Data Science Applications and Techniques

Lecture 1

Dr David Weston

Birkbeck, University of London

Acknowledgement: Many of these slides are based on slides accompanying the course text: OTexts.org/fpp3/

Lectures, Labs, Assessments

- This module will be assessed by:
  - 70% written exam (2 hours).
  - 30% one coursework exercise.
- This module is delivered in the Autumn Term.
  - Lecture and Lab each Tuesday evening 18:00 – 21:00
  - Coursework - Released in Week 1 to start in Week 6. 2 weeks to submit.
  - Exam - Online in January (exact date to be fixed).

Accessing module resources

- You can access copies of the lecture notes, lab sheets, class work, class work solutions etc. from Moodle: http://moodle.bbk.ac.uk/

Reading Materials

All the following books are available to read online.

- Rob J Hyndman and George Athanasopoulos, "Forecasting: Principles and Practice" (3rd ed)
- Laura Igual and Santi Seguí, "Introduction to Data Science", Springer.
- Joel Grus, "Data Science from scratch", (2nd ed.)
- Selected Research Papers
- Tidyverse
  - Garrett Grolemund and Hadley Wickham, "R for data science" (2nd ed)

Syllabus

- Week 1 TidyVerse and Introduction to Applied Forecasting.
- Week 2 Time Series Decomposition.
- Week 3 The Forecasters' Toolbox.
- Week 4 Regression and Exponential Smoothing.
- Week 5 Arima and Neural Approaches to Forecasting.
- Week 6 Networks Analysis.
- Week 7 Recommender Systems.
- Week 8 Handling Missing Data. Pitfalls in analysis.
- Week 9 What Do Data Scientists Do? Real World Data, its Curation and Validation.
- Week 10 *Revision week.*

Aims of Week 1

- In this lecture you will learn:
  - Rstudio and the tidyverse
  - What makes things hard/easy to forecast
  - Time series data and tsibble objects.
  - Time series plots
- Reading:
  - Chapters 1 and 2 from Forecasting: Principles and Practice
  - Chapters 1 to 8 from R for data science

R and the Tidyverse

- R is a language and environment for statistical computing and graphics (https://www.r-project.org/about.html)
- R can be easily extended with the creation of *Packages*. These are libraries, many of which reside on the CRAN family of websites. They can be easily installed by using the `install.packages()` command.
- Tidyverse is a collection of packages designed for data science (https://www.tidyverse.org/).
- Rstudio (more specifically the Rstudio workbench) is an IDE that can be used for both R and Python programming.

Downloading Software

Installing R and RStudio

- Both R and RStudio are available on the computers in our labs.
- If you want to work using your own computer:
    1. Download and install the latest edition of R from https://cran.ma.imperial.ac.uk/
    2. Then download and install RStudio desktop from https://posit.co/download/rstudio-desktop/
    3. Open RStudio and type in the Console window:
        - `install.packages('tidyverse','fpp3')`

(The package `fpp3` contains data from the course text)

Forecasting

What can we Forecast?

Vast array practical applications of forecasting examples include:

- Telecommunication routing
- Call Centre hourly/daily/weekly volumes of calls
- Stock Inventory

How well we can predict the future depends on several factors including:

1. How well we understand the factors that contribute to it.
2. How much data is available.
3. How similar the future is to the past.
4. Whether the forecasts can affect the thing we are trying to forecast.

Example 1 - Electricity Demand

1. How well we understand the factors that contribute to it.
   - Largely driven by temperature. Some other contributing factors: calender variations, holidays.
2. How much data is available.
   - Usually plenty of past data of actual demand. Also need weather condition data .
3. How similar the future is to the past.
   - From 1, we can be confident that this is the case
4. Whether the forecasts can affect the thing we are trying to forecast.
   - No it doesn't.

Examples 2 and 3

Example 2 - Currency Exchange Rates

- Only 2 is satisfied.
- This is a good example of a forecast that affects the thing it is forecasting.
  - e.g. publicized forecasts that predict the exchange rate will increase, will result in people adjusting the price their are willing to pay, which results in an increase.

Example 3 - National Lottery

- Only 2 is satisfied.There is enough data to show it cannot be predicted with any accuracy

Handling Timeseries in TidyVerse with ffp3 package

R for Python Programmers I

Python code

```
year = list(range(2012,2016))
print(year == [2012,2013,2014,2015]) # note how a list is created
```

```
True
```

```
print("year[0] is " + str(year[0]) )  # note the index value
```

```
year[0] is 2012
```

```
print(year[-1]) # note the negative index value
```

```
2015
```

R for Python Programmers II

R Code

```
year <- 2012:2015
print(year == c(2012,2013,2014,2015))
```

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

```
print(paste("year[1] is", year[1]))  # note the index value
```

```
[1] "year[1] is 2012"
```

```
print(year[-1]) # note the negative index value
```

```
[1] 2013 2014 2015
```

Pipes and `tibble` objects

- The pipe operator `%>%` or `|>` helps to make composing functions clearer to see.
  - `a %>% b %>% c` is equivalent to `c(b(a))`
- R has a fundamental data structure called a `data.frame`. This is equivalent to a table in a database.
- The tidyverse introduces a data structure called a `tibble` which is an enhanced the basic `data.frame`. See
- When a tidyverse function requires to use R functionality it might need to cast a `tibble`, to a `data.frame`. The function for casting is `as.data.frame()`
- This leads to ...

`tsibble` objects

- A `tsibble` allows storage and manipulation of multiple time series in R. (A times series tibble.)
- It contains:
  - An index: time information about the observation
  - Measured variable(s): numbers of interest
  - Key variable(s): optional unique identifiers for each series
- It works with tidyverse functions.

The tsibble index

```
mydata <- tsibble(
    year = 2012:2016,
    y = c(123, 39, 78, 52, 110),
    index = year
)
mydata
```

```
## # A tsibble: 5 x 2 [1Y]
##    year     y
##   <int> <dbl>
## 1  2012   123
## 2  2013    39
## 3  2014    78
## 4  2015    52
## 5  2016   110
```

The `tsibble` index

For observations more frequent than once per year, we need to use a time class function on the index. Consider a tibble called `z`:

```
z
```

```
# A tibble: 5 x 2
  Month    Observation
  <chr>          <dbl>
1 2019 Jan          50
2 2019 Feb          23
3 2019 Mar          34
4 2019 Apr          30
5 2019 May          25
```

The tsibble index

For observations more frequent than once per year, we need to use a time class function on the index.

