

# Applied Economic Forecasting

## 2. Exploring & Visualizing Time series

- 1 Time series in R
- 2 Time plots
- 3 Seasonal plots
- 4 Scatterplots
- 5 Lag plots and autocorrelation
- 6 White Noise Process

# Section 1

## Time series in R

# What is a Time Series?

- A time series can be thought of as a list of observations (the measurements), along with some information about what times those numbers were recorded (the index).
- A basic way of storing a time series is as a `ts` object.
- The `fpp3` package stores this information as a `tsibble` object.

## ts objects and ts function

Let us get familiar with `ts` objects before tackling `tstibble` objects.

### Example

Year	Obs.
2012	123
2013	39
2014	78
2015	52
2016	110

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

## ts objects and the ts function

For observations that are more frequent than once per year, we need to add a **frequency** argument. We usually think of the **frequency** argument as the number of time the series will be observed (collected) in a year.

E.g., monthly data (**frequency** = 12) stored as a numerical vector **z**:

```
y <- ts(z, frequency=12, start=c(2003, 1))
```

## ts objects and ts function

`ts(data, frequency, start, end)`

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

We do not need to always specify both the **start** and **end** arguments. In many cases, just one will suffice.

# Let's Practice!!!

- ① Set a seed as  $10^3$
- ② Generate a random normal variable ( $\mathbf{x}$ ) with 200 observations, mean = 75, and sd = 5
- ③ Declare  $\mathbf{x}$  as a *quarterly* **ts** object ending December 2018 ( $\mathbf{x.ts}$ )
- ④ Now repeat step 3 with weekly, monthly and annual frequencies (*How about using a loop?*)



## tsibble function

The `tsibble` function offers a nice mix between the `ts` objects/function and the `tidyverse` package.

Let's assume I have the following data:

Year	GDP
2010	155
2011	134
2012	80
2013	100
2014	120
2015	110

# The tsibble function

We can record this as a `tsibble` object using the following syntax:

```
y <- tsibble(Year = 2010:2015,  
             GDP = c(155,134,80,100,120,110),  
             index = Year  
            )
```

```
y  
  
## # A tsibble: 6 x 2 [1Y]  
##   Year    GDP  
##   <int> <dbl>  
## 1  2010    155  
## 2  2011    134  
## 3  2012     80  
## 4  2013    100  
## 5  2014    120  
## 6  2015    110
```

# The `tsibble()` function

Similar to the `ts()` function earlier, we have the ability to work with variables observed at higher frequencies, say quarterly, monthly, daily, etc. . .

For instance:

Month	Observation
2020 Jan	50
2020 Feb	23
2020 Mar	45
2020 Apr	67
2020 May	78
2020 Jun	98

# The `tsibble()` function

We would first need to convert our dataset to a `tsibble` object.

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

```
## # A tsibble: 6 x 2 [1M]  
##      Month Observation  
##      <mth>          <dbl>  
## 1 2020 Jan           50  
## 2 2020 Feb           23  
## 3 2020 Mar           45  
## 4 2020 Apr           67  
## 5 2020 May           78  
## 6 2020 Jun           98
```

## Note:

It is worth mentioning the difference between the `as_tsibble` function in the code above and `tsibble`. The former is used when you would like to convert a dataframe or variable that *already exists*. `tsibble` is usually reserved for creating a new variable.

## Other Classes we might encounter:

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

# Working with tsibble objects

In many instances, we will be required to forecast a single (or subset of) variable(s) from our dataset. This is where `dplyr` functions such as `mutate()`, `filter()`, and `select()` come in handy.

For example, say we are interested in the winning times in the Olympic track events stored in the `olympic_running` data.

```
olympic_running
```

```
## # A tsibble: 312 x 4 [4Y]
## # Key:           Length, Sex [14]
##       Year Length Sex    Time
##   <int>  <int> <chr> <dbl>
## 1  1896    100 men     12
## 2  1900    100 men     11
## 3  1904    100 men     11
## 4  1908    100 men    10.8
## 5  1912    100 men    10.8
```

# Working with tsibble objects

We can filter to see only the female races.

```
olympic_running %>% filter(Sex == "women")
```

```
## # A tsibble: 104 x 4 [4Y]
## # Key:           Length, Sex [7]
##       Year Length Sex      Time
##   <int>  <int> <chr>  <dbl>
## 1  1928     100 women   12.2
## 2  1932     100 women   11.9
## 3  1936     100 women   11.5
## 4  1940     100 women    NA
## 5  1944     100 women    NA
## 6  1948     100 women   11.9
## 7  1952     100 women   11.5
## 8  1956     100 women   11.5
## 9  1960     100 women    11
## 10 1964     100 women   11.4
```

# Exercise Time

*Let us explore a few exercises for selecting, mutating, and summarizing the data.*



## Section 2

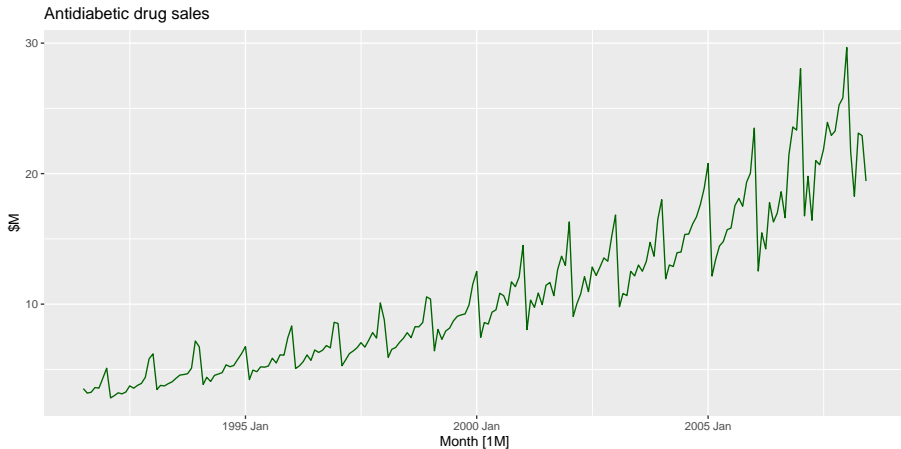
### Time plots

```
melsyd <- ansett %>%  
  filter(Airports == "MEL-SYD", Class == "Economy") %>%  
  mutate(Passengers = Passengers/1000)  
  
melsyd %>% autoplot(Passengers, col = "red") +  
  labs(title = "Ansett airlines Economy class",  
       subtitle = "Melbourne-Sydney", y = "Passengers ('000)")
```

Ansett airlines Economy class  
Melbourne-Sydney



```
a10 <- PBS %>% filter(ATC2 == "A10") %>%  
  summarise(TC = sum(Cost)/1e6)  
  
autoplot(a10, TC, col = "darkgreen") +  
  labs(y = "$M", title = "Antidiabetic drug sales")
```



# Let's Practice

- 1 Using the data from our earlier exercise: Plot `x.ts` at the monthly, quarterly and yearly frequencies.
  - 2 Create plots of the following time series: `Bricks` from `aus_production`, `Lynx` from `pelt`, `Close` from `gafa_stocks`. Redo your plot for `Close` focusing in on `GOOG` instead.
- Use `help()` or `?seriesName` to find out about the data in each series.
  - For each of the plots, be sure to modify the axis labels and title.

