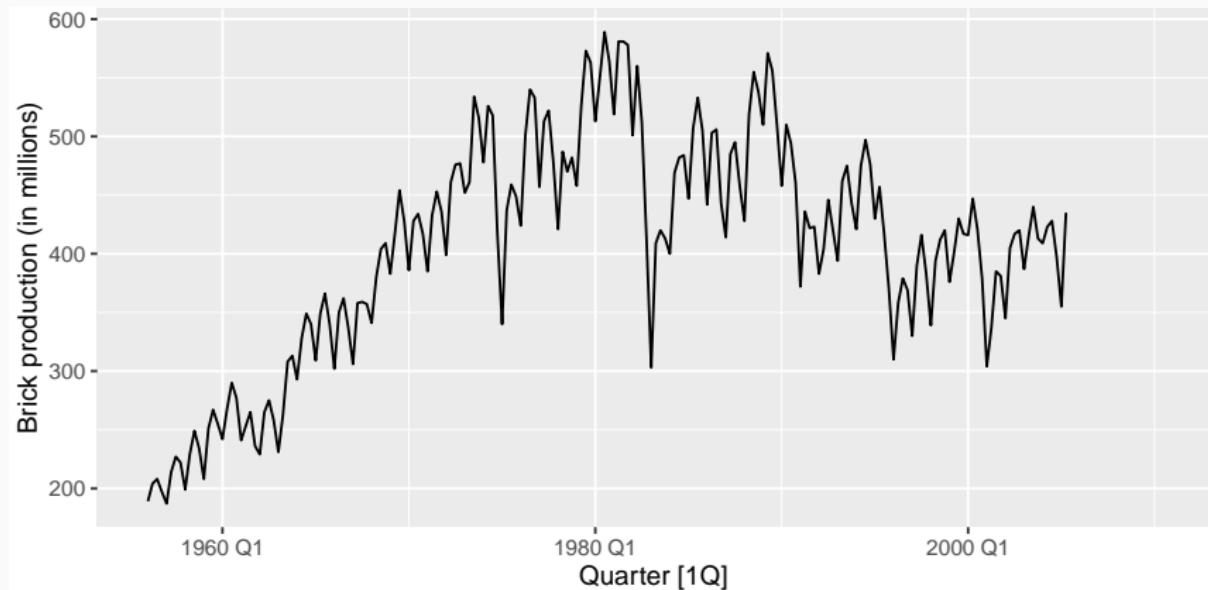
# Changes in Google closing stock price

```
gafa_stock %>%
  filter(Symbol == "GOOG", year(Date) == 2015) %>%
  autoplot(difference(Close)) +
  ylab("Changes in Google closing stock price ($USD)")
```



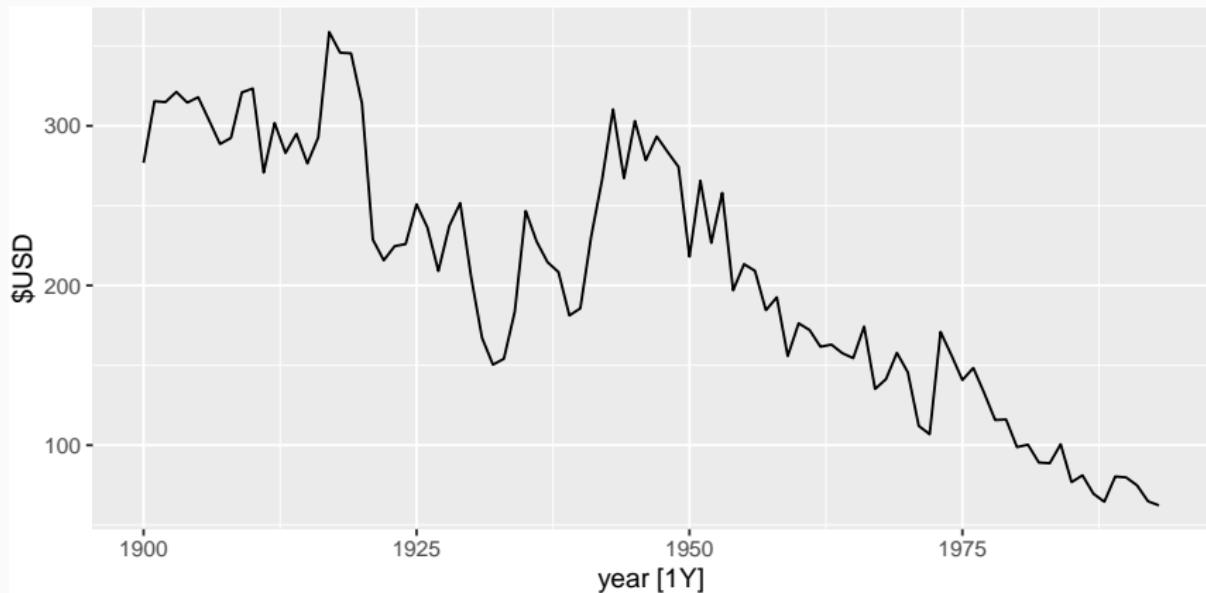
# Australian brick production

```
aus_production %>%
  autoplot(Bricks) +
  ylab("Brick production (in millions)")
```



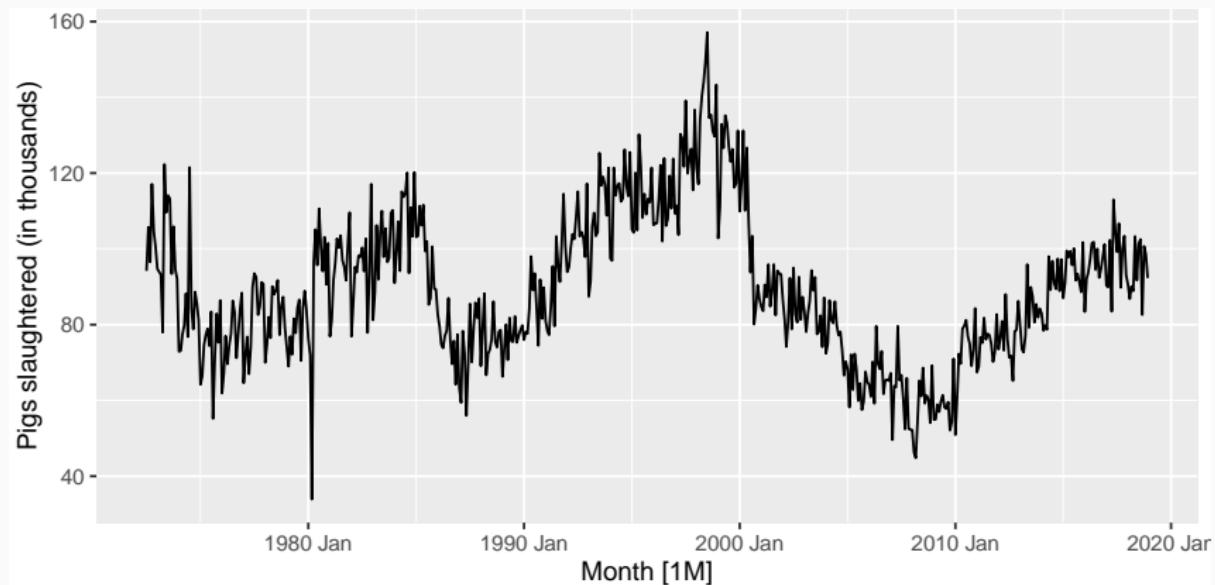
# Egg prices in US

```
prices %>%
  filter(!is.na(eggs)) %>%
  autoplot(eggs) +
  ylab("$USD")
```



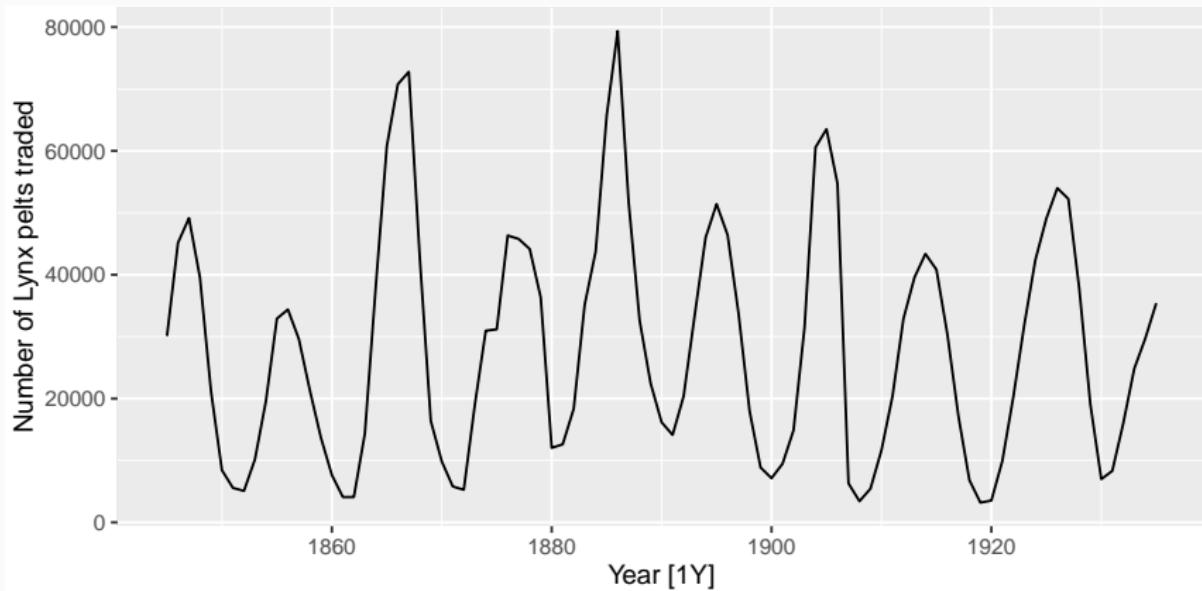
# Pigs slaughtered in Victoria

```
aus_livestock %>%
  filter(State == "Victoria", Animal == "Pigs") %>%
  autoplot(Count/1e3) +
  ylab("Pigs slaughtered (in thousands)")
```



## Canadian Lynx pelts traded

```
pelt %>%
  autoplot(Lynx) +
  ylab("Number of Lynx pelts traded")
```

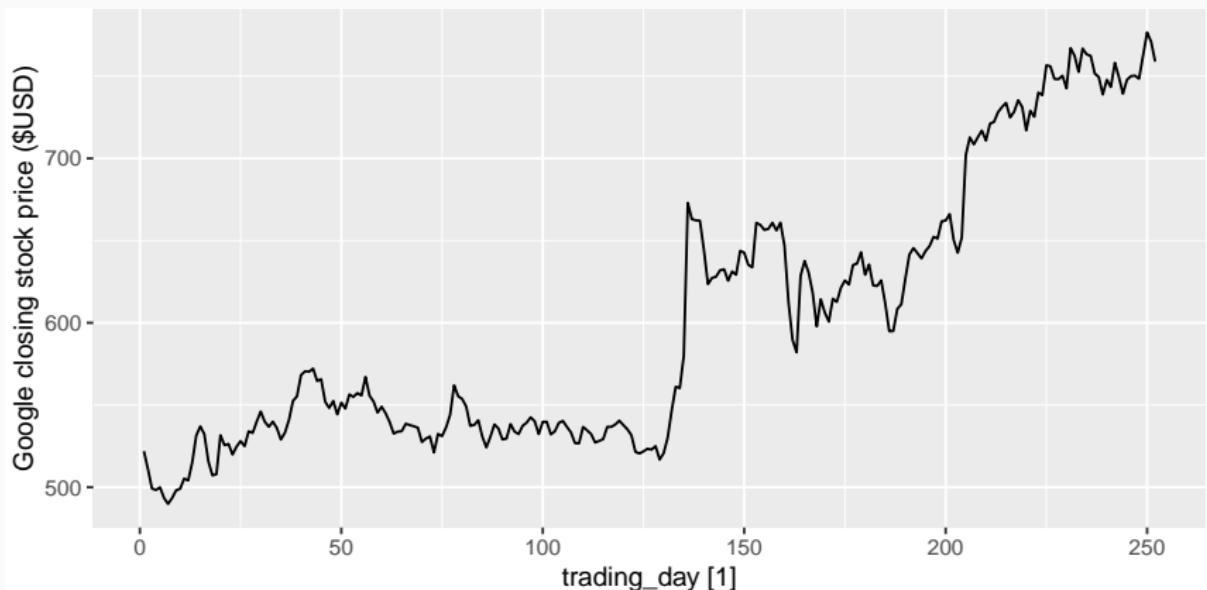


## Identifying non-stationary time series

- Time plot.
- ACF plot.
- For **stationary** time series, the ACF will drop to zero relatively quickly.
- For **non-stationary** time series, the ACF will decrease slowly. The value of  $r_1$  is often large and positive.

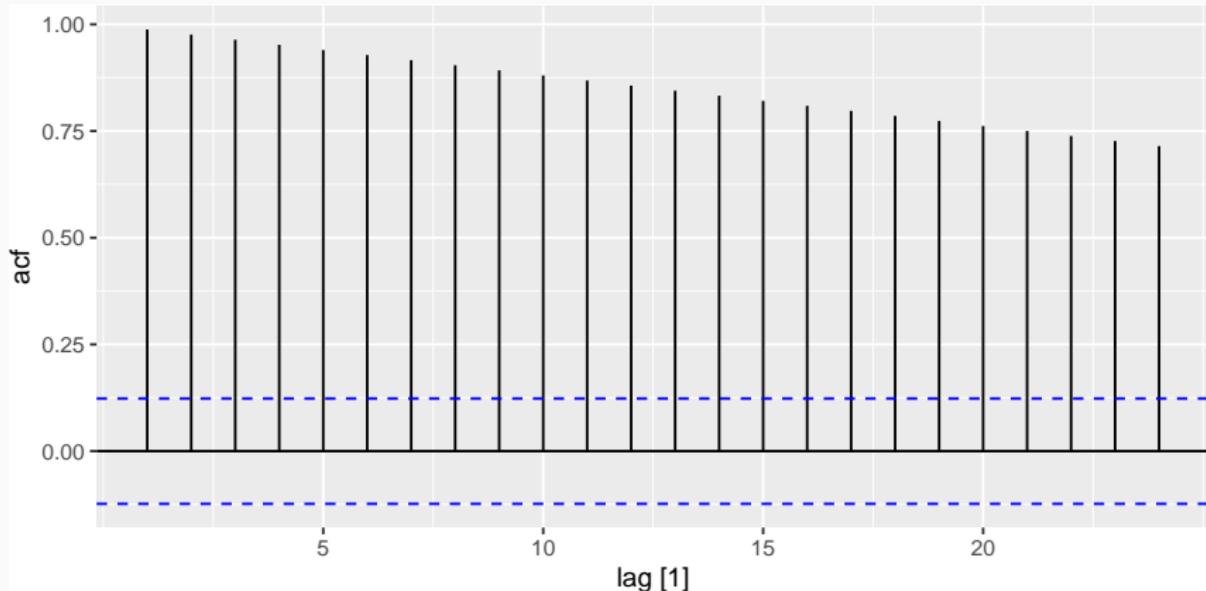
## Google closing stock price

```
google_2015 <- gafa_stock %>%
  filter(Symbol == "GOOG", year(Date) == 2015) %>%
  mutate(trading_day = row_number()) %>%
  update_tsibble(index = trading_day, regular = TRUE)
google_2015 %>% autoplot(Close) + ylab("Google closing stock price ($USD)")
```



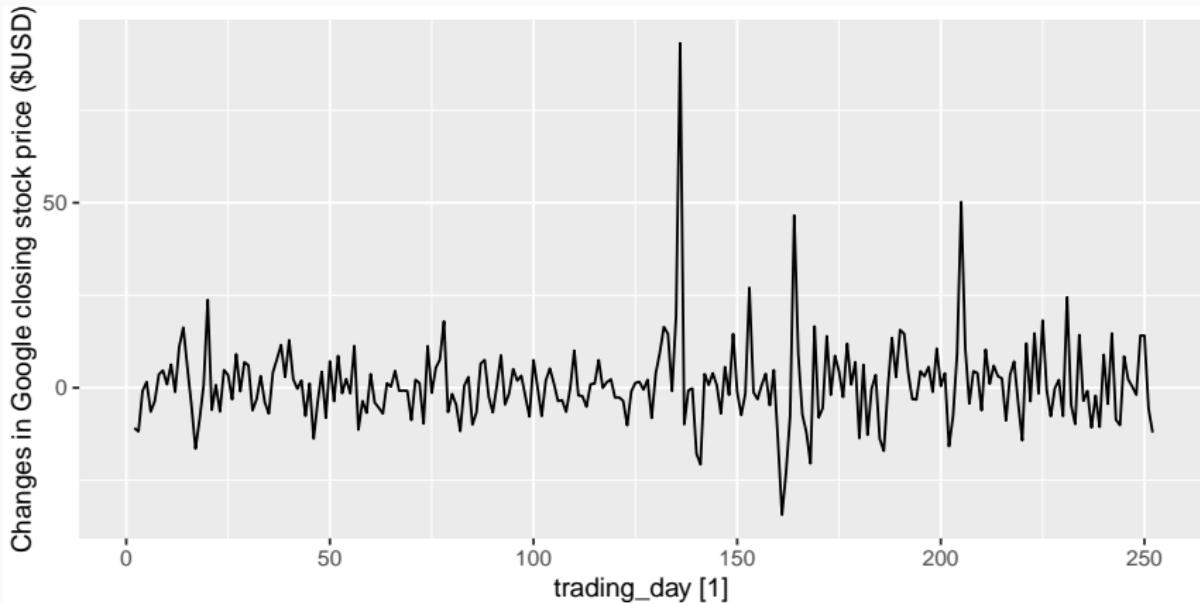
## Google closing stock price

```
google_2015 %>%  
  ACF() %>% autoplot()
```



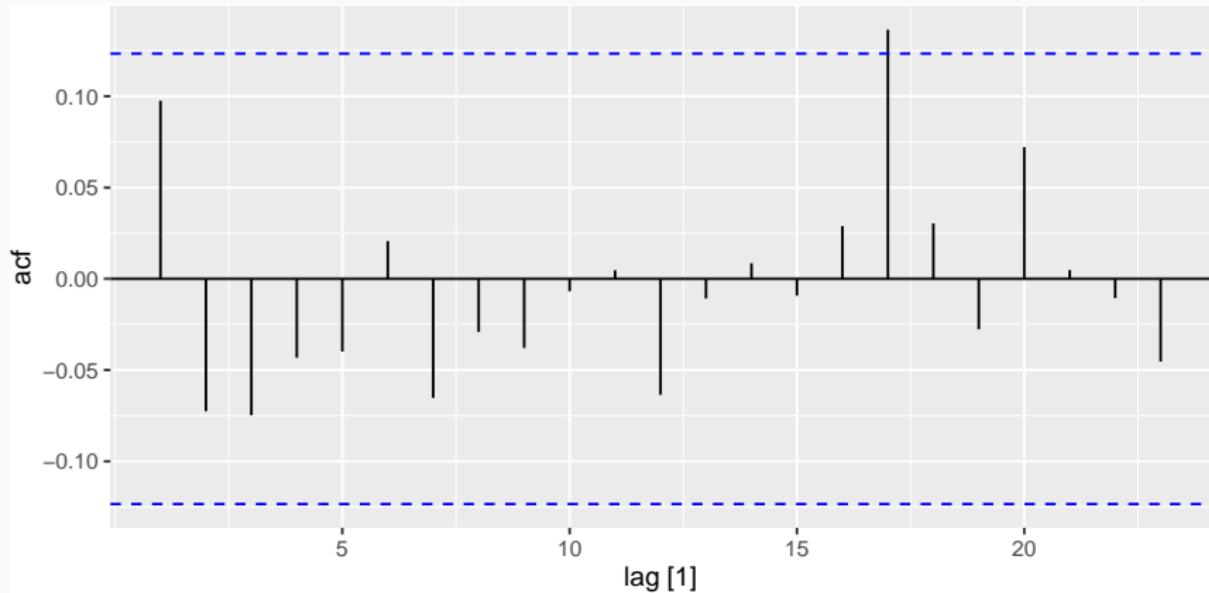
## Changes in Google closing stock price

```
google_2015 %>%
  autoplot(difference(Close)) +
  ylab("Changes in Google closing stock price ($USD)")
```



# Changes in Google closing stock price

```
google_2015 %>%  
  ACF(difference(Close)) %>%  
  autoplot()
```



## Differencing

- Mathematical transformations help to stabilise the variance of a time series.
- For ARIMA models, we need to stabilise the mean of a time series.
- Differencing can be helpful to remove changes in the level of a time series.
  - ▶ Eliminates (or reduces) trend and seasonality.
- The differenced series is the change between consecutive observations in the original series:

$$y'_t = y_t - y_{t-1}, \quad t = 2, 3, \dots, T.$$

- The differenced series will have only  $T - 1$  observations.

## Random walk model

- When the differenced series is white noise:

$$y_t - y_{t-1} = \varepsilon_t \implies y_t = y_{t-1} + \varepsilon_t,$$

where  $\varepsilon_t$  denotes a white noise series.

- These models are widely used in financial and economic data.
- The forecasts from a random walk model are equal to the last observation (naïve forecasts).
- When the differences have a non-zero mean:

$$y_t - y_{t-1} = c + \varepsilon_t,$$

where  $c$  is called the drift.

- We refer to this as random walk with drift model.
- If  $c$  is positive,  $y_t$  will tend to drift upwards and if  $c$  is negative,  $y_t$  will tend to drift downwards.