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

```
# A tsibble: 5 x 2 [1M]
     Month Observation
     <mth>       <dbl>
1 2019 Jan          50
2 2019 Feb          23
3 2019 Mar          34
4 2019 Apr          30
5 2019 May          25
```

The `tsibble` index

Common time index variables can be created with these functions:

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

Read a csv file to `tsibble`

Suppose we have the following quarterly data stored in a csv file (only the first 10 rows are shown). This data set provides information on the size of the prison population in Australia, disaggregated by state, gender, legal status and indigenous status. (Here, ATSI stands for Aboriginal or Torres Strait Islander.)

```
prison <- readr::read_csv("data/prison_population.csv")
```

```
# 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  Other          2
 3 2005-03-01 ACT   Female Sentenced ATSI           0
 4 2005-03-01 ACT   Female Sentenced Other          0
 5 2005-03-01 ACT   Male   Remanded  ATSI           7
 6 2005-03-01 ACT   Male   Remanded  Other         58
 7 2005-03-01 ACT   Male   Sentenced ATSI           0
 8 2005-03-01 ACT   Male   Sentenced Other          0
 9 2005-03-01 NSW   Female Remanded  ATSI          51
10 2005-03-01 NSW   Female Remanded  Other        131
# i 3,062 more rows
```

Read a csv file to tsibble

```
prison <- readr::read_csv("data/prison_population.csv") %>%
  mutate(Quarter = yearquarter(date))
```

```
# A tibble: 3,072 x 7
   date       state gender legal     indigenous count Quarter
   <date>     <chr> <chr>  <chr>     <chr>      <dbl>   <qtr>
 1 2005-03-01 ACT   Female Remanded  ATSI           0 2005 Q1
 2 2005-03-01 ACT   Female Remanded  Other          2 2005 Q1
 3 2005-03-01 ACT   Female Sentenced ATSI           0 2005 Q1
 4 2005-03-01 ACT   Female Sentenced Other          0 2005 Q1
 5 2005-03-01 ACT   Male   Remanded  ATSI           7 2005 Q1
 6 2005-03-01 ACT   Male   Remanded  Other         58 2005 Q1
 7 2005-03-01 ACT   Male   Sentenced ATSI           0 2005 Q1
 8 2005-03-01 ACT   Male   Sentenced Other          0 2005 Q1
 9 2005-03-01 NSW   Female Remanded  ATSI          51 2005 Q1
10 2005-03-01 NSW   Female Remanded  Other        131 2005 Q1
# i 3,062 more rows
```

Read a csv file to tsibble

```
prison <- readr::read_csv("data/prison_population.csv") %>%
  mutate(Quarter = yearquarter(date)) %>%
  select(-date)
```

```
# 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  Other          2 2005 Q1
 3 ACT   Female Sentenced ATSI           0 2005 Q1
 4 ACT   Female Sentenced Other          0 2005 Q1
 5 ACT   Male   Remanded  ATSI           7 2005 Q1
 6 ACT   Male   Remanded  Other         58 2005 Q1
 7 ACT   Male   Sentenced ATSI           0 2005 Q1
 8 ACT   Male   Sentenced Other          0 2005 Q1
 9 NSW   Female Remanded  ATSI          51 2005 Q1
10 NSW   Female Remanded  Other        131 2005 Q1
# i 3,062 more rows
```

Read a csv file to tsibble

```r
prison <- readr::read_csv("data/prison_population.csv") %>%
  mutate(Quarter = yearquarter(date)) %>%
  select(-date) %>%
  as_tsibble(
    index = Quarter,
    key = c(state, gender, legal, indigenous)
  )
```

```
# 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
 6 ACT   Female Remanded ATSI           1 2006 Q2
 7 ACT   Female Remanded ATSI           1 2006 Q3
 8 ACT   Female Remanded ATSI           0 2006 Q4
 9 ACT   Female Remanded ATSI           0 2007 Q1
10 ACT   Female Remanded ATSI           1 2007 Q2
# i 3,062 more rows
```

Working with `tsibble` objects

Example: Australian pharmaceutical sales

The Pharmaceutical Benefits Scheme (PBS) is the Australian government drugs subsidy scheme.

- Many drugs bought from pharmacies are subsidised to allow more equitable access to modern drugs.
- The cost to government is determined by the number and types of drugs purchased. Currently nearly 1% of GDP.
- The total cost is budgeted based on forecasts of drug usage.
- Costs are disaggregated by drug type (ATC1 x15 / ATC2 x84), concession category (x2) and patient type (x2). For ATC2 this gives $84 \times 2 \times 2 = 336$ time series.

Working with `tsibble` objects

```
PBS
```

```
# A tsibble: 67,596 x 9 [1M]
# Key:       Concession, Type, ATC1, ATC2 [336]
      Month Concession   Type  ATC1  ATC1_desc ATC2  ATC2_desc Scripts  Cost
      <mth> <chr>        <chr> <chr> <chr>     <chr> <chr>       <dbl> <dbl>
 1 1991 Jul Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   18228 67877
 2 1991 Aug Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   15327 57011
 3 1991 Sep Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   14775 55020
 4 1991 Oct Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   15380 57222
 5 1991 Nov Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   14371 52120
 6 1991 Dec Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   15028 54299
 7 1992 Jan Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   11040 39753
 8 1992 Feb Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   15165 54405
 9 1992 Mar Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   16898 61108
10 1992 Apr Concession~ Co-p~ A     Alimenta~ A01   STOMATOL~   18141 65356
# i 67,586 more rows
```

Working with `tsibble` objects

We can use the `filter()` function to select rows.

```r
PBS %>%
  filter(ATC2 == "A10")
```

```
## # A tsibble: 816 x 9 [1M]
## # Key:       Concession, Type, ATC1, ATC2 [4]
##       Month Concession Type  ATC1  ATC1_desc ATC2  ATC2_desc Scripts   Cost
##       <mth> <chr>      <chr> <chr> <chr>     <chr> <chr>       <dbl>  <dbl>
##  1 1991 Jul Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   89733 2.09e6
##  2 1991 Aug Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   77101 1.80e6
##  3 1991 Sep Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   76255 1.78e6
##  4 1991 Oct Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   78681 1.85e6
##  5 1991 Nov Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   70554 1.69e6
##  6 1991 Dec Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   75814 1.84e6
##  7 1992 Jan Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   64186 1.56e6
##  8 1992 Feb Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   75899 1.73e6
##  9 1992 Mar Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   89445 2.05e6
## 10 1992 Apr Concessio~ Co-p~ A     Alimenta~ A10   ANTIDIAB~   97315 2.23e6
## # i 806 more rows
```

Working with `tsibble` objects

We can use the `select()` function to select columns.