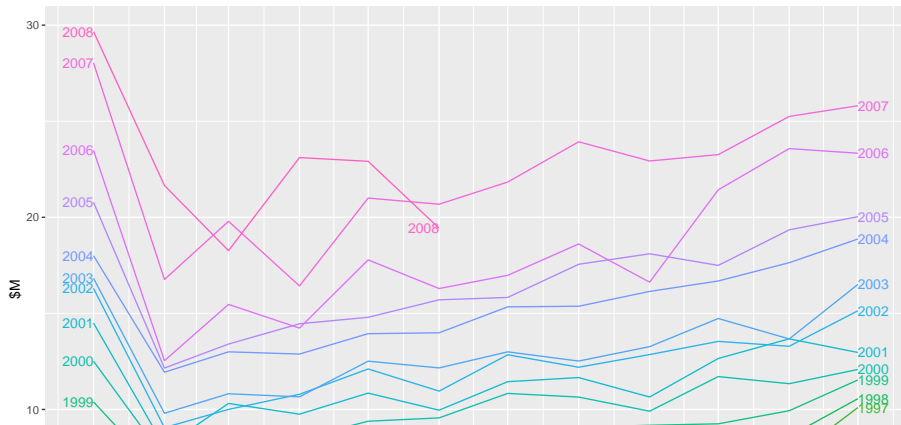
## Section 3

### Seasonal plots

# Seasonal plots

```
a10 %>% gg_season(TC, labels = "both") +  
  labs( y = "$M",  
        title = "Seasonal plot: Antidiabetic drug sales") +  
  #Add a bit of space on both sides of the graph.  
  expand_limits(x = ymd(c("1972-12-28", "1973-12-05")))
```

Seasonal plot: Antidiabetic drug sales

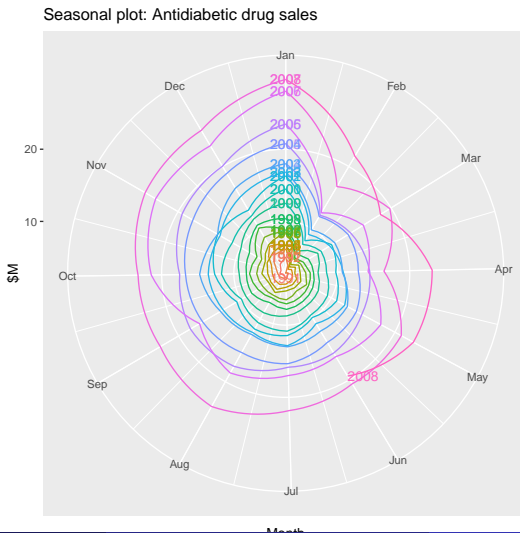


# Seasonal plots

- 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 polar plots

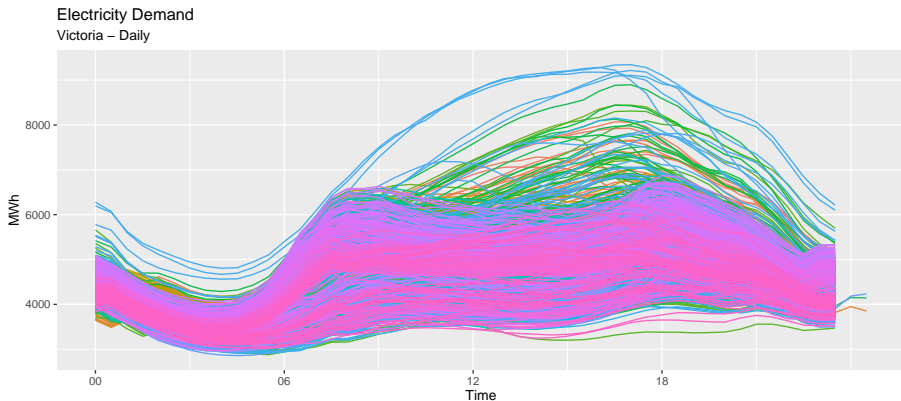
```
a10 %>% gg_season(TC, labels = "both", polar = TRUE) +  
  labs( y = "$M", title = "Seasonal plot: Antidiabetic drug sales")
```





# Multiple Seasonal Plots

```
# Plot daily pattern  
vic_elec %>% gg_season(Demand, period = "day") +  
  theme(legend.pos = "none") +  
  labs(title = "Electricity Demand", subtitle = "Victoria - Daily",  
       y = "MWh")
```

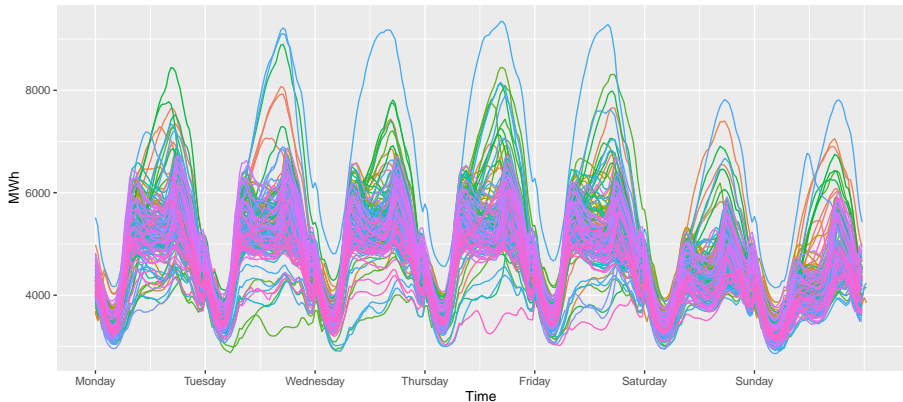


```
# Plot weekly pattern
```

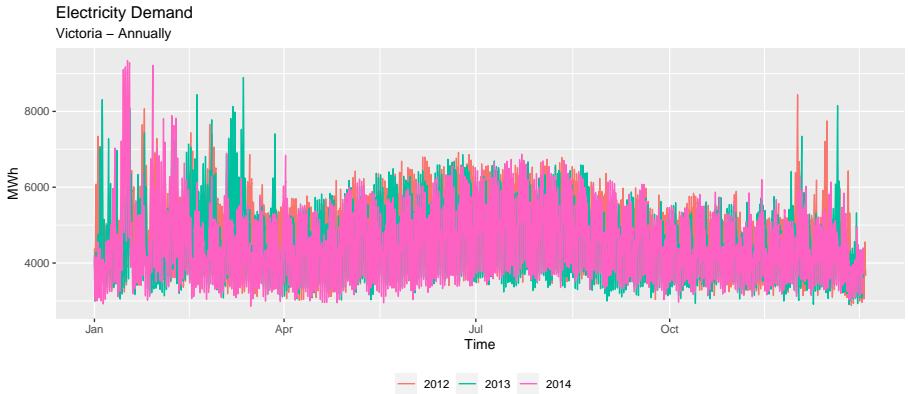
```
vic_elec %>% gg_season(Demand, period = "week") +  
  theme(legend.pos = "none") +  
  labs(title = "Electricity Demand", subtitle = "Victoria - Weekly",  
       y = "MWh")
```