## Second order differencing

- Occasionally the differenced data will not appear to be stationary.
- Therefore, it may be necessary to difference the data twice:

$$\begin{aligned}y_t'' &= y_t' - y_{t-1}' \\&= (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) \\&= y_t - 2y_{t-1} + y_{t-2},\end{aligned}$$

for  $t = 3, 4, \dots, T$ .

- $y_t''$  will have only  $T - 2$  observations.
- The series represents the “change in changes” of the original data.
- In practice, it is almost never necessary to go beyond second-order differencing.

## Seasonal differencing

- A seasonal difference is the difference between an observation and the previous observation from the same season.

$$y'_t = y_t - y_{t-m},$$

where  $m$  is the seasonal period.

- ▶ For monthly data  $m = 12$ .
- ▶ For quarterly data  $m = 4$ .

- If seasonally differenced series appears to be white noise:

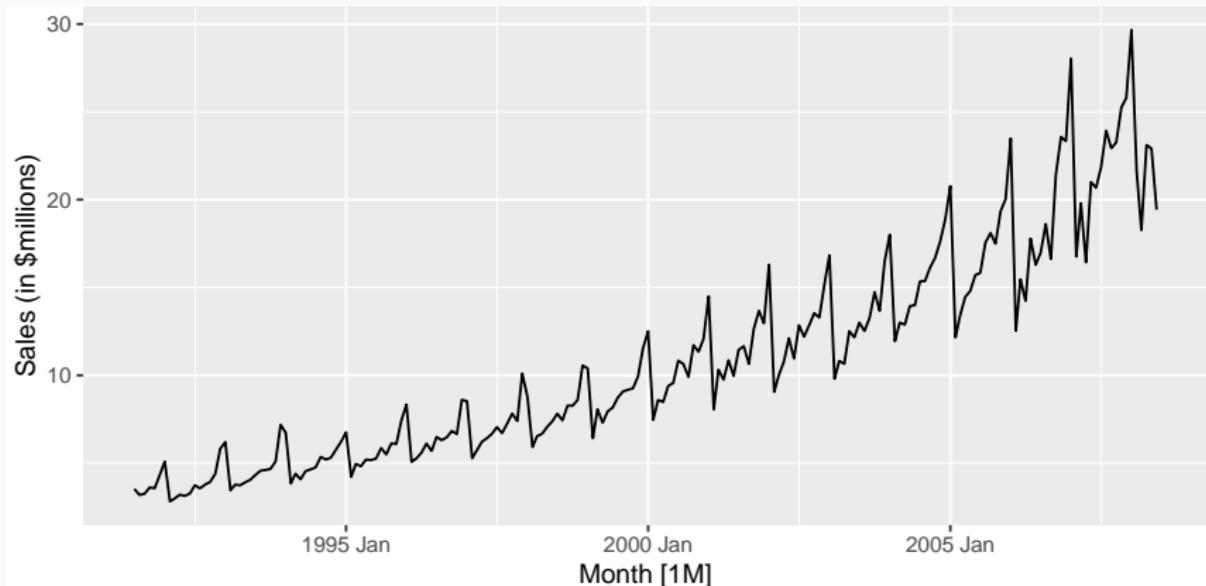
$$y_t - y_{t-m} = \varepsilon_t \implies y_t = y_{t-m} + \varepsilon_t,$$

where  $\varepsilon_t$  denotes a white noise series.

- Forecasts from the model are equal to the observations from the same season last year.
- The model behind seasonal naïve forecasts.

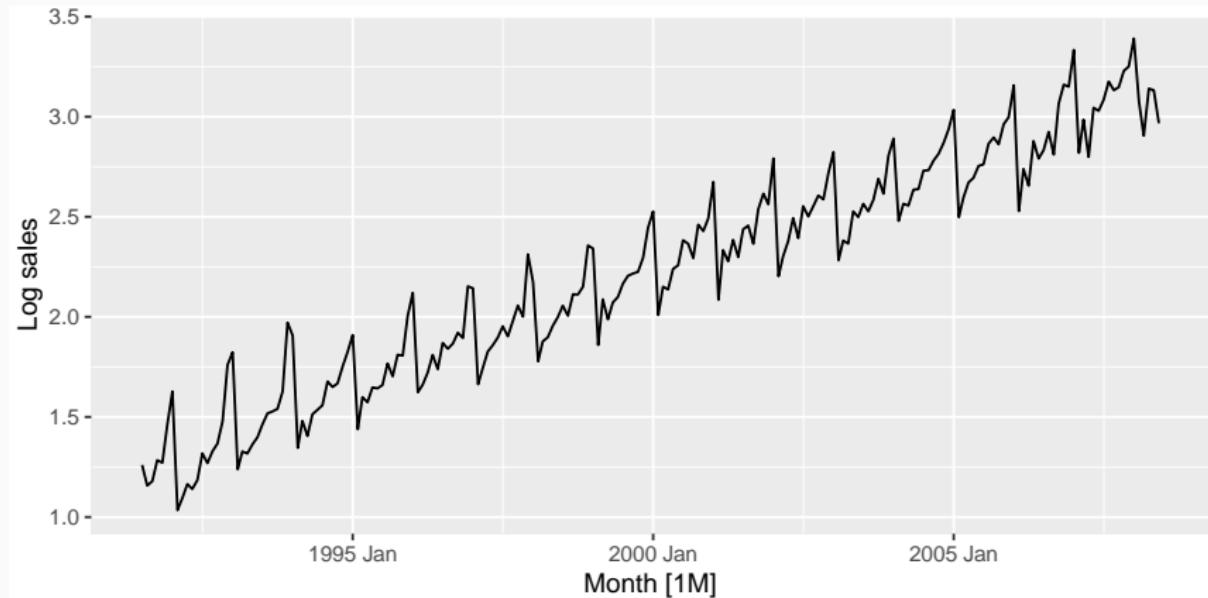
# Antidiabetic drug sales

```
a10 <- PBS %>%
  filter(ATC2 == "A10") %>%
  summarise(Cost = sum(Cost)/1e6)
a10 %>% autoplot(Cost) +
  ylab("Sales (in $millions)")
```



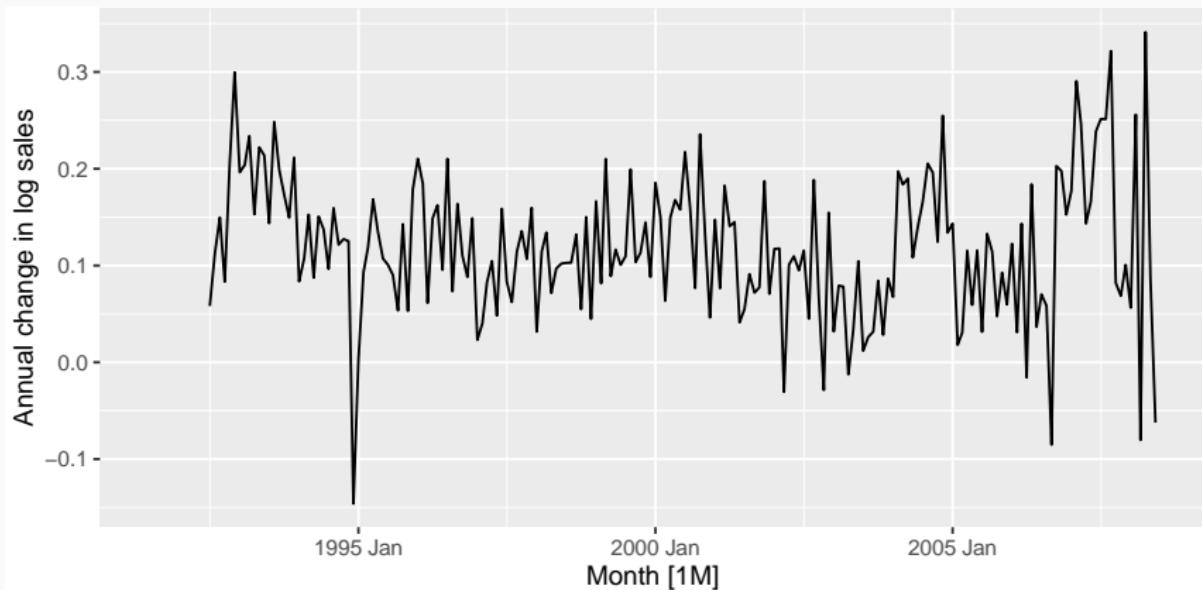
## Antidiabetic drug sales

```
a10 %>% autoplot(log(Cost)) +  
  ylab("Log sales")
```



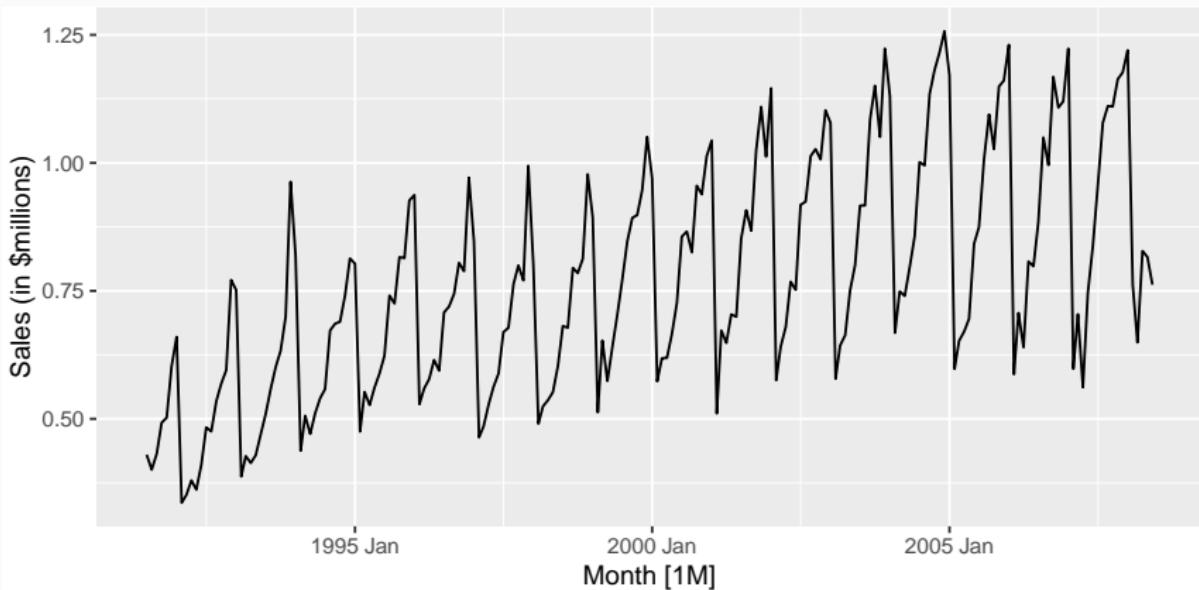
# Antidiabetic drug sales

```
a10 %>%  
  autoplot(log(Cost) %>% difference(12)) +  
  ylab("Annual change in log sales")
```



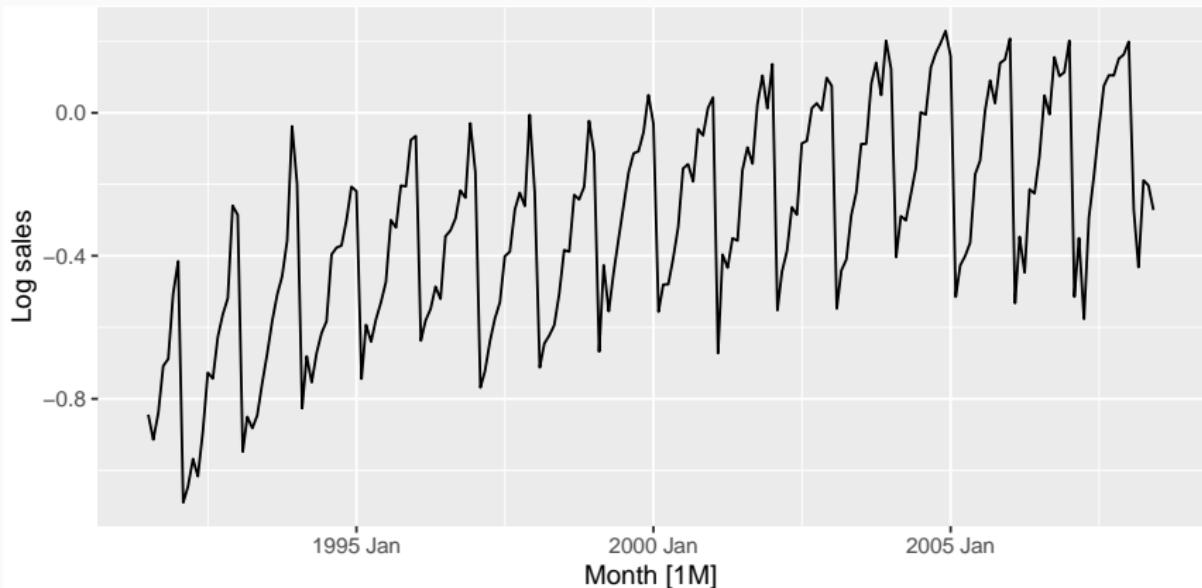
# Corticosteroid drug sales

```
h02 <- PBS %>%
  filter(ATC2 == "H02") %>%
  summarise(Cost = sum(Cost)/1e6)
h02 %>% autoplot(Cost) +
  ylab("Sales (in $millions)")
```



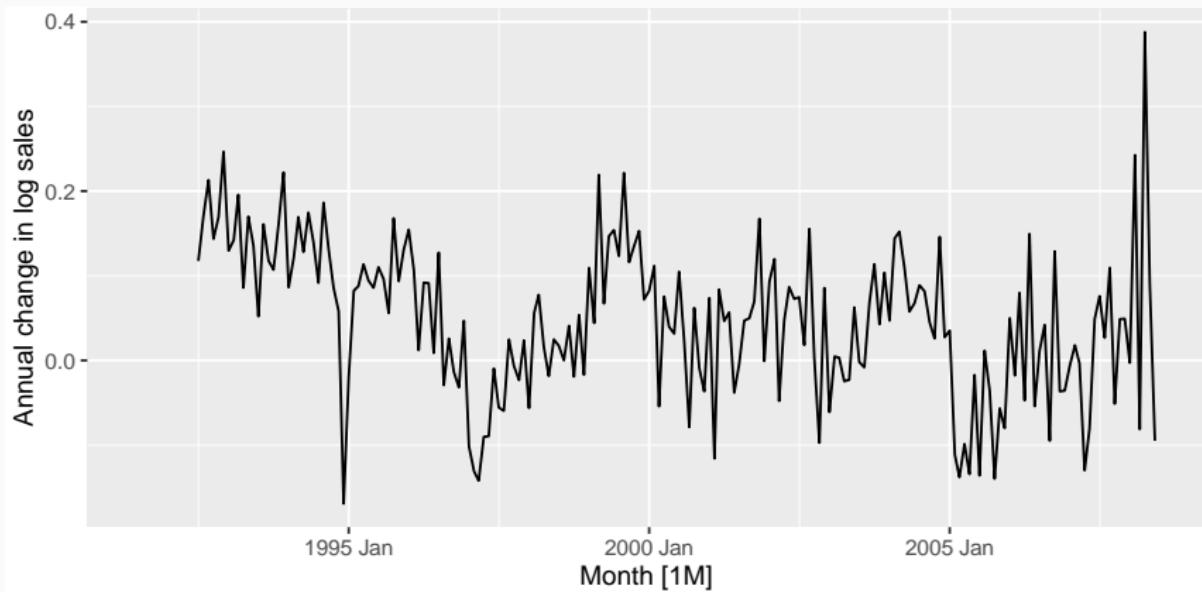
## Corticosteroid drug sales

```
h02 %>% autoplot(log(Cost)) +  
  ylab("Log sales")
```



## Corticosteroid drug sales

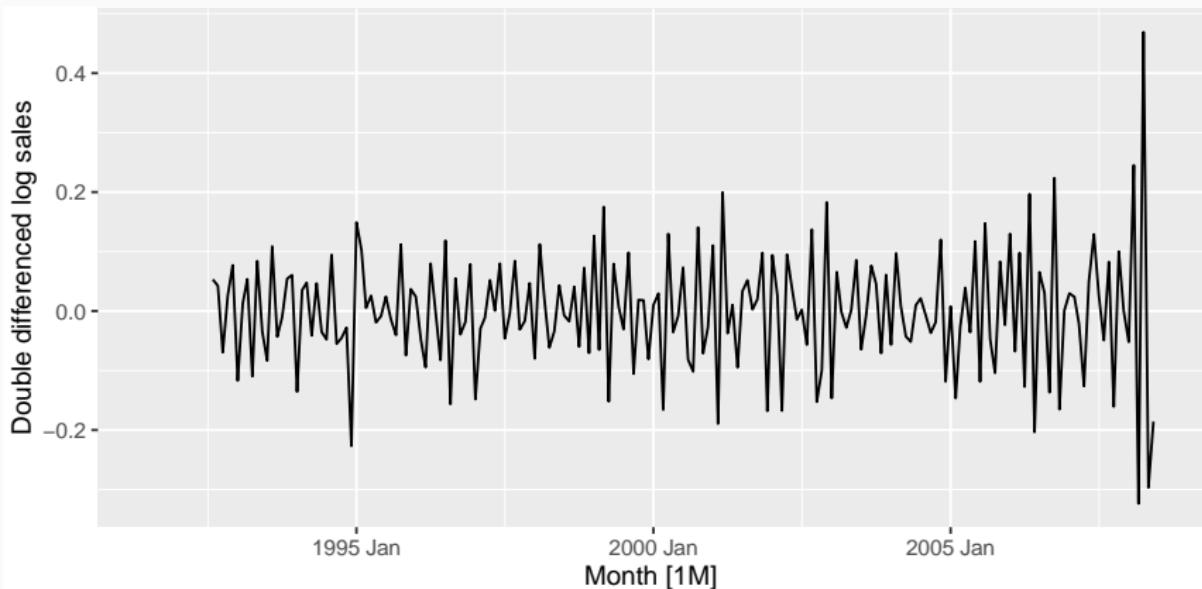
```
h02 %>%
  autoplot(log(Cost) %>% difference(12)) +
  ylab("Annual change in log sales")
```



# Corticosteroid drug sales

```
h02 %>%
```

```
  autoplot(log(Cost) %>% difference(12) %>% difference(1)) +  
  ylab("Double differenced log sales")
```



## Corticosteroid drug sales

If  $y'_t = y_t - y_{t-m}$  denotes a seasonally differenced series. Then the twice-differenced series is

$$\begin{aligned}y''_t &= y'_t - y'_{t-1} \\&= (y_t - y_{t-m}) - (y_{t-1} - y_{t-m-1}) \\&= y_t - y_{t-1} - y_{t-m} + y_{t-m-1}\end{aligned}$$

- When both seasonal and first differences are applied:
  - ▶ it makes no difference which is applied first—the end result will be the same.
  - ▶ If the data have a strong seasonal pattern, it is recommended to apply seasonal differencing first, because sometimes the resulting series will be stationary and there will be no need for a further first difference.

## Beware

- Applying more differences than required will induce false dynamics or autocorrelations that do not exist in the original time series.
- Perform as few differences as necessary to obtain a stationary time series.

## Interpretation of differences

- First differences are the change between one observation and the next.
- Seasonal differences are the change between one year to the next.
- Other lags are unlikely to have a meaningful interpretation and should be avoided.

- These are statistical hypothesis tests to determine whether a time series needs differencing or not.
- A number of unit root tests are available.
- Augmented Dickey Fuller (ADF) test: null hypothesis is that the data are non-stationary and non-seasonal.
- Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test: null hypothesis is that the data are stationary and non-seasonal.
- Phillips-Perron (PP) test: the null hypothesis is that the data are non-stationary and non-seasonal.
- Tests are available for seasonal data.

## KPSS test

```
google_2015 %>%  
  features(Close, unitroot_kpss)
```

```
## # A tibble: 1 x 3  
##   Symbol kpss_stat kpss_pvalue  
##   <chr>     <dbl>        <dbl>  
## 1 GOOG      3.56       0.01
```

```
google_2015 %>%  
  mutate(diff_close = difference(Close)) %>%  
  features(diff_close, unitroot_kpss)
```

```
## # A tibble: 1 x 3  
##   Symbol kpss_stat kpss_pvalue  
##   <chr>     <dbl>        <dbl>  
## 1 GOOG      0.0989      0.1
```

## Automatically selecting number of differences

```
google_2015 %>%
  features(Close, unitroot_ndiffs)

## # A tibble: 1 x 2
##   Symbol ndiffs
##   <chr>   <int>
## 1 GOOG      1
```

## Seasonal differencing

- We use seasonal strength of an STL decomposition to identify the need for a seasonal difference.
- This is not a statistical hypothesis test.
- STL decomposition:  $y_t = T_t + S_t + R_t$ .
- For strongly seasonal data, the detrended data should have much more variation than the remainder component:  
 $\text{Var}(R_t)/\text{Var}(S_t + R_t)$  should be relatively small.
- For data with weak seasonality or no seasonality, the variances should be roughly the same.
- Seasonal strength:  $F_s = \max \left( 0, 1 - \frac{\text{Var}(R_t)}{\text{Var}(S_t + R_t)} \right)$
- If  $F_s > 0.64$ , one seasonal difference is suggested, otherwise no seasonal difference.