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

```
# 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
 9 1992 Mar Concessional Co-payments 2046102
10 1992 Apr Concessional Co-payments 2225977
# i 806 more rows
```

Working with tsibble objects

We can use the `summarise()` function to summarise over keys.

```
PBS %>%
  filter(ATC2 == "A10") %>%
  select(Month, Concession, Type, Cost) %>%
  summarise(total_cost = sum(Cost))
```

```
# 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
 9 1992 Mar    2985811
10 1992 Apr    3204780
# i 194 more rows
```

Working with `tsibble` objects

We can use the `mutate()` function to create new variables.

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

```
# A tsibble: 204 x 2 [1M]
      Month total_cost
      <mth>      <dbl>
 1 1991 Jul       3.53
 2 1991 Aug       3.18
 3 1991 Sep       3.25
 4 1991 Oct       3.61
 5 1991 Nov       3.57
 6 1991 Dec       4.31
 7 1992 Jan       5.09
 8 1992 Feb       2.81
 9 1992 Mar       2.99
10 1992 Apr       3.20
# i 194 more rows
```

Working with tsibble objects

We can use the `mutate()` function to create new variables.

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

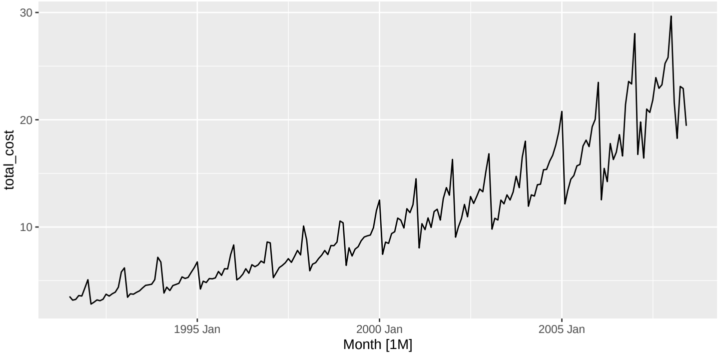
```
a10
```

```
## # A tsibble: 204 x 2 [1M]
##       Month total_cost
##       <mth>      <dbl>
##  1 1991 Jul       3.53
##  2 1991 Aug       3.18
##  3 1991 Sep       3.25
##  4 1991 Oct       3.61
##  5 1991 Nov       3.57
##  6 1991 Dec       4.31
##  7 1992 Jan       5.09
##  8 1992 Feb       2.81
##  9 1992 Mar       2.99
## 10 1992 Apr       3.20
## # i 194 more rows
```

Time plots

`autoplot` produces an appropriate plot based on the data of the first argument. Notice the increasing trend and the pattern of jumps at each year end.

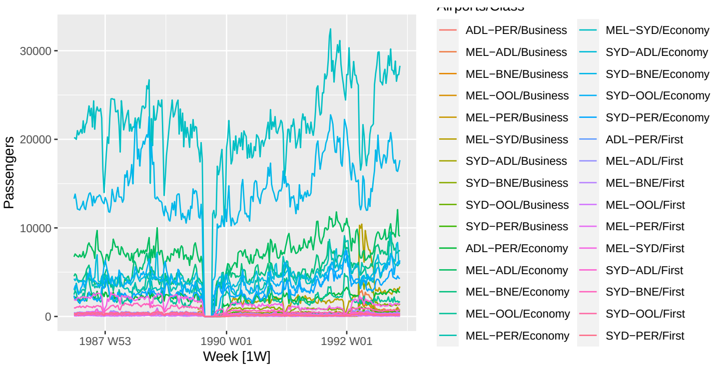
```
a10 %>%
  autoplot(total_cost)
```



Time plots Example

Passenger numbers on Ansett airline flights.

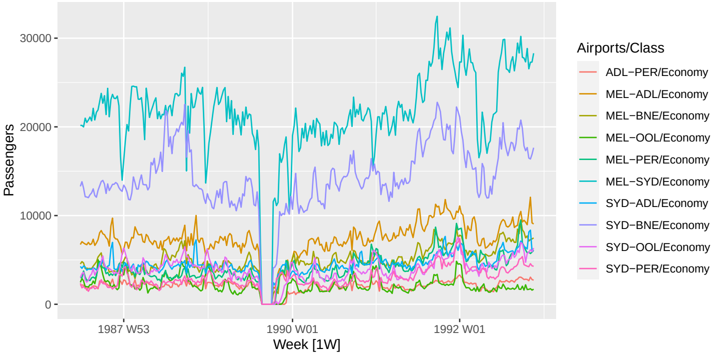
```
ansett %>%
  autoplot(Passengers)
```



Time plots Example

Drilling down into the data

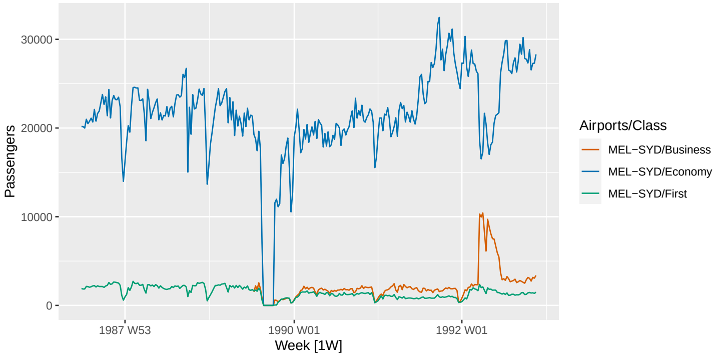
```
ansett %>%
  filter(Class == "Economy") %>%
  autoplot(Passengers)
```



Time plots Example

Selecting only a specific airport. Notice the economy passenger numbers fall to zero for a short period.

