Chapter 9 ARIMA models

Ankit Gupta

18/01/2023

ARIMA models provide another approach to time series forecasting. Exponential smoothing and ARIMA models are the two most widely used approaches to time series forecasting, and provide complementary approaches to the problem. While exponential smoothing models are based on a description of the trend and seasonality in the data, ARIMA models aim to describe the autocorrelations in the data.

Before we introduce ARIMA models, we must first discuss the concept of stationarity and the technique of differencing time series.

9.1 Stationarity and differencing

Stationarity

Definition

If $\{y_t\}$ is a *stationary time series* then for all s, the distribution of $(y_t, ..., y_{t+s})$ does not depend on t.

So a stationary time series is one for which when we look at different parts of the time series, the joint probability density of those different parts of the time series will look almost identical.

So, if we take 2 parts of the time series graph and make the histogram for both then they look almost identical. Hence data may not be the same but density or probability distribution will look very very similar.

Features

A stationary time series is:

- roughly horizontal (no trend)
- constant variance (homosadestic)
- no patterns predictable in long-term (no seasonality)
- Transformation helps to *stabilize the variance*
- For ARIMA modelling, we also need to *stabilize the mean.*

Identifying non-stationary series

- time plot.
- The ACF of stationary data drops to zero relatively quickly
- The ACF of non-stationary data decreases slowly
- For non-stationary data, the value of r_1 (first order correlation) is often large and positive.

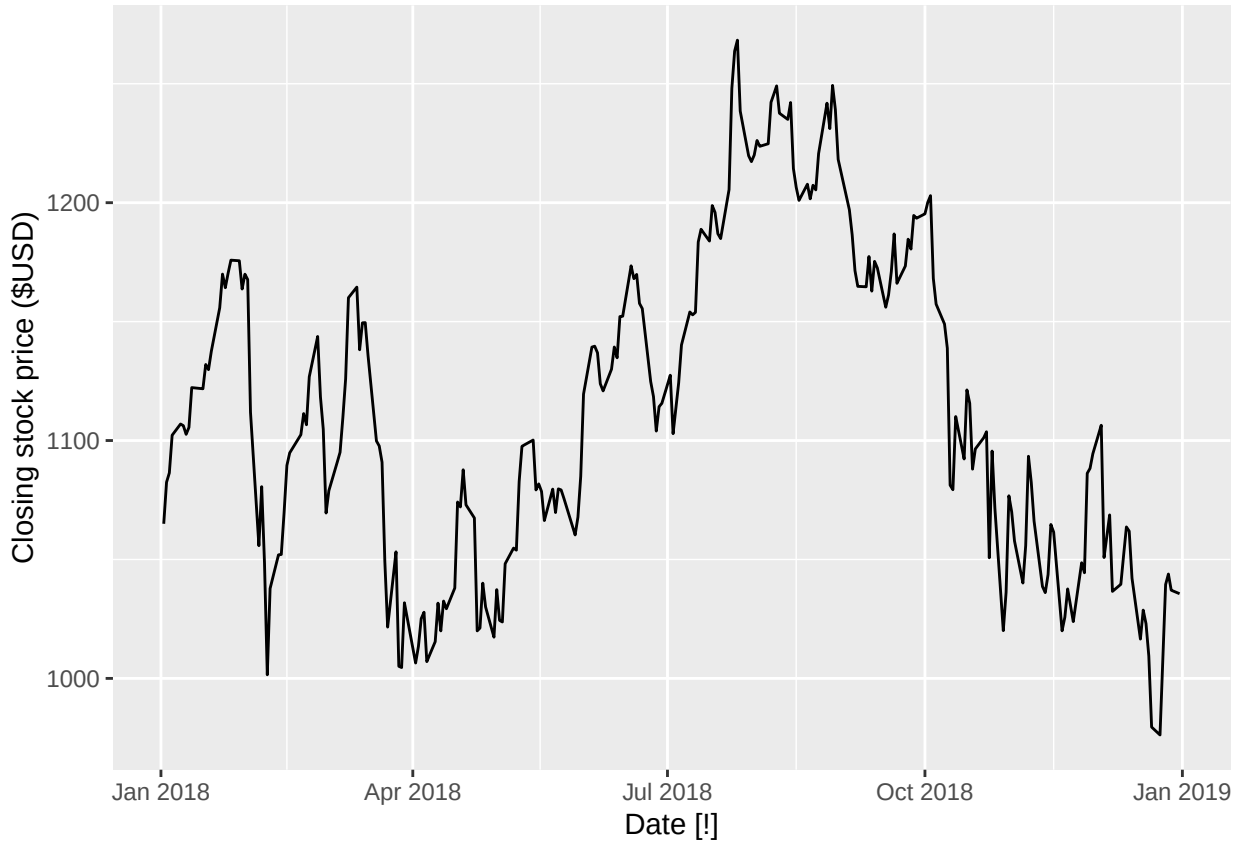
Example: Google Stock Price

```
library(fpp3)
```

```
## -- Attaching packages --------------------------------------- fpp3 0.5 --

## v tibble      3.2.1     v tsibble     1.1.3
## v dplyr       1.1.3     v tsibbledata 0.4.1
## v tidyr       1.3.0     v feasts      0.3.1
## v lubridate   1.9.3     v fable       0.3.3
## v ggplot2     3.4.4     v fabletools  0.3.4

## -- Conflicts ------------------------------------------ fpp3_conflicts --
## x lubridate::date()    masks base::date()
## x dplyr::filter()      masks stats::filter()
## x tsibble::intersect() masks base::intersect()
## x tsibble::interval()  masks lubridate::interval()
## x dplyr::lag()         masks stats::lag()
## x tsibble::setdiff()   masks base::setdiff()
## x tsibble::union()     masks base::union()
```

```
google_2018 <- gafa_stock |>
  filter(Symbol == "GOOG", year(Date) == 2018)
```

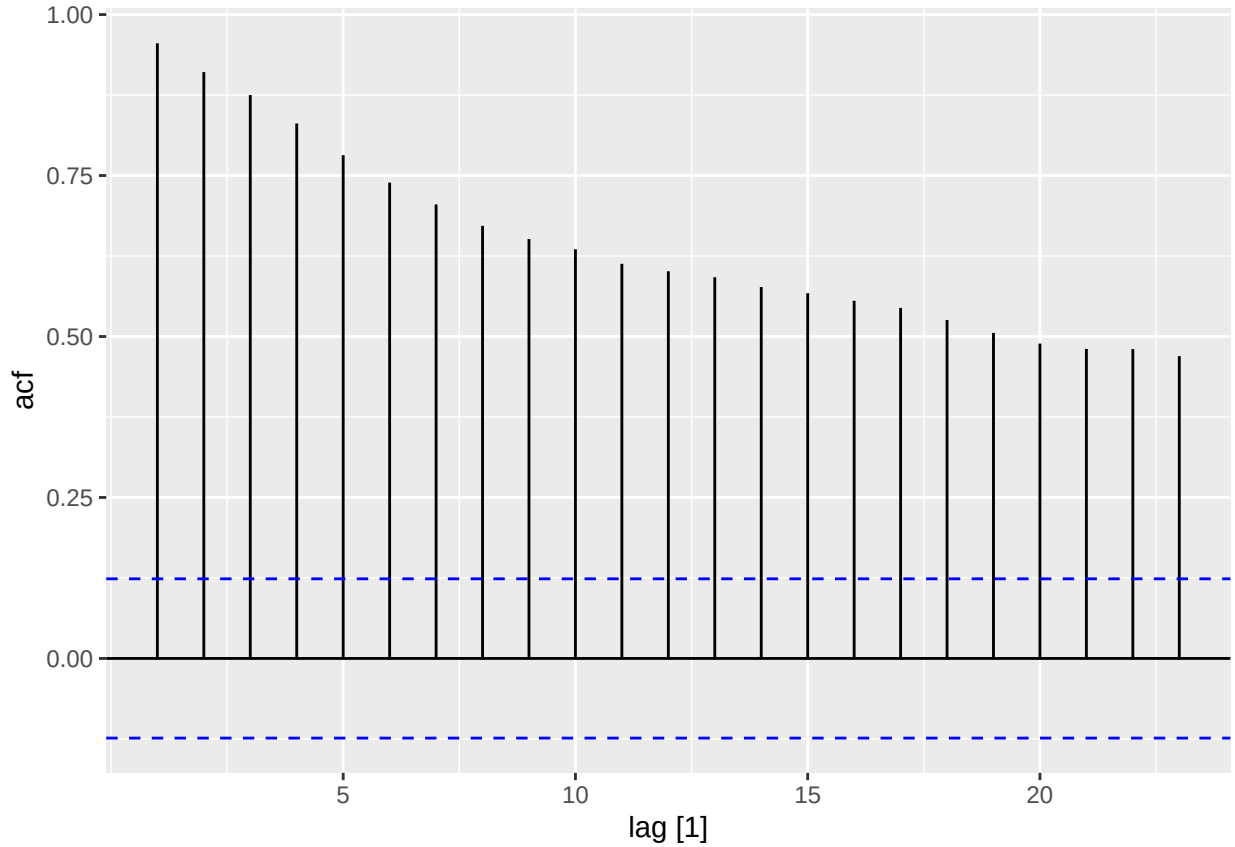
```
google_2018 |>
  autoplot(Close) +
  labs(y="Closing stock price ($USD)")
```



Here for different window, it looks different and this is typical wandering behavior of financial time series.

```
google_2018 |>
  ACF(Close) |>
  autoplot()
```

```
## Warning: Provided data has an irregular interval, results should be treated
## with caution. Computing ACF by observation.
```

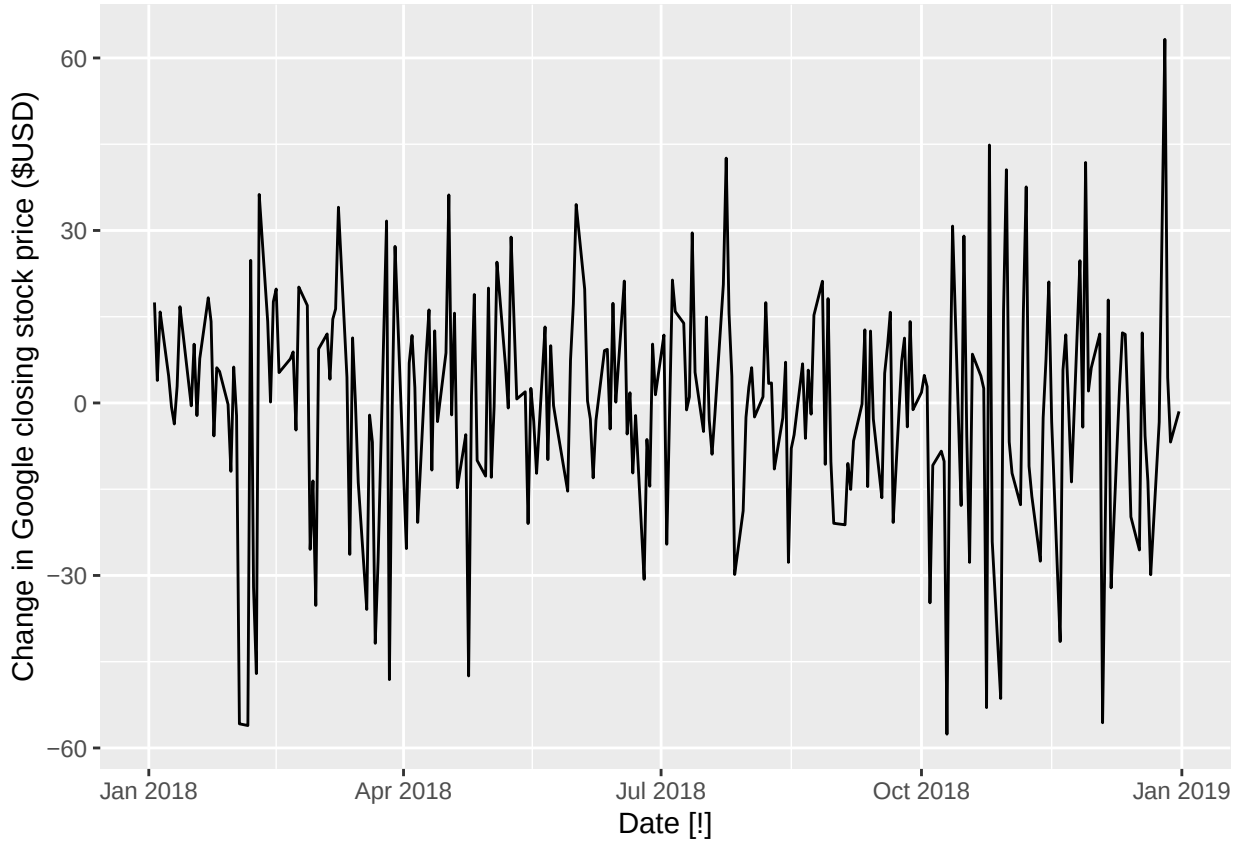


It is very slowly decaying, so it is non-stationary.