# Seasonal differencing

```
h02 %>%
  mutate(log_cost = log(Cost)) %>%
  features(log_cost, feat_stl)

## # A tibble: 1 x 9
##   trend_strength seasonal_strength_year seasonal_peak_year
##           <dbl>                  <dbl>                  <dbl>
## 1          0.957                 0.955                   6
## # ... with 6 more variables: seasonal_trough_year <dbl>,
## #   spikiness <dbl>, linearity <dbl>, curvature <dbl>,
## #   stl_e_acf1 <dbl>, stl_e_acf10 <dbl>
```

## Automatically selecting number of differences

```
h02 %>%  
  mutate(log_cost = log(Cost)) %>% features(log_cost, unitroot_nsdiffs)
```

```
## # A tibble: 1 x 1  
##   nsdiffs  
##     <int>  
## 1       1
```

```
h02 %>% mutate(d_log_cost = difference(log(Cost), 12)) %>%  
  features(d_log_cost, unitroot_ndiffs)
```

```
## # A tibble: 1 x 1  
##   ndiffs  
##     <int>  
## 1       1
```

- These functions suggest we should apply one seasonal difference and one first difference.

- The backward shift operator  $B$  is useful when working with time series lags:

$$By_t = y_{t-1}.$$

- The effect of applying  $B$  on  $y_t$  is to shift the data back by one period.
- Applying  $B$  twice on  $y_t$ :

$$B^2 y_t = B(By_t) = By_{t-1} = y_{t-2}$$

- For monthly data, if we wish to consider “the same month last year”:

$$B^{12} y_t = y_{t-12}.$$

- The backward shift operator is convenient for describing the process of differencing.
- A first difference:

$$y'_t = y_t - y_{t-1} = y_t - By_t = (1 - B)y_t.$$

- We can write second-order differences (first differences of first differences) as:

$$y''_t = y_t - 2y_{t-1} + y_{t-2} = (1 - 2B + B^2)y_t = (1 - B)^2 y_t.$$

- Second-order difference is not the same as a second difference  $(1 - B^2)$ .

- A first difference can be represented by  $(1 - B)$ .
- Second order difference can be represented by  $(1 - B)^2$ .
- In general, a  $d$ -th order difference can be written as  $(1 - B)^d$ .
- A seasonal difference followed by a first difference:

$$\begin{aligned}(1 - B)(1 - B^m)y_t &= (1 - B - B^m + B^{m+1})y_t \\ &= y_t - y_{t-1} - y_{t-m} + y_{t-m-1},\end{aligned}$$

where  $m$  is the seasonal period.

# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

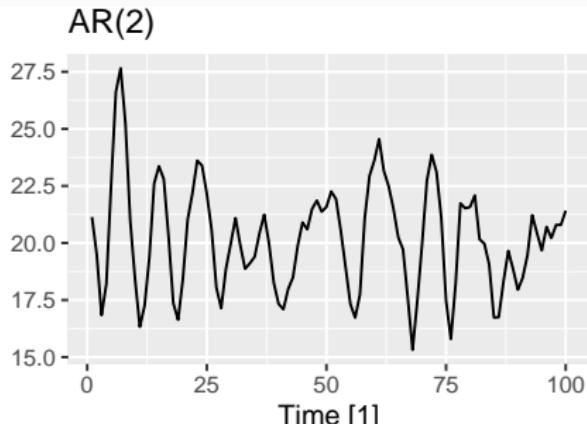
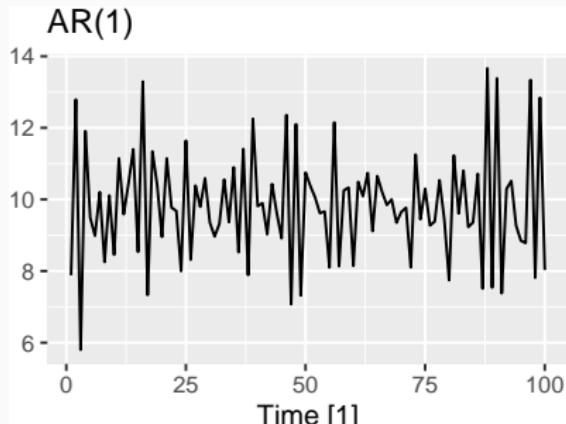
# Autoregressive models

- In multiple linear regression models, we forecast the variable of interest using a linear combination of predictors.
- An autoregressive model is a regression model where we use lagged values of the variable of interest as predictors.

## Autoregressive model of order $p$ : AR( $p$ )

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t,$$

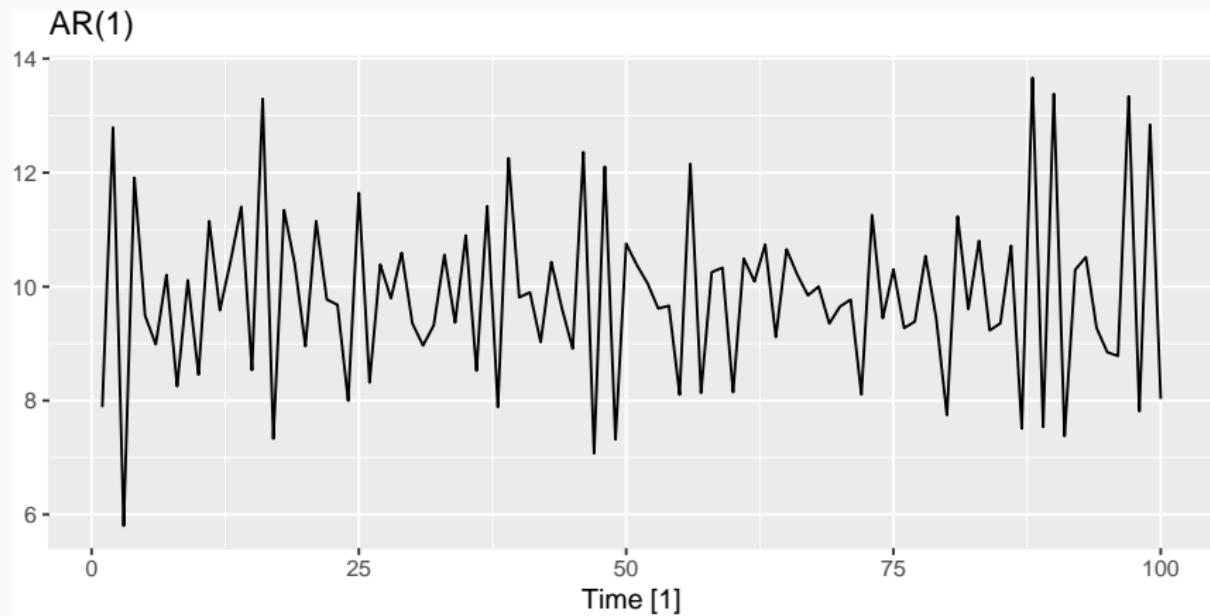
where  $\varepsilon_t$  is a white noise series.



## AR(1) model

$$y_t = 18 - 0.8y_{t-1} + \varepsilon_t,$$

where  $\varepsilon_t \sim N(0, 1)$  and  $t = 1, 2, \dots, 100$ .



## AR(1) model

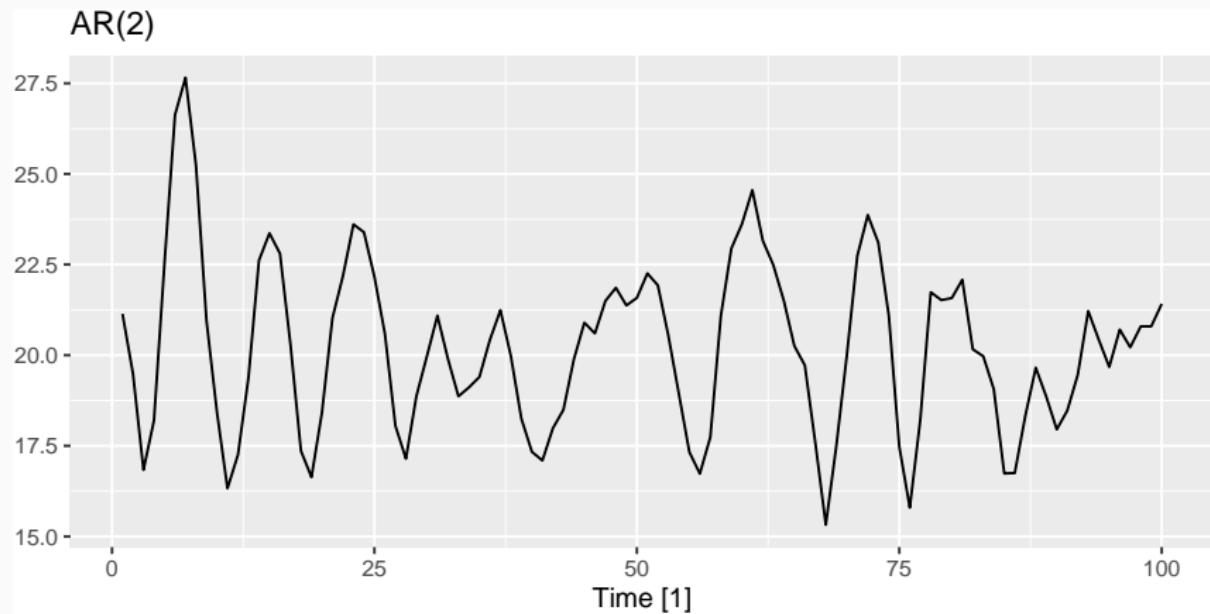
$$y_t = c + \phi_1 y_{t-1} + \varepsilon_t.$$

- When  $\phi_1 = 0$ , and  $c = 0$ ,  $y_t$  is equivalent to a white noise series.
- When  $\phi_1 = 1$  and  $c = 0$ ,  $y_t$  is equivalent to a random walk.
- When  $\phi_1 = 1$  and  $c \neq 0$ ,  $y_t$  is equivalent to a random walk with drift.
- When  $\phi_1 < 0$ ,  $y_t$  tends to oscillate around the mean.

## AR(2) model

$$y_t = 8 + 1.3y_{t-1} - 0.7y_{t-2} + \varepsilon_t,$$

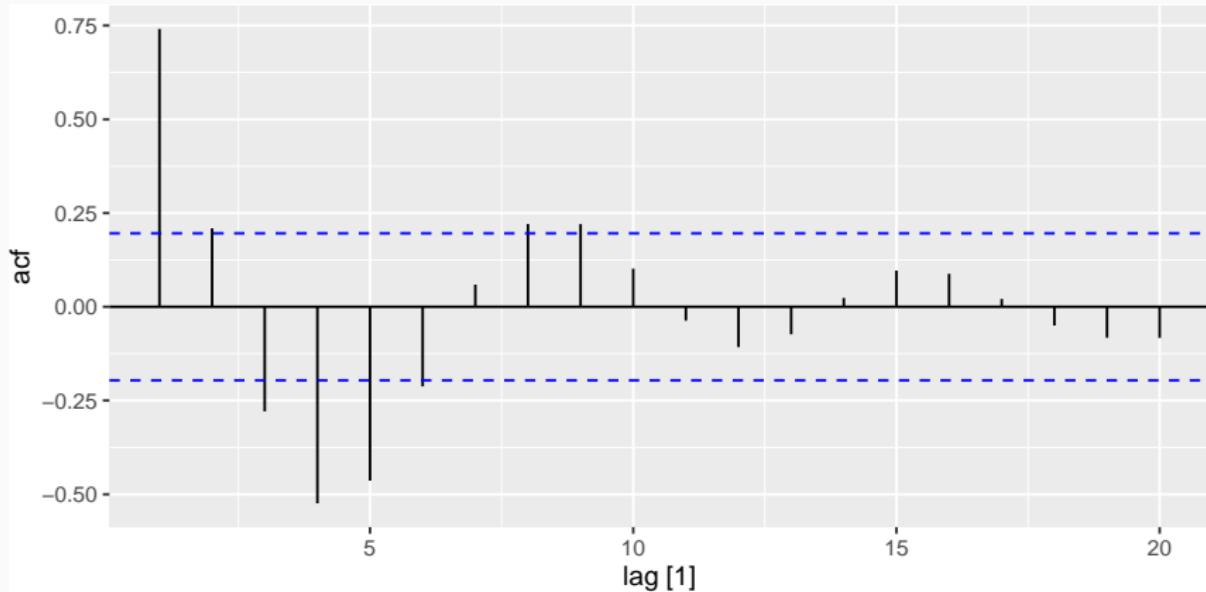
where  $\varepsilon_t \sim N(0, 1)$  and  $t = 1, 2, \dots, 100$ .



## AR(2) model

- A cyclic behavior can occur if  $\phi_1^2 + 4\phi_2 < 0$ .
- The average period of the cycles is

$$\frac{2\pi}{\text{arc cos}(-\phi_1(1 - \phi_2)/(4\phi_2))}.$$



## Stationarity conditions

- We normally restrict autoregressive models to stationary data.
- We need to define some constraints on the values of the parameters:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \varepsilon_t.$$

### Conditions for stationarity

The roots of  $\phi(z) = 1 - \phi_1 z - \phi_2 z^2 - \dots - \phi_p z^p$  lie outside the unit circle on the complex plane.

- For an AR(1):  $-1 < \phi_1 < 1$ .
- For an AR(2):  $-1 < \phi_2 < 1, \phi_1 + \phi_2 < 1, \phi_2 - \phi_1 < 1$ .
- For an AR( $p$ ) model where  $p \geq 3$ : complicated conditions exist.
- The fable package checks these restrictions when estimating a model.

# Moving average models

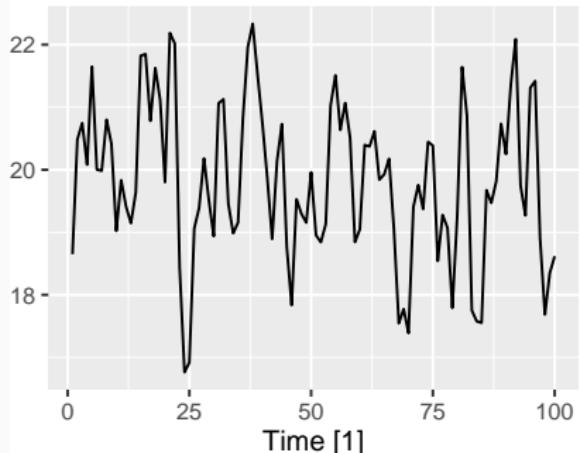
- These models use past forecast errors in a regression-like model.

## Moving average model of order $q$ : MA( $q$ )

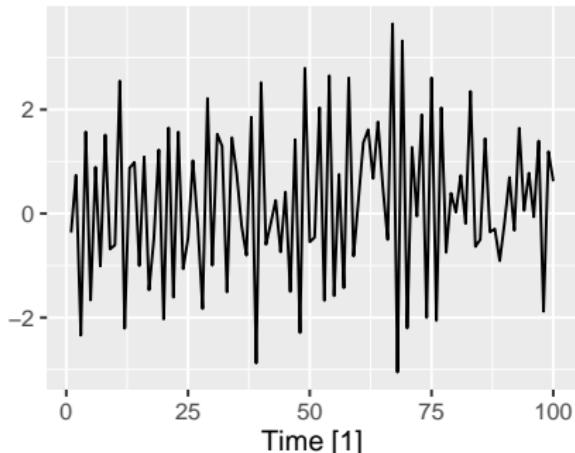
$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

where  $\varepsilon_t$  is a white noise series.

MA(1)



MA(2)



# Moving average models

- These models use past forecast errors in a regression-like model.

## Moving average model of order $q$ : MA( $q$ )

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

where  $\varepsilon_t$  is a white noise series.

MA(1)



MA(2)



### Don't get confused

- Moving average models and moving average smoothing are different.
- A moving average model is used for forecasting, whereas moving average smoothing is used for estimating the trend-cycle of past observations.

Time [1]

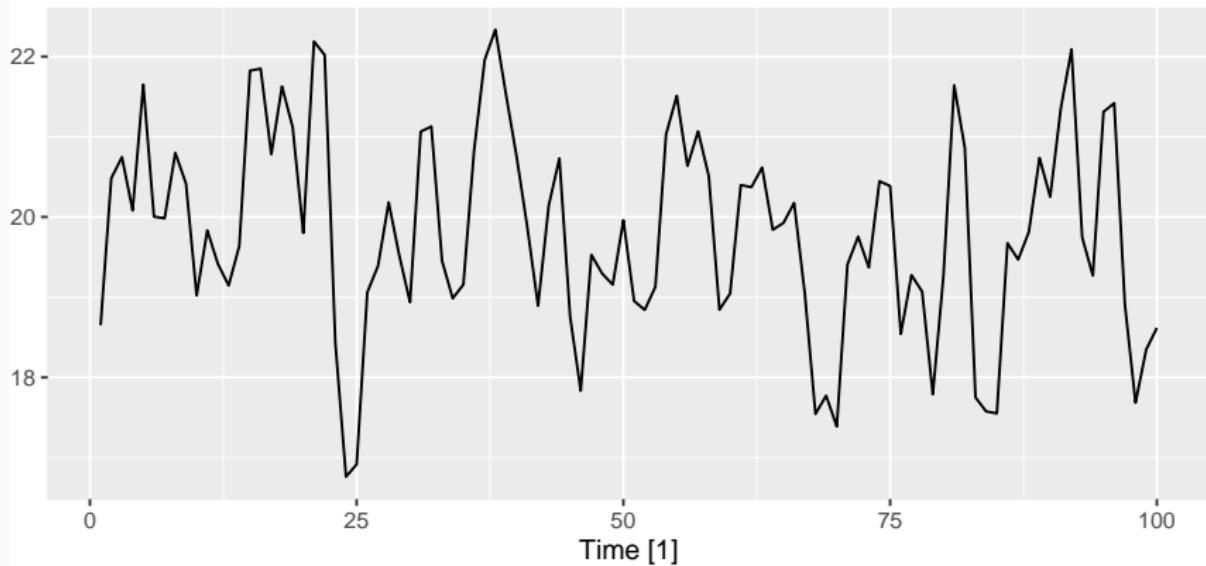
Time [1]

## MA(1) model

$$y_t = 20 + \varepsilon_t + 0.8\varepsilon_{t-1},$$

where  $\varepsilon_t \sim N(0, 1)$  and  $t = 1, 2, \dots, 100$ .

MA(1)

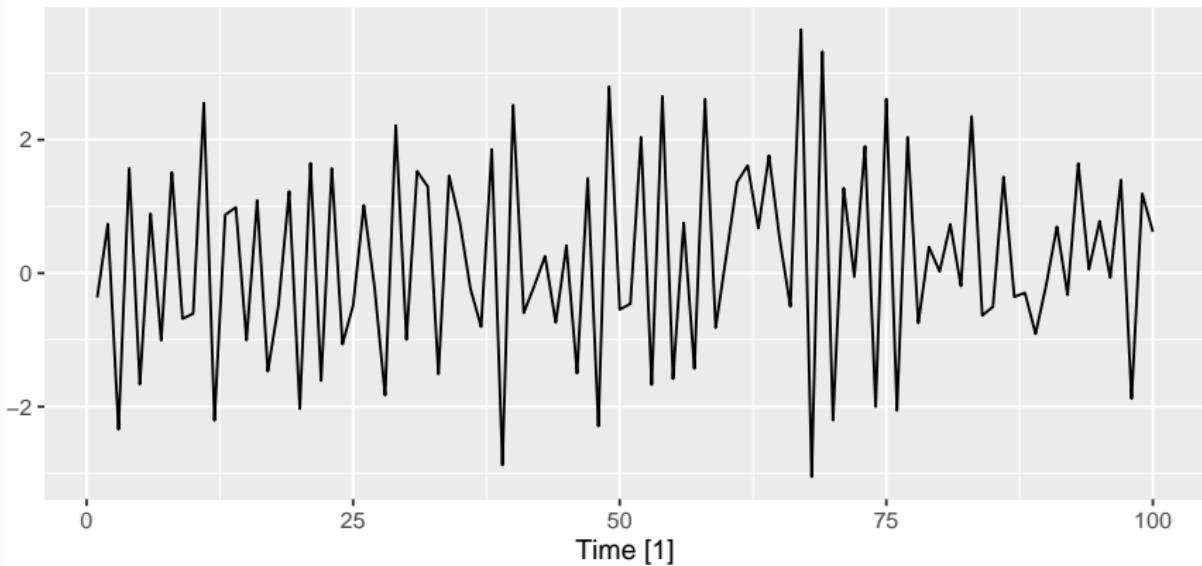


## MA(2) model

$$y_t = \varepsilon_t - \varepsilon_{t-1} + 0.8\varepsilon_{t-2},$$

where  $\varepsilon_t \sim N(0, 1)$  and  $t = 1, 2, \dots, 100$ .

MA(2)



## AR( $p$ ) to MA( $\infty$ )

Any stationary AR( $p$ ) process can be written as an MA( $\infty$ ) process.

### AR(1)

$$\begin{aligned}y_t &= \phi_1 y_{t-1} + \varepsilon_t \\&= \phi_1(\phi_1 y_{t-2} + \varepsilon_{t-1}) + \varepsilon_t = \phi_1^2 y_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\&= \phi_1^3 y_{t-3} + \phi_1^2 \varepsilon_{t-2} + \phi_1 \varepsilon_{t-1} + \varepsilon_t \\&\vdots\end{aligned}$$

Eventually we get

$$y_t = \varepsilon_t + \phi_1 \varepsilon_{t-1} + \phi_1^2 \varepsilon_{t-2} + \phi_1^3 \varepsilon_{t-3} + \dots \quad \text{MA}(\infty)$$

- As  $-1 < \phi_1 < 1$ , the value of  $\phi_1^k$  will become smaller and smaller as  $k$  gets larger.

## Invertibility

- Any MA( $q$ ) process can be written as an AR( $\infty$ ) process if we impose constraints on the MA parameters.
- Then we say that the MA model is invertible.