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



Time series patterns

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

Seasonal pattern exists when a series is influenced by seasonal factors (e.g. the quarter of the year, the month, or day of the week).

Cyclic pattern exists when data exhibit rises and falls that are *not of fixed period.*

Time series patterns

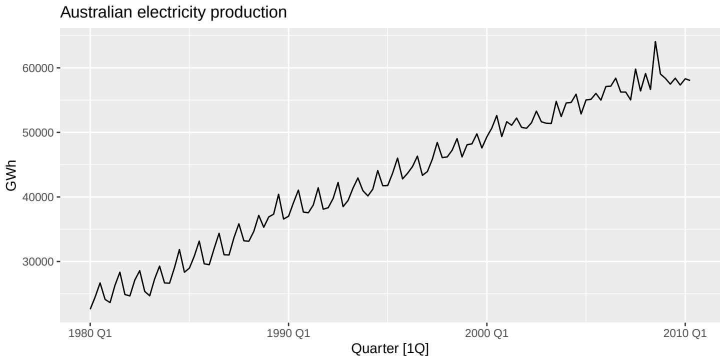
Differences between seasonal and cyclic patterns:

- seasonal pattern constant length; cyclic pattern variable length
- average length of cycle longer than length of seasonal pattern
- magnitude of cycle more variable than magnitude of seasonal pattern

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

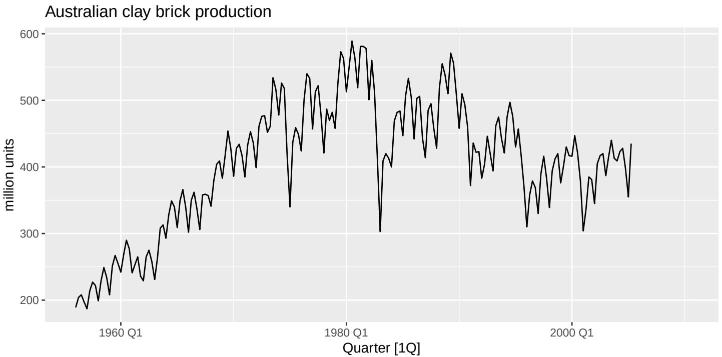
Time series patterns Example 1

```
aus_production %>%
  filter(year(Quarter) >= 1980) %>%
  autoplot(Electricity) +
  labs(y = "GWh", title = "Australian electricity production")
```



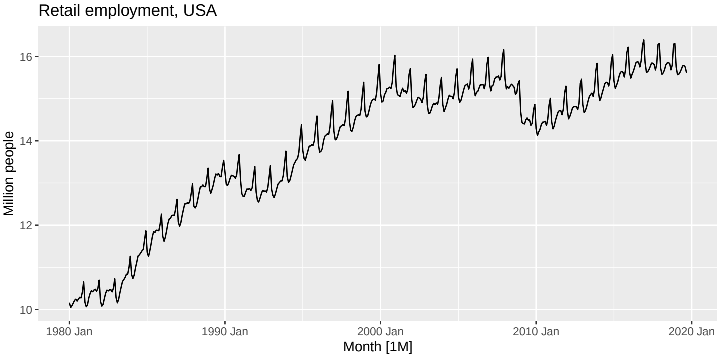
Time series patterns Example 2

```
aus_production %>%
  autoplot(Bricks) +
  labs(y = "million units", title = "Australian clay brick product
```



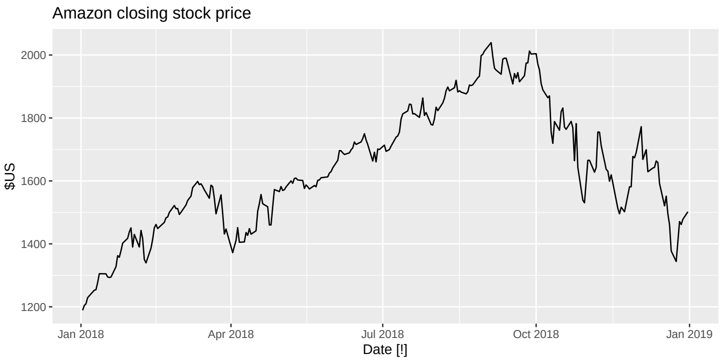
Time series patterns Example 3

```r
us_employment %>%
  filter(Title == "Retail Trade", year(Month) >= 1980) %>%
  autoplot(Employed / 1e3) +
  labs(y = "Million people", title = "Retail employment, USA")
```



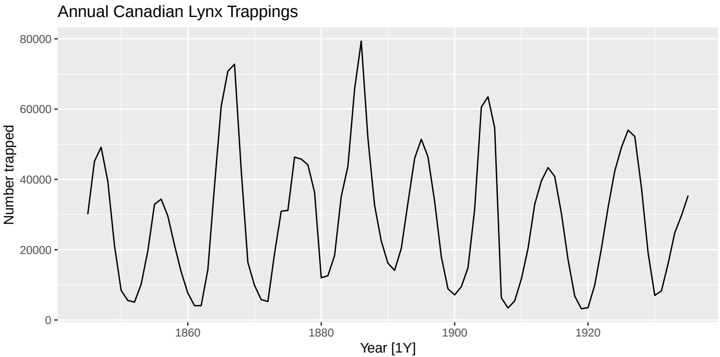
Time series patterns Example 4

```
gafa_stock %>%
  filter(Symbol == "AMZN", year(Date) >= 2018) %>%
  autoplot(Close) +
  labs(y = "$US", title = "Amazon closing stock price")
```



Time series patterns Example 5

```
pelt %>%
  autoplot(Lynx) +
  labs(y="Number trapped", title = "Annual Canadian Lynx Trappings
```