If we see the change in price:

```
google_2018 |>
  autoplot(difference(Close)) +
  labs(y= "Change in Google closing stock price ($USD)")
```

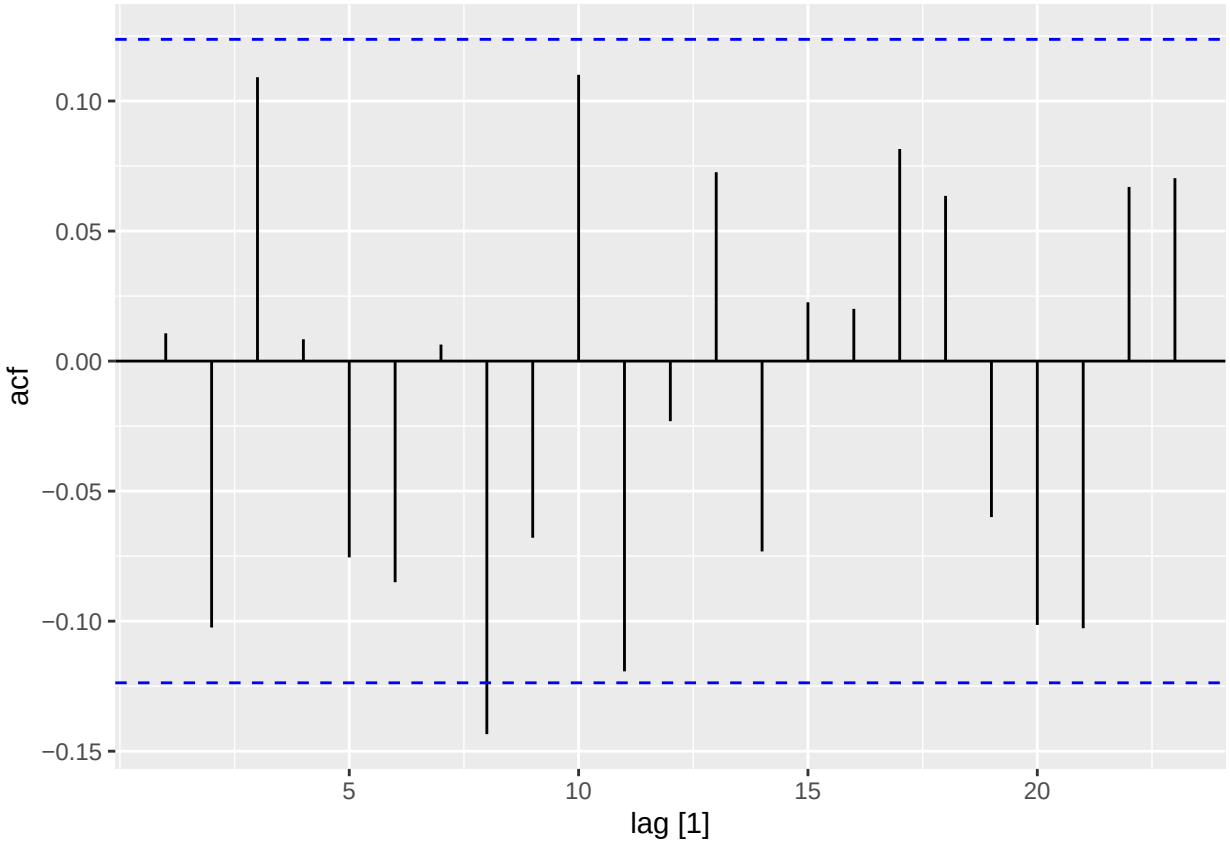
```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```



It is $y'_t = y_t - y_{t-1}$. It looks stationary i.e. flat, horizontal, no trending behaviour and no seasonality, now if we see the ACF:

```
google_2018 |>
  ACF(difference(Close)) |>
  autoplot()
```

```
## Warning: Provided data has an irregular interval, results should be treated
## with caution. Computing ACF by observation.
```



Here, we get very non-eventful ACF, so there is not so many spikes. So, this data is stationary.

Differencing

- Differencing helps to *stabilize the mean.*
- The differenced series is the *change* between each observation in the original series: $y'_t = y_t - y_{t-1}$.
- The differenced series will have *only T-1 values* since it is not possible to calculate difference y'_1 for the first observation.

Second order differencing Sometimes first order difference is not enough to stabilize the mean hence we may want to take the second order difference. So,

$y''_t = y'_t - y'_{t-1} = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2y_{t-1} + y_{t-2}$

- y''_t have $T-2$ values.
- In practice, it is almost never necessary to go beyond second order differences.

Seasonal differencing

- A *seasonal difference* is the difference between an observation and the corresponding observation from the previous year.

$y'_t = y_t - y_{t-m}$

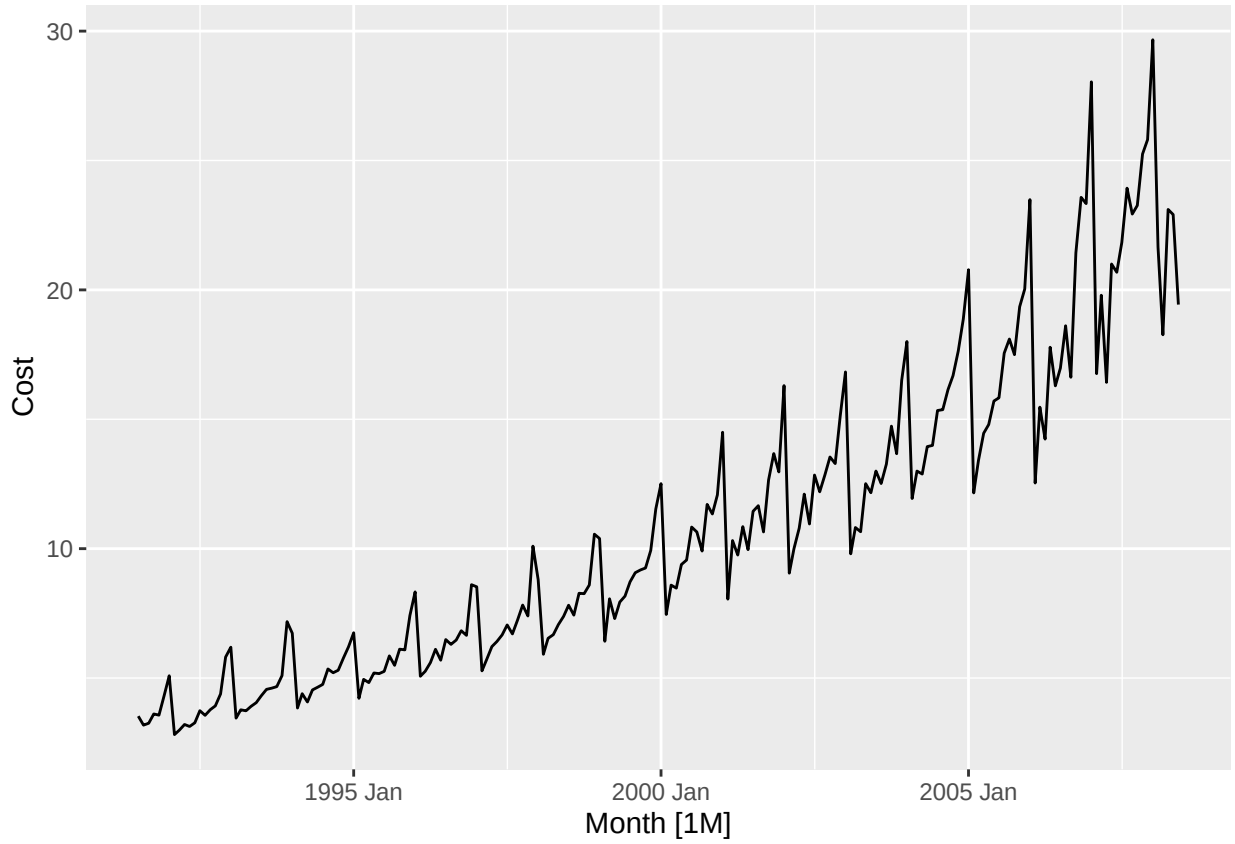
where m is the number of seasons. For monthly data, $m = 12$ and for quarterly data, $m = 4$ and seasonal difference series have $T - m$ observations.

- Seasonal differencing can help us to stabilize the mean also and deal with the seasonality.

Example: Antidiabetic drug sales

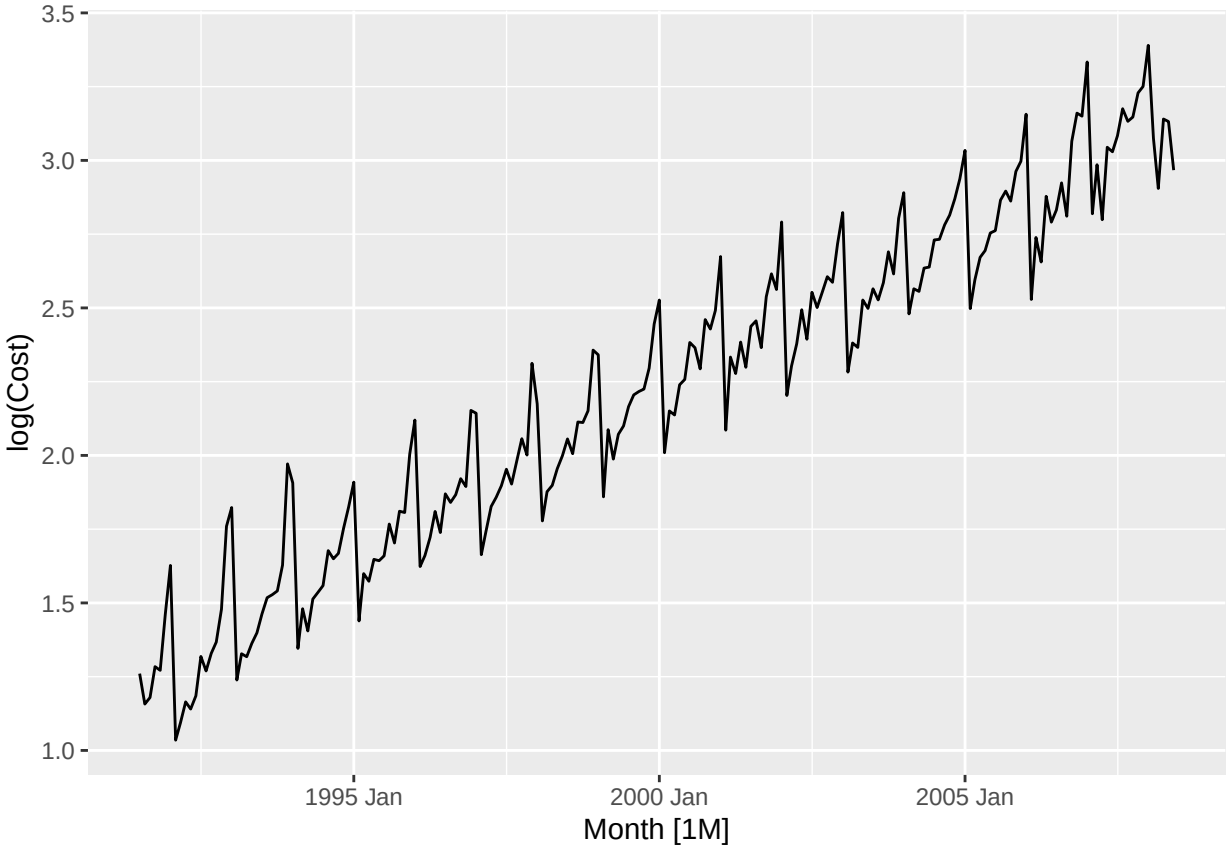
```
a10 <- PBS |>
  filter(ATC2 == "A10") |>
  summarise(Cost = sum(Cost)/1e6)
```

```
a10 |>
  autoplot(Cost)
```



Clearly this is non-stationary data, so the variance increases as the level increases. To deal with it, we will use the log-transformation.

```
a10 |>
  autoplot(log(Cost))
```

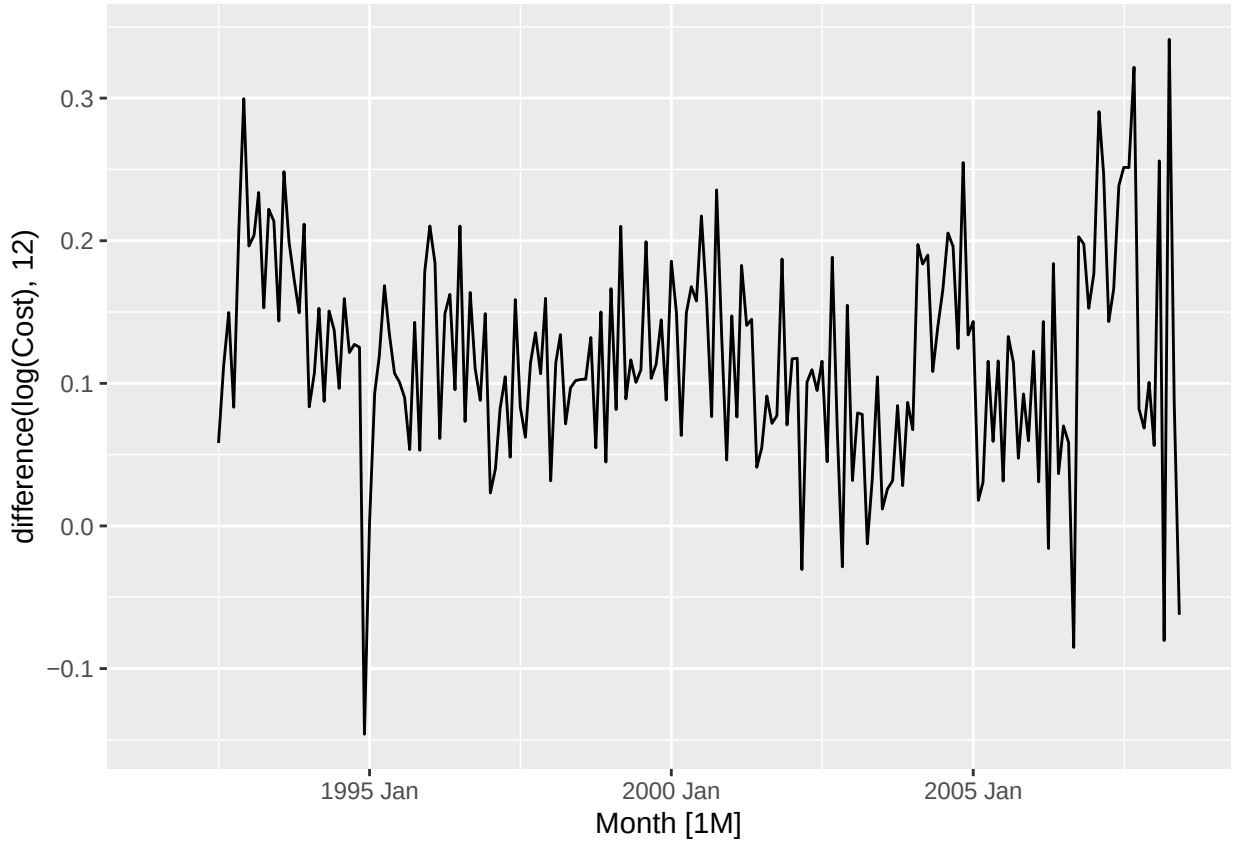


It is still non-stationary. We still have trend and seasonality.

Let's deal with the seasonal component.