### MA(1)

$$\begin{aligned}\varepsilon_t &= y_t - \theta_1 \varepsilon_{t-1} \\&= y_t - \theta_1(y_{t-1} - \theta_1 \varepsilon_{t-2}) = y_t - \theta_1 y_{t-1} + \theta_1^2 \varepsilon_{t-2} \\&= y_t - \theta_1 y_{t-1} + \theta_1^2(y_{t-2} - \theta_1 \varepsilon_{t-3}) \\&\vdots\end{aligned}$$

Eventually, we get

$$\varepsilon_t = \sum_{j=0}^{\infty} (-\theta_1)^j y_{t-j}.$$

For an MA(1) process:

$$\varepsilon_t = \sum_{j=0}^{\infty} (-\theta_1)^j y_{t-j}$$

- When  $|\theta_1| > 1$ , the weights increase as lags increase.
  - ▶ So the more distant the observations the greater their influence on the current error.
- When  $|\theta_1| = 1$ , the weights are constant in size.
  - ▶ The distant observations have the same influence as the recent observations.
- When  $|\theta_1| < 1$ , the weights decay as lags increase.
  - ▶ More recent observations have higher weight than the observations from the distant past.

# Invertibility conditions

$$y_t = c + \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q}.$$

## Conditions for invertibility

The roots of  $\theta(z) = 1 + \theta_1 z + \theta_2 z^2 + \dots + \theta_q z^q$  lie outside the unit circle on the complex plane.

- For an MA(1):  $-1 < \theta_1 < 1$ .
- For an MA(2):  $-1 < \theta_2 < 1, \theta_1 + \theta_2 > -1, \theta_1 - \theta_2 < 1$ .
- For an MA( $q$ ) model where  $q \geq 3$ : complicated conditions exist.
- The fable package checks these restrictions when estimating a model.

## ARIMA models

### Autoregressive moving average models: ARMA( $p, q$ )

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_p y_{t-p} + \\ \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

where  $\varepsilon_t$  is a white noise series.

- The model includes both lagged values of  $y_t$  and lagged errors.
- The conditions on AR coefficients ensure stationarity.
- The conditions on MA coefficients ensure invertibility.

### Autoregressive integrated moving average model: ARIMA( $p, d, q$ )

Let  $y'_t = (1 - B)^d y_t$ .

$$y'_t = c + \phi_1 y'_{t-1} + \phi_2 y'_{t-2} + \cdots + \phi_p y'_{t-p} + \\ \varepsilon_t + \theta_1 \varepsilon_{t-1} + \theta_2 \varepsilon_{t-2} + \cdots + \theta_q \varepsilon_{t-q},$$

where  $\varepsilon_t$  is a white noise series.

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

$$(1 - \phi_1 B - \cdots - \phi_p B^p) \uparrow \quad (1 - B)^d y_t \uparrow = c + (1 + \theta_1 B + \cdots + \theta_q B^q) \varepsilon_t \uparrow$$

AR( $p$ )                     $d$  differences                    MA( $q$ )

- White noise model: ARIMA(0,0,0) with no constant
- Random walk model: ARIMA(0,1,0) with no constant
- Random walk with drift: ARIMA(0,1,0) with a constant
- AR( $p$ ) model: ARIMA( $p, 0, 0$ )
- MA( $q$ ) model: ARIMA( $0, 0, q$ )

# Parameterization of ARIMA models

Let  $y'_t = (1 - B)^d y_t$ .

## Intercept form

$$(1 - \phi_1 B - \cdots - \phi_p B^p) y'_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q) \varepsilon_t.$$

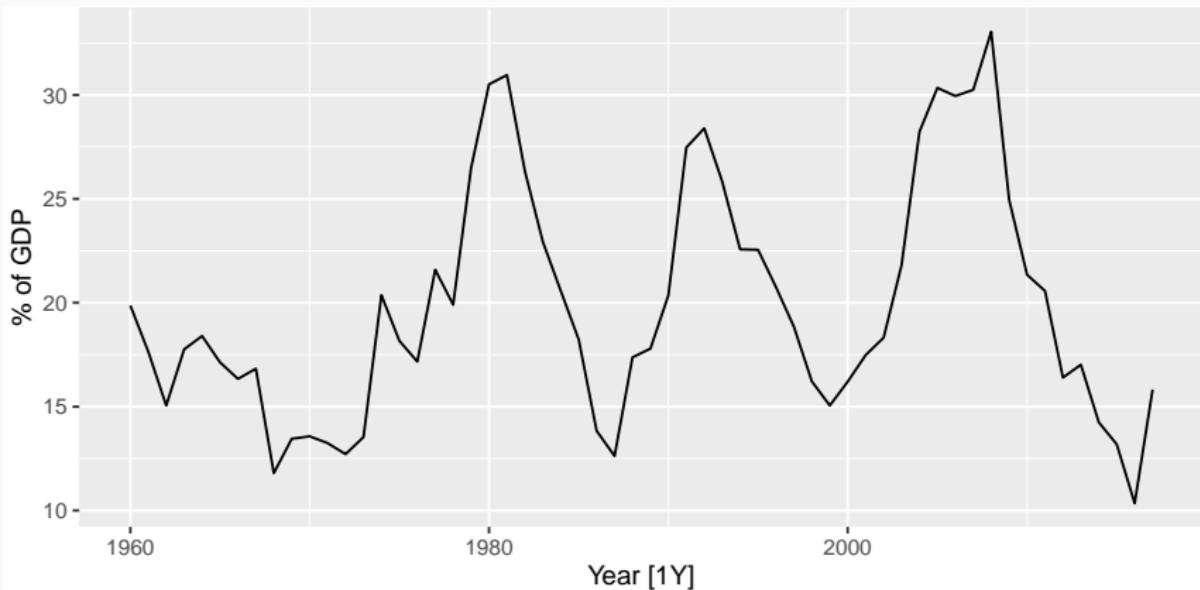
## Mean form

$$(1 - \phi_1 B - \cdots - \phi_p B^p)(y'_t - \mu) = (1 + \theta_1 B + \cdots + \theta_q B^q) \varepsilon_t.$$

- $\mu$  is the mean of  $y'_t$ .
- $c = \mu(1 - \phi_1 - \cdots - \phi_p)$ .
- The fable package uses intercept form.

# Egyptian exports data

```
global_economy %>%
  filter(Code == "EGY") %>%
  autoplot(Exports) +
  ylab("% of GDP")
```



## Egyptian exports data

```
fit <- global_economy %>% filter(Code == "EGY") %>%
  model(ARIMA(Exports))
report(fit)

## Series: Exports
## Model: ARIMA(2,0,1) w/ mean
##
## Coefficients:
##             ar1      ar2      ma1  constant
##             1.676   -0.8034  -0.690     2.562
## s.e.    0.111    0.0928   0.149     0.116
##
## sigma^2 estimated as 8.046: log likelihood=-142
## AIC=293   AICc=294   BIC=303
```

# Egyptian exports data

```
fit <- global_economy %>% filter(Code == "EGY") %>%  
  model(ARIMA(Exports))  
  report(fit)
```

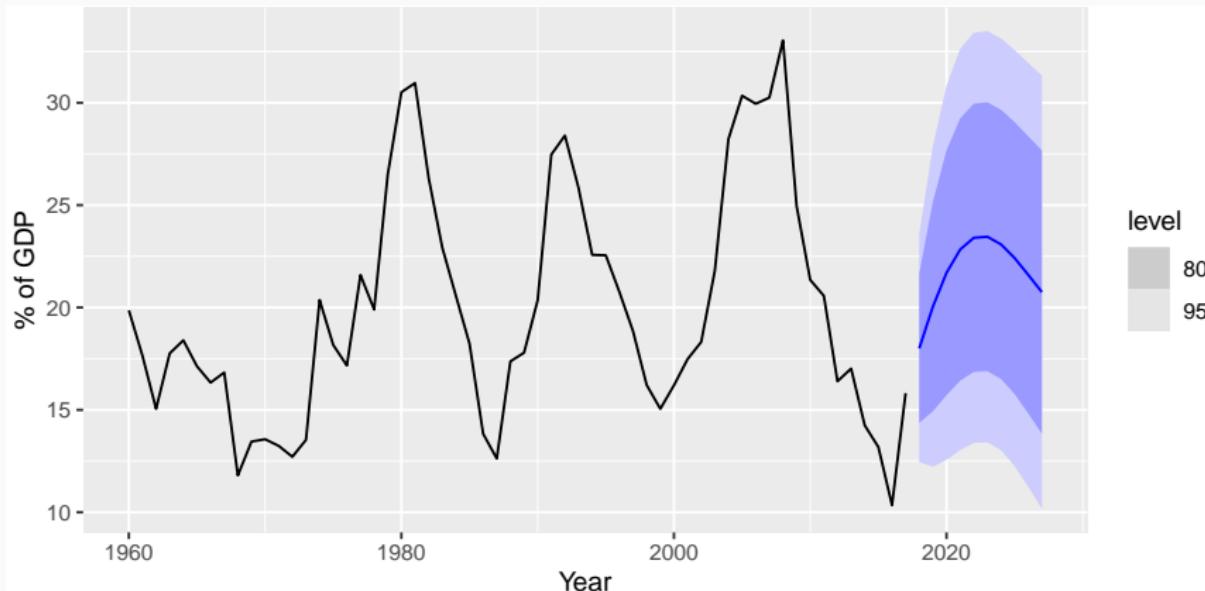
```
## Series: Exports  
## Model: ARIMA(2,0,1) w/ mean  
##  
## Coefficients:  
##          ar1      ar2      ma1  constant  
##          1.676   -0.8034  -0.690     2.562  
## s.e.    0.111    0.0928   0.149     0.116  
##  
# ARIMA(2,0,1) model
```

$$y_t = 2.56 + 1.68y_{t-1} - 0.8y_{t-2} - 0.69\varepsilon_{t-2} + \varepsilon_t,$$

where  $\varepsilon_t$  is a white noise series with zero mean and variance 8.05.

## Egyptian exports data

```
fit %>% forecast(h = 10) %>%
  autoplot(global_economy) +
  ylab("% of GDP")
```

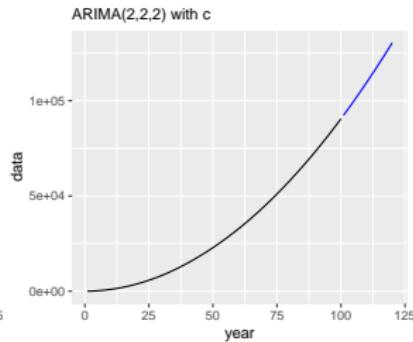
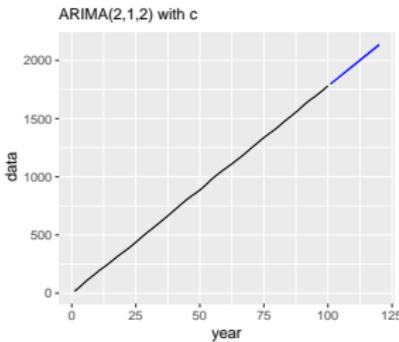
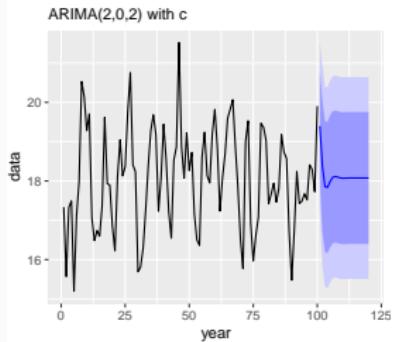
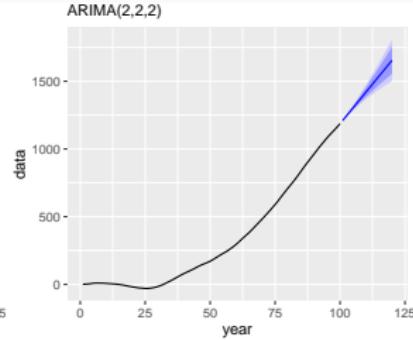
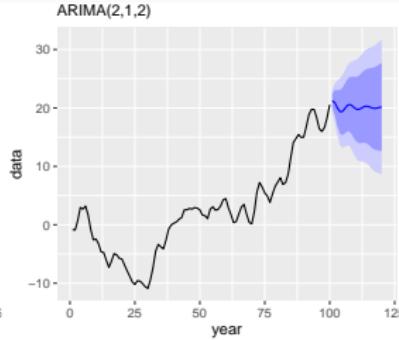
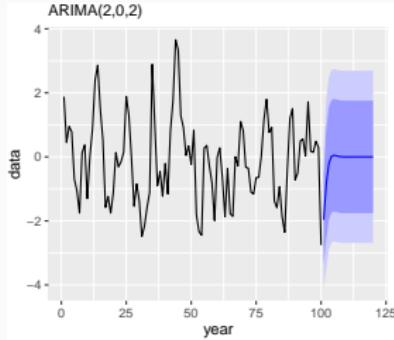


- The model has been able to pick up the cycles seen in the data

## Long-term forecasts from ARIMA models

- If  $c = 0$  and  $d = 0$ , the long-term forecasts will go to zero.
- If  $c = 0$  and  $d = 1$ , the long-term forecasts will go to a non-zero constant.
- If  $c = 0$  and  $d = 2$ , the long term forecasts will follow a straight line.
- If  $c \neq 0$  and  $d = 0$ , the long-term forecasts will go to the mean of the data.
- If  $c \neq 0$  and  $d = 1$ , the long-term forecasts will follow a straight line.
- If  $c \neq 0$  and  $d = 2$ , the long-term forecasts will follow a quadratic trend.

# Long-term forecasts from ARIMA models



## Prediction intervals and order of differencing

- Higher the value of  $d$ , the more rapidly the prediction intervals increase in size.
- For  $d = 0$ , the long-term forecast variance will go the variance of the historical data.

# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

## Partial autocorrelations

- Autocorrelations measure the linear relationship between  $y_t$  and  $y_{t-k}$  for different values of  $k$ .
  - ▶ If  $y_t$  and  $y_{t-1}$  are correlated then  $y_{t-1}$  and  $y_{t-2}$  must be correlated. Then  $y_t$  and  $y_{t-2}$  may be correlated, because they are connected by  $y_{t-1}$ .
- To overcome this problem, we use **partial autocorrelations**.

### Partial autocorrelation at lag $k$ : $\alpha_k$

This measures the correlation between  $y_t$  and  $y_{t-k}$  with the linear dependence of  $\{y_{t-1}, y_{t-2}, \dots, y_{t-k+1}\}$  on each, removed.

- The first partial autocorrelation is identical to the first autocorrelation.

## Partial autocorrelations

$\alpha_k$  = kth partial autocorrelation coefficient

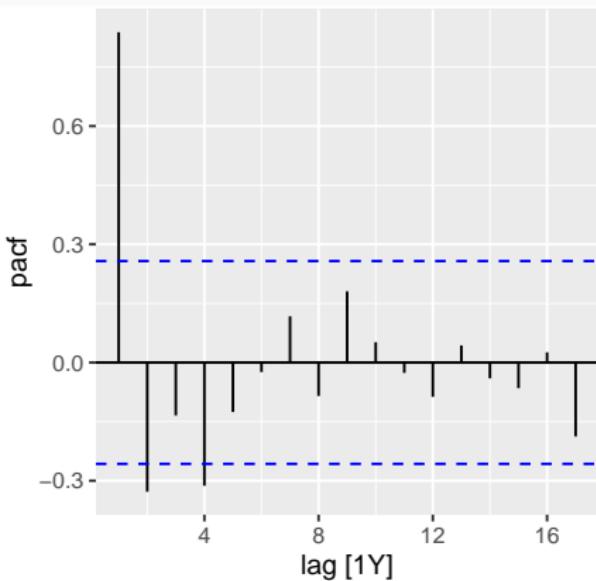
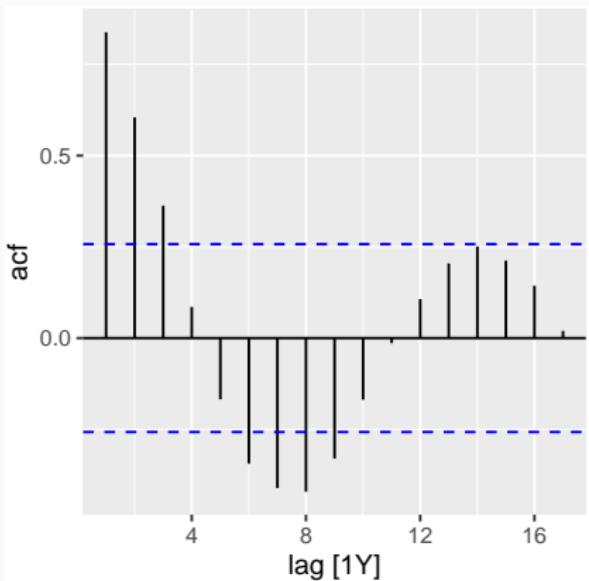
= equal to the coefficient  $\phi_k$  in the following model:

$$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + \cdots + \phi_k y_{t-k} + \varepsilon_t.$$

- Varying the number of terms on RHS gives  $\alpha_k$  for different  $k$ .
- In practice, there are more efficient algorithms for computing  $\alpha_k$  than fitting all of these models.
- Partial autocorrelations use the same critical values  $\pm 1.96/\sqrt{T}$  as for ACF.

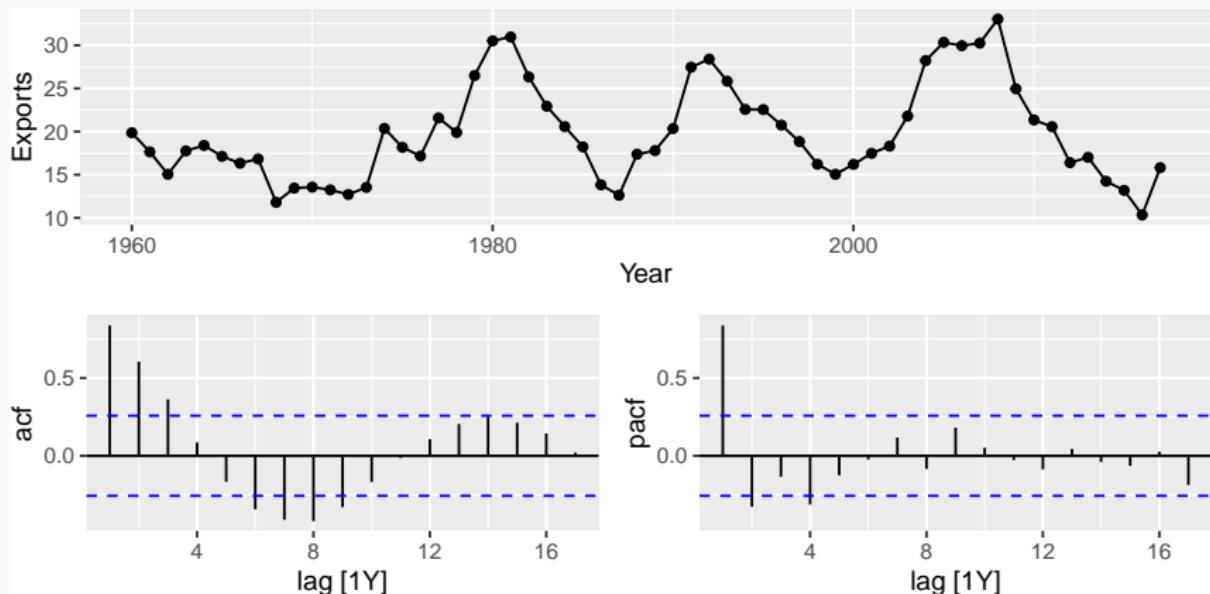
# Egyptian exports data

```
egypt <- global_economy %>% filter(Code == "EGY")
egypt %>% ACF(Exports) %>% autoplot()
egypg %>% PACF(Exports) %>% autoplot()
```



# Egyptian exports data

```
egypt %>% gg_tsdisplay(Exports, plot_type = "partial")
```



## ACF and PACF interpretation for AR models

Let  $\rho_k$  denotes the autocorrelation coefficient at lag  $k$ .

### AR(1)

$$\rho_k = \phi_1^k, \quad \text{for } k = 1, 2, \dots,$$

$$\alpha_1 = \phi_1, \quad \alpha_k = 0, \quad \text{for } k = 2, 3, \dots$$

Features of an AR(1) model

- autocorrelations exponentially decay.
- there is one significant partial autocorrelation.

### AR( $p$ )

- Autocorrelations are exponentially decaying or show a damped sine-wave.
- All partial autocorrelations are zero after the  $p$ th lag.

### Features of an AR( $p$ ) model

- the ACF is exponentially decaying or sinusoidal.
- there is a significant spike at lag  $p$  in the PACF, but none beyond lag  $p$ .

## ACF and PACF interpretation for MA models

### MA(1)

$$\rho_1 = \frac{\theta_1}{1 + \theta_1^2}, \quad \rho_k = 0, \quad \text{for } k = 2, 3, \dots,$$

$$\alpha_k = \frac{-(-\theta_1)^k}{1 + \theta_1^2 + \dots + \theta_1^{2k}}, \quad \text{for } k = 1, 2, \dots$$

Features of an MA(1) model

- there is one significant spike in ACF.
- the PACF is exponentially decaying.

### MA( $q$ )

- All autocorrelations are zero after the  $q$ th lag.
- Partial autocorrelations are exponentially decaying or show a damped sine-wave.

### Features of an MA( $q$ ) model

- there is a significant spike at lag  $q$  in the ACF, but none beyond lag  $q$ .
- the PACF is exponentially decaying or sinusoidal.