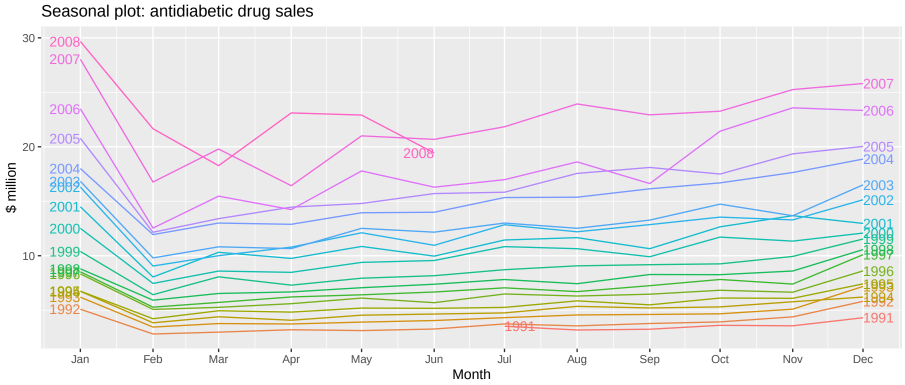
Seasonal and Subseries Plots

Seasonal plots

- Data plotted against the individual "seasons" in which the data were observed. (In this case a "season" is a month.)
- Something like a time plot except that the data from each season are overlapped.
- Enables the underlying seasonal pattern to be seen more clearly, and also allows any substantial departures from the seasonal pattern to be easily identified.
- In R: `gg_season()`

Seasonal plots - Example

```r
a10 %>% gg_season(total_cost, labels = "both") +
  labs(y = "$ million", title = "Seasonal plot: antidiabetic drug
```

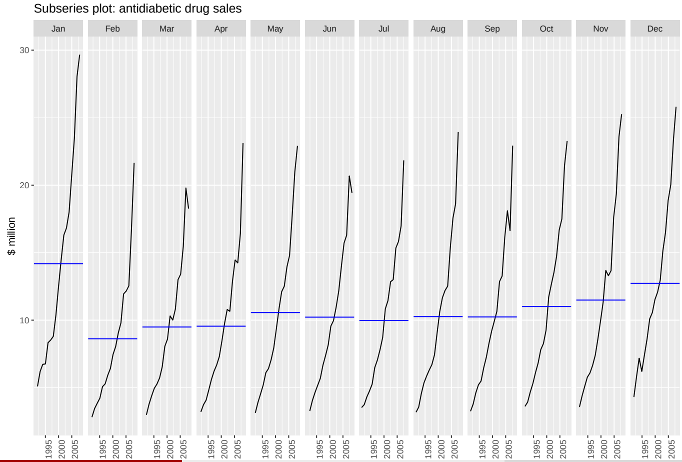


Seasonal subseries plots

- Data for each season collected together in time plot as separate time series.
- Enables the underlying seasonal pattern to be seen clearly, and changes in seasonality over time to be visualized.
- In R: `gg_subseries()`

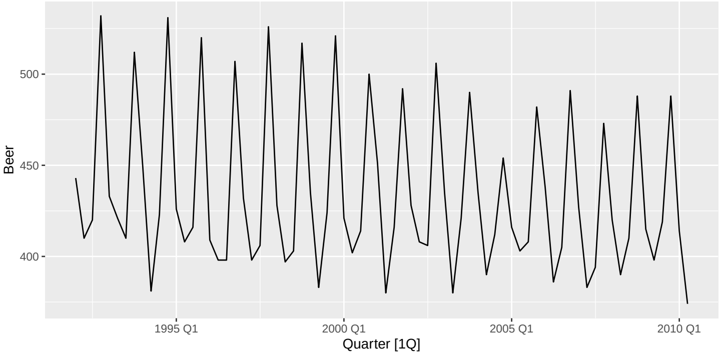
Seasonal Subseries Plots - Example

```r
a10 %>%
  gg_subseries(total_cost) +
  labs(y = "$ million", title = "Subseries plot: antidiabetic drug
```