Electricity Demand

Victoria - Weekly



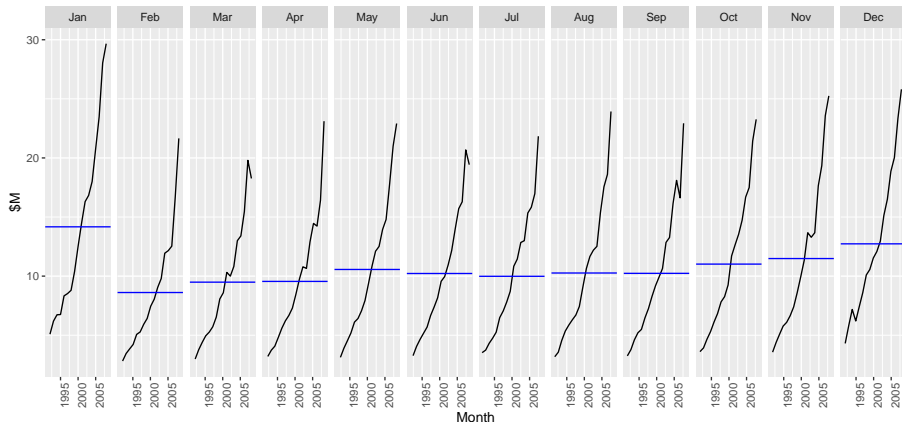
```
# Plot yearly pattern
vic_elec %>% gg_season(Demand, period = "year") +
  theme(legend.pos = "bottom") +
  labs(title = "Electricity Demand",
       subtitle = "Victoria - Annually",
       y = "MWh")
```



# Seasonal subseries plots

```
a10 %>% gg_subseries(TC) +  
  labs(y = "$M", title = "Subseries plot: Antidiabetic drug sales")
```

Subseries plot: Antidiabetic drug sales



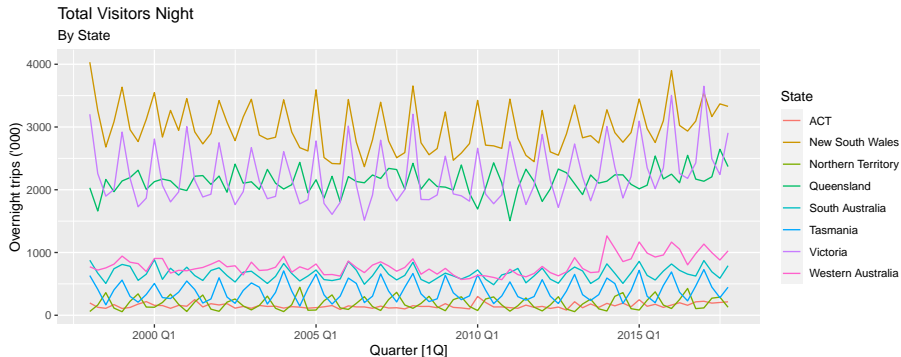
# Seasonal subseries plots

- 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()`

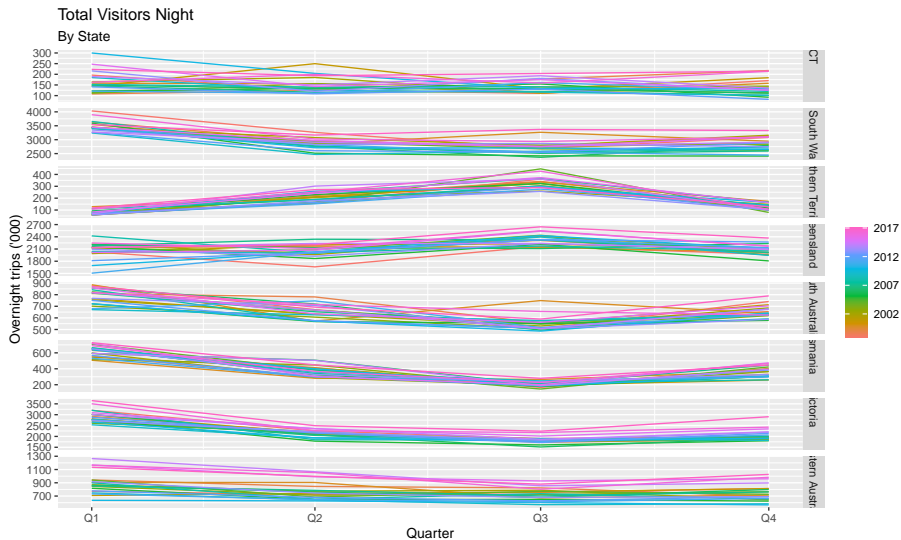
# Australian Tourism Data

```
holiday <- tourism %>% filter(Purpose == "Holiday") %>%  
  #Group the data by state then aggregate  
  group_by(State) %>% summarise(total = sum(Trips))
```

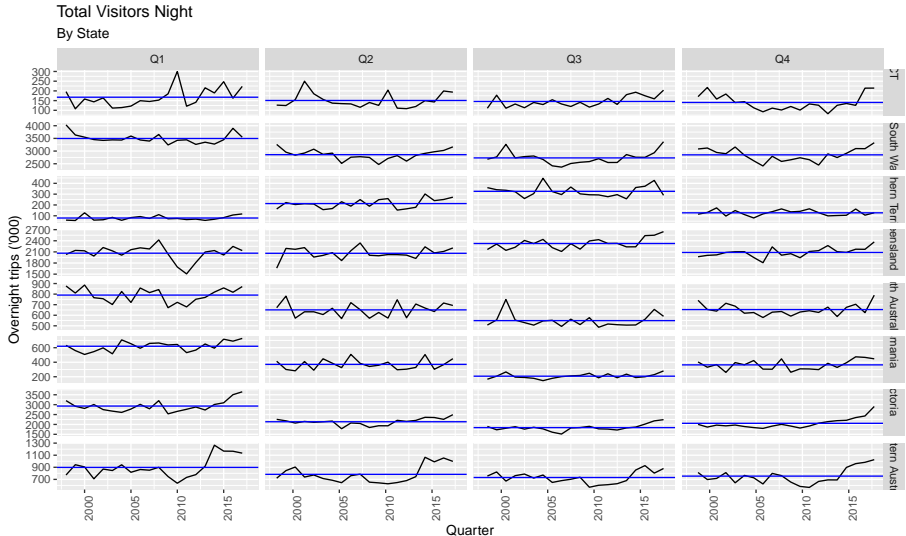
```
holiday %>% autoplot(total) +  
  labs(title = "Total Visitors Night", subtitle = "By State",  
       y = "Overnight trips ('000)")
```



```
holiday %>% gg_season(total) +  
  labs(title = "Total Visitors Night", subtitle = "By State",  
        y = "Overnight trips ('000)")
```



```
holiday %>% gg_subseries(total) +
  labs(title = "Total Visitors Night", subtitle = "By State",
       y = "Overnight trips ('000)")
```





# Let's Practice!!!

The `aus_arrivals` data set reports quarterly international arrivals to Australia from Japan, New Zealand, UK, and the US.