## Behaviour of ACF and PACF for ARMA models

	AR( $p$ )	MA( $q$ )	ARMA( $p, q$ )
ACF	tails off	Cuts off after lag $q$	Tails off
PACF	Cuts off after lag $p$	Tails off	Tails off

# Egyptian exports data

```
egypt %>%
  model(ARIMA(Exports ~ pdq(4, 0, 0))) %>%
  report()
```

```
## Series: Exports
## Model: ARIMA(4,0,0) w/ mean
##
## Coefficients:
##             ar1      ar2      ar3      ar4  constant
##             0.986   -0.172   0.181   -0.328     6.692
## s.e.    0.125    0.186   0.186    0.127     0.356
##
## sigma^2 estimated as 7.885: log likelihood=-141
## AIC=293    AICc=295    BIC=305
```

## Information theoretic criteria

### Akaike's Information Criterion (AIC)

$$\text{AIC} = -2\log(L) + 2(p + q + k + 1),$$

### Corrected AIC

$$\text{AICc} = \text{AIC} + \frac{2(p + q + k + 1)(p + q + k + 2)}{T - p - q - k - 2},$$

### Bayesian Information Criterion

$$\text{BIC} = \text{AIC} + [\log(T) - 2](p + q + k + 1),$$

where  $L$  is the likelihood of the data,  $k = 1$  if  $c \neq 0$  and  $k = 0$  if  $c = 0$ .

- We select the best models by minimizing the AIC, AICc or BIC. We would prefer to use AICc.
- These information criteria are not appropriate for selecting the order of differencing ( $d$ ) of a model.

## Parameter estimation

Once the model order has been identified, we need to estimate the parameters  $c, \phi_1, \phi_2, \dots, \phi_p, \theta_1, \theta_2, \dots, \theta_q$ .

- We use maximum likelihood estimation (MLE).
  - ▶ i.e., it finds the values of the parameters which maximizes the probability of observing the data from the specified model.
- MLE is similar to the least squares estimation obtained by minimizing
$$\sum_{t=1}^T \varepsilon_t^2.$$
- The ARIMA() function also allows CSS, MLE, or CSS-MLE.
- Non-linear optimization must be used in all the cases.
- Different software will give slightly different estimates.

# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

## Stepwise algorithm in ARIMA()

For non-seasonal ARIMA models, we need to estimate  $p$ ,  $d$ ,  $q$  and  $c$ .

The fable package uses a variation of the Hyndman and Khandakar (2008) algorithm.

- 1 Select  $0 \leq d \leq 2$  using repeated KPSS test.
- 2 Select  $p$ ,  $q$  and  $c$  by minimizing AICc.

$$\text{AICc} = -2\log(L) + 2(p + q + k + 1) \left[ 1 + \frac{p + q + k + 2}{T - p - q - k - 2} \right],$$

where  $L$  is the maximized likelihood fitted to the differenced data,  $k = 1$ , if  $c \neq 0$  and  $k = 0$ , otherwise.

- Rather than considering every possible combination of  $p$ ,  $q$  and  $c$ , the algorithm uses a stepwise search to traverse the model space.

# Stepwise algorithm in ARIMA()

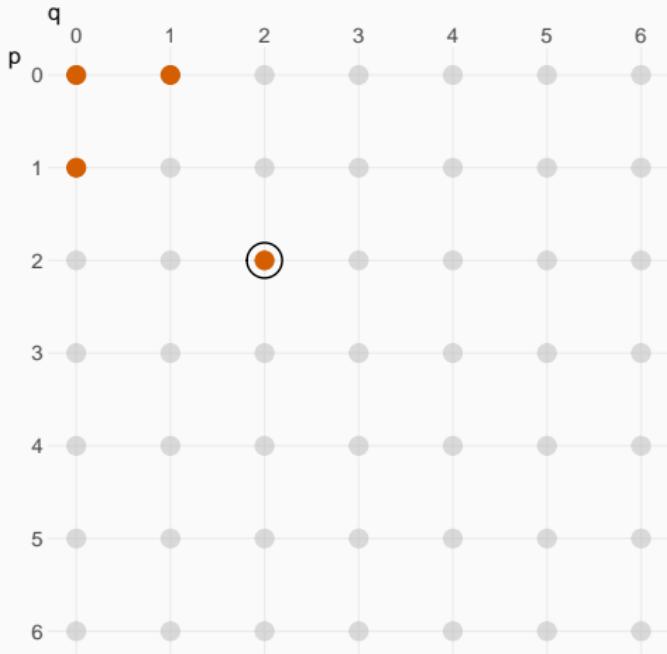
## Step 1

Four initial models are fitted.

- ARIMA(0, $d$ ,0)
- ARIMA(2, $d$ ,2)
- ARIMA(1, $d$ ,0)
- ARIMA(0, $d$ ,1)

A constant is included unless  $d = 2$ .

Select the one with smallest AICc.



Picture credit: Rob J Hyndman

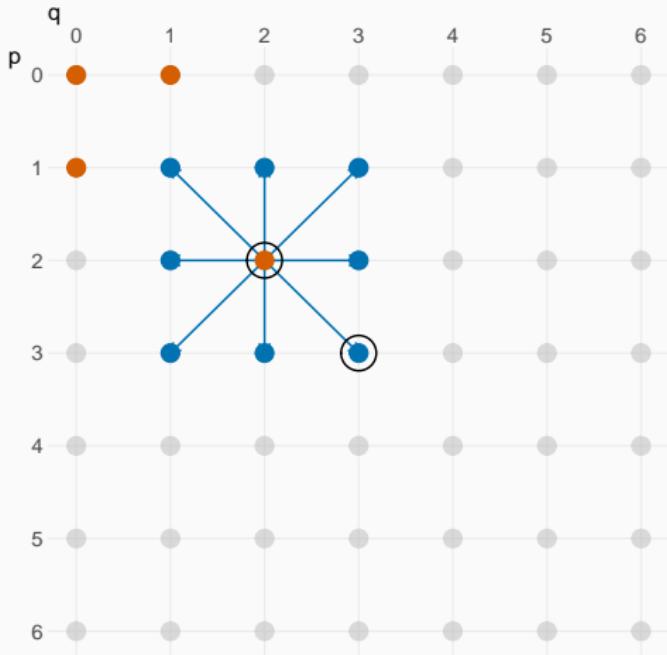
# Stepwise algorithm in ARIMA()

## Step 2

Define the neighborhood of the current model

- Vary  $p$ ,  $q$  and  $c$  such that the Euclidean distance from the current model is  $\leq \sqrt{2}$ .

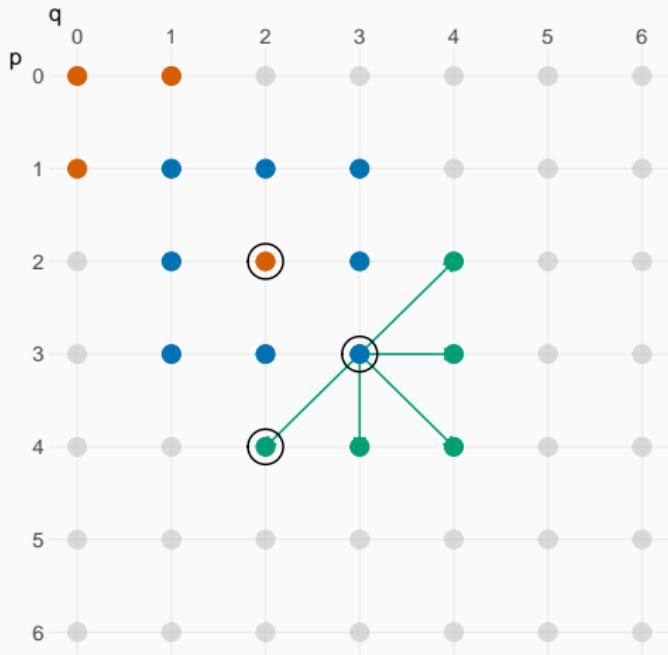
Select the model with smallest AICc.



Picture credit: Rob J Hyndman

# Stepwise algorithm in ARIMA()

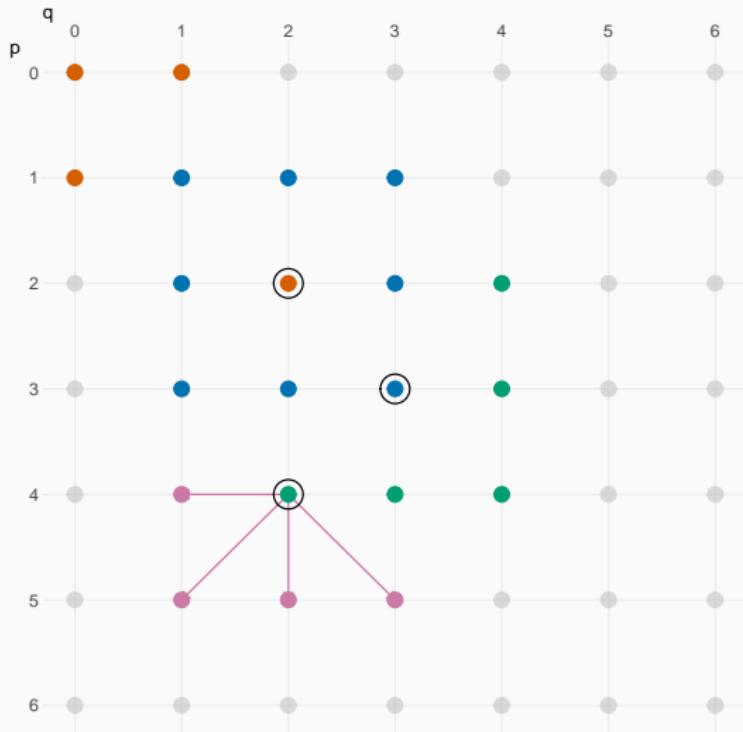
## Step 3



Repeat step 2 until no lower AICc can be found.

Picture credit: Rob J Hyndman

## Stepwise algorithm in ARIMA()



Picture credit: Rob J Hyndman

## Default behaviour of ARIMA()

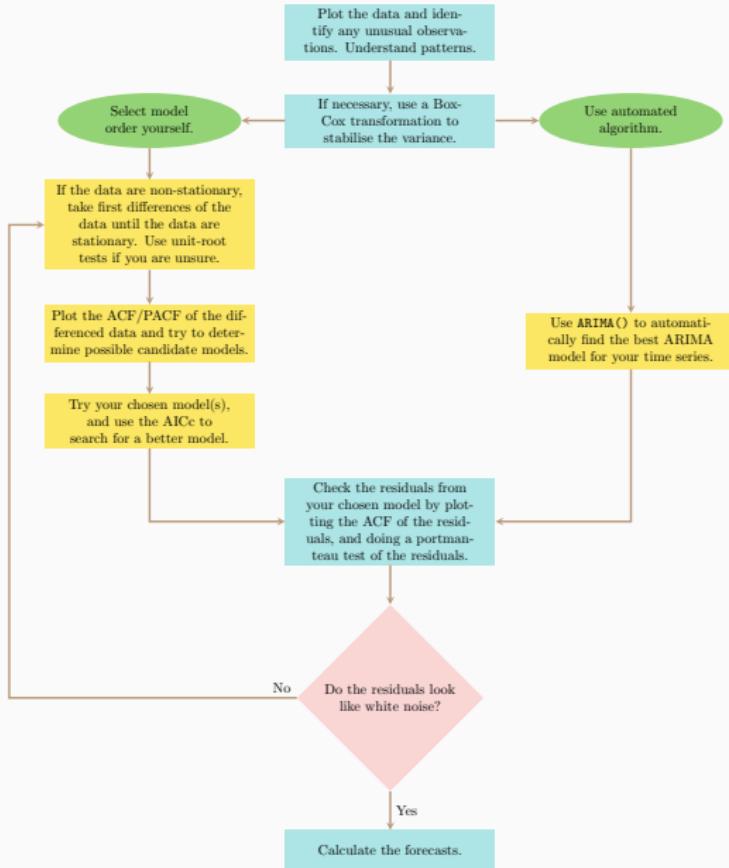
The default procedure uses some approximations to speed up the search.

The function sets

- `stepwise` = TRUE: Performs previously described stepwise algorithm.
- `greedy` = TRUE: Stepwise search moves to the next best option immediately.
- `approximation` = TRUE: If  $T > 150$  or  $m > 12$ , CSS will be used during model selection. The final model is estimated using maximum likelihood estimation.

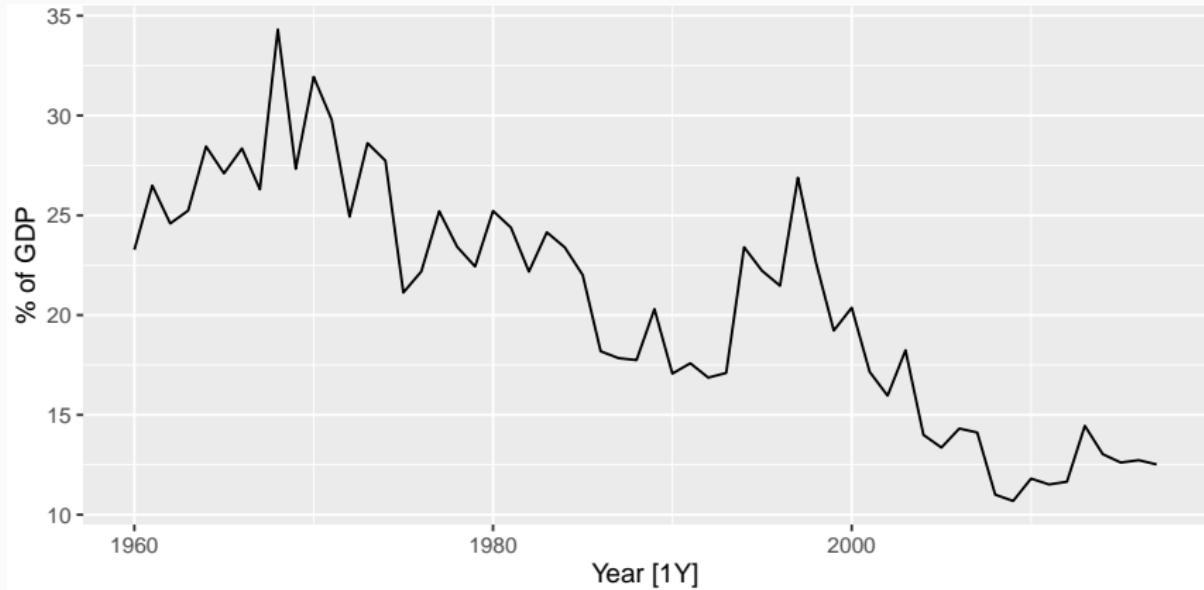
A much larger set of models will be searched if we set `stepwise` = FALSE.

# Modelling procedure with ARIMA models



## Central African Republic exports data

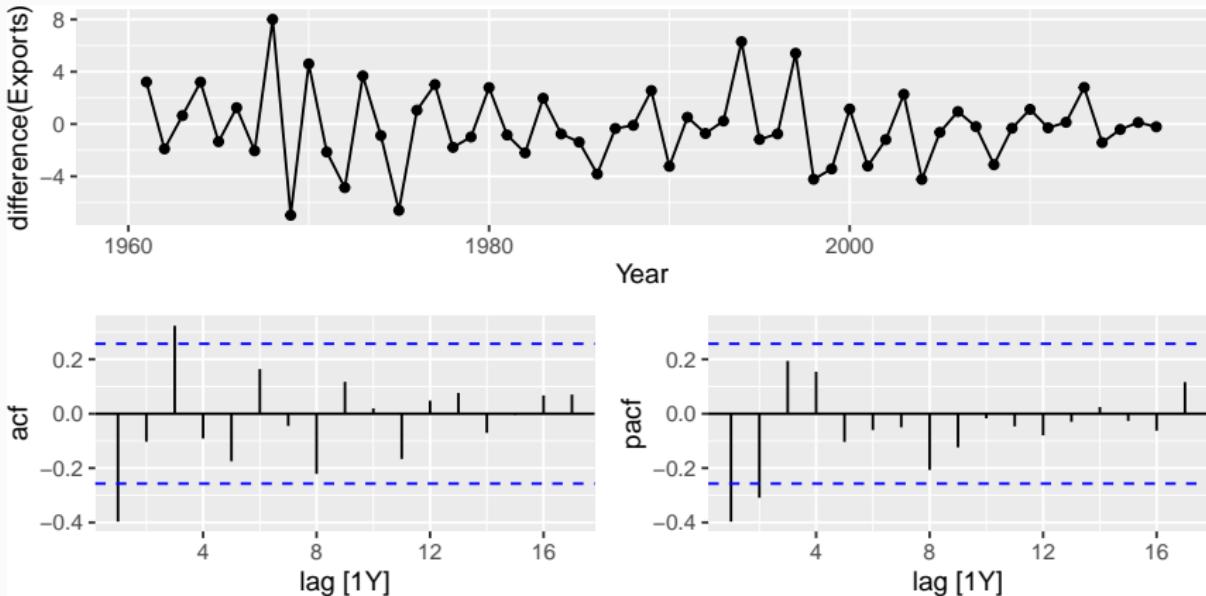
```
global_economy %>% filter(Code == "CAF") %>%  
  autoplot(Exports) + ylab("% of GDP")
```



- An overall downward trend—non-stationary time series.
- No evidence of changing variance.

# Central African Republic exports data

```
global_economy %>% filter(Code == "CAF") %>%  
  gg_tsdisplay(difference(Exports), plot_type = "partial")
```



Candidate models: ARIMA(0,1,3) or ARIMA(2,1,0).

# Central African Republic exports data

```
fit <- global_economy %>%
  filter(Code == "CAF") %>%
  model(arima013 = ARIMA(Exports ~ pdq(0,1,3)),
        arima210 = ARIMA(Exports ~ pdq(2,1,0)),
        stepwise = ARIMA(Exports),
        search = ARIMA(Exports, stepwise = FALSE))
fit %>% pivot_longer(-Country, names_to = "Model name",
                      values_to = "Order")
```

```
## # A mable: 4 x 3
## # Key:   Country, Model name [4]
##   Country           `Model name`      Order
##   <fct>             <chr>          <model>
## 1 Central African Republic arima013    <ARIMA(0,1,3)>
## 2 Central African Republic arima210    <ARIMA(2,1,0)>
## 3 Central African Republic stepwise    <ARIMA(2,1,2)>
## 4 Central African Republic search     <ARIMA(3,1,0)>
```

# Central African Republic exports data

```
glance(fit) %>% arrange(AICc) %>% select(.model:BIC)
```

```
## # A tibble: 4 x 6
##   .model    sigma2 log_lik     AIC     AICc     BIC
##   <chr>     <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1 search     6.52   -133.   274.   275.   282.
## 2 arima210   6.71   -134.   275.   275.   281.
## 3 arima013   6.54   -133.   274.   275.   282.
## 4 stepwise   6.42   -132.   274.   275.   284.
```

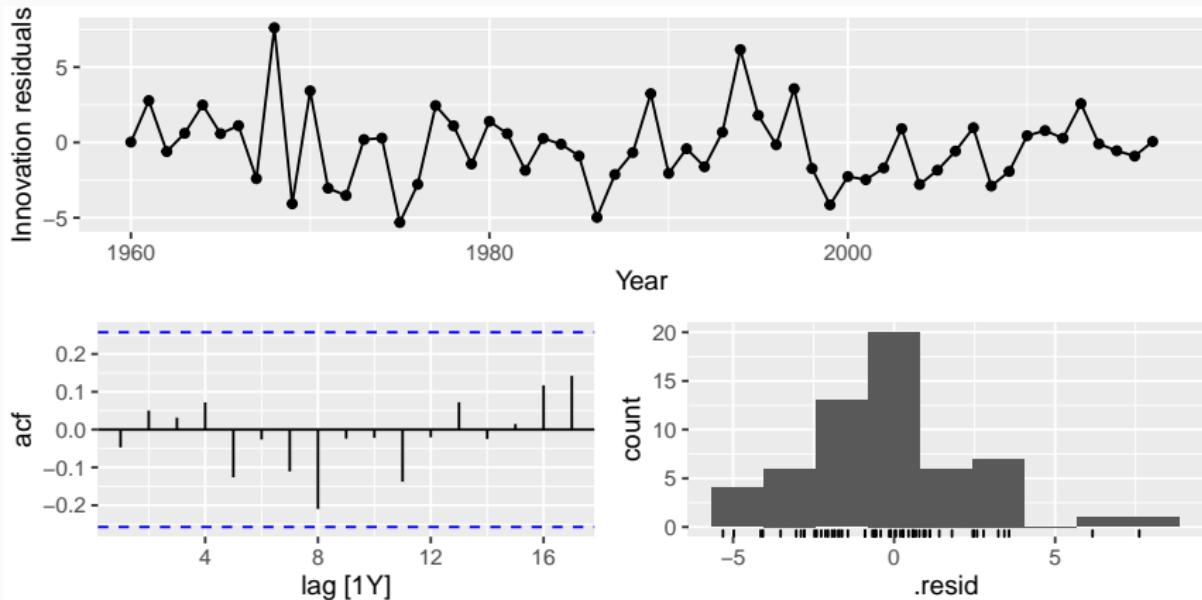
# Central African Republic exports data

```
fit %>%  
  select(search) %>%  
  report()
```

```
## Series: Exports  
## Model: ARIMA(3,1,0)  
##  
## Coefficients:  
##          ar1      ar2      ar3  
##        -0.442   -0.185   0.205  
## s.e.    0.130    0.139   0.127  
##  
## sigma^2 estimated as 6.519:  log likelihood=-133  
## AIC=274    AICc=275    BIC=282
```