```
a10 |>
  autoplot(log(Cost) |> difference(12))
```

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```

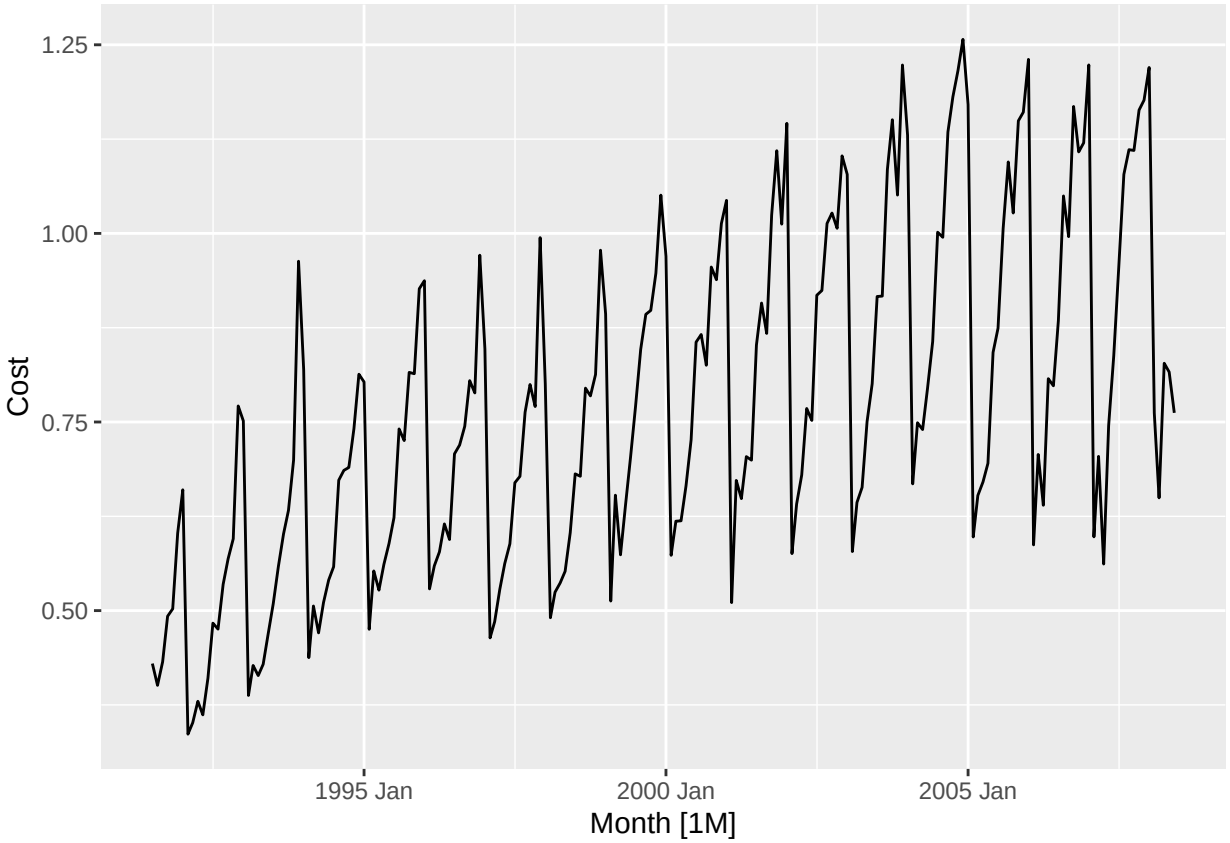


Here, difference(12) means $y_t - y_{t-12}$. So, we are taking the difference of 12 observations. Now, this time series looks fairly stationary.

Another Example

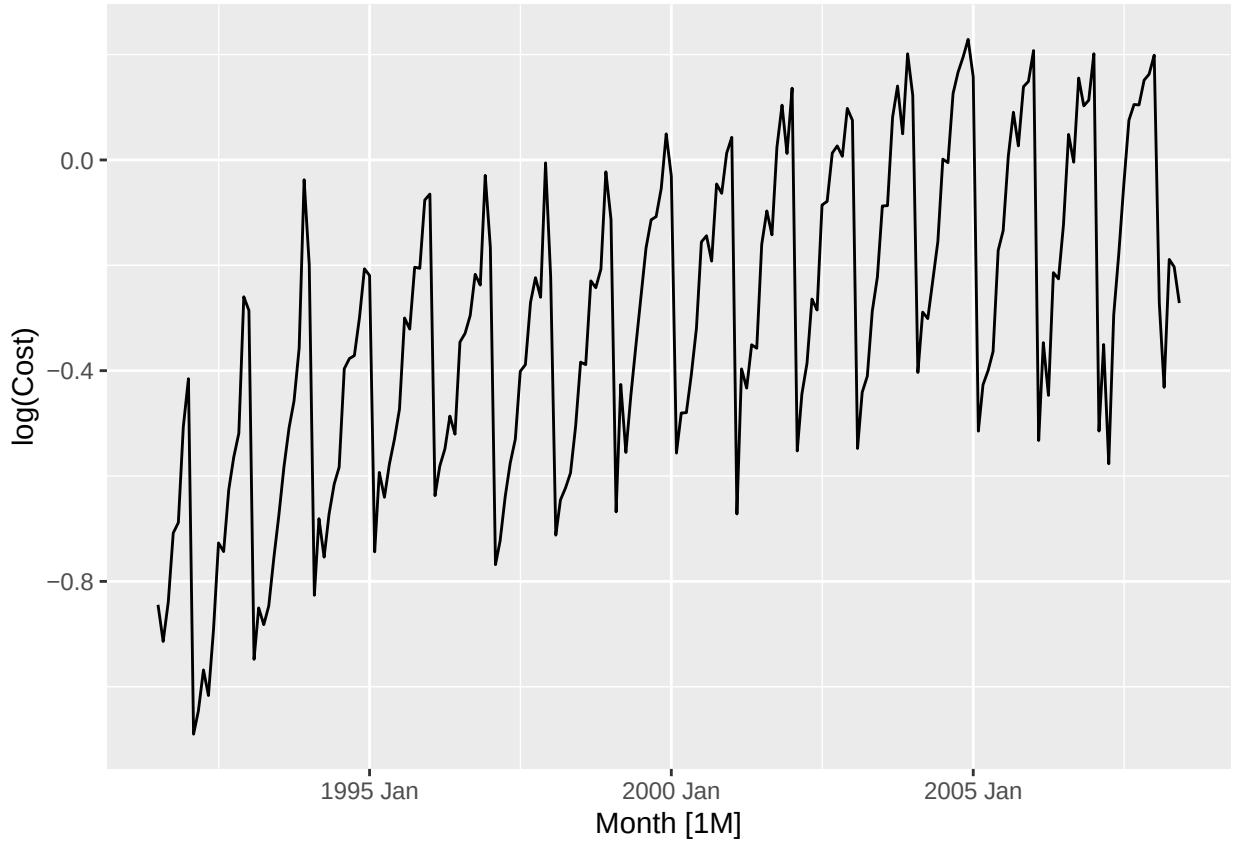
```
h02 <- PBS |>
  filter(ATC2 == "H02") |>
  summarise(Cost = sum(Cost)/1e6)
```

```
h02 |> autoplot(Cost)
```



Again it seems variance is increasing as level is increasing, hence we take a log-transformation.

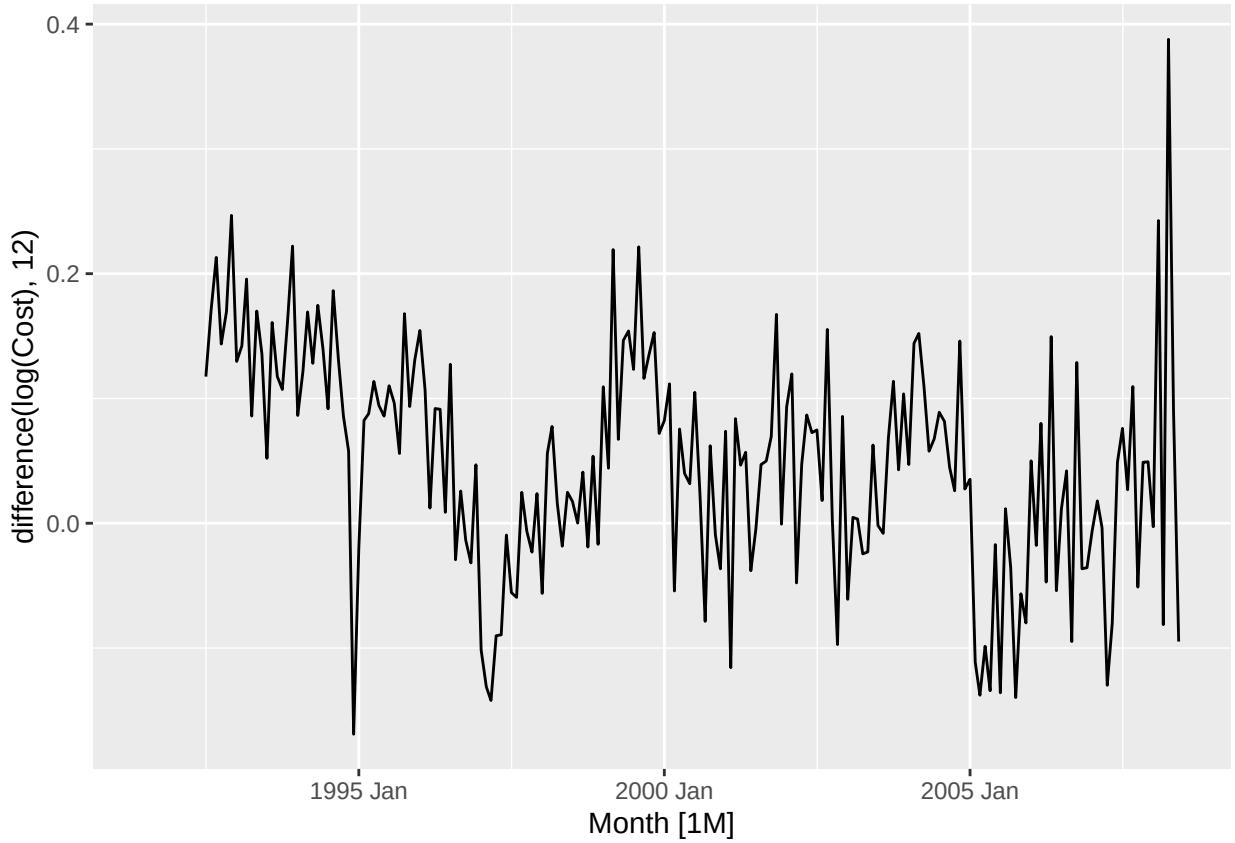
```
h02 |> autoplot(log(Cost))
```



It is still non-stationary. It has a seasonal and trending component, so taking a seasonal difference.

```
h02 |> autoplot(log(Cost) |> difference(12))
```

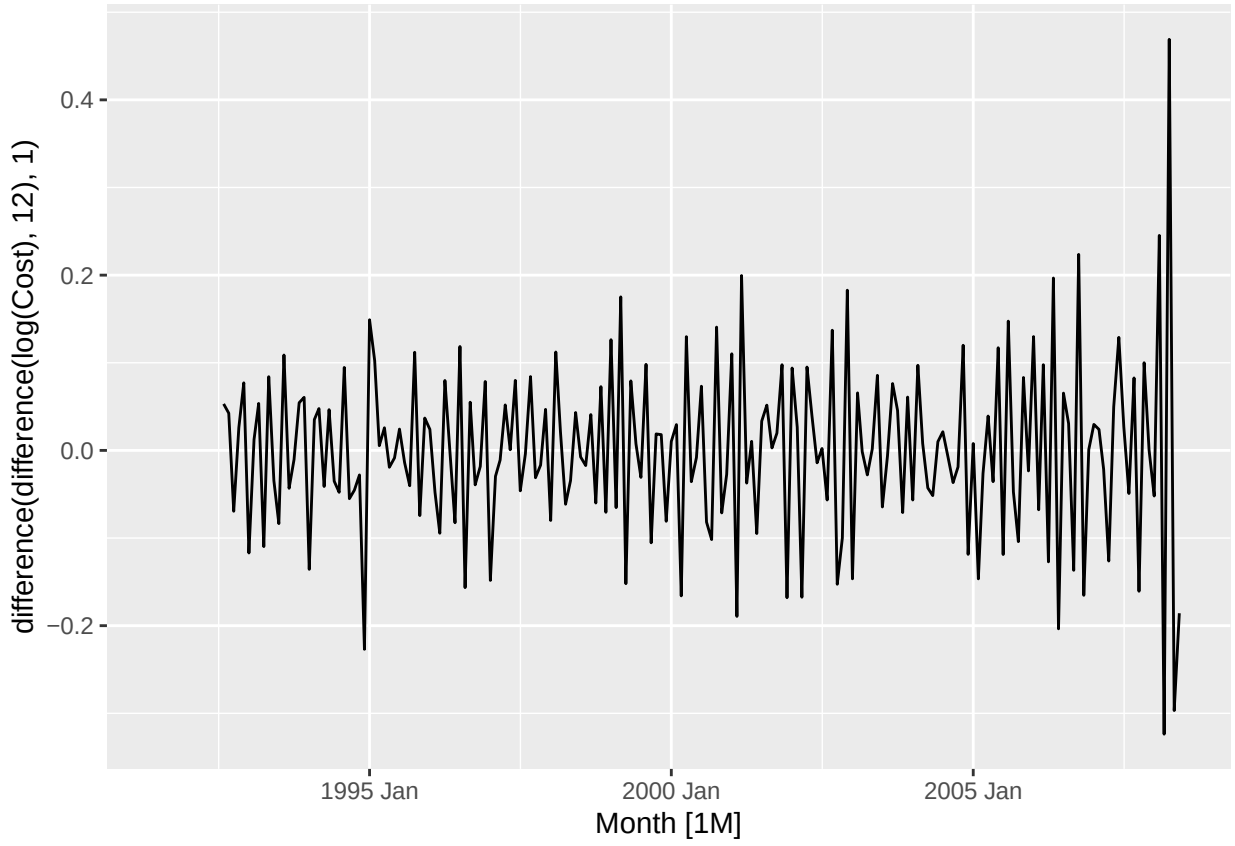
```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```



Here also we can see the trending behavior. Hence this is also not stationary. Now we take the first order difference.

```
h02 |> autoplot(log(Cost) |> difference(12) |> difference(1))
```

```
## Warning: Removed 13 rows containing missing values (`geom_line()`).
```



Now the data looks fairly stationary. Now, notice:

- Seasonally differenced series is closer to being stationary.
- Remaining non-stationarity can be removed with further list difference.