- Using the `autoplot()` and `gg_season()`, and `gg_subseries()` functions, **discuss** and **compare** the arrivals from each of these.
- Can you identify any unusual observations?

## Section 4

### Scatterplots

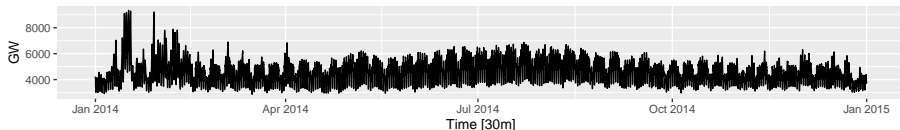
# Scatterplots: Relationship *Between* Time Series

Scatterplots are commonly used tools for visualizing the relationship between 2 variables. Take the text example of Electricity Demand and Temperature in Victoria, Australia:

```
p1 <- vic_elec %>% filter(year(Date) == 2014) %>% autoplot(Demand) +  
  labs(title = "Electricity Demand in Victoria (2014)",  
        subtitle = "Half-Hourly", y = "GW")  
p2 <- vic_elec %>% filter(year(Date) == 2014) %>% autoplot(Temperature) +  
  labs(title = "Temperature in Victoria (2014)", subtitle = "Half-Hourly",  
        y = "GW")  
gridExtra::grid.arrange(p1,p2, ncol = 1)
```

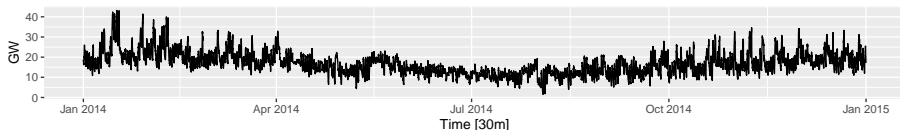
Electricity Demand in Victoria (2014)

Half-Hourly



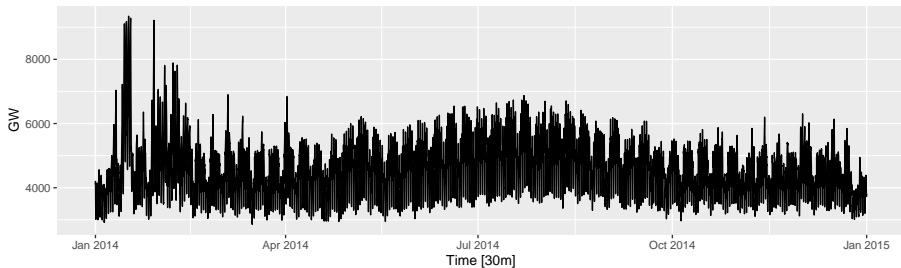
Temperature in Victoria (2014)

Half-Hourly



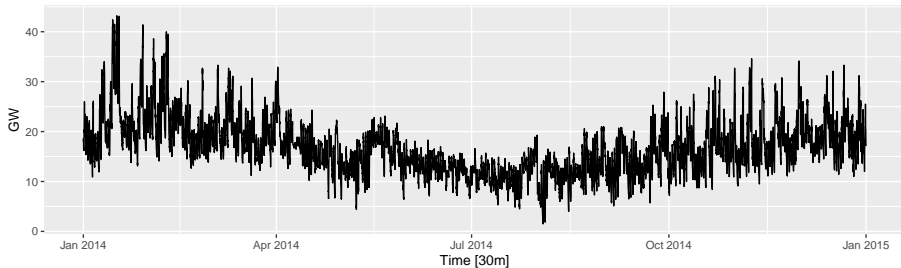
## Electricity Demand in Victoria (2014)

Half-Hourly



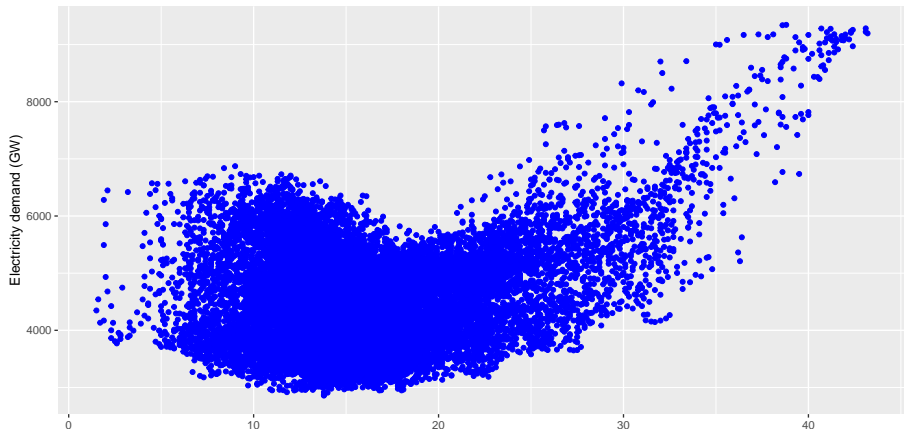
## Temperature in Victoria (2014)

Half-Hourly



# What would you make of this relationship?

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



# Scatterplots: Relationship *Between* Time Series

- Higher temperatures are associated with higher demand for electricity.
- Some high demand for heating on the lower end of the graph.