# Central African Republic exports data

```
fit %>% select(search) %>%  
  gg_tsresiduals()
```



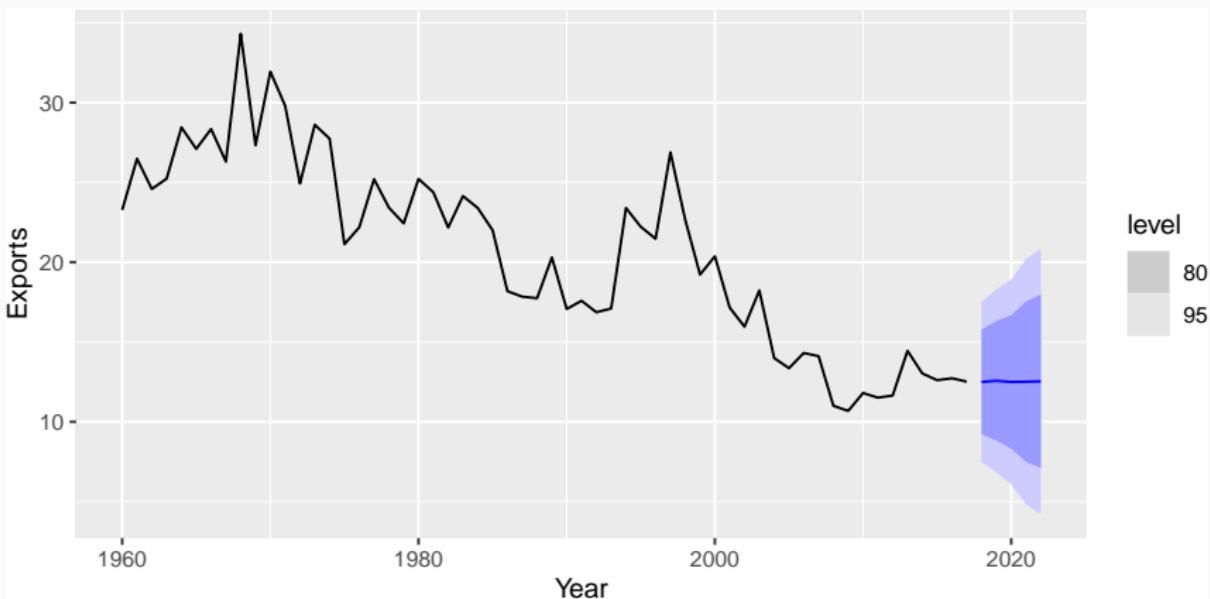
## Central African Republic exports data

```
fit %>% select(search) %>%
  augment() %>%
  features(.resid, ljung_box, lag = 10, dof = 3)
```

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 search    5.75     0.569
```

## Central African Republic exports data

```
fit %>% select(Country, search) %>%  
forecast(h = 5) %>% autoplot(global_economy)
```



The mean point forecasts are similar to what we would get with a random walk model.

## Understanding constants in R

### Intercept form

$$(1 - \phi_1 B - \cdots - \phi_p B^p)(1 - B)^d y_t = c + (1 + \theta_1 B + \cdots + \theta_q B^q) \varepsilon_t.$$

### Mean form

$$(1 - \phi_1 B - \cdots - \phi_p B^p)(1 - B)^d (y_t - \mu t^d / d!) = (1 + \theta_1 B + \cdots + \theta_q B^q) \varepsilon_t,$$

where  $\mu$  is the mean of  $(1 - B)^d y_t$  and  $c = \mu(1 - \phi_1 - \cdots - \phi_p)$ .

- The inclusion of a constant in a non-stationary ARIMA model is equivalent to inducing a polynomial trend of order  $d$  in the forecasts.
- If the constant is omitted, the forecasts include a polynomial trend of order  $d - 1$ .
- If  $d = 0$ ,  $\mu$  is the mean of  $y_t$ .

## Understanding constants in R

- By default, ARIMA() will automatically determine if a constant should be included.
- For  $d = 0$  or  $d = 1$ , a constant will be included if it improves the AICc value.
- If  $d > 1$  the constant is always omitted as it considers a quadratic or higher order trend.
- The constant can be specified manually by including 0 or 1 in the model formula.
  - ▶ To include a constant: ARIMA(y ~ 1 + ...).
  - ▶ To exclude a constant: ARIMA(y ~ 0 + ...).

# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

## Calculation of point forecasts

- 1 Expand the ARIMA equation and rewrite it so  $y_t$  is on LHS.
- 2 Rewrite the equation by replacing  $t$  by  $T + h$ .
- 3 On RHS, replace future observations by their forecasts, future errors by zero, and past errors by the corresponding residuals.

Start with  $h = 1$  and repeat these steps for  $h = 2, 3, \dots$

## Example: Step 1

Consider ARIMA(3,1,1) model without a constant.

$$(1 - \hat{\phi}_1 B - \hat{\phi}_2 B^2 - \hat{\phi}_3 B^3)(1 - B)y_t = (1 + \hat{\theta}_1 B)\varepsilon_t.$$

$$[1 - (1 + \hat{\phi}_1)B + (\hat{\phi}_1 - \hat{\phi}_2)B^2 + (\hat{\phi}_2 - \hat{\phi}_3)B^3 + \hat{\phi}_3 B^4]y_t = \\ (1 + \hat{\theta}_1 B)\varepsilon_t$$

$$y_t - (1 + \hat{\phi}_1)y_{t-1} + (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} + (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} + \\ \hat{\phi}_3 y_{t-4} = \varepsilon_t + \hat{\theta}_1 \varepsilon_{t-1}$$

$$y_t = (1 + \hat{\phi}_1)y_{t-1} - (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} - (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} - \\ \hat{\phi}_3 y_{t-4} + \varepsilon_t + \hat{\theta}_1 \varepsilon_{t-1}$$

## Example: Steps 2 and 3

$$y_t = (1 + \hat{\phi}_1)y_{t-1} - (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} - (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} - \hat{\phi}_3 y_{t-4} + \varepsilon_t + \hat{\theta}_1 \varepsilon_{t-1}$$

### Step 2

$$y_{T+1} = (1 + \hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \varepsilon_{T+1} + \hat{\theta}_1 \varepsilon_T$$

### Step 3

$$\hat{y}_{T+1|T} = (1 + \hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \hat{\theta}_1 \hat{\varepsilon}_T$$

## Example: 2-step-ahead point forecasts

$$y_t = (1 + \hat{\phi}_1)y_{t-1} - (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} - (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} - \\ \hat{\phi}_3 y_{t-4} + \varepsilon_t + \hat{\theta}_1 \varepsilon_{t-1}$$

### Step 2

$$y_{T+2} = (1 + \hat{\phi}_1)y_{T+1} - (\hat{\phi}_1 - \hat{\phi}_2)y_T - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-1} - \\ \hat{\phi}_3 y_{T-2} + \varepsilon_{T+2} + \hat{\theta}_1 \varepsilon_{T+1}$$

### Step 3

$$\hat{y}_{T+2|T} = (1 + \hat{\phi}_1)\hat{y}_{T+1|T} - (\hat{\phi}_1 - \hat{\phi}_2)y_T - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-1} - \\ \hat{\phi}_3 y_{T-2}$$

## Prediction intervals

### 95% prediction interval

$$\hat{y}_{T+h|T} \pm 1.96 \hat{\sigma}_h$$

where  $\hat{\sigma}_h^2$  is the estimated forecast variance.

- $\hat{\sigma}_1^2 = \hat{\sigma}^2$  for all ARIMA models, where  $\hat{\sigma}^2$  is the variance of the residuals.
- Multi-step prediction intervals for MA( $q$ ):

$$y_t = \varepsilon_t + \sum_{j=1}^q \theta_j \varepsilon_{t-j}.$$

$$\hat{\sigma}_h^2 = \hat{\sigma}^2 \left[ 1 + \sum_{j=1}^{h-1} \hat{\theta}_j^2 \right], \quad \text{for } j = 2, 3, \dots$$

- For AR(1), use its MA( $\infty$ ) model and apply the above formula.
- Other models are hard to calculate manually and beyond the scope of this course.

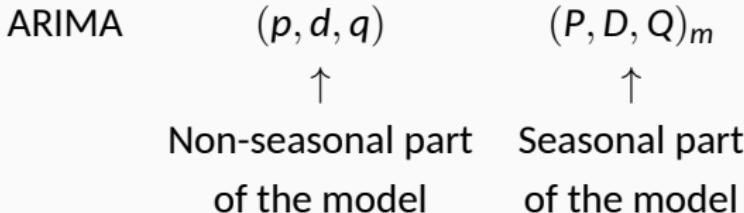
## Prediction intervals (PI)

- Generally PIs from ARIMA models increase as the forecast horizon increases.
  - ▶ For stationary models (i.e.  $d = 0$ ), the PIs for longer horizons are the same.
  - ▶ For  $d \geq 1$ , PIs will continue to grow into the future.
- Calculations assume residuals are uncorrelated and normally distributed.
  - ▶ If normality assumption is highly violated we can set `bootstrap = TRUE` in the `forecast` function.
- PIs tend to be too narrow.
  - ▶ the uncertainty in the parameters and model orders has not been taken into account.
  - ▶ the calculations assumes that the historical patterns that have been modelled will continue into the forecast period.
  - ▶ ARIMA model assumes that the future errors are uncorrelated.

# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

## Seasonal ARIMA models



$m$ : seasonal period.

Define

$$\Delta^d y_t = (1 - B)^d y_t, \quad \Delta_m^D y_t = (1 - B^m)^D y_t,$$

$$\phi(B) = 1 - \phi_1 B - \phi_2 B^2 - \cdots - \phi_p B^p,$$

$$\theta(B) = 1 + \theta_1 B + \theta_2 B^2 + \cdots + \theta_q B^q,$$

$$\Phi_P(B^m) = 1 - \Phi_1 B^m - \Phi_2 B^{2m} - \cdots - \Phi_P B^{Pm},$$

$$\Theta_Q(B^m) = 1 + \Theta_1 B^m + \Theta_2 B^{2m} + \cdots + \Theta_Q B^{Qm}.$$

$$\Phi_P(B^m) \phi(B) \Delta_m^D \Delta^d y_t = c + \Theta_Q(B^m) \theta(B) \varepsilon_t.$$

## Seasonal ARIMA models

e.g. ARIMA(1,1,1)(1,1,1)<sub>4</sub> model without a constant for quarterly data:

$$(1 - \Phi_1 B^4)(1 - \phi_1 B)(1 - B^4)(1 - B)y_t = (1 + \Theta_1 B^4)(1 + \theta_1 B)\varepsilon_t.$$

The factors can be multiplied out to get the general model:

$$\begin{aligned}y_t = & (1 + \phi_1)y_{t-1} - \phi_1 y_{t-2} + (1 + \Phi_1)y_{t-4} - \\& (1 + \phi_1 + \Phi_1 + \phi_1\Phi_1)y_{t-5} + \phi_1(1 + \Phi_1)y_{t-6} - \\& \Phi_1 y_{t-8} + \Phi_1(1 + \phi_1)y_{t-9} - \phi_1\Phi_1 y_{t-10} + \\& \varepsilon_t + \theta_1\varepsilon_{t-1} + \Theta_1\varepsilon_{t-4} + \theta_1\Theta_1\varepsilon_{t-5}.\end{aligned}$$

## Behaviour of ACF and PACF for pure seasonal ARMA models

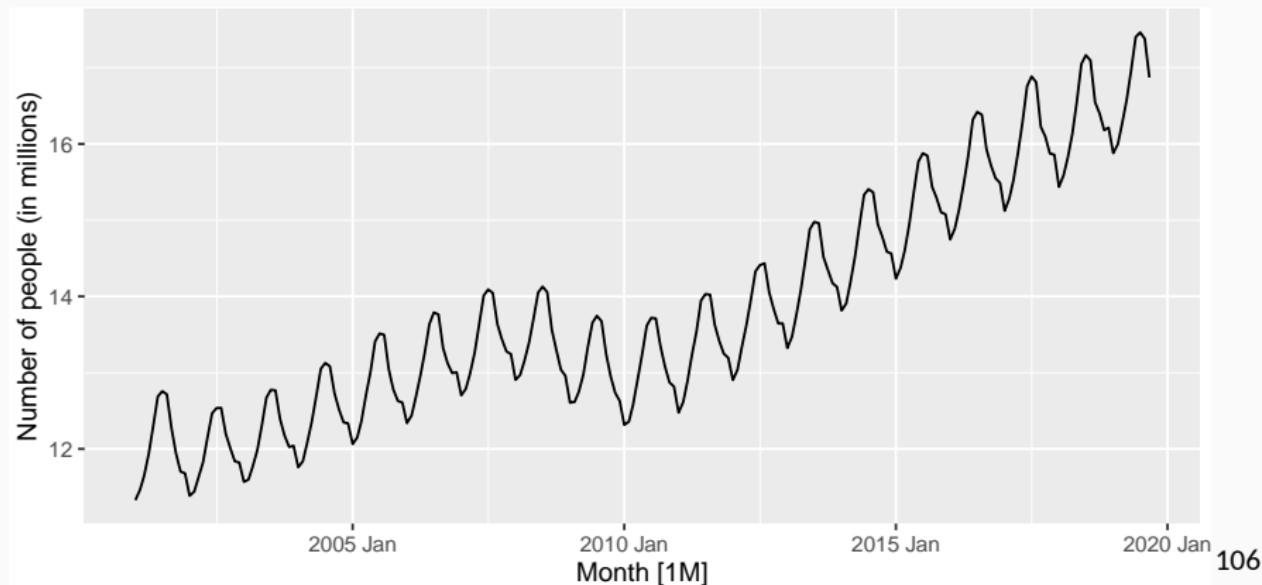
Seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF.

	$AR(P)_m$	$MA(Q)_m$	$ARMA(P, Q)_m$
ACF	Tails off at lags $km$ $k = 1, 2, \dots,$	Cuts off after lag $Qm$	Tails off at lags $km$
PACF	Cuts off after lag $Pm$	Tails off at lags $km$ $k = 1, 2, \dots,$	Tails off at lags $km$

- For seasonal ARIMA models, we need to select both seasonal and non-seasonal orders.
- To select appropriate seasonal orders, focus on the seasonal lags.

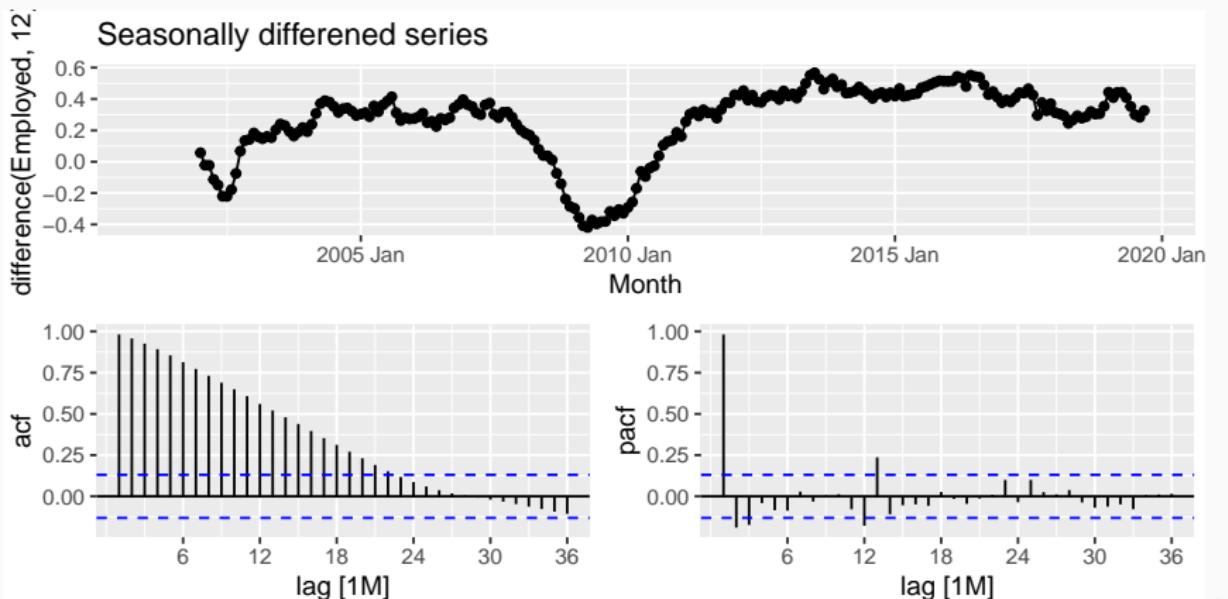
# US leisure and hospitality employment

```
leisure <- us_employment %>%
  filter>Title == "Leisure and Hospitality", year(Month) > 2000) %>%
  mutate(Employed = Employed / 1e3) %>%
  select(Month, Employed)
autoplot(leisure) + ylab("Number of people (in millions)")
```



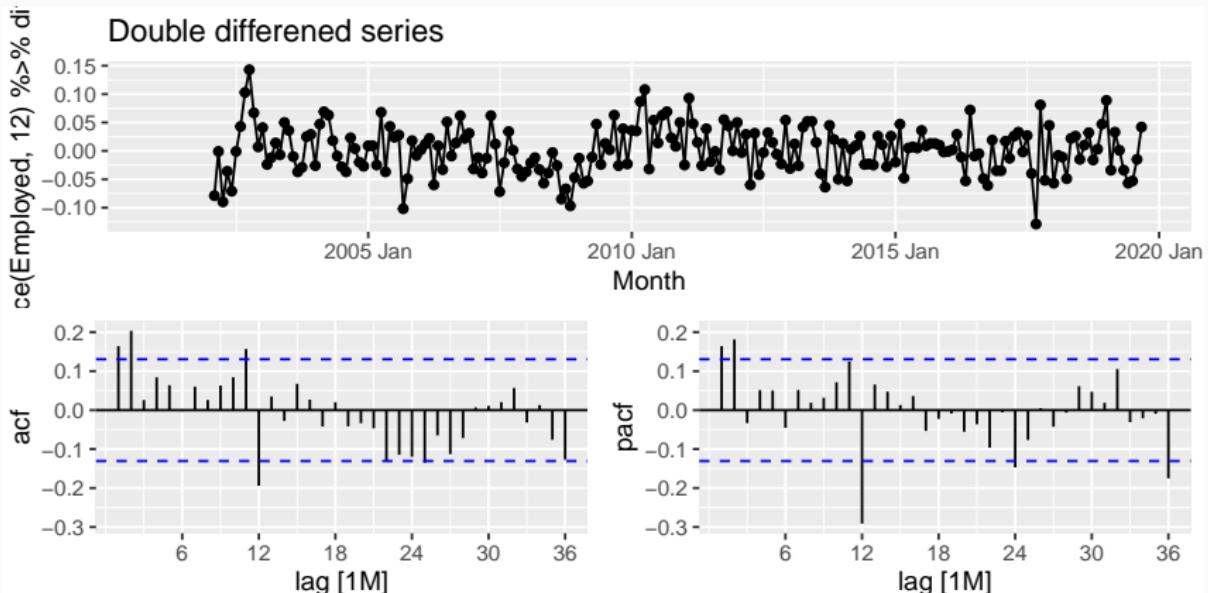
# US leisure and hospitality employment

```
leisure %>%  
  gg_tsdisplay(difference(Employed, 12),  
               plot_type = "partial", lag_max = 36) +  
  labs(title = "Seasonally differenced series")
```



# US leisure and hospitality employment

```
leisure %>%  
  gg_tsdisplay(difference(Employed, 12) %>% difference(),  
               plot_type = "partial", lag_max = 36) +  
  labs(title = "Double differenced series")
```



# US leisure and hospitality employment

```
fit <- leisure %>%
  model(arima012011 = ARIMA(Employed ~ pdq(0,1,2) + PDQ(0,1,1)),
        arima210011 = ARIMA(Employed ~ pdq(2,1,0) + PDQ(0,1,1)),
        auto = ARIMA(Employed, stepwise = FALSE, approximation = FALSE))
fit %>% pivot_longer(everything(), names_to = "Model_name",
                      values_to = "Orders")
```

```
## # A mable: 3 x 2
## # Key:   Model_name [3]
##   Model_name          Orders
##   <chr>                <model>
## 1 arima012011 <ARIMA(0,1,2)(0,1,1)[12]>
## 2 arima210011 <ARIMA(2,1,0)(0,1,1)[12]>
## 3 auto       <ARIMA(2,1,0)(1,1,1)[12]>
```

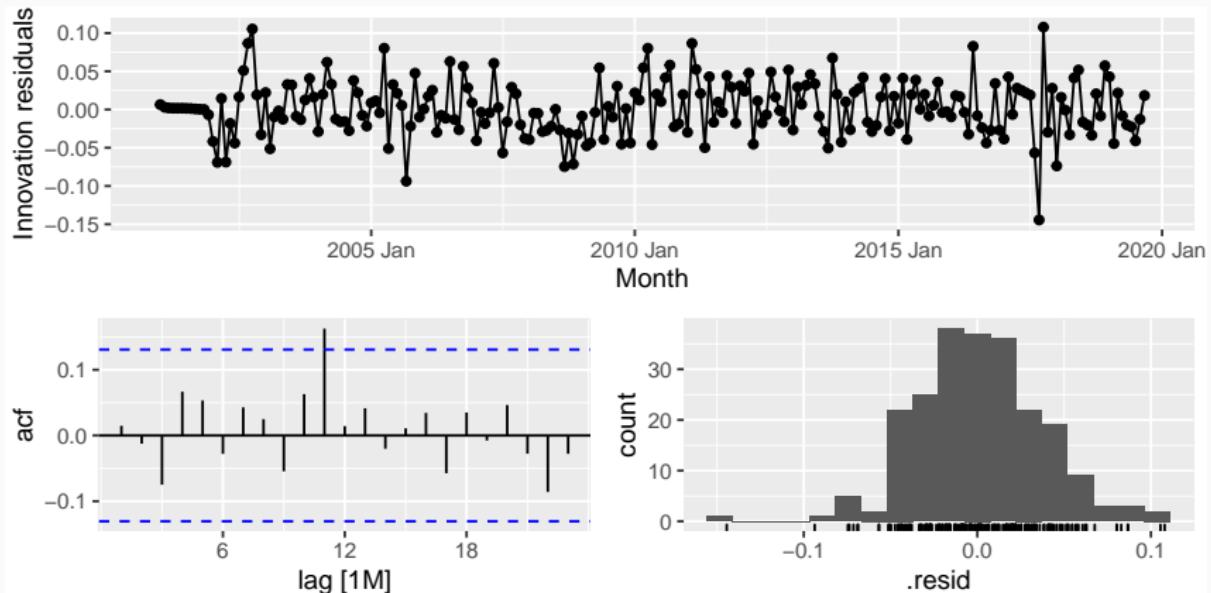
# US leisure and hospitality employment