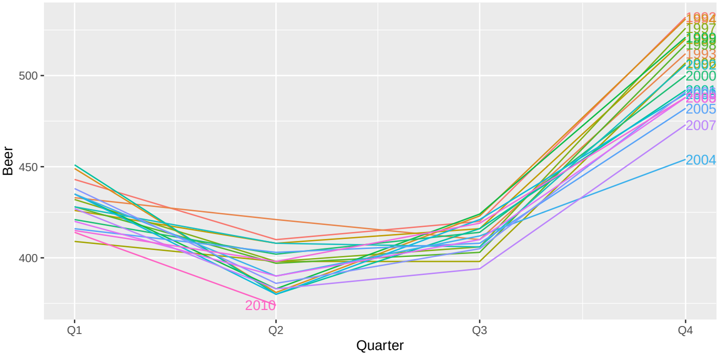
Quarterly Australian Beer Production

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



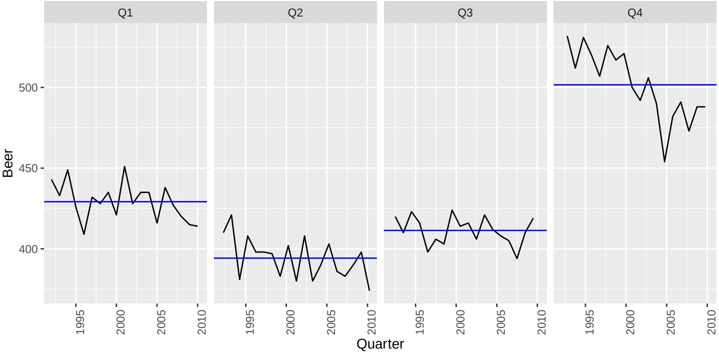
Quarterly Australian Beer Production

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



Quarterly Australian Beer Production

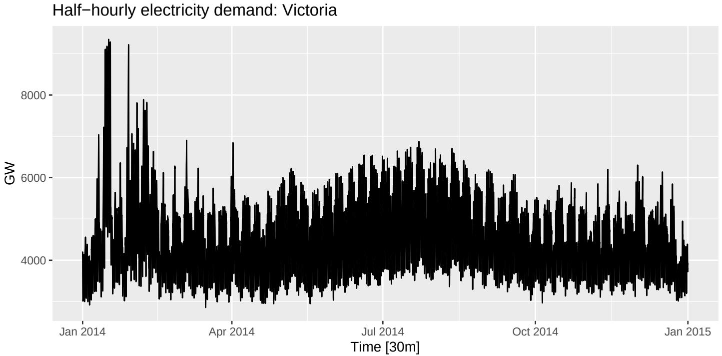
```r
beer %>% gg_subseries(Beer)
```



Scatterplots

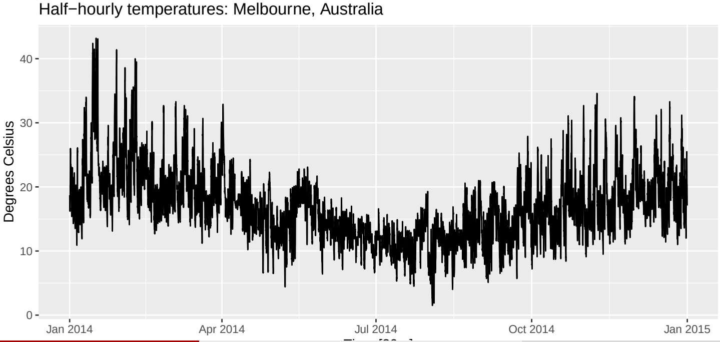
Electricity Demand

```r
vic_elec %>%
  filter(year(Time) == 2014) %>%
  autoplot(Demand) +
  labs(y = "GW",
       title = "Half-hourly electricity demand: Victoria")
```



Temperature Demand

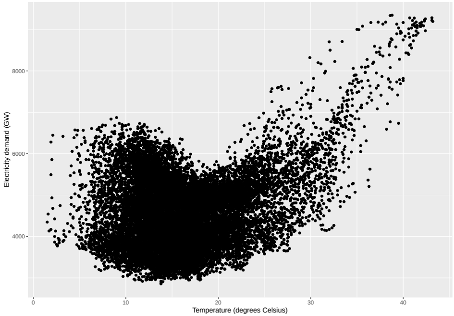
```
vic_elec %>%
  filter(year(Time) == 2014) %>%
  autoplot(Temperature) +
  labs(
    y = "Degrees Celsius",
    title = "Half-hourly temperatures: Melbourne, Australia"
  )
```



Scatterplot

Plot demand against temp

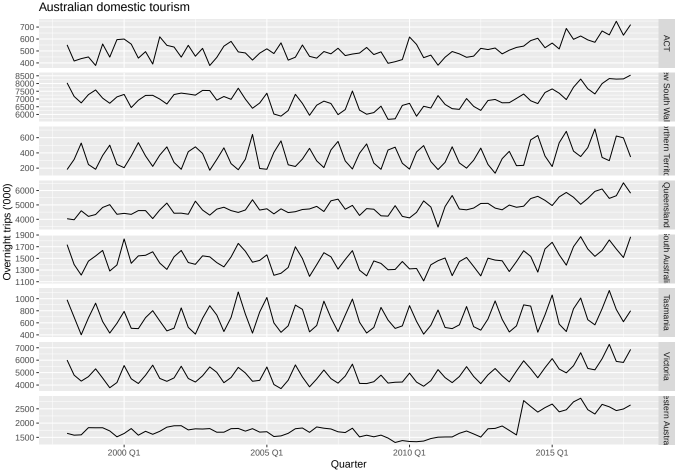
```
vic_elec %>%
  filter(year(Time) == 2014) %>%
  ggplot(aes(x = Temperature, y = Demand)) +
  geom_point() +
  labs(x = "Temperature (degrees Celsius)",
       y = "Electricity demand (GW)")
```



Scatterplot Matrices - Many Variables

```
visitors <- tourism %>%
  group_by(State) %>%
  summarise(Trips = sum(Trips))
visitors %>%
  ggplot(aes(x = Quarter, y = Trips)) +
  geom_line() +
  facet_grid(vars(State), scales = "free_y") +
  labs(title = "Australian domestic tourism",
       y= "Overnight trips ('000)")
```

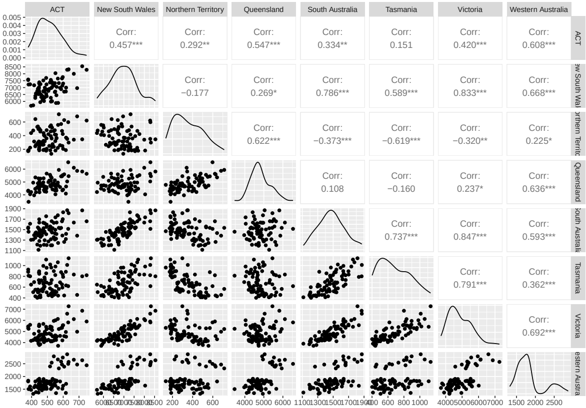
Scatterplot Matrices - Many Variables Output



Scatterplot Matrices

Plot predictor variables against each other:

```
visitors %>%
  pivot_wider(values_from=Trips, names_from=State) %>%
  GGally::ggpairs(columns = 2:9)
```