## Measuring the strength of this relationship?

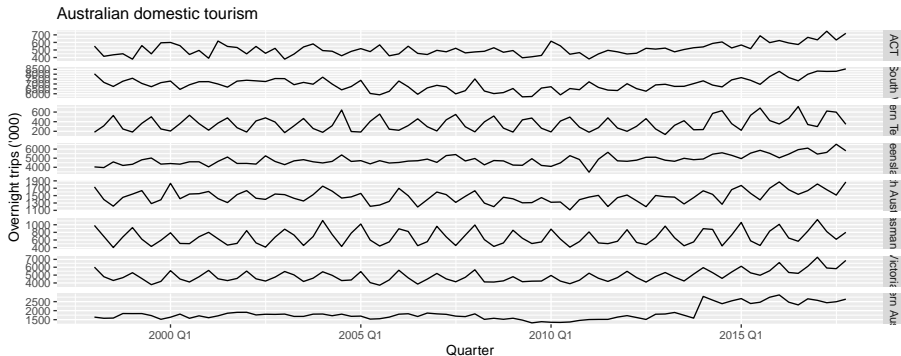
- For this, we will use the **correlation coefficient**:

$$\rho_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sigma_x^2} \sqrt{\sigma_y^2}}, \quad -1 \leq \rho_{xy} \leq 1$$

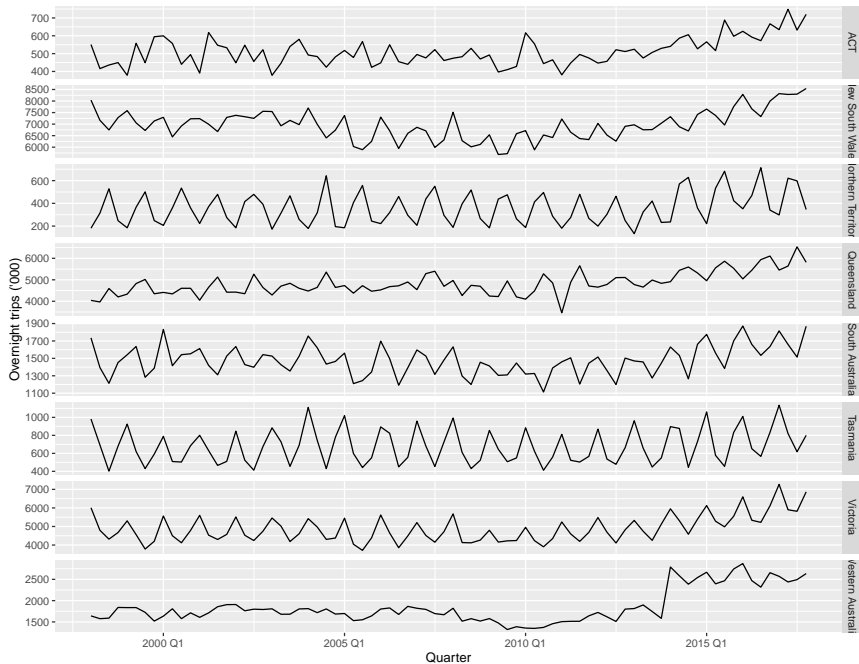
- Negative values indicate a negative *linear* relationship between  $x$  and  $y$ .
- Positive values indicate a positive *linear* relationship between  $x$  and  $y$ .

```
visitors <- tourism %>% group_by(State) %>%
  summarise(Trips = sum(Trips))

visitors %>% ggplot(aes(x = Quarter, y = Trips)) +
  geom_line() +
  #plot grid by state and with indep. y-axis scalings
  facet_grid(vars(State), scales = "free_y") +
  labs(title = "Australian domestic tourism", y = "Overnight trips ('000)")
```



Australian domestic tourism

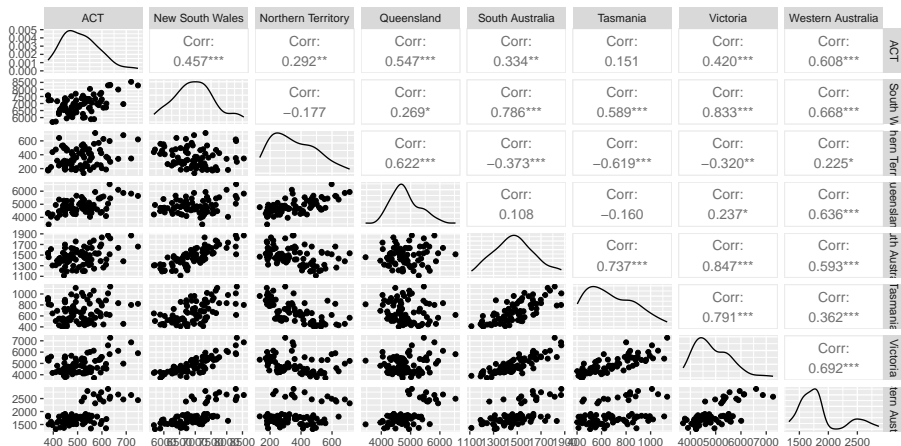




# Scatterplots (Multiple Series){plain}

We can visualize the correlation between multiple series using a scatterplot matrix. In R, we will use the `ggpairs()` command from the `GGally` package.

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

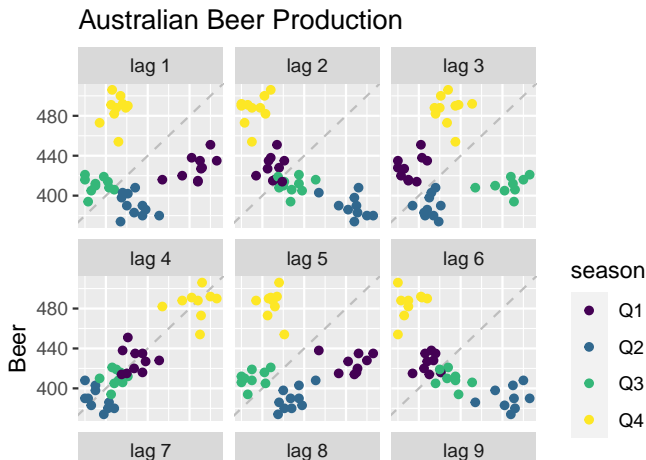


## Section 5

### Lag plots and autocorrelation

# Example: Beer production

```
recent_production <- aus_production %>%  
  filter(year(Quarter) >= 2000)  
recent_production %>% gg_lag(Beer, geom = "point") +  
  labs(x = "lag(Beer, k)", title = "Australian Beer Production")
```



# Lagged scatterplots

- Each graph shows  $y_t$  plotted against  $y_{t-k}$  for different values of  $k$ .
- Strongly positive relationships at lags 4 and 8, reflecting the strong seasonality in the data.
- The negative relationship at lags 2 and 6 occurs because peaks (in Q4) are plotted against troughs (in Q2).
- The autocorrelations are the correlations associated with these scatterplots.

**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

Let us denote the sample autocovariance at lag  $k$  as  $\gamma_k$  and the sample autocorrelation at lag  $k$  by  $\rho_k$ . Then define

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

$$\text{and} \quad \rho_k = \frac{\gamma_k}{\gamma_0}$$

- It is easy to see that  $\gamma_0$  is actually the variance of  $y$ . Let  $k = 0$  then:

$$\gamma_0 = \frac{1}{T} \sum_{t=1}^T (y_t - \bar{y})(y_t - \bar{y}) = \frac{1}{T} \sum_{t=1}^T (y_t - \bar{y})^2$$

- $\rho_1$  indicates how successive values of  $y$  relate to each other
- $\rho_2$  indicates how  $y$  values two periods apart relate to each other
- $\rho_k$  is *almost* the same as the sample correlation between  $y_t$  and  $y_{t-k}$ .