```
glance(fit) %>% arrange(AICc) %>% select(.model:BIC)
```

```
## # A tibble: 3 x 6
##   .model      sigma2 log_lik    AIC    AICc    BIC
##   <chr>      <dbl>   <dbl>  <dbl>  <dbl>  <dbl>
## 1 auto       0.00142   395. -780. -780. -763.
## 2 arima210011 0.00145   392. -776. -776. -763.
## 3 arima012011 0.00146   391. -775. -775. -761.
```

# US leisure and hospitality employment

```
fit %>% select(auto) %>% gg_tsresiduals()
```



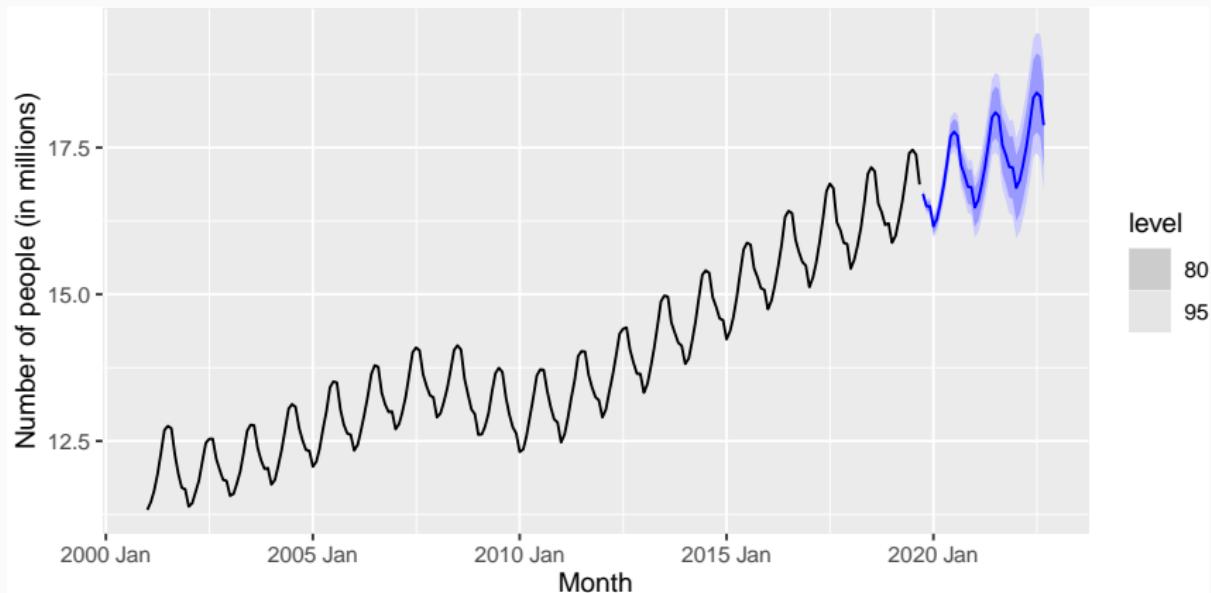
# US leisure and hospitality employment

```
fit %>% select(auto) %>%
  augment() %>%
  features(.resid, ljung_box, lag = 24, dof = 4)
```

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 auto      16.6     0.680
```

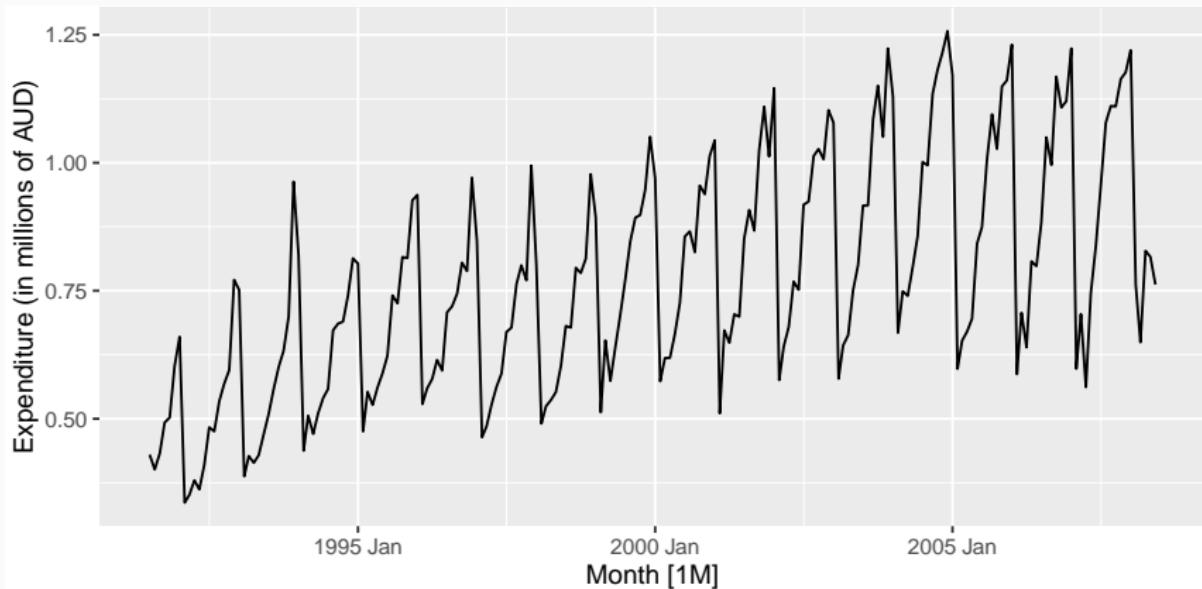
# US leisure and hospitality employment

```
fit %>% select(auto) %>%  
  forecast(h = "3 years") %>%  
  autoplot(leisure) + ylab("Number of people (in millions)")
```



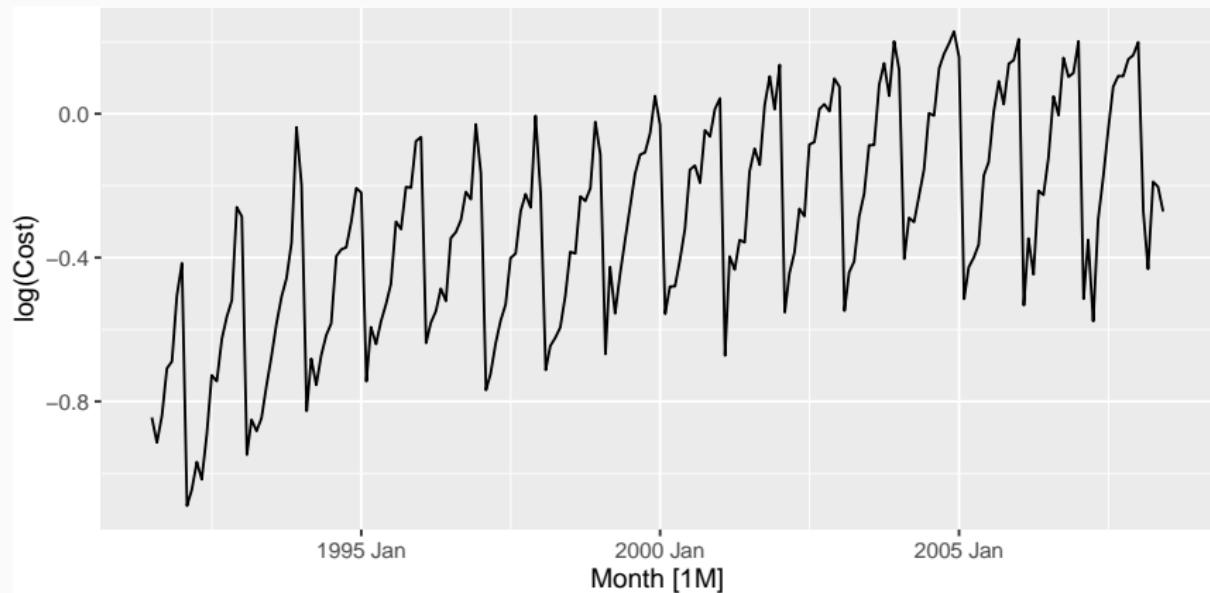
# Corticosteroid drug sales in Australia

```
h02 <- PBS %>%
  filter(ATC2 == "H02") %>% summarise(Cost = sum(Cost)/1e6)
h02 %>% autoplot(Cost) + ylab("Expenditure (in millions of AUD)")
```



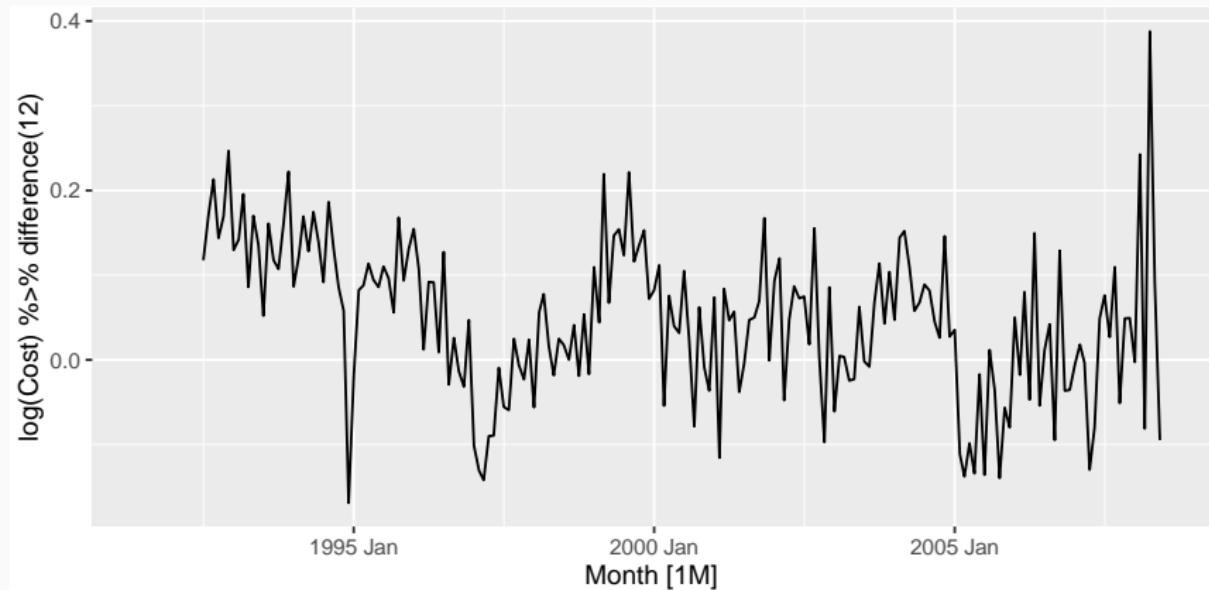
# Corticosteroid drug sales in Australia

```
h02 %>% autoplot(log(Cost))
```



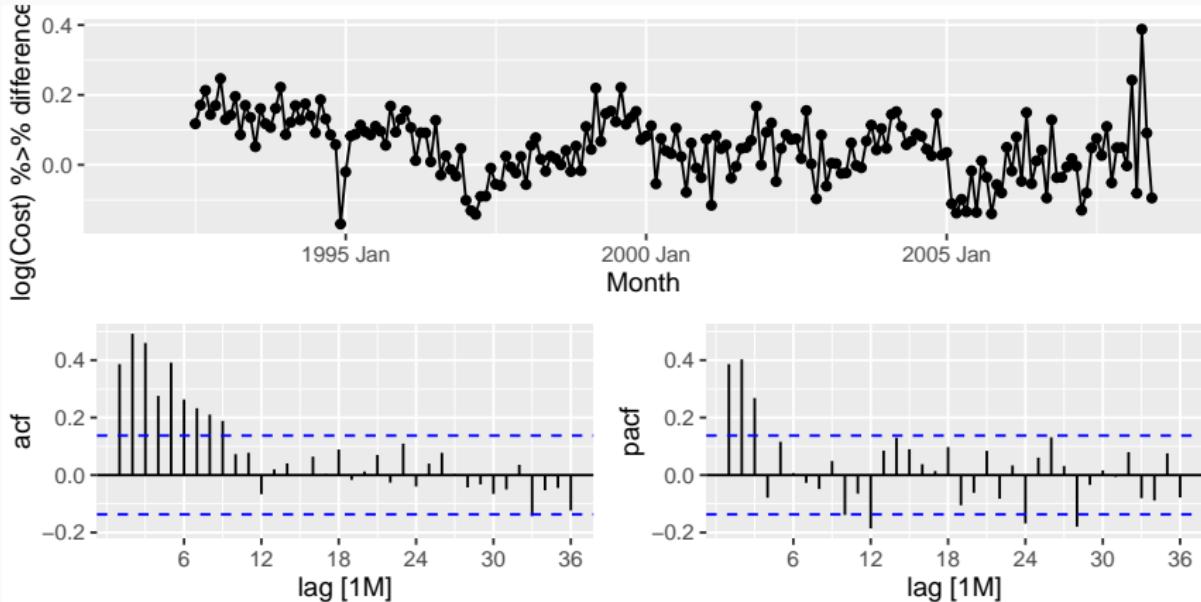
# Corticosteroid drug sales in Australia

```
h02 %>% autoplot(log(Cost) %>% difference(12))
```



# Corticosteroid drug sales in Australia

```
h02 %>% gg_tsdisplay(log(Cost) %>% difference(12),  
                      plot_type = "partial", lag_max = 36)
```



Initial candidate model: ARIMA(3,0,0)(2,1,0)<sub>12</sub>.

## Corticosteroid drug sales in Australia

.model	AICc
ARIMA(3,0,1)(0,1,2)[12]	-485
ARIMA(3,0,1)(1,1,1)[12]	-484
ARIMA(3,0,1)(0,1,1)[12]	-484
ARIMA(3,0,1)(2,1,0)[12]	-476
ARIMA(3,0,0)(2,1,0)[12]	-475
ARIMA(3,0,2)(2,1,0)[12]	-475
ARIMA(3,0,1)(1,1,0)[12]	-463

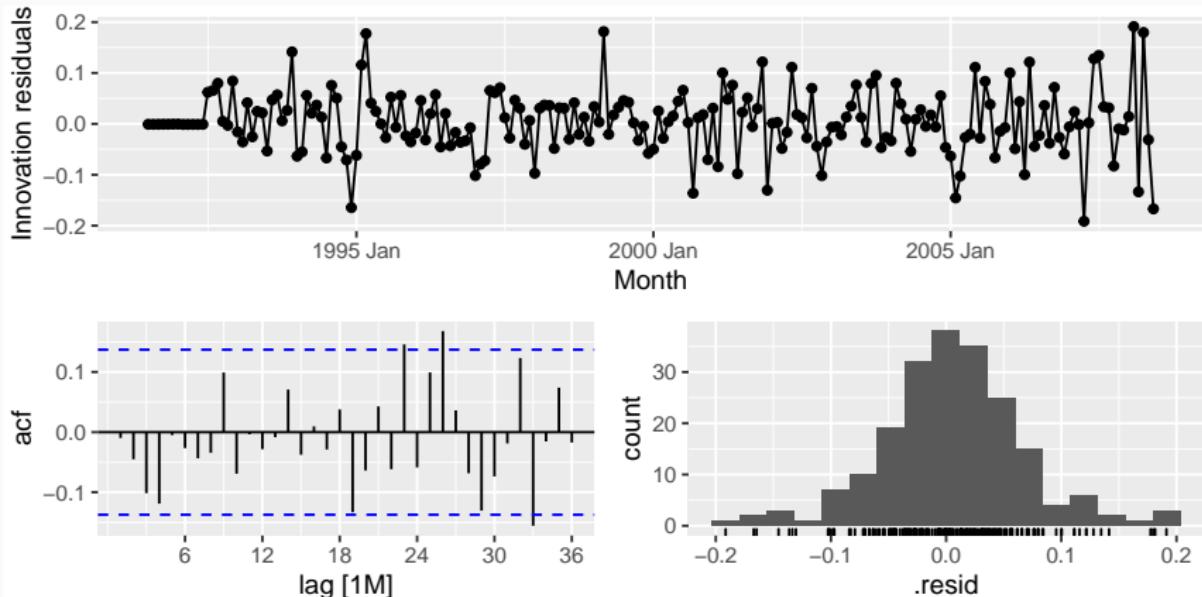
# Corticosteroid drug sales in Australia

```
fit <- h02 %>%
  model(best = ARIMA(log(Cost) ~ 0 + pdq(3,0,1) + PDQ(0,1,2)))
report(fit)
```

```
## Series: Cost
## Model: ARIMA(3,0,1)(0,1,2)[12]
## Transformation: log(Cost)
##
## Coefficients:
##          ar1      ar2      ar3      ma1      sma1      sma2
##          -0.160   0.5481   0.5678   0.383   -0.5222   -0.1768
##  s.e.    0.164   0.0878   0.0942   0.190    0.0861    0.0872
##
## sigma^2 estimated as 0.004278:  log likelihood=250
## AIC=-486    AICc=-485    BIC=-463
```

# Corticosteroid drug sales in Australia

```
gg_tsresiduals(fit, lag_max = 36)
```



## Corticosteroid drug sales in Australia

```
augment(fit) %>%
  features(.innov, ljung_box, lag = 36, dof = 6)
```

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 best      50.7     0.0104
```

- The model fails the Ljung-Box test.
- The model can still be useful for forecasting.
- The prediction intervals may not be accurate due to correlated residuals.

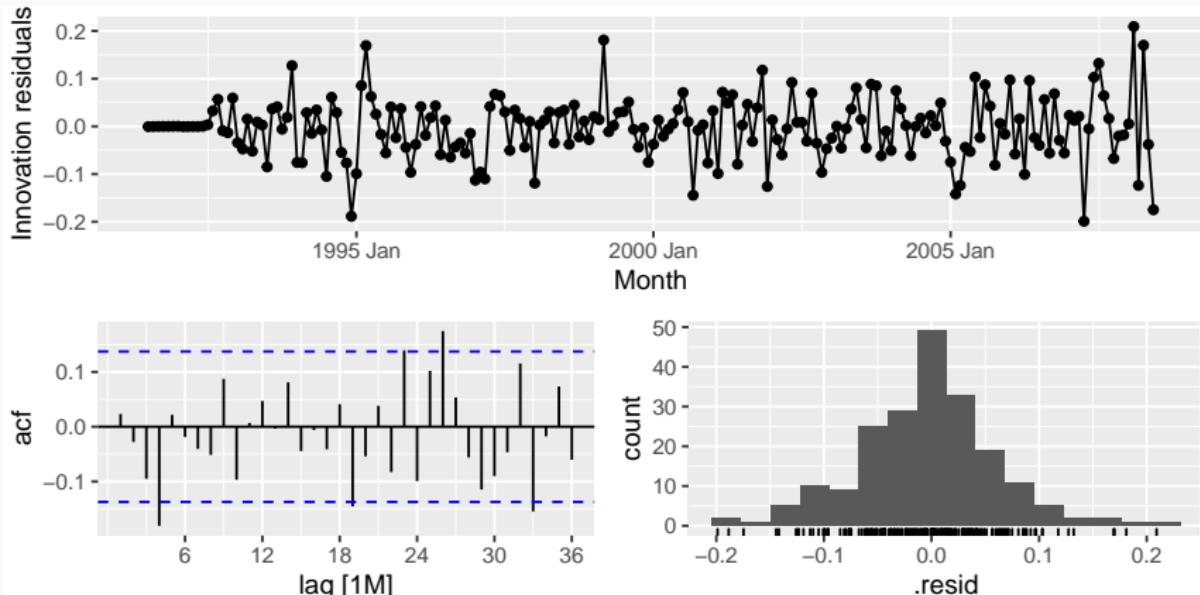
## Corticosteroid drug sales in Australia

```
fit <- h02 %>% model(auto = ARIMA(log(Cost)))
report(fit)

## Series: Cost
## Model: ARIMA(2,1,0)(0,1,1)[12]
## Transformation: log(Cost)
##
## Coefficients:
##             ar1      ar2      sma1
##             -0.8491  -0.4207  -0.6401
## s.e.    0.0712   0.0714   0.0694
##
## sigma^2 estimated as 0.004387:  log likelihood=245
## AIC=-483   AICc=-483   BIC=-470
```

# Corticosteroid drug sales in Australia

```
gg_tsresiduals(fit, lag_max = 36)
```



## Corticosteroid drug sales in Australia

```
augment(fit) %>%  
  features(.innov, ljung_box, lag = 36, dof = 3)
```

```
## # A tibble: 1 x 3  
##   .model lb_stat lb_pvalue  
##   <chr>     <dbl>      <dbl>  
## 1 auto      59.3    0.00332
```

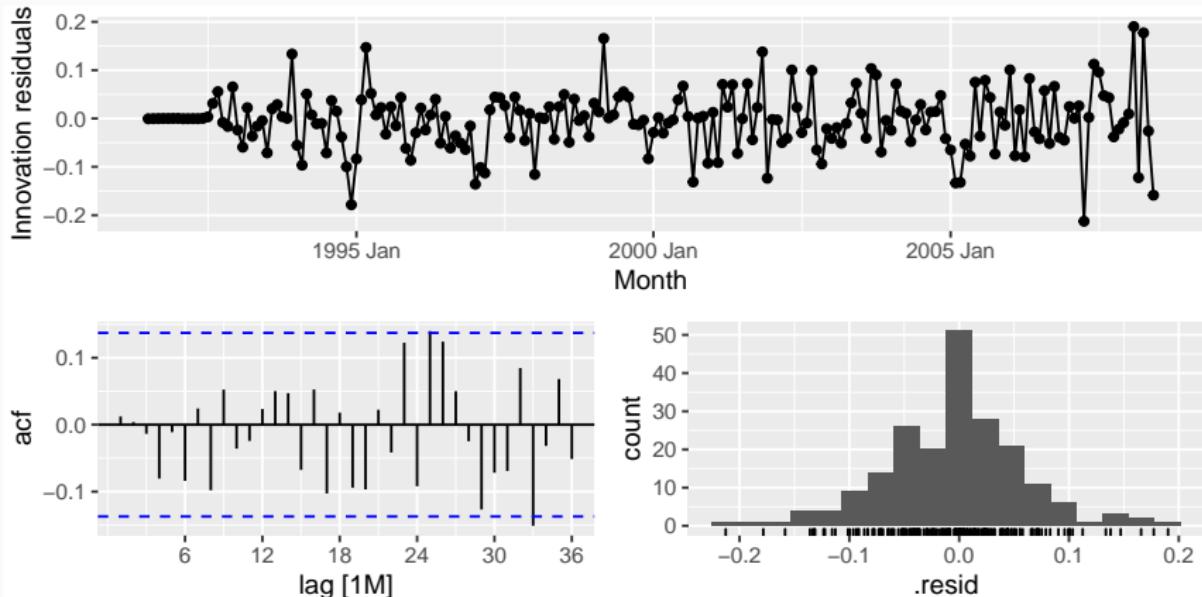
## Corticosteroid drug sales in Australia

```
fit <- h02 %>%
  model(best = ARIMA(log(Cost), stepwise = FALSE,
                     approximation = FALSE))
report(fit)

## Series: Cost
## Model: ARIMA(2,1,3)(0,1,1)[12]
## Transformation: log(Cost)
##
## Coefficients:
##             ar1      ar2      ma1      ma2      ma3      sma1
##             -1.019   -0.835   0.172   0.258   -0.421   -0.6528
## s.e.     0.165    0.120   0.208   0.118    0.106    0.0657
##
## sigma^2 estimated as 0.004203:  log likelihood=251
## AIC=-488    AICc=-487    BIC=-465
```

# Corticosteroid drug sales in Australia

```
gg_tsresiduals(fit, lag_max = 36)
```



# Corticosteroid drug sales in Australia

```
augment(fit) %>%  
  features(.innov, ljung_box, lag = 36, dof = 6)
```

```
## # A tibble: 1 x 3  
##   .model lb_stat lb_pvalue  
##   <chr>     <dbl>      <dbl>  
## 1 best       46.1      0.0301
```

## Corticosteroid drug sales in Australia

Training set: July 1991–June 2006. Test set: July 2006–June 2008 (2 years).