If $y'_t = y_t - y_{t-12}$ denotes seasoally differenced series then the twice difference series is:

$y^*_t = y'_t - y'_{t-1} = (y_t - y_{t-12}) - (y_{t-1} - y_{t-13}) = y_t - y_{t-1} - y_{t-12} - y_{t-13}$

When both seasonal and first differences are applied

- It makes no difference which is done first- the result will be same.
- If seasonality is strong, we recommend that seasonal differencing be done first because sometimes the resulting series will be stationary and there will no need for further first difference.

——————————— Summary ———————————-

A stationary time series is one whose statistical properties do not depend on the time at which the series is observed.18 Thus, time series with trends, or with seasonality, are not stationary — the trend and seasonality will affect the value of the time series at different times. On the other hand, a white noise series is stationary — it does not matter when you observe it, it should look much the same at any point in time.

Some cases can be confusing — a time series with cyclic behaviour (but with no trend or seasonality) is stationary. This is because the cycles are not of a fixed length, so before we observe the series we cannot be sure where the peaks and troughs of the cycles will be.

In general, a stationary time series will have no predictable patterns in the long-term. Time plots will show the series to be roughly horizontal (although some cyclic behaviour is possible), with constant variance.

Differencing

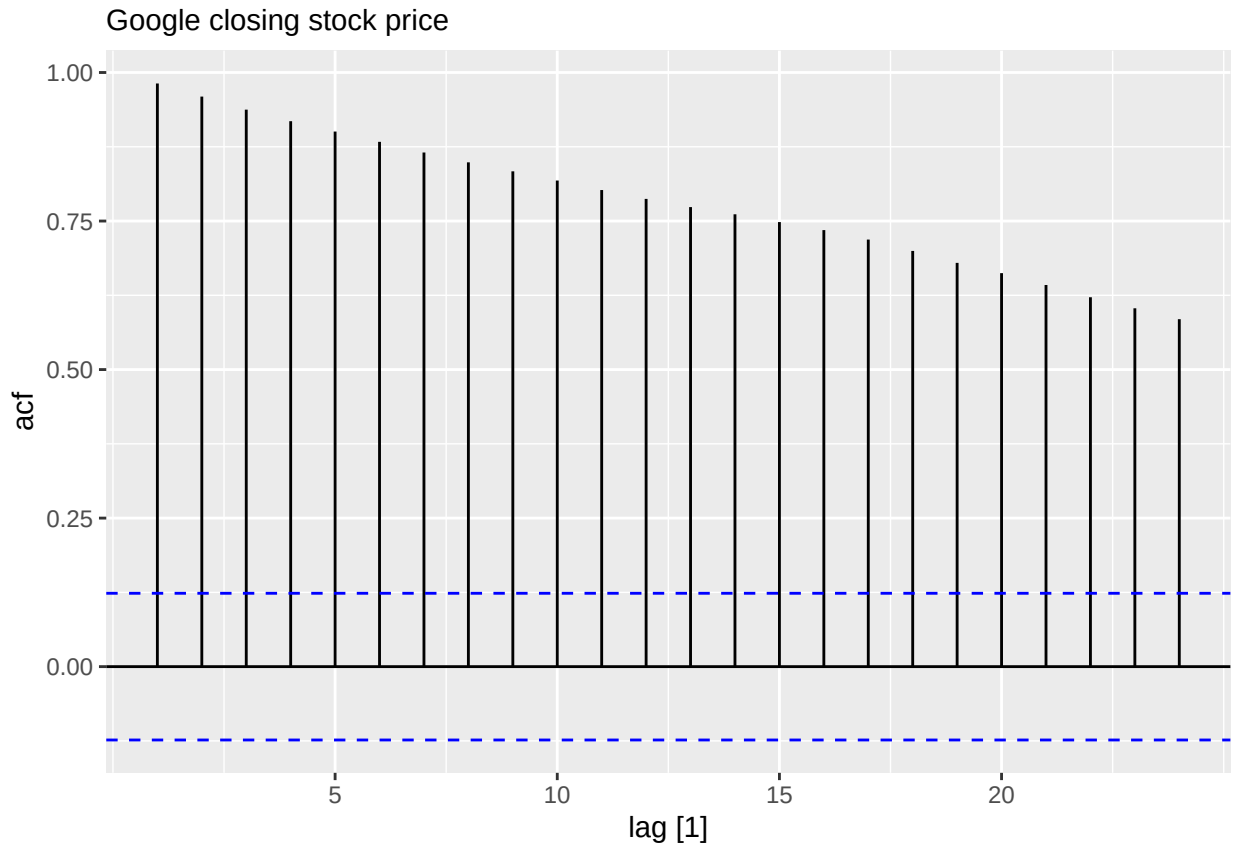
In Figure 9.1, note that the Google stock price was non-stationary in panel (a), but the daily changes were stationary in panel (b). This shows one way to make a non-stationary time series stationary — compute the differences between consecutive observations. This is known as differencing.

Transformations such as logarithms can help to stabilise the variance of a time series. Differencing can help stabilise the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality.

As well as the time plot of the data, the ACF plot is also useful for identifying non-stationary time series. For a stationary time series, the ACF will drop to zero relatively quickly, while the ACF of non-stationary data decreases slowly. Also, for non-stationary data, the value of r_1 is often large and positive.

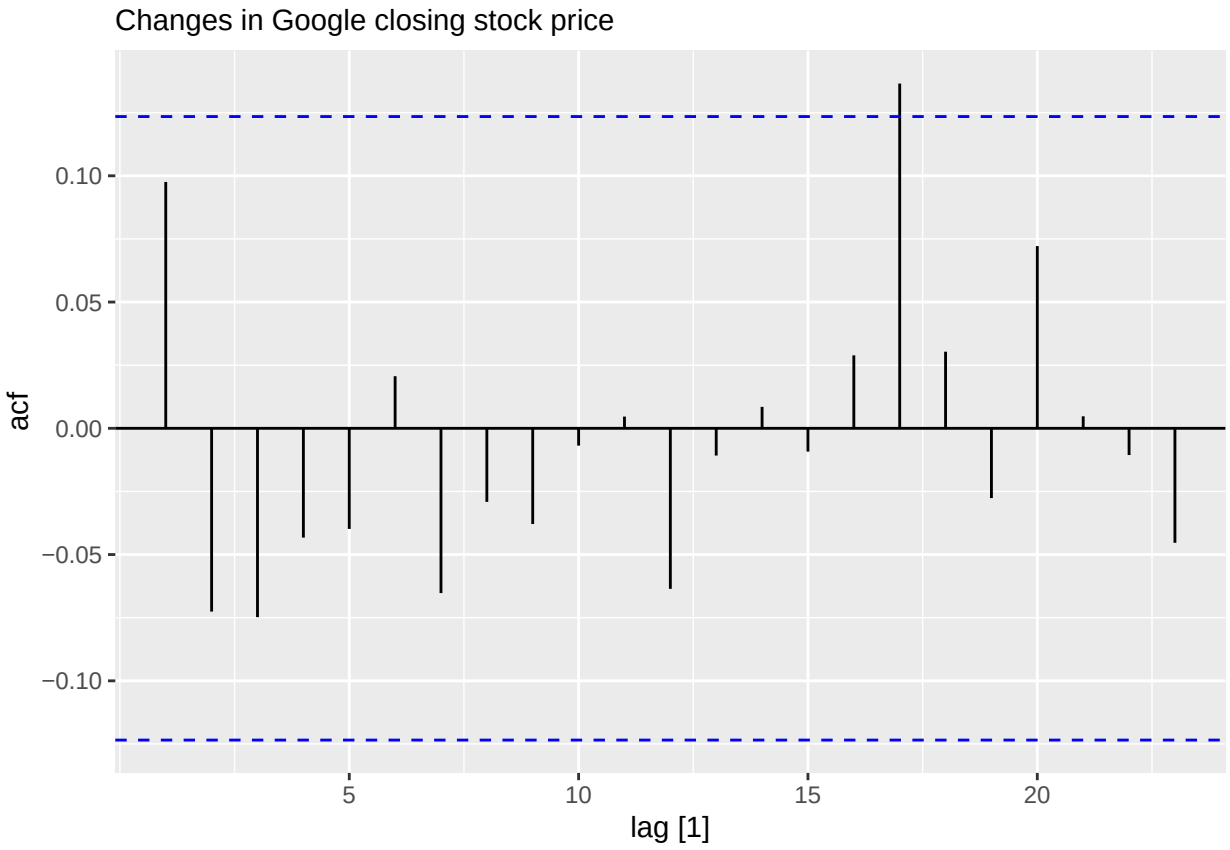
```
google_2015 <- gafa_stock |>
  filter(Symbol == "GOOG", year(Date) == 2015)
google_2015 |> ACF(Close) |>
  autoplot() + labs(subtitle = "Google closing stock price")
```

```
## Warning: Provided data has an irregular interval, results should be treated
## with caution. Computing ACF by observation.
```




```
google_2015 |> ACF(difference(Close)) |>
  autoplot() + labs(subtitle = "Changes in Google closing stock price")
```

```
## Warning: Provided data has an irregular interval, results should be treated
## with caution. Computing ACF by observation.
```




```
google_2015 |>
  mutate(diff_close = difference(Close)) |>
  features(diff_close, ljung_box, lag = 10)
```

```
## # A tibble: 1 x 3
##   Symbol lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 GOOG      7.91     0.637
```

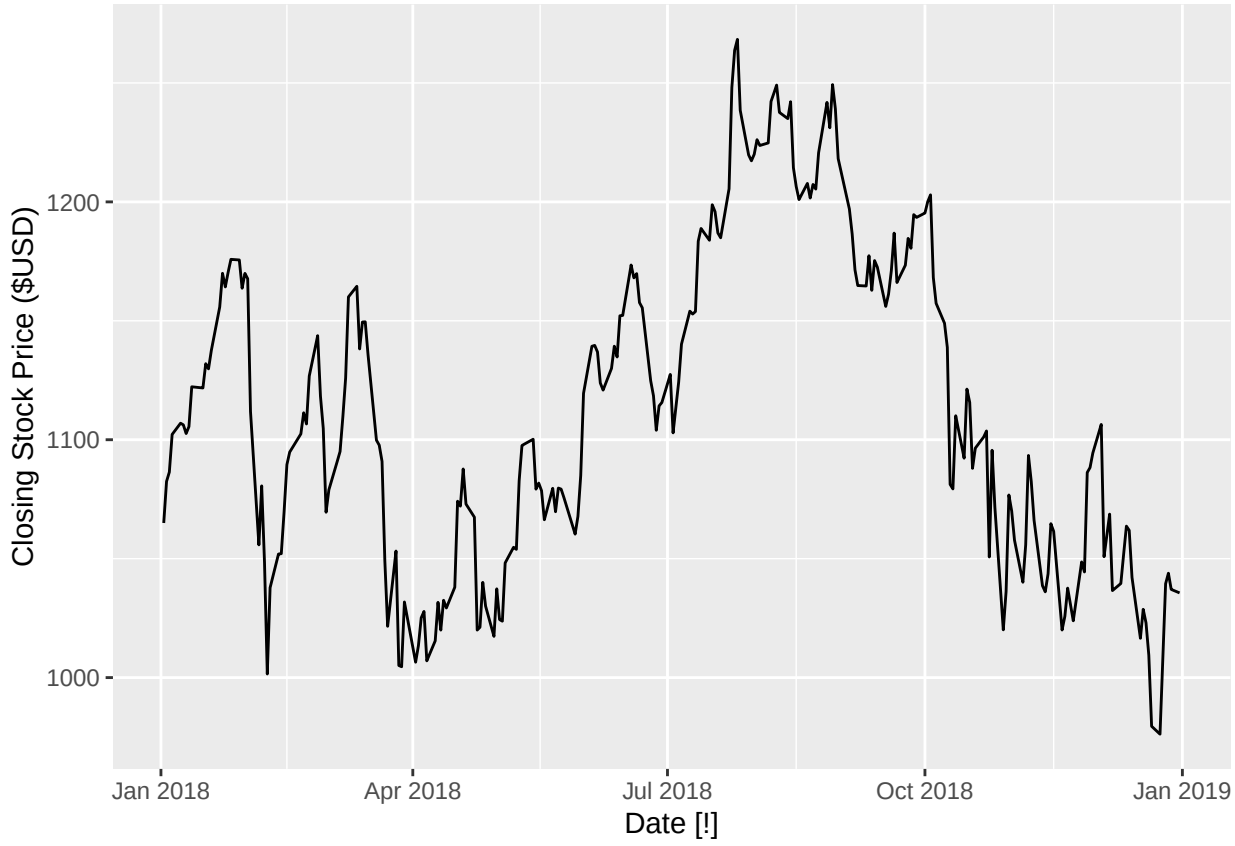
```
#> # A tibble: 1 × 3
#>   Symbol lb_stat lb_pvalue
#>   <chr>    <dbl>     <dbl>
#> 1 GOOG      7.91     0.637
```

The ACF of the differenced Google stock price looks just like that of a white noise series. Only one autocorrelation is outside of the 95% limits, and the Ljung-Box Q^* statistic has a p-value of 0.637 (for h=10). This suggests that the daily change in the Google stock price is essentially a random amount which is uncorrelated with that of previous days.