# Autocorrelation

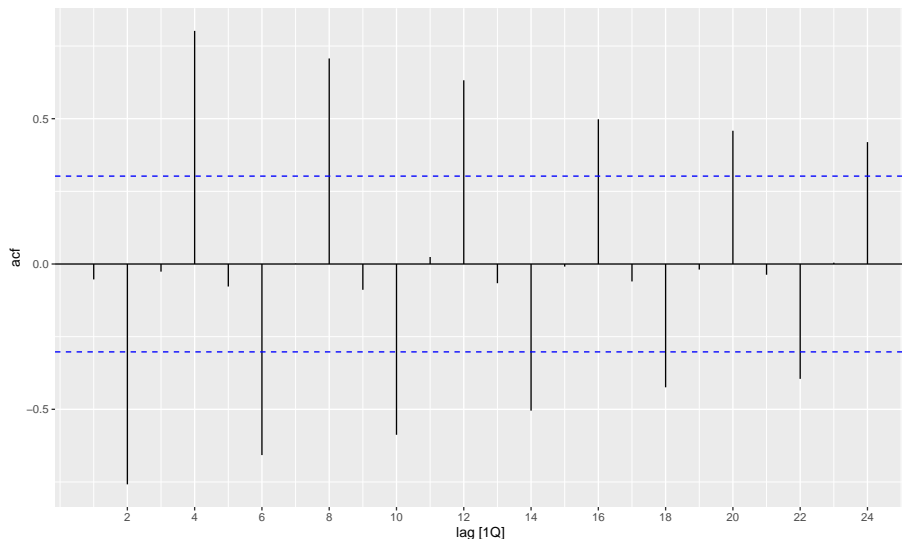
Results for first 9 lags for the Beer production data:

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

```
## # A tibble: 9 x 2 [1Q]
##       lag      acf
##   <cf_lag>    <dbl>
## 1      1Q -0.0530
## 2      2Q -0.758
## 3      3Q -0.0262
## 4      4Q  0.802
## 5      5Q -0.0775
## 6      6Q -0.657
## 7      7Q  0.00119
## 8      8Q  0.707
## 9      9Q -0.0888
```

# Autocorrelation

```
recent_production %>% ACF(Beer, lag_max = 24) %>% autoplot()
```





We again have confirmation that:

- $\rho_4$  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.
- $\rho_2$  is more negative than for the other lags because troughs tend to be 2 quarters behind peaks.
- Together, the autocorrelations at lags 1, 2, ..., make up the **autocorrelation function** or ACF.
- The plot is known as a **correlogram**

# Trend and seasonality in ACF plots

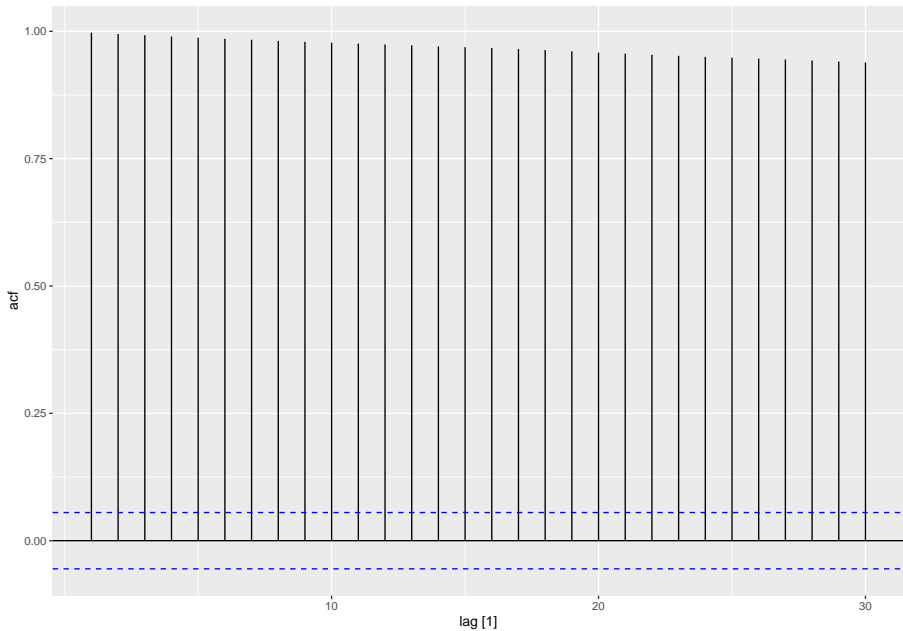
- When data have a trend, the autocorrelations for small lags tend to be large (near 1) and positive.
- When data are seasonal, the autocorrelations will be larger at the seasonal lags (i.e., at multiples of the seasonal frequency)
- When data are trended and seasonal, you see a combination of these effects.

# Google Stock Prices

```
gafa_stock %>% filter(Symbol == "GOOG") %>% autoplot(Close)
```

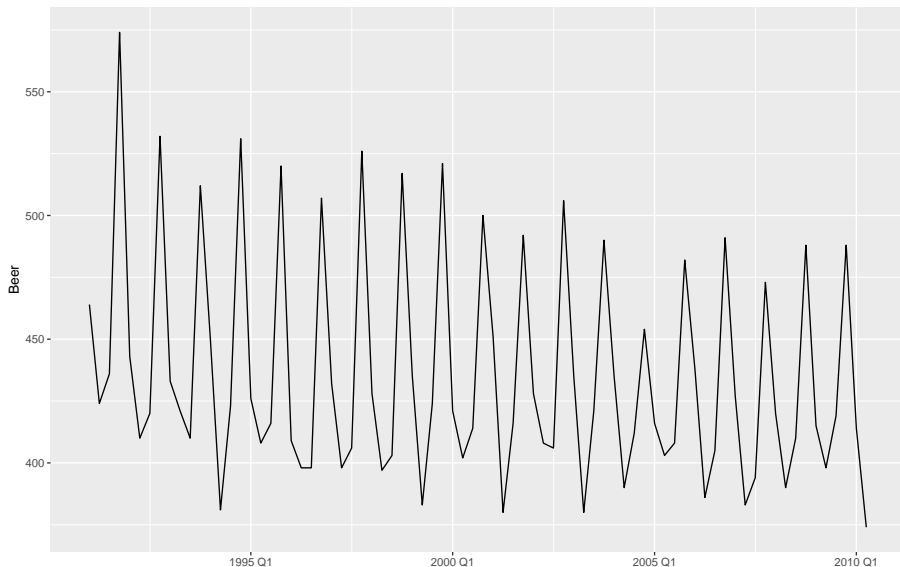


```
gafa_stock %>% filter(Symbol == "GOOG") %>% ACF(Close) %>% autoplot()
```