```
fit <- h02 %>%
  filter_index(~ "2006 June") %>%
  model(ARIMA(log(Cost) ~ 0 + pdq(3,0,0) + PDQ(2,1,0)),
        ARIMA(log(Cost) ~ 0 + pdq(3,0,1) + PDQ(2,1,0)),
        ARIMA(log(Cost) ~ 0 + pdq(3,0,2) + PDQ(2,1,0)),
        ARIMA(log(Cost) ~ 0 + pdq(3,0,1) + PDQ(1,1,0)),
        # ...
      )
fit %>%
  forecast(h = "2 years") %>%
  accuracy(h02)
```

## Corticosteroid drug sales in Australia

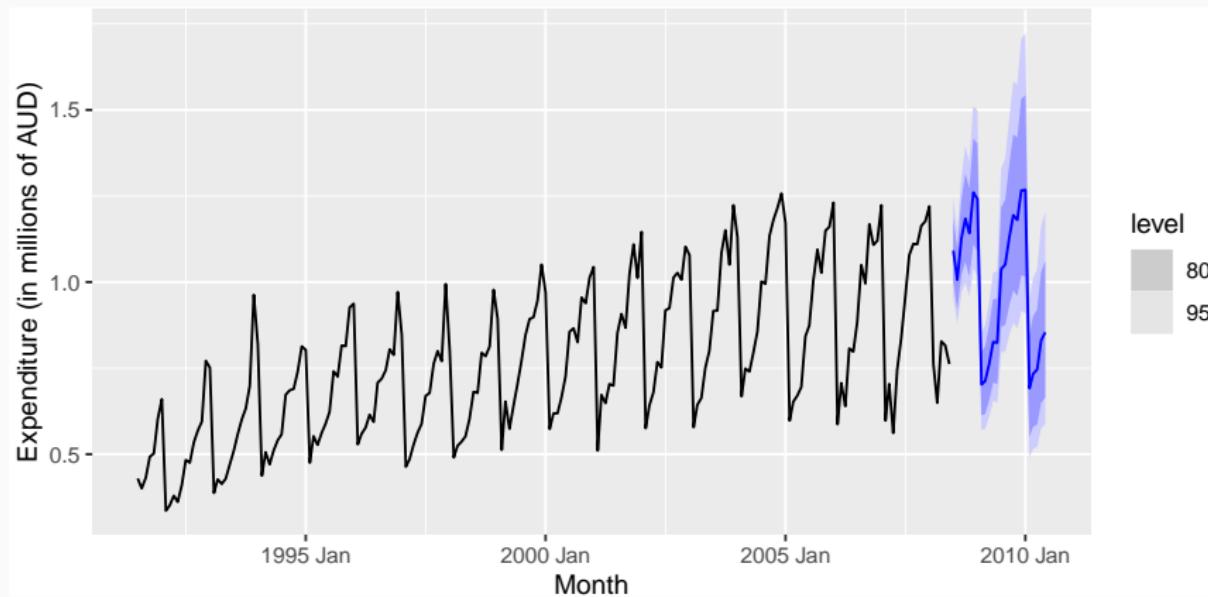
.model	RMSE
ARIMA(3,0,1)(1,1,1)[12]	0.0619
ARIMA(3,0,1)(0,1,2)[12]	0.0621
ARIMA(2,1,3)(0,1,1)[12]	0.0628
ARIMA(3,0,1)(0,1,1)[12]	0.0630
ARIMA(2,1,0)(0,1,1)[12]	0.0630
ARIMA(3,0,2)(2,1,0)[12]	0.0651
ARIMA(3,0,1)(2,1,0)[12]	0.0653
ARIMA(3,0,1)(1,1,0)[12]	0.0666
ARIMA(3,0,0)(2,1,0)[12]	0.0668

When comparing models using

- test set, it doesn't matter how the forecasts were produced.
- AICc, it matters the order of differencing used—the model must have same order of differencing.

# Corticosteroid drug sales in Australia

```
h02 %>%
  model(ARIMA(log(Cost) ~ 0 + pdq(3,0,1) + PDQ(0,1,2))) %>%
  forecast() %>%
  autoplot(h02) + ylab("Expenditure (in millions of AUD)")
```

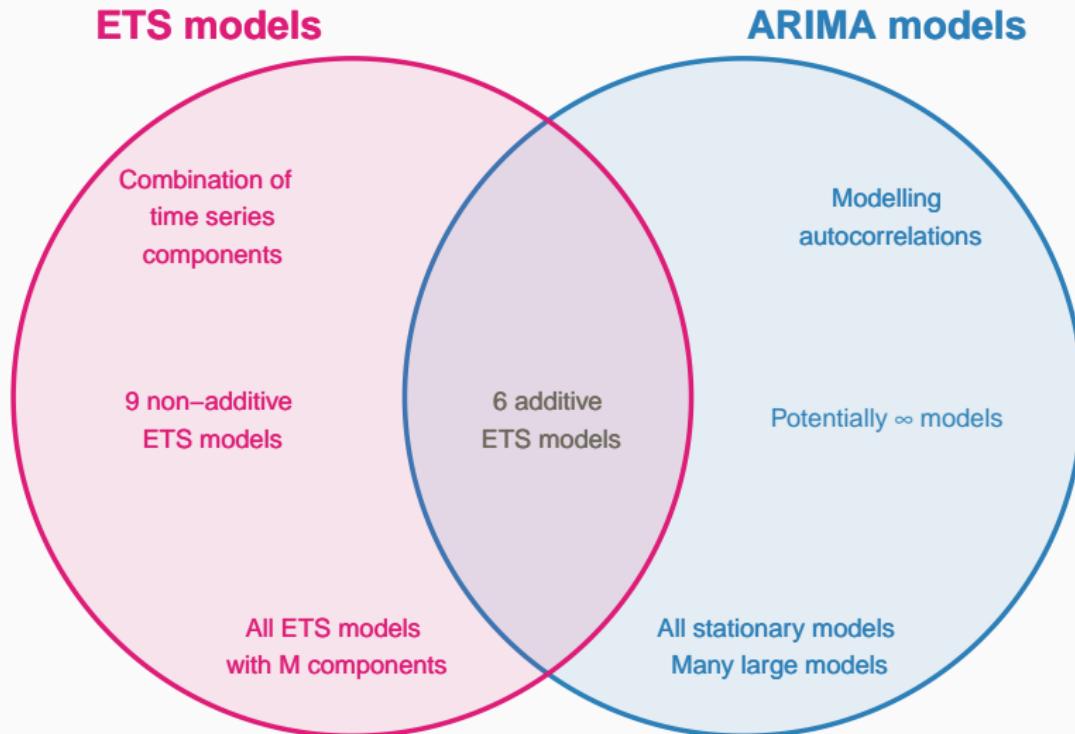


# Outline

- 1 Stationarity and differencing
- 2 Non-seasonal ARIMA models
- 3 Order selection and estimation
- 4 ARIMA modelling in R
- 5 Forecasting
- 6 Seasonal ARIMA models
- 7 ARIMA vs ETS models

- It is commonly held myth that ARIMA models are more general than ETS models.
- Linear ETS models are special cases of ARIMA models.
- Non-linear ETS models have no equivalent ARIMA counterparts.
- Many ARIMA models have no ETS counterparts.
- All ETS models are non-stationary.
- There are stationary ARIMA models.

# ARIMA vs ETS models



Picture credit: Rob J Hyndman

## Equivalence between ETS and ARIMA

ETS model	ARIMA model	Parameters
ETS(A,N,N)	ARIMA(0,1,1)	$\theta_1 = \alpha - 1$
ETS(A,A,N)	ARIMA(0,2,2)	$\theta_1 = \alpha + \beta - 2$
		$\theta_2 = 1 - \alpha$
ETS(A,A <sub>d</sub> ,N)	ARIMA(1,1,2)	$\phi_1 = \phi$
		$\theta_1 = \alpha + \phi\beta - 1 - \phi$
		$\theta_2 = (1 - \alpha)\phi$
ETS(A,N,A)	ARIMA(0,0,m)(0,1,0) <sub>m</sub>	
ETS(A,A,A)	ARIMA(0,1,m+1)(0,1,0) <sub>m</sub>	
ETS(A,A <sub>d</sub> ,A)	ARIMA(1,0,m+1)(0,1,0) <sub>m</sub>	

- ETS models with seasonality or non-damped trend (or both) have two unit roots.
- All other ETS models have one unit root.

- AICc is useful for selecting models in the same class.
- It cannot be used to compare ETS and ARIMA models.
  - ▶ The model classes are different;
  - ▶ The likelihood is computed differently.

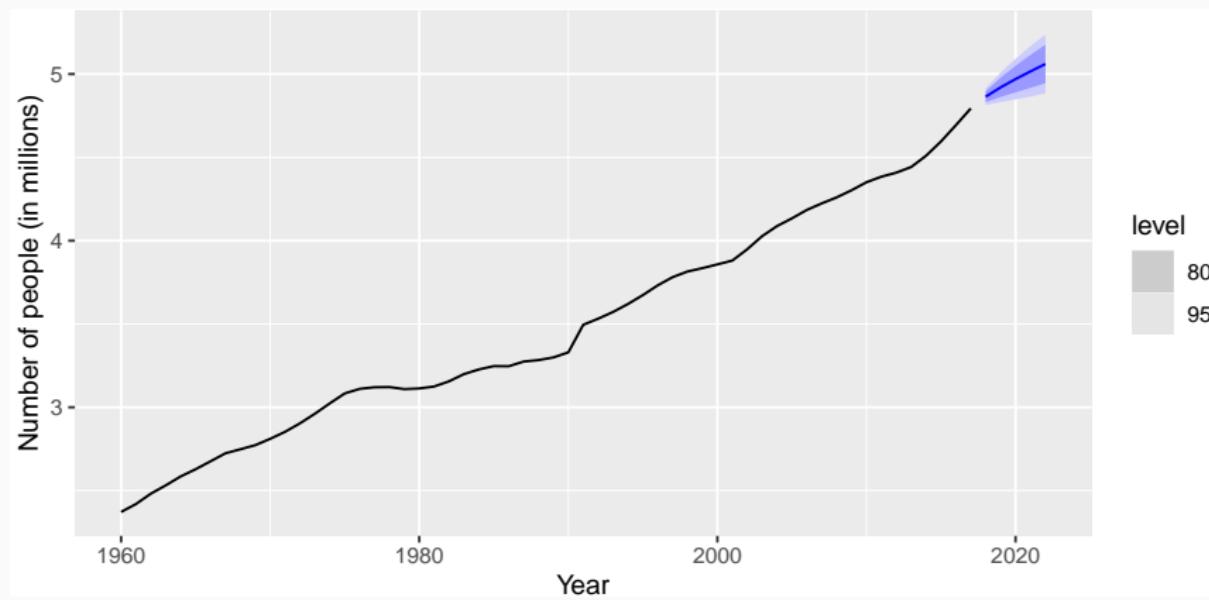
# NZ population data

```
nz_economy <- global_economy %>%  
  filter(Code == "NZL") %>%  
  mutate(Population = Population/1e6)  
  
nz_economy %>%  
  stretch_tsibble(.init = 10) %>%  
  model(ets = ETS(Population),  
         arima = ARIMA(Population)) %>%  
  forecast(h = 1) %>%  
  accuracy(nz_economy) %>%  
  select(.model, RMSE:MAPE)
```

```
## # A tibble: 2 x 5  
##   .model     RMSE     MAE     MPE    MAPE  
##   <chr>     <dbl>   <dbl>   <dbl>   <dbl>  
## 1 arima  0.0296  0.0191  0.118  0.533  
## 2 ets    0.0338  0.0221  0.0444  0.623
```

# NZ population data

```
nz_economy %>%
  model(ARIMA(Population)) %>%
  forecast(h = 5) %>%
  autoplot(nz_economy) + ylab("Number of people (in millions)")
```



# Australian cement production

```
cement <- aus_production %>%
  select(Cement) %>%
  filter_index("1988 Q1" ~ .)
train <- cement %>% filter_index(. ~ "2007 Q4")
fit <- train %>%
  model(ets = ETS(Cement),
        arima = ARIMA(Cement))
```

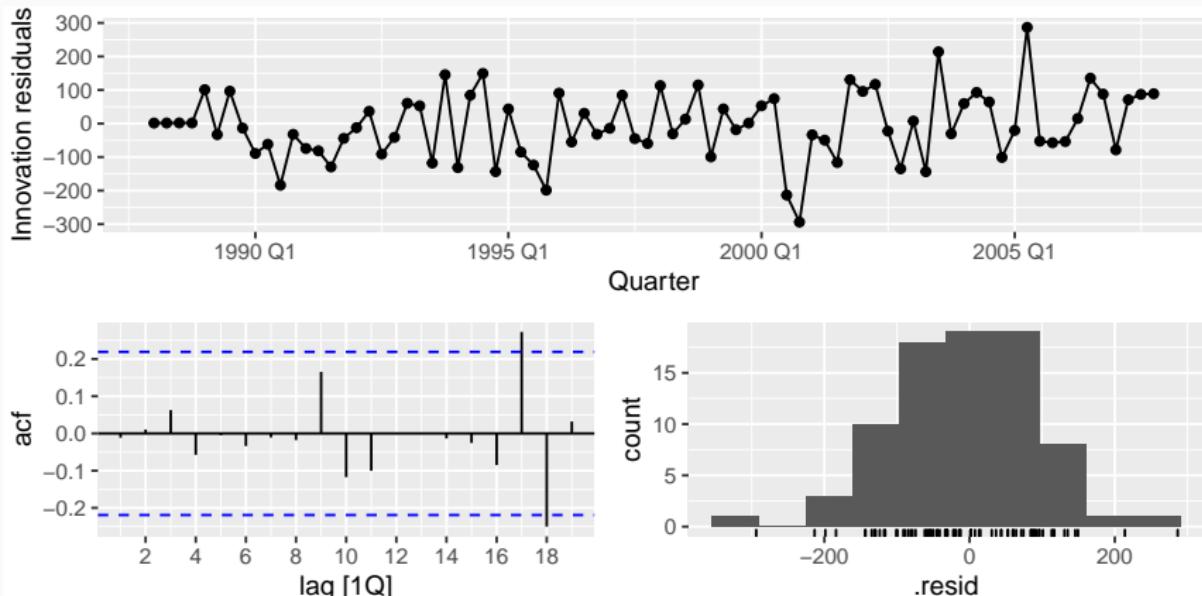
# Australian cement production

```
fit %>% select(arima) %>% report()

## Series: Cement
## Model: ARIMA(1,0,1)(2,1,1)[4] w/ drift
##
## Coefficients:
##             ar1      ma1     sar1     sar2     sma1   constant
##             0.8886 -0.237  0.081  -0.234  -0.898      5.39
## s.e.    0.0842  0.133  0.157   0.139   0.178      1.48
##
## sigma^2 estimated as 11456: log likelihood=-464
## AIC=941    AICc=943    BIC=957
```

# Australian cement production

```
fit %>% select(arima) %>% gg_tsresiduals()
```



# Australian cement production

```
fit %>%
  select(arima) %>%
  augment() %>%
  features(.resid, ljung_box, lag = 18, dof = 6)
```

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 arima     20.7     0.0542
```

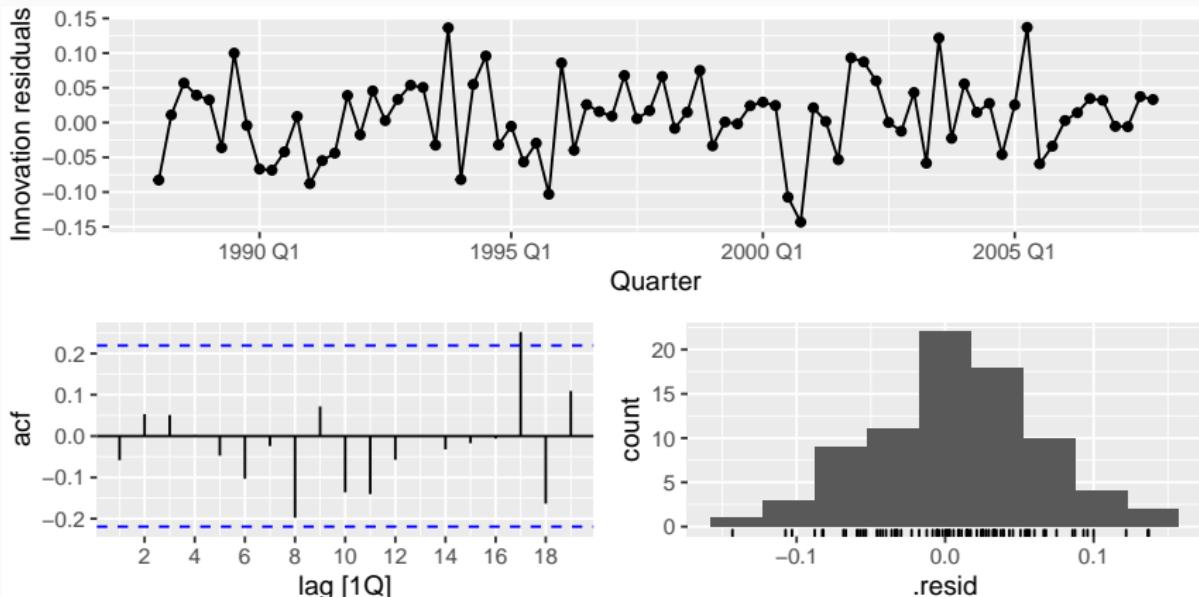
# Australian cement production

```
fit %>% select(ets) %>% report()

## Series: Cement
## Model: ETS(M,N,M)
## Smoothing parameters:
##     alpha = 0.753
##     gamma = 1e-04
##
## Initial states:
## l[0] s[0] s[-1] s[-2] s[-3]
## 1695 1.03 1.05 1.01 0.912
##
## sigma^2: 0.0034
##
## AIC AICc BIC
## 1104 1106 1121
```

# Australian cement production

```
fit %>% select(ets) %>% gg_tsresiduals()
```



# Australian cement production

```
fit %>%
  select(ets) %>%
  augment() %>%
  features(.innov, ljung_box, lag = 18, dof = 6)
```

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 ets      19.5     0.0777
```

# Australian cement production

```
fit %>%  
  forecast(h = "2 years 6 months") %>%  
  accuracy(cement) %>%  
  select(.model, RMSE:MASE)  
  
## # A tibble: 2 x 6  
##   .model    RMSE     MAE     MPE     MAPE     MASE  
##   <chr>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>  
## 1 arima    216.    186.   -7.71    8.68    1.27  
## 2 ets      222.    191.   -8.07    8.85    1.30
```

# Australian cement production

```
cement %>%
  model(ARIMA(Cement)) %>%
  forecast(h = "2 years") %>%
  autoplot(cement) + ylab("Tonnes (in thousands)")
```