Random walk model

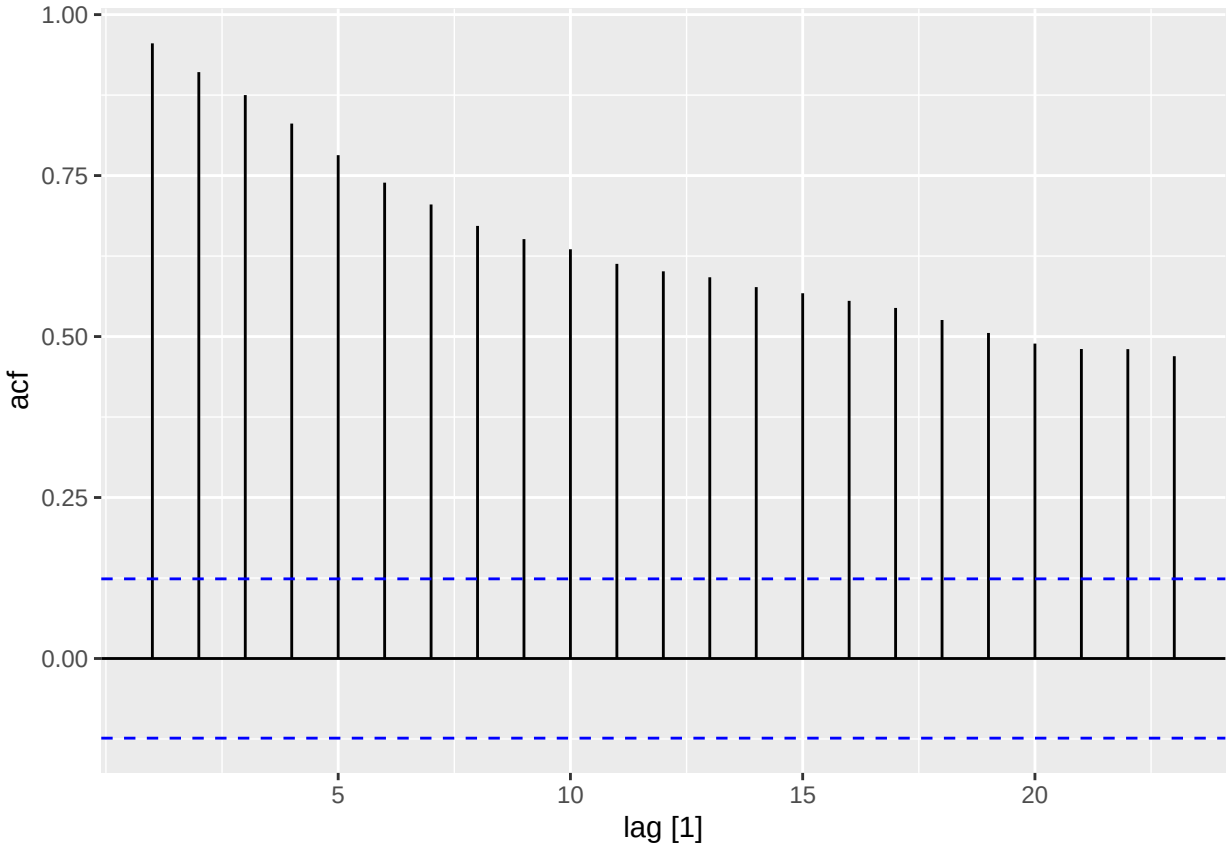
```
google_2018 <- gafa_stock |>
  filter(Symbol=="GOOG",year(Date)==2018)
```

```
google_2018 |>
  autoplot(Close)+
  labs(y="Closing Stock Price ($USD)")
```




```
google_2018 |>
  ACF(Close) |>
  autoplot()
```

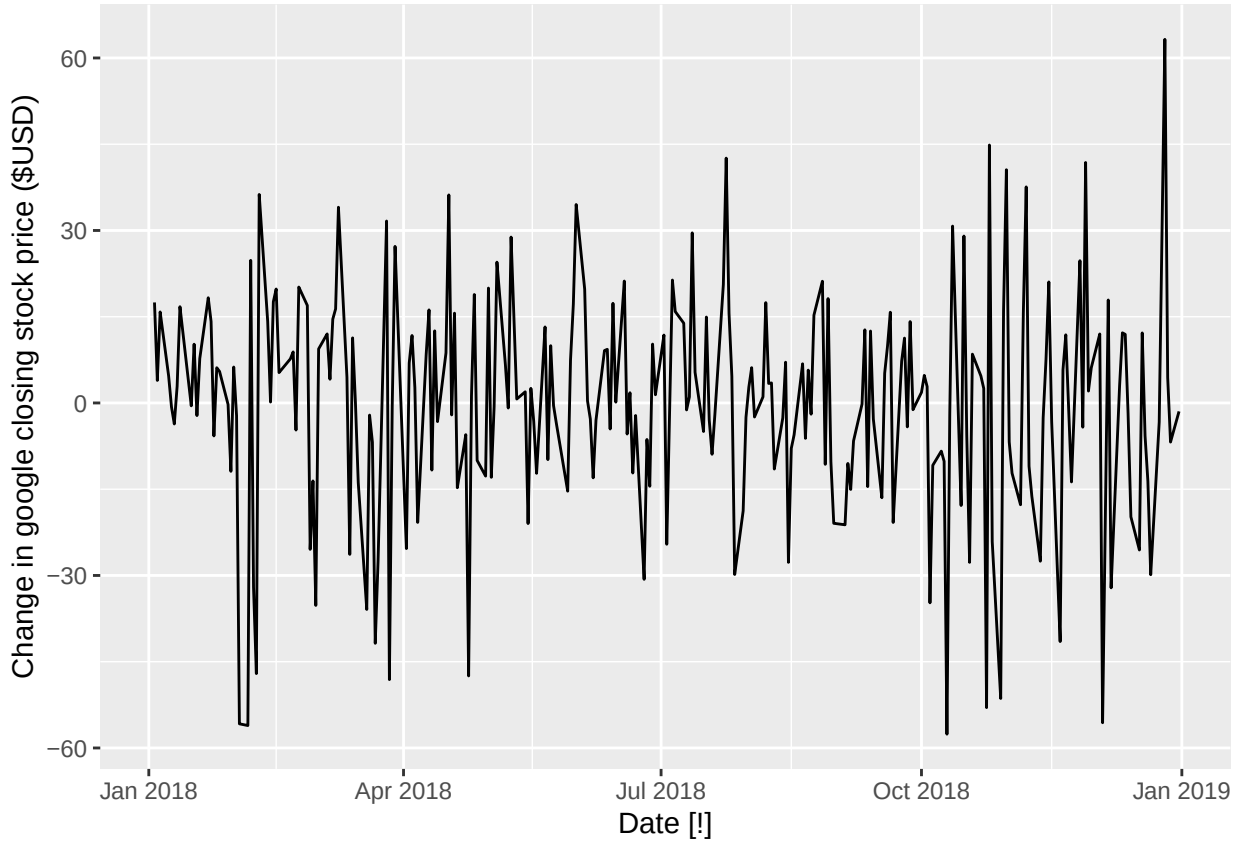
```
## Warning: Provided data has an irregular interval, results should be treated
## with caution. Computing ACF by observation.
```



It is slowly decaying so this time series is non-stationary.

```
google_2018 |>
  autoplot(difference(Close))+
  labs(y="Change in google closing stock price ($USD)")
```

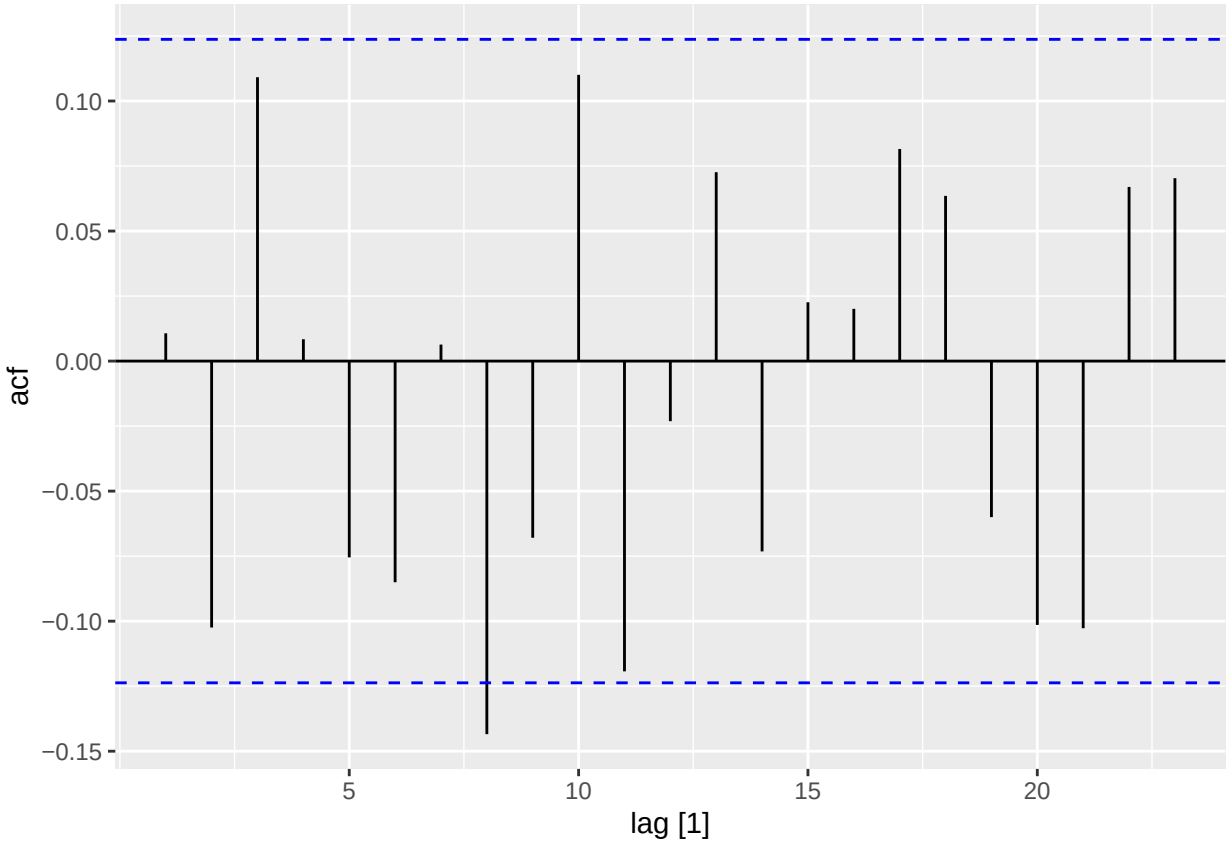
```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```



Now, we can see a different behavior and it is now stationary time series.

```
google_2018 |>
  ACF(difference(Close))|>
  autoplot()
```

```
## Warning: Provided data has an irregular interval, results should be treated
## with caution. Computing ACF by observation.
```



Now, we can conclude that this time series is not only stationary but is a white noise. We can ignore one spike due to probably type 1 error.

- The differences are *day-to-day* changes.
- Now, the series looks just like a white noise series and no significant autocorrelations are outside the 95% limits.

White Noise implies the Stationarity means a white noise process is always stationary but reverse is not true i.e. stationarity does not imply white noise.

- *Conclusion:* The daily change in Google stock price is essentially a random amount uncorrelated with previous days.

Random Walk Model

If differenced time series is white noise with zero mean:

$y_t - y_{t-1} = \epsilon_t$ or $y_t = y_{t-1} + \epsilon_t$

where $\epsilon_t \sim NID(0, \sigma^2)$

- Very widely used for non-stationary data.
- This is the model behind the *Naive Model*. We can show it as for $t = 1, 2, ..., T$ $E(y_{T+1|T}) = y_T + E(\epsilon_{T+1|T}) = y_T$ and similarly $E(y_{T+2|T}) = E(y_{T+1|T}) + E(\epsilon_{T+2|T}) = y_T$ and we can generalize it as $E(y_{T+n|T}) = y_T$
- Random walks typically have:

(1) long periods of apparent trends up or down

(2) Sudden/unpredictable changes in direction

Both (1) and (2) are generally refered to as *stochastic trend*

- Forecast are equal to last observation (naive)

(1) future movements up or down are equally likely.

Random walk model with drift model Now, if the difference series is white noise with non-zero mean:

$y_t - y_{t-1} = c + \epsilon_t$ or $y_t = y_{t-1} + c + \epsilon_t$

where $\epsilon_t \sim N(0, \sigma^2)$

- "c" is the *non-zero average change* between the consecutive observations.
- If $c > 0$, y_t will tend to drift upwards and vice-versa.
- This is the model behind the *drift method*

This model has a stochastic trend due to random walk and also deterministic trend due to the drift factor.

Random walk models are widely used for non-stationary data, particularly financial and economic data. Random walks typically have:

long periods of apparent trends up or down sudden and unpredictable changes in direction.

The forecasts from a random walk model are equal to the last observation, as future movements are unpredictable, and are equally likely to be up or down. Thus, the random walk model underpins naïve forecasts, first introduced in Section 5.2.

Summary -

Seasonal differencing

A seasonal difference is the difference between an observation and the previous observation from the same season. So

$y'_t = y_t - y_{t-m}$,

where m= the number of seasons. These are also called "lag- m differences", as we subtract the observation after a lag of m periods.

If seasonally differenced data appear to be white noise, then an appropriate model for the original data is

$y_t = y_{t-m} + \varepsilon_t$.

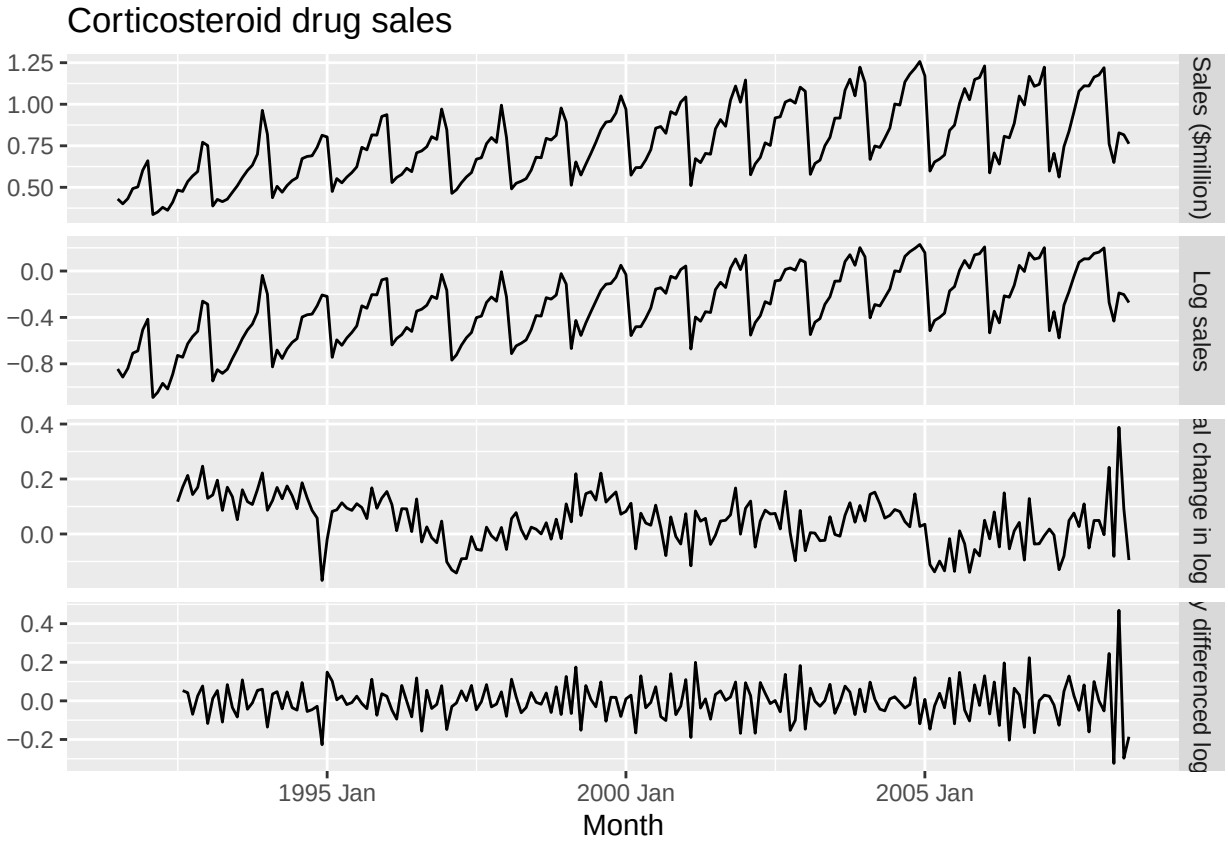
Forecasts from this model are equal to the last observation from the relevant season. That is, this model gives seasonal naïve forecasts, introduced in Section 5.2.

The bottom panel in Figure 9.3 shows the seasonal differences of the logarithm of the monthly scripts for A10 (antidiabetic) drugs sold in Australia. The transformation and differencing have made the series look relatively stationary.

To distinguish seasonal differences from ordinary differences, we sometimes refer to ordinary differences as "first differences", meaning differences at lag 1.

Sometimes it is necessary to take both a seasonal difference and a first difference to obtain stationary data. Figure 9.4 plots Australian corticosteroid drug sales ($AUD) (top panel). Here, the data are first transformed using logarithms (second panel), then seasonal differences are calculated (third panel). The data still seem somewhat non-stationary, and so a further lot of first differences are computed (bottom panel).