Lag plots and autocorrelation

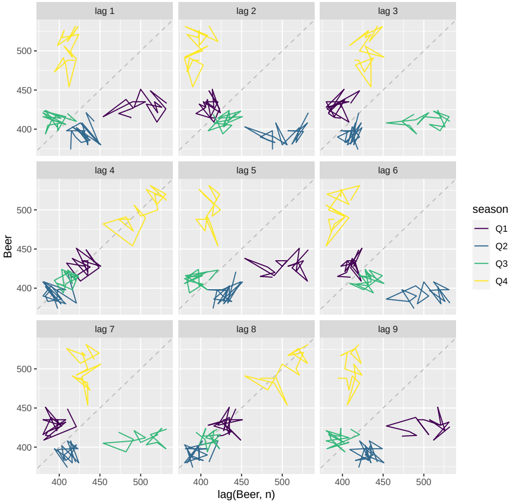
Beer production

```r
new_production <- aus_production %>%
  filter(year(Quarter) >= 1992)
new_production
```

```
# A tsibble: 74 x 7 [1Q]
   Quarter  Beer Tobacco Bricks Cement Electricity   Gas
     <qtr> <dbl>   <dbl>  <dbl>  <dbl>       <dbl> <dbl>
 1 1992 Q1   443    5777    383   1289       38332   117
 2 1992 Q2   410    5853    404   1501       39774   151
 3 1992 Q3   420    6416    446   1539       42246   175
 4 1992 Q4   532    5825    420   1568       38498   129
 5 1993 Q1   433    5724    394   1450       39460   116
 6 1993 Q2   421    6036    462   1668       41356   149
 7 1993 Q3   410    6570    475   1648       42949   163
 8 1993 Q4   512    5675    443   1863       40974   138
 9 1994 Q1   449    5311    421   1468       40162   127
10 1994 Q2   381    5717    475   1755       41199   159
# i 64 more rows
```

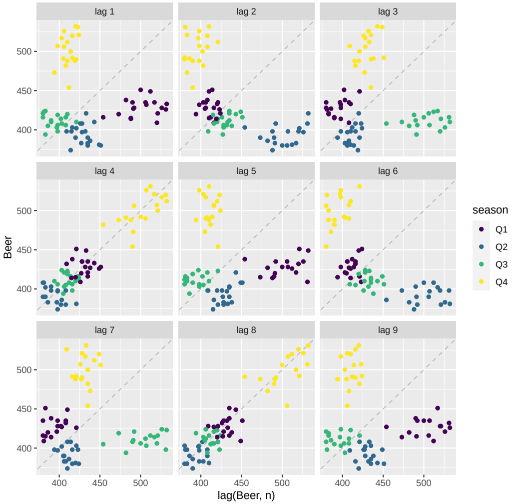
Beer production

```
new_production %>% gg_lag(Beer)
```



Beer production

```
new_production %>% gg_lag(Beer, geom='point')
```



Lagged scatterplots

- Each graph shows y_t plotted against y_{t-k} for different values of k.
- The autocorrelations are the correlations associated with these scatterplots.
- ACF (autocorrelation function):
  - $r_1 = \text{Correlation}(y_t, y_{t-1})$
  - $r_2 = \text{Correlation}(y_t, y_{t-2})$
  - $r_3 = \text{Correlation}(y_t, y_{t-3})$
  - etc.

Autocorrelation

Autocorrelation

Covariance and **correlation**: measure extent of **linear relationship** between two variables (y and X).

Autocovariance and **autocorrelation**: measure linear relationship between **lagged values** of a time series y.

We measure the relationship between:

- y_t and y_{t-1}
- y_t and y_{t-2}
- y_t and y_{t-3}
- etc.

Autocorrelation

We denote the sample autocovariance at lag k by c_k and the sample autocorrelation at lag k by r_k. Then define

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

and

$$r_k = c_k / c_0$$

- r_1 indicates how successive values of y relate to each other
- r_2 indicates how y values two periods apart relate to each other
- r_k is *almost* the same as the sample correlation between y_t and y_{t-k}.

Autocorrelation

Results for first 9 lags for beer data:

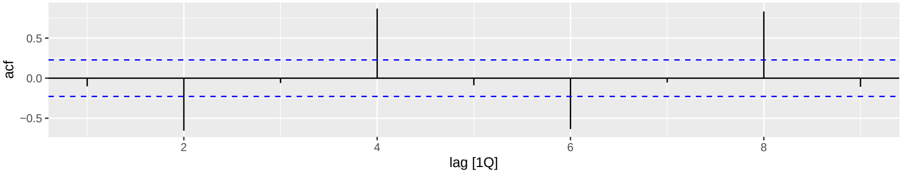
```
new_production %>% ACF(Beer, lag_max = 9)
```

```
# A tsibble: 9 x 2 [1Q]
        lag      acf
  <cf_lag>    <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

Results for first 9 lags for beer data:

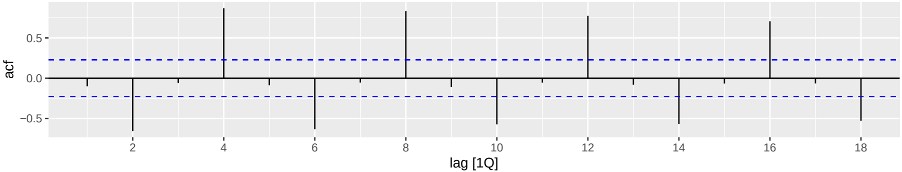
```
new_production %>% ACF(Beer, lag_max = 9) %>% autoplot()
```



- Together, the autocorrelations at lags 1, 2, . . . , make up the *autocorrelation* or ACF.
- The plot is known as a **correlogram**

Autocorrelation

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



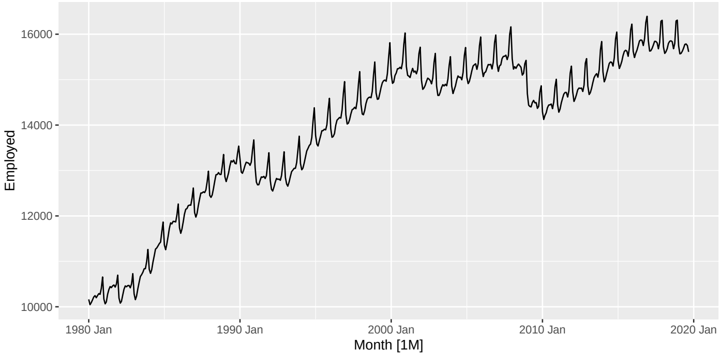
- r_4 higher than for the other lags. This is due to **the seasonal pattern in the data**: the peaks tend to be **4 quarters** apart and the troughs tend to be **2 quarters** apart.
- r_2 is more negative than for the other lags because troughs tend to be 2 quarters behind peaks.

Trend and seasonality in ACF plots

- When data have a trend, the autocorrelations for small lags tend to be large and positive.
- When data are seasonal, the autocorrelations will be larger at the seasonal lags (i.e., at multiples of the seasonal frequency)
- When data are trended and seasonal, you see a combination of these effects.

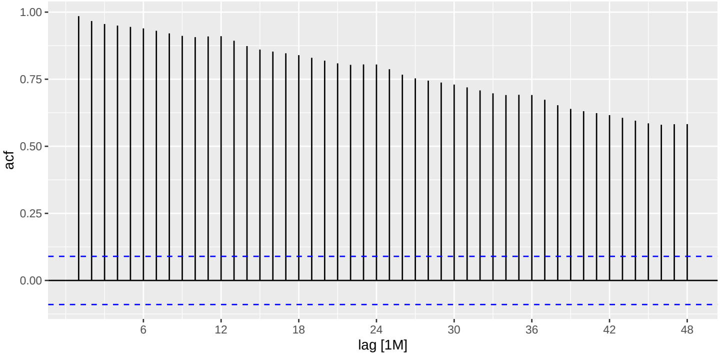
Autocorrelation functions