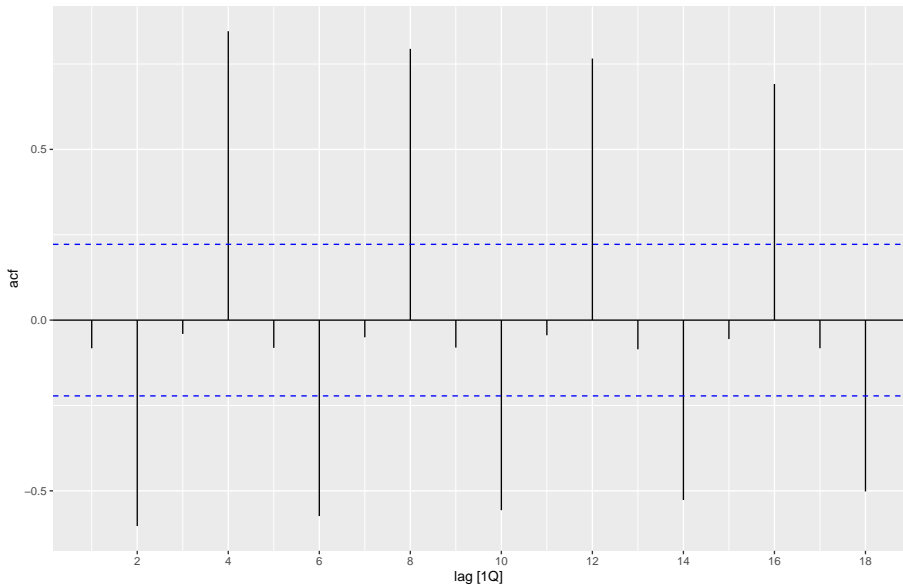


# Aus Beer production

```
aus_production %>% filter(year(Quarter)> 1990) %>% autoplot(Beer)
```

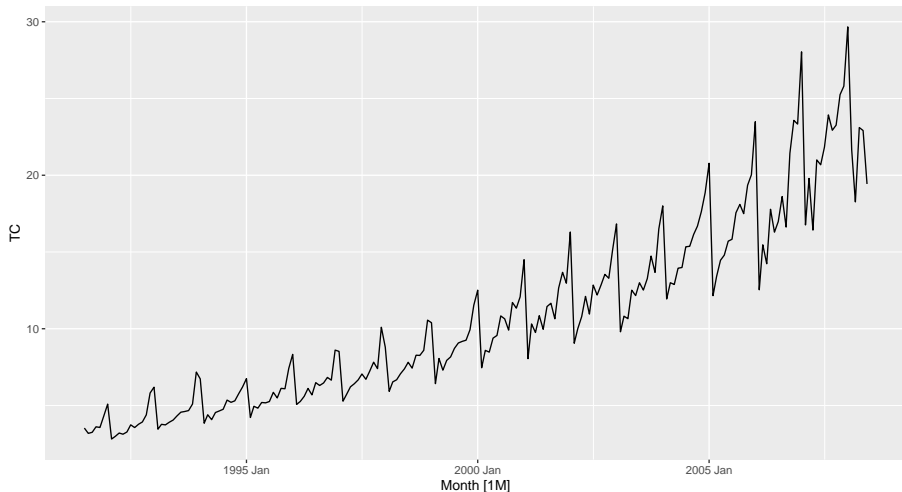


```
aus_production %>% filter(year(Quarter)> 1990) %>%  
  ACF(Beer) %>% autoplot()
```