```
PBS |>
  filter(ATC2 == "H02") |>
  summarise(Cost = sum(Cost)/1e6) |>
  transmute(
    `Sales ($million)` = Cost,
    `Log sales` = log(Cost),
    `Annual change in log sales` = difference(log(Cost), 12),
    `Doubly differenced log sales` =
                    difference(difference(log(Cost), 12), 1)
  ) |>
  pivot_longer(-Month, names_to="Type", values_to="Sales") |>
  mutate(
    Type = factor(Type, levels = c(
      "Sales ($million)",
      "Log sales",
      "Annual change in log sales",
      "Doubly differenced log sales"))
  ) |>
  ggplot(aes(x = Month, y = Sales)) +
  geom_line() +
  facet_grid(vars(Type), scales = "free_y") +
  labs(title = "Corticosteroid drug sales", y = NULL)
```



There is a degree of subjectivity in selecting which differences to apply. The seasonally differenced data in Figure 9.3 do not show substantially different behaviour from the seasonally differenced data in Figure 9.4. In the latter case, we could have decided to stop with the seasonally differenced data, and not done an

extra round of differencing. In the former case, we could have decided that the data were not sufficiently stationary and taken an extra round of differencing. Some formal tests for differencing are discussed below, but there are always some choices to be made in the modelling process, and different analysts may make different choices.

When both seasonal and first differences are applied, it makes no difference which is done first—the result will be the same. However, if the data have a strong seasonal pattern, we recommend that seasonal differencing be done first, because the resulting series will sometimes be stationary and there will be no need for a further first difference. If first differencing is done first, there will still be seasonality present.

Beware that applying more differences than required will induce false dynamics or autocorrelations that do not really exist in the time series. Therefore, do as few differences as necessary to obtain a stationary series.

It is important that if differencing is used, the differences are interpretable. 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 make much interpretable sense and should be avoided.

Unit root tests

Statistical tests to determine the required order of differencing.

- 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.
- Other tests available for seasonal data.

The difference between the above two test is: ADF test assumes the NULL hypothesis as $H_0 : Non - Stationary$ and KPSS test assumes the NULL hypothesis as $H_0 : Stationary$

For econometric modelling of inference, ADF test is preferred because we are interested in controlling for the probability of type-1 error so rejecting H_0 while is true and we want α to be low.

In forecasting, we prefer the KPSS test, we don't want a difference unless we really have to unless we find a strong evidence of rejecting the null hypothesis of stationarity. So, in fable package, the default setting is to use the KPSS test.

Example

```
google_2018 %>%
  features(Close,unitroot_kpss)
```

```
## # A tibble: 1 x 3
##   Symbol kpss_stat kpss_pvalue
##   <chr>      <dbl>       <dbl>
## 1 GOOG       0.573      0.0252
```

Alternative way to find the number of differences:

```
google_2018 %>%
  features(Close, unitroot_ndiffs)
```

```
## # A tibble: 1 x 2
##   Symbol ndiffs
##   <chr>   <int>
## 1 GOOG        1
```

So, we need only 1 difference to make the data stationary.

Now, for seasonal data:

- STL decomposition: $y_t = T_t + S_t + R_t$
- Seasonal Strength: $F_s = \max(0, 1 - \frac{Var(R_t)}{Var(S_t+R_t)})$

As $S_t \to 0$ then $Ratio \to 1$ and so $F_s \to 0$

As $S_t \to \infty$(high seasonal component) then $Ratio \to 0$ and so $F_s \to 1$

- If $F_s > 0.64$ do one seasonal difference.

```
h02 %>% mutate(log_sales=log(Cost)) %>%
  features(log_sales,feat_stl)
```

```
## # A tibble: 1 x 9
##   trend_strength seasonal_strength_year seasonal_peak_year seasonal_trough_year
##            <dbl>                  <dbl>              <dbl>                <dbl>
## 1          0.957                  0.955                  6                    8
## # i 5 more variables: spikiness <dbl>, linearity <dbl>, curvature <dbl>,
## #   stl_e_acf1 <dbl>, stl_e_acf10 <dbl>
```

Since, here, $0.95 > 0.64$, so we need a seasonal difference.

Alternatively,

```
h02 %>% mutate(log_sales=log(Cost)) %>%
  features(log_sales,unitroot_nsdiffs)
```

```
## # A tibble: 1 x 1
##   nsdiffs
##     <int>
## 1       1
```

Here "ns_diff" means number of seasonal differences.

```
h02 %>% mutate(d_log_sales=difference(log(Cost),12)) %>%
  features(d_log_sales,unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      1
```

- Summary

One way to determine more objectively whether differencing is required is to use a unit root test. These are statistical hypothesis tests of stationarity that are designed for determining whether differencing is required.

A number of unit root tests are available, which are based on different assumptions and may lead to conflicting answers. In our analysis, we use the Kwiatkowski-Phillips-Schmidt-Shin (KPSS) test (Kwiatkowski et al., 1992). In this test, the null hypothesis is that the data are stationary, and we look for evidence that the null hypothesis is false. Consequently, small p-values (e.g., less than 0.05) suggest that differencing is required. The test can be computed using the unitroot_kpss() function.

For example, let us apply it to the Google stock price data.

```
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
```

```
#> # A tibble: 1 × 3
#>   Symbol kpss_stat kpss_pvalue
#>   <chr>      <dbl>       <dbl>
#> 1 GOOG        3.56        0.01
```

The KPSS test p-value is reported as a number between 0.01 and 0.1. If the actual p-value is less than 0.01, it is reported as 0.01; and if the actual p-value is greater than 0.1, it is reported as 0.1. In this case, the p-value is shown as 0.01 (and therefore it may be smaller than that), indicating that the null hypothesis is rejected. That is, the data are not stationary. We can difference the data, and apply the test again.

```
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
```

```
#> # A tibble: 1 × 3
#>   Symbol kpss_stat kpss_pvalue
#>   <chr>      <dbl>       <dbl>
#> 1 GOOG      0.0989         0.1
```

This time, the p-value is reported as 0.1 (and so it could be larger than that). We can conclude that the differenced data appear stationary.

This process of using a sequence of KPSS tests to determine the appropriate number of first differences is carried out using the unitroot_ndiffs() feature.

```
google_2015 |>
  features(Close, unitroot_ndiffs)
```

```
## # A tibble: 1 x 2
##   Symbol ndiffs
##   <chr>   <int>
## 1 GOOG        1
```

```
#> # A tibble: 1 × 2
#>   Symbol ndiffs
#>   <chr>   <int>
#> 1 GOOG        1
```

As we saw from the KPSS tests above, one difference is required to make the google_2015 data stationary.

A similar feature for determining whether seasonal differencing is required is unitroot_nsdiffs(), which uses the measure of seasonal strength introduced in Section 4.3 to determine the appropriate number of seasonal differences required. No seasonal differences are suggested if $F_S < 0.64$, otherwise one seasonal difference is suggested.

We can apply unitroot_nsdiffs() to the monthly total Australian retail turnover.

```
aus_total_retail <- aus_retail |>
  summarise(Turnover = sum(Turnover))
aus_total_retail |>
  mutate(log_turnover = log(Turnover)) |>
  features(log_turnover, unitroot_nsdiffs)
```

```
## # A tibble: 1 x 1
```

```
##   nsdiffs
##     <int>
## 1       1
```

```
#> # A tibble: 1 × 1
#>   nsdiffs
#>     <int>
#> 1       1
```

```
aus_total_retail |>
  mutate(log_turnover = difference(log(Turnover), 12)) |>
  features(log_turnover, unitroot_ndiffs)
```