```r
retail <- us_employment %>%
  filter(Title == "Retail Trade", year(Month) >= 1980)
retail %>% autoplot(Employed)
```



Autocorrelation functions

```
retail %>%
  ACF(Employed, lag_max = 48) %>%
  autoplot()
```



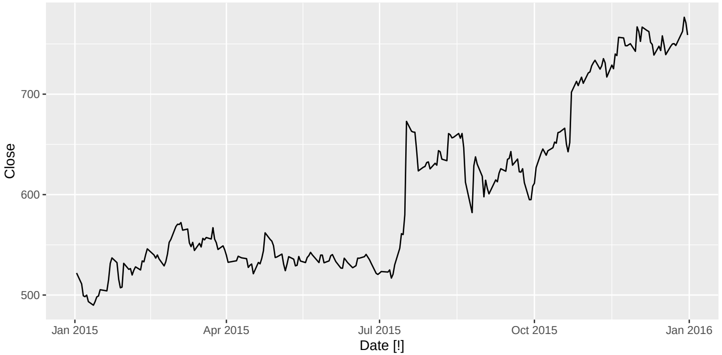
Google stock price

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

```
# A tsibble: 252 x 2 [!]
   Date       Close
   <date>     <dbl>
 1 2015-01-02  522.
 2 2015-01-05  511.
 3 2015-01-06  499.
 4 2015-01-07  498.
 5 2015-01-08  500.
 6 2015-01-09  493.
 7 2015-01-12  490.
 8 2015-01-13  493.
 9 2015-01-14  498.
10 2015-01-15  499.
# i 242 more rows
```

Google stock price

```
google_2015 %>% autoplot(Close)
```



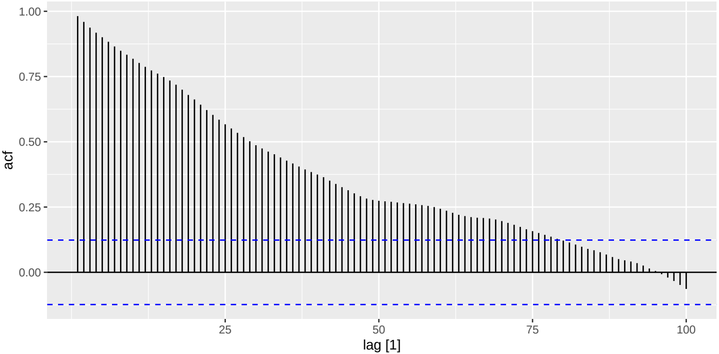
Google stock price

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

```
# A tsibble: 100 x 2 [1]
         lag   acf
    <cf_lag> <dbl>
 1         1 0.982
 2         2 0.959
 3         3 0.937
 4         4 0.918
 5         5 0.901
 6         6 0.883
 7         7 0.865
 8         8 0.849
 9         9 0.834
10        10 0.818
# i 90 more rows
```

Google stock price

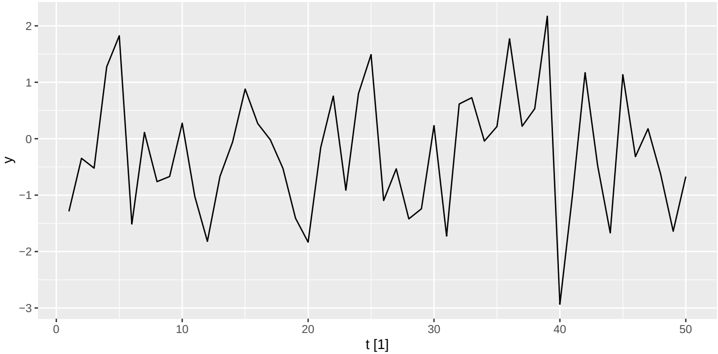
```r
google_2015 %>%
  ACF(Close, lag_max = 100) %>%
  autoplot()
```



White noise

White noise - Example

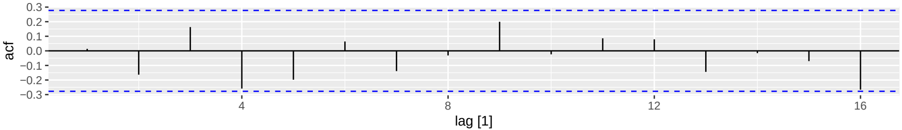
```r
set.seed(30)  # set the random seed, so result is reproducible
wn <- tsibble(t = 1:50, y = rnorm(50), index = t)
# rnorm generates samples from a standard normal
wn %>% autoplot(y)
```



White noise ACF plot

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

r_1	r_2	r_3	r_4	r_5	r_6	r_7	r_8	r_9	r_{10}
0.014	-0.163	0.163	-0.259	-0.198	0.064	-0.139	-0.032	0.199	-0.024



- Sample autocorrelations for white noise series.
- Expect each autocorrelation to be close to zero.
- Blue lines show 95% critical values (see next slide).

Sampling distribution of autocorrelations

Sampling distribution of r_k for white noise data is asymptotically N(0,1/T).

- 95% of all r_k for white noise must lie within $\pm 1.96/\sqrt{T}$.
- If this is not the case, the series is probably not White Noise.
- Common to plot lines at $\pm 1.96/\sqrt{T}$ when plotting ACF. These are the *critical values*.