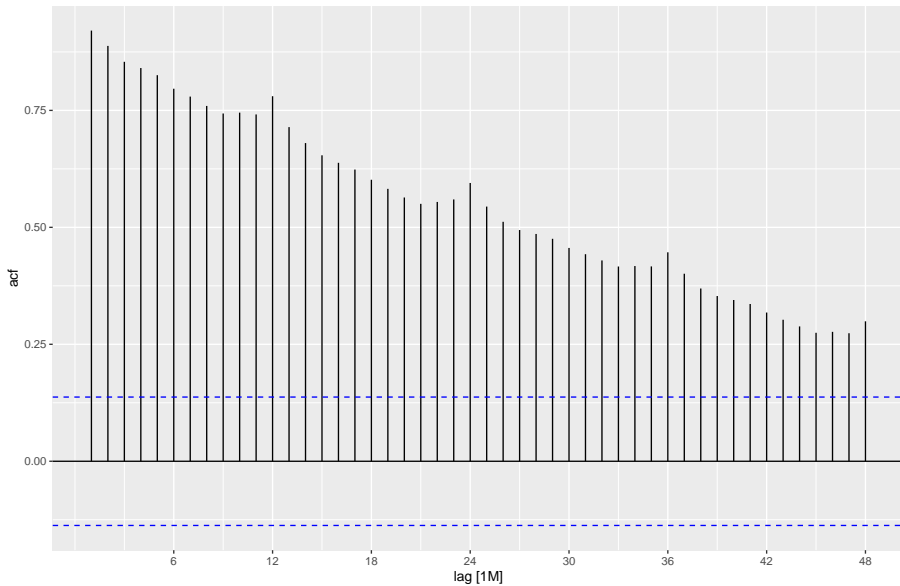


# Aus Antidiabetic Sales

```
a10 %>% autoplot(TC)
```



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





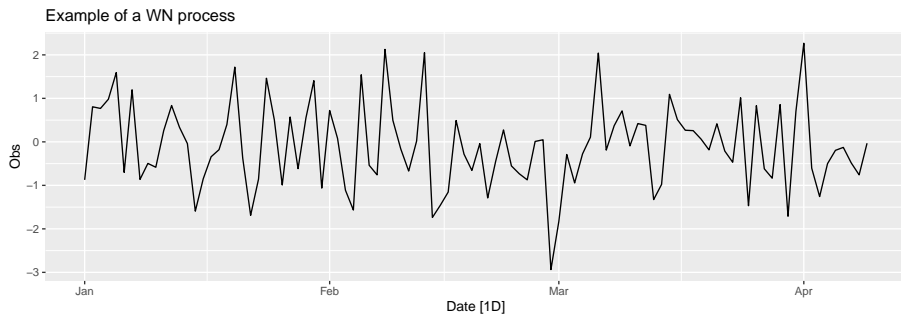
## Section 6

### White Noise Process

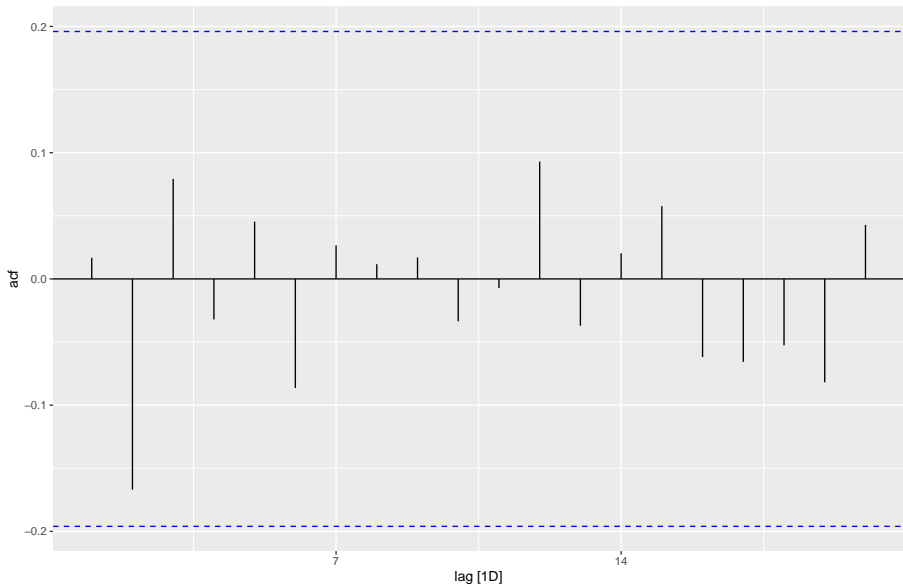
# White Noise Process

A time series that displays no autocorrelation is called **white noise**.

```
set.seed(1256)
x <- tsibble(Date = as.Date("2020-01-01") + 0:99,
              Obs = rnorm(100), index = Date)
x %>% autoplot(Obs) + labs(title = "Example of a WN process")
```

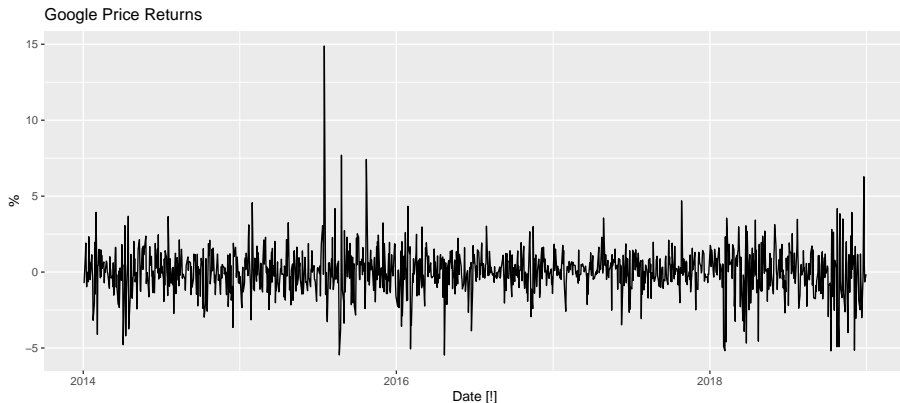


```
x %>% ACF(0bs) %>% autoplot()
```



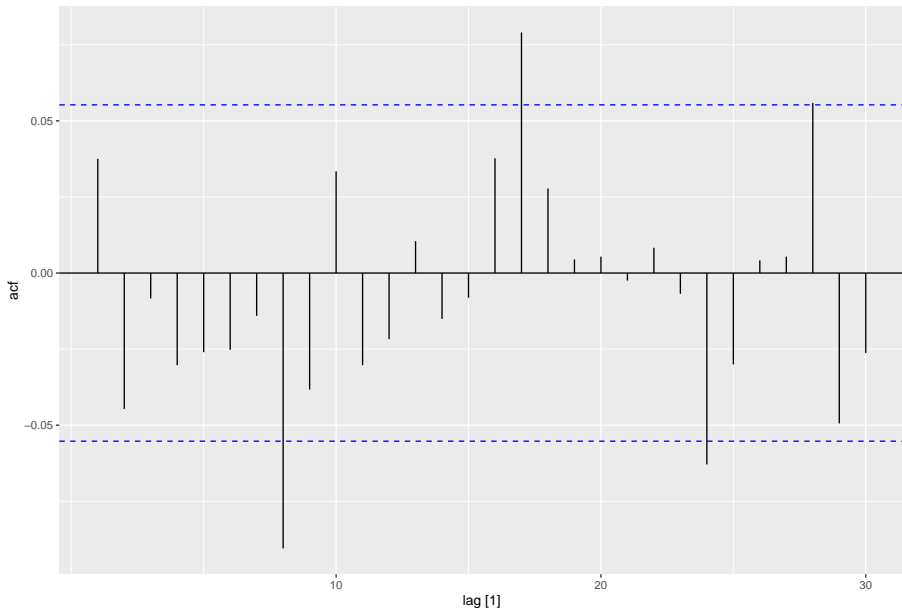
# Google Returns

```
dgoog <- gafa_stock %>% filter(Symbol == "GOOG") %>%  
  select(Date,Close) %>%  
  mutate(l.price = log(Close),  
         return = c(NA,diff(l.price))*100)  
  
dgoog %>% autoplot(return) + labs(title = "Google Price Returns",  
                                 y = "%")
```



```
dgoog %>% ACF(return) %>% autoplot() +  
  labs(title = "ACF: Google Price Returns")
```

ACF: Google Price Returns



# WN: What to look for

- We expect each autocorrelation to be **close to zero**.
  - They will not be exactly equal to zero as there is some random variation.
- We expect 95% of the spikes in the ACF to lie within  $\pm 2/\sqrt{T}$ , where  $T$  is the length of the time series (number of observations).
- These bounds are represented as the blue dashed lines above. If
  - i. one or more large spikes are outside these bounds, or
  - ii. if substantially more than 5% of spikes are outside these bounds, then the series is probably not white noise.

# White Noise Processes: Formally

A sequence  $\{y_t\}$  is a **white noise** process if each value in the sequence has:

- 1 A mean of zero, such that  $\mathbf{E}(y_t) = 0, \forall t$
- 2 A constant (finite) variance, such that  $\mathbf{var}(y_t) = \sigma^2, \forall t$
- 3 Is uncorrelated with all other realizations (past values), i.e. There is no serial correlation. Such that  $\rho_k = 0, \forall k \neq 0$

A slightly stronger condition is that they are independent from one another; this is an “independent white noise process.”

Note: 1. and 2. are under the assumption that  $\{y_t\}$  is normally distributed.

**Otherwise, each value in the sequence should have a finite mean and variance.**

# Do you Agree??

A linear combination of white noises **IS necessarily** also a White noise sequence.



# Let's Practice!!!

In this lesson, we explored the following graphics functions:

- `autoplot`
- `gg_season`
- `gg_subseries`
- `gg_lag`
- `ACF`

Using these functions, as appropriate, explore the meat production in Australia dataset, `aus_livestock`. Your interest is in the **Pigs** slaughtered in **Western Australia**.

- You might need to first view the data to get familiar with the structure.
- The `distinct()` function should come in handy.
- Can you spot any seasonality, cycle, and trend? What do you learn about the series?