```
## # A tibble: 1 x 1
##   ndiffs
##    <int>
## 1      1
```

```
#> # A tibble: 1 × 1
#>   ndiffs
#>    <int>
#> 1      1
```

Because unitroot_nsdiffs() returns 1 (indicating one seasonal difference is required), we apply the unitroot_ndiffs() function to the seasonally differenced data. These functions suggest we should do both a seasonal difference and a first difference.

9.2 Backshift notation

It is used for shifting backward in time.

A very useful notational device is the backward shift operator B which is used as follows:

$By_t = y_{t-1}$

In other words, B, operating on y_t has the effect of *shifting the data back one period.*

Two applications of B to y_t, *shifts the data back two periods*

$B(By_t) = B^2y_t = y_{t-2}$

For monthly data, if we wish to shift attention to "the same month last year", then B^{12} is used and the notation is $B^{12}y_t = y_{t-12}$.

The backward shift operator is convenient for describing the process of *differencing.*

A first order difference can be written as:

$y'_t = y_t - y_{t-1} = y_t - By_t = (1-B)y_t$

Similarly, if second-order differences (i.e. first differences of first differences) have to be computed, then:

$y''_t = y_t - 2y_{t-1} + y_{t-2} = (1-B)^2y_t$

- Second-order difference is denoted: $(1-B)^2$
- Second-order difference is not the same as a second-difference which would be denoted $1-B^2$
- In general, a dth- order difference can be written as

$(1-B)^dy_t$

- A seasonal difference followed by a first difference can be written as:

$(1-B)(1-B^m)y_t$

where m is the period of the seasonality. This expression is same as $(1-B-B^m+B^{m+1})y_t = y_t - y_{t-1} - y_{t-m} + y_{t-m-1}$

(Some references use L for "lag" instead of B for "backshift".)

9.3 Autoregressive models

Here, we think and learn about the ARIMA model. ARIMA stands for "Auto Regressive Integrated Moving Average" models. In the previous two sections of the book, we focused on the integrated bit in the model which is the reverse of the differencing. Here, we focus on autoregressive component of the model.

Autoregressive model - AR(p) (Autoregressive model of order p):

$y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_p y_{t-p} + \epsilon_t$

where ϵ_t is white noise. This is a multiple regression with *lagged values* of y_t as predictors. Since it is a lagged values as predictors so it is autoregressive.

AR(1) Model: $y_t = -0.8y_{t-1} + \epsilon_t$

we can generate pattern for $\epsilon_t \sim N(0,1)$ and $T = 100$

- The general $AR(1)$ is $y_t = \phi_1 y_{t-1} + \epsilon_t$
- When $\phi_1 = 0$, y_t is *equivalent to a WN(White Noise)*
- When $\phi_1 = 1$, y_t is *equivalent to a RW(Random Walk)*
- We require $|\phi_1| < 1$ for stationarity. The closer ϕ_1 is to the bounds the more the process wanders above or below, it's unconditional mean (zero in this case)
- When $\phi_1 < 0$, y_t tends to *oscillates between positive and negative values*

AR(1) model including a constant

$y_t = c + \phi_1 y_{t-1} + \epsilon_t$

- When $\phi_1 = 0$ and $c = 0$, y_t is *equivalent to WN(White Noise)*
- When $\phi_1 = 1$ and $c = 0$, y_t is *equivalent to RW(Random Walk)*
- When $\phi_1 = 0$ and $c \neq 0$, y_t is *equivalent to a RW with drift*
- When $\phi_1 < 0$, y_t tends to oscillate around the mean.
- c is related to mean of y_t but not mean itself.
- Let $E(y_t) = \mu$

So, $\mu = c + \phi_1 \mu$ which gives $\mu = \frac{c}{1-\phi_1}$

- ARIMA() takes care of whether you need a constant or not, or you can override it.
- If constant included then estimated model returns constant with mean.

AR(2) Model

- Here we use 2 lags of y.
- Example: $y_t = 8 + 1.3y_{t-1} - 0.7y_{t-2} + \epsilon_t$

Stationarity Conditions

We normally restrict autoregressive models to stationary data, and then some constraints on the values of the parameters are required.

General condition for stationarity

Complex roots of $1 - \phi_1 z - \phi_2 z^2 - ... - \phi_p z^p$ lie outside the unit circle on the complex plane.

Special Cases

- For $p = 1 : -1 < \phi_1 < 1$.
- For $p = 2 : -1 < \phi_2 < 1$ and $\phi_2 + \phi_1 < 1$ and $\phi_2 - \phi_1 < 1$.
- More complicated conditions hold for $p \geq 3$
- Estimation software and ARIMA() function takes care of this. The fable package takes care of these restrictions when estimating a model.

Summary

In a multiple regression model, introduced in Chapter 7, we forecast the variable of interest using a linear combination of predictors. In an autoregression model, we forecast the variable of interest using a linear combination of past values of the variable. The term autoregression indicates that it is a regression of the variable against itself.

This is like a multiple regression but with lagged values of y_t as predictors. We refer to this as an AR(p) model, an autoregressive model of order p.

Autoregressive models are remarkably flexible at handling a wide range of different time series patterns. The two series in Figure 9.5 show series from an AR(1) model and an AR(2) model. Changing the parameters $\phi_1, ..., \phi_p$ results in different time series patterns. The variance of the error term ε_t will only change the scale of the series, not the patterns.

9.4 Moving average models

This is the other half of the ARIMA model. It's little bit like a regression but it's against the past errors rather than past observed values.

$y_t = c + \epsilon_t + \theta_1 \epsilon_{t-1} + \theta_2 \epsilon_{t-2} + ... + \theta_q \epsilon_{t-q}$;

Where ϵ_t is white noise. This is a multiple regression with *past errors* as predictors. Don't confuse this with moving average smoothing.

We call this a "moving average" because it's a little bit like a weighted moving average model of the past errors.

MA(1) [Moving average of order 1]

Example: $y_t = 20 + \epsilon_t + 0.8\epsilon_{t-1}$ where $\epsilon_t \sim (0, 1)$ so errors are standard normal random variable.

MA(2) [Moving average of order 2]

Example: $y_t = \epsilon_t - \epsilon_{t-1} + 0.8\epsilon_{t-2}$ where $\epsilon_t \sim (0, 1)$

MA(∞) Model

It is possible to write any stationary AR(p) process as an MA(∞) process.

Example: AR(1):

$y_t = \phi_1 y_{t-1} + \epsilon_t$

$= \phi_1(\phi_1 y_{t-2} + \epsilon_{t-1}) + \epsilon_t$

$= \phi_1^2 y_{t-2} + \phi_1 \epsilon_{t-1} + \epsilon_t$

$= \phi_1^3 y_{t-3} + \phi_1^2 \epsilon_{t-2} + \phi_1 \epsilon_{t-1} + \epsilon_t$

$= ...$

Provided $-1 < \phi_1 < 1$ because if $\phi_1 > 1$ then its power gets blow up and things will not be numerically stable.

Invertibility

- Any MA(q) process can be written as an AR(∞) process if we impose some constraints on the MA parameters.
- Then the MA model is called "invertible"
- Invertible models have some mathematical properties that make them easier to use in practice.
- Invertibility of an ARIMA model is equivalent to forecastability of an ETS model.

General condition for invertability

- Complex roots of $1 + \theta_1 z + \theta_2 z^2 + ... + \theta_q z^q$ lie outside the unit circle on the complex plane.
- For $q = 1 : -1 < \theta_1 < 1$
- For $q = 2 : -1 < \theta_2 < 1$ and $\theta_2 + \theta_1 > -1$ and $\theta_1 - \theta_2 < 1$
- More complicated conditions hold for $q \geq 3$
- Estimation software fable package takes care of this.

Summary

Rather than using past values of the forecast variable in a regression, a moving average model uses past forecast errors in a regression-like model.

We refer to this as an MA(q) model, a moving average model of order q. Of course, we do not observe the values of ε_t, so it is not really a regression in the usual sense.

Notice that each value of y_t can be thought of as a weighted moving average of the past few forecast errors (although the coefficients will not normally sum to one). However, moving average models should not be confused with the moving average smoothing we discussed in Chapter 3. A moving average model is used for forecasting future values, while moving average smoothing is used for estimating the trend-cycle of past values.

Figure 9.6 shows some data from an MA(1) model and an MA(2) model. Changing the parameters $\theta_1, ..., \theta_q$ results in different time series patterns. As with autoregressive models, the variance of the error term ε_t will only change the scale of the series, not the patterns.

It is possible to write any stationary AR(p) model as an MA(∞) model. For example, using repeated substitution, we can demonstrate this for an AR(1) model:

Provided $1 < \phi_1 < 1$, the value of ϕ_1^k will get smaller as k gets larger.

The reverse result holds if we impose some constraints on the MA parameters. Then the MA model is called invertible. That is, we can write any invertible MA(q) process as an AR(∞) process. Invertible models are not simply introduced to enable us to convert from MA models to AR models. They also have some desirable mathematical properties.

For example, consider the MA(1) process, $y_t = \varepsilon_t + \theta_1 \varepsilon_{t-1}$. In its AR(∞) representation, the most recent error can be written as a linear function of current and past observations:

$\varepsilon_t = \Sigma_{j=0}^{\infty}(-\theta_1)^j y_{t-j}$.

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, and the distant observations have the same influence as the recent observations. As neither of these situations make much sense, we require $|\theta_1| < 1$,

so the most recent observations have higher weight than observations from the more distant past. Thus, the process is invertible when $|\theta_1| < 1$.

The invertibility constraints for other models are similar to the stationarity constraints.

- For an MA(1) model: $-1 < \theta_1 < 1$.
- For an MA(2) model: $-1 < \theta_2 < 1$, and $\theta_2 + \theta_1 > -1$, and $\theta_1 - \theta_2 < 1$.

More complicated conditions hold for $q \geq 3$. Again, the fable package will take care of these constraints when estimating the models.

9.5 Non-seasonal ARIMA models

ARIMA Models

- AR: Autoregressive (lagged observations as inputs)
- I: integrated(opposite of differencing) (differencing to make series stationary)
- MA: moving average (lagged errors as inputs)

ARIMA models don't have the same nice interpretation in terms of things like trends and seasonality. So, An ARIMA model is rarely interpretable in terms of visible data structures like trend and seasonality. On the other hand, it can capture a huge range of time series patterns. With some time series, we can only model with ARIMA model and we could not do with ETS model.

Autoregressive Moving Average Models

$y_t = c + \phi_1 y_{t-1} + ... + \phi_p y_{t-p} + \theta_1 \epsilon_{t-1} + ... + \theta_q \epsilon_{t-q} + \epsilon_t$

- Predictors include both *lagged value of* y_t *and lagged errors.*
- Conditions on AR coefficients ensure stationarity.
- Conditions on MA coefficients ensure invertibility.

Without "I" for the differencing, it is called *ARMA* model.

Autoregressive Integrated Moving Average Models

- Combine ARMA model with differencing (could be more than 1 differencing).
- $(1-B)^d y_t$ follows an ARMA model.

ARIMA models

Autoregressive Integrated Moving Average models

ARIMA(p,d,q) model

AR: $p =$ order of the autoregressive part

I: $d =$ degree of first differencing involved (how many differences we have used)

MA: $q =$ order of the moving average part.

- White noise model: ARIMA(0,0,0) i.e. $y_t = \epsilon_t$
- Random Walk: ARIMA(0,1,0) with no constant i.e. $y_t - y_{t-1} = \epsilon_t \Rightarrow y_t = y_{t-1} + \epsilon_t$
- Random walk with drift: ARIMA(0,1,0) with const. i.e. $y_t - y_{t-1} = c + \epsilon_t$
- AR(p): ARIMA(p,0,0)
- MA(q): ARIMA(0,0,q)

Backshift Notation for ARIMA

- ARMA model:

$y_t = c+ _1By_t +…+ _pB^py_t + _t+ _1B _t+…+ _qB^q _t $

(or) $(1 - \phi_1 B - ... - \phi_p B^p) y_t = c + (1 + \theta_1 B + ... + \theta_q B^q) \epsilon_t$

- ARIMA(1,1,1) model:

(or) $(1 - \phi_1 B)(1 - B) y_t = c + (1 + \theta_1 B) \epsilon_t$

where $(1 - \phi_1 B)$ represents AR(1).

(1-B) represents first difference.

$(1 - \theta_1 B)$ represents MA(1).

Here, first we take the first difference and then apply the AR(1) on the left side and do the MA(1) on right side.

If we expand the above equation, we get,

$y_t = c + y_{t-1} + \phi_1 y_{t-1} - \phi_1 y_{t-2} + \theta_1 \epsilon_{t-1} + \epsilon_t$

R model

Various textbooks follow different form for the above equation.

Intercept form

$(1 - \phi_1 B - ... - \phi_p B^p) y'_t = c + (1 + \theta_1 B + ... + \theta_q B^q) \epsilon_t$

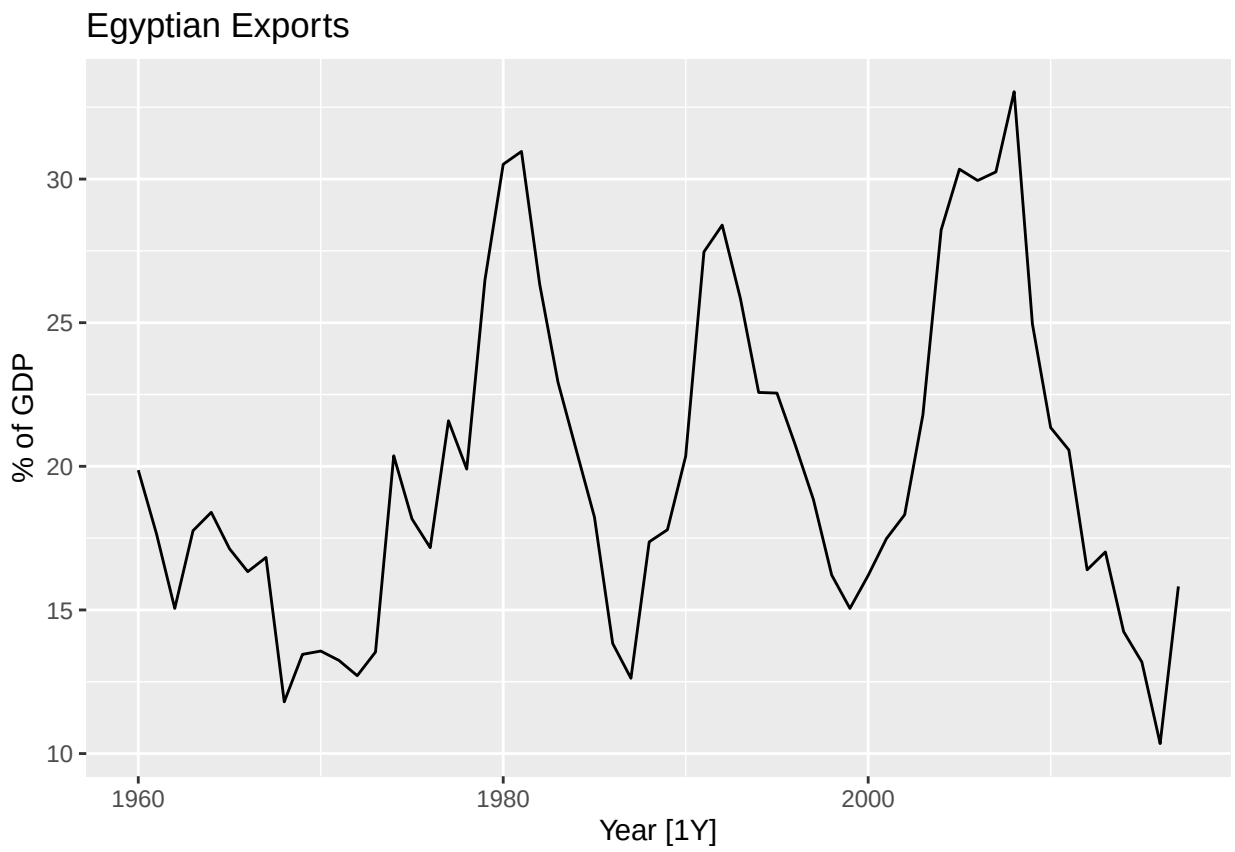
Mean form

$(1 - \phi_1 B - ... - \phi_p B^p)(y'_t - \mu) = (1 + \theta_1 B + ... + \theta_q B^q) \epsilon_t$

- $y'_t = (1 - B)^d y_t$
- μ is the mean of y'_t.
- $c = \mu(1 - \phi_1 - ... - \phi_p)$
- fable uses intercept form.

Example

```
global_economy |>
  filter(Code == "EGY") |>
  autoplot(Exports) +
  labs(y= "% of GDP", title = "Egyptian Exports")
```



It looks like little bit of cyclic behavior and we can get the cyclic behavior in autoregressive model provided "p" is atleast 2. One of the things in autoregressive for an ARIMA model can do that an ETS model can't do. ETS model can't handle the cyclic behavior in the data, it can handle seasonality but in ARIMA model, we can handle both seasonality and cyclic behavior.

```
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
##       1.6764  -0.8034  -0.6896    2.5623
## s.e.  0.1111   0.0928   0.1492    0.1161
##
## sigma^2 estimated as 8.046:  log likelihood=-141.57
## AIC=293.13   AICc=294.29   BIC=303.43
```

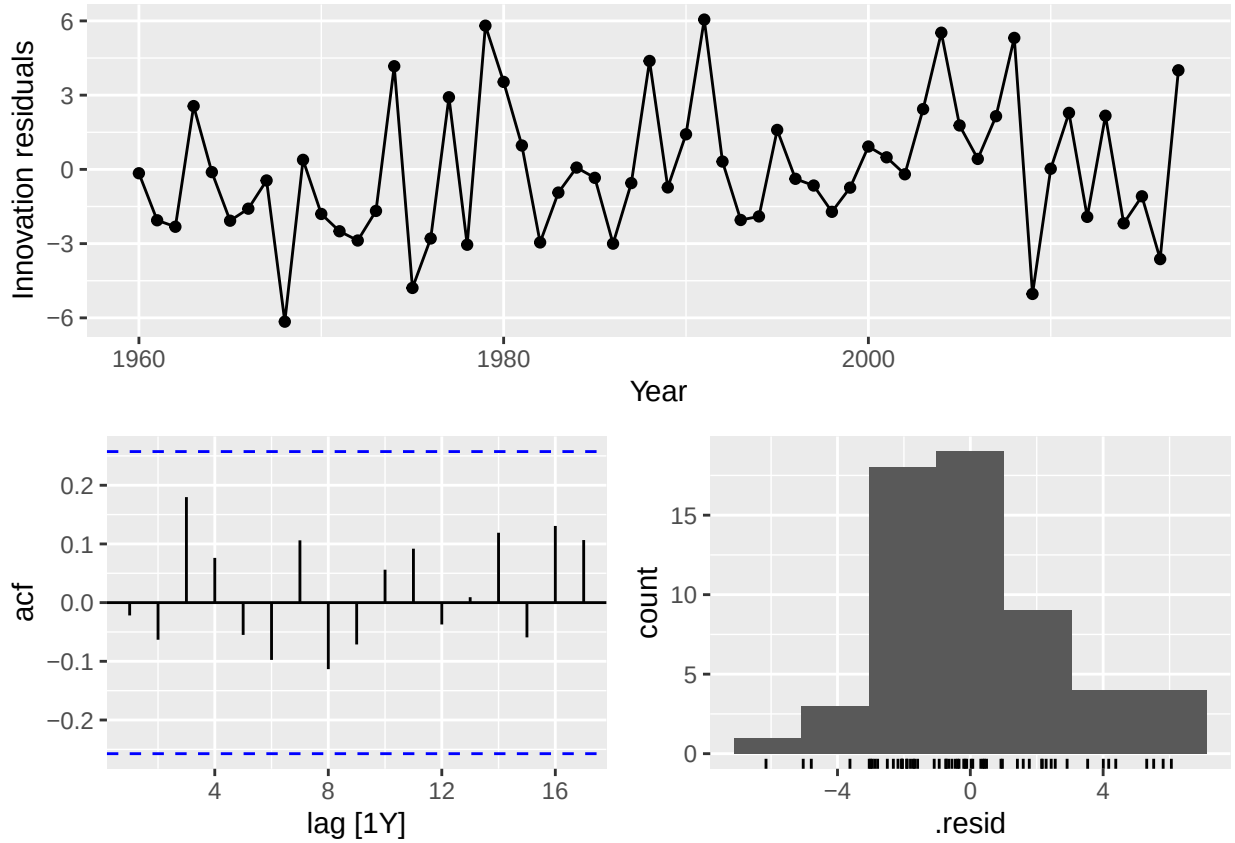
It chooses the model automatically. Here, ARIMA(2,0,1) with constant is the best model.

From this output, we can write the equation as:

$y_t = 2.56 + 1.68y_{t-1} - 0.80y_{t-2} - 0.69\epsilon_{t-1} + \epsilon_t$,

where ϵ_t is white noise with standard deviation of $2.837 = \sqrt{8.046}$

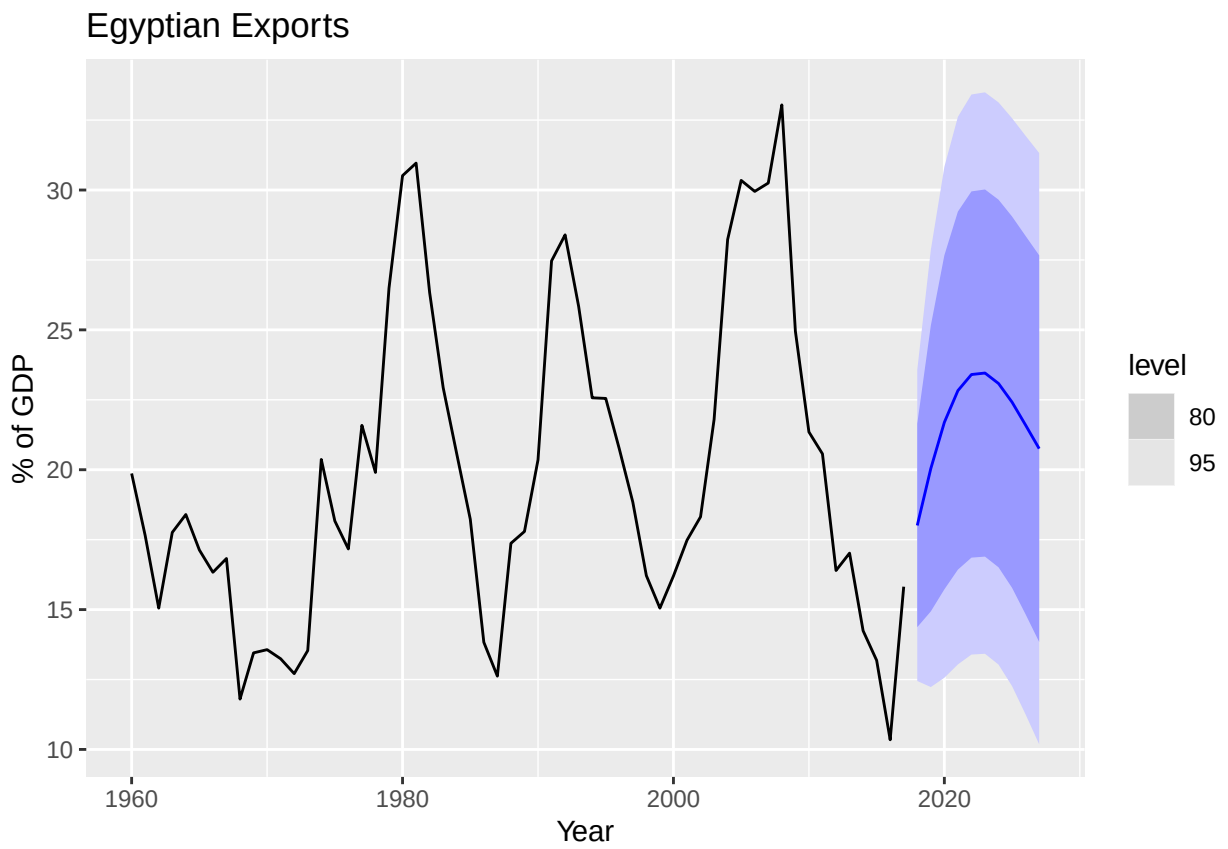
```
gg_tsresiduals(fit)
```



It looks nice as residuals are centered around zero, there is no obvious patterns in the time plot. All the spikes are well inside the blue line.

Now, if we go for forecast.

```
fit |>
  forecast(h=10) |>
  autoplot(global_economy)+
  labs(y="% of GDP", title = "Egyptian Exports")
```



Here, forecast have actually picked up the cyclic nature of the data. It is picking increasing part of cycle and decreasing part of the cycle. Prediction Intervals are quite wide because cyclic behavior is very hard to predict.

Now, we go for some features of ARIMA model.

Understanding ARIMA models

For the long-term forecasting, the most important part of the model is whether or not we include the constant and how many differences we include. p and q only affects only short term part of the forecasts. c and d affects only long term part of the forecasts.

- If $c = 0$ and $d = 0$ then there is no intercepting and there is no differencing then in the long run, forecast will go to zero. So point forecast tends towards zero or converge 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.(not recommended)

Forecast Variance and d

- The higher the value of d, the more rapidly the prediction intervals increase in size.

- For $d = 0$, the long-term forecast standard deviation will go to the standard deviation of the historical data.

Cyclic behavior

- For cyclic forecasts, $p \geq 2$ and some restrictions on coefficients are required.
- If $p = 2$, we need $\phi_1^2 + 4\phi_2 < 0$. The average cycle of length $\frac{2\pi}{\arccos(-\phi_1(1-\phi_2)/(4\phi_2))}$

Partial autocorrelations

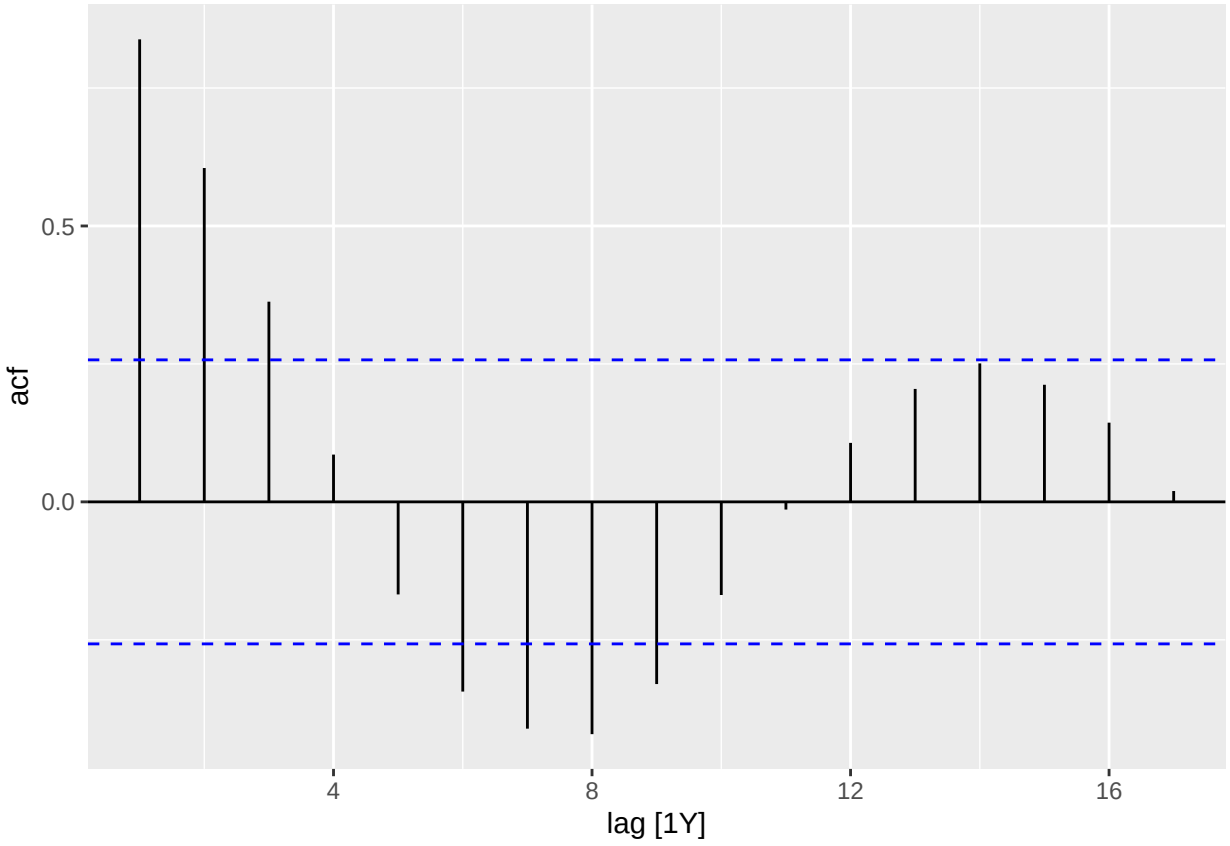
Partial autocorrelations measure the relationship between y_t and y_{t-k} when the effects of other time lags 1,2,3,...k-1 are removed.

$\alpha_k =$ kth partial autocorrelation coefficient = equal to the estimate of the ϕ_k in regression $y_t = c + \phi_1 y_{t-1} + \phi_2 y_{t-2} + ... + \phi_k y_{t-k} + \epsilon_t$

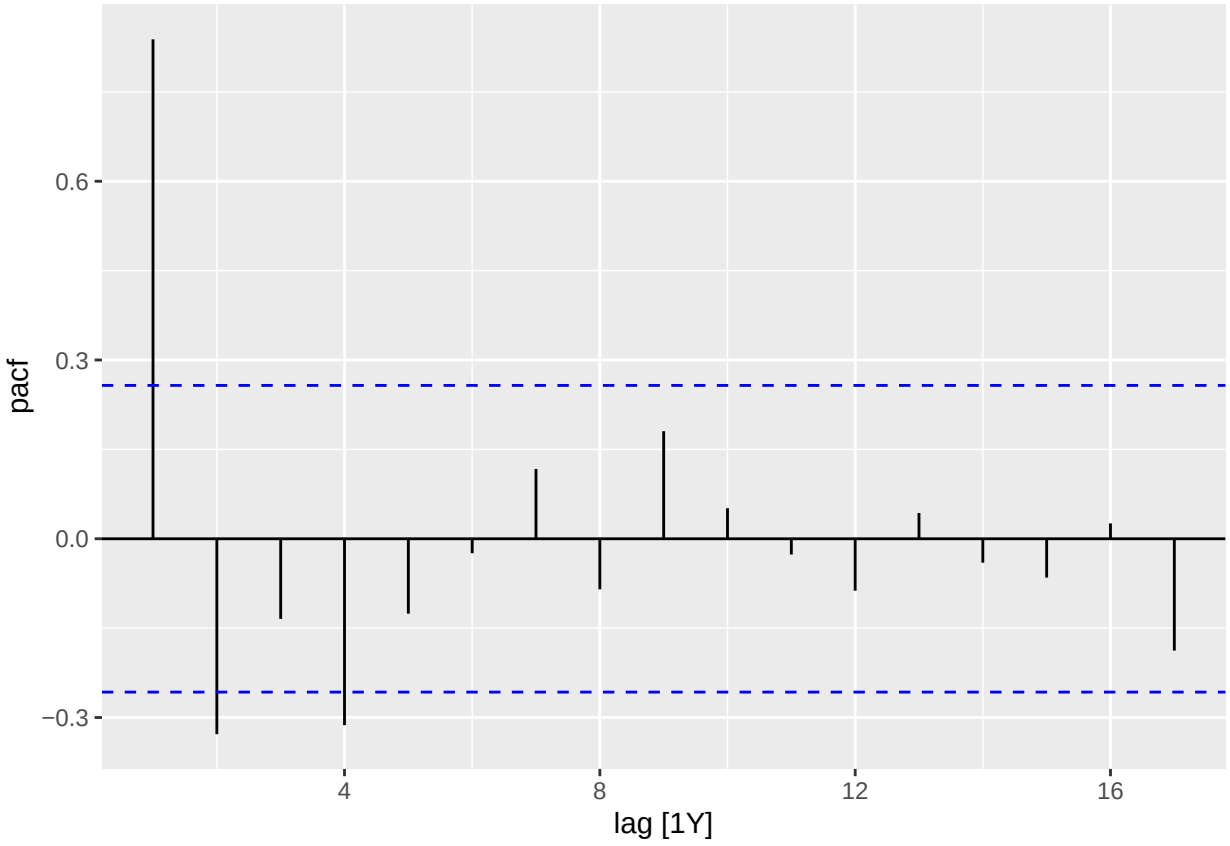
- Varying the number of terms on RHS gives α_k for different values of k.
- $\alpha_1 = \rho_1$
- Same critical values of $\pm 1.96/\sqrt{T}$ as for the ACF. It means blue line is same for both acf and pacf graph.
- Last significant α_k indicates the order of an AR model.

Example

```
egypt <- global_economy |> filter(Code == "EGY")
egypt |> ACF(Exports) |> autoplot()
```




```
egypt |> PACF(Exports) |> autoplot()
```



As we have mentioned that last significant α_k decides the order of AR model. So, here, in the pacf graph, last significant spike is for value as 4. So, it suggests that AR(4) might be a reasonable model for egyptian export data. We can use this trick if we have a pure AR model. We can't use it for mixing AR and MA terms.

So, here we have 2 possible models for this data, ARIMA(2,1,1) as seen previous as well as AR(4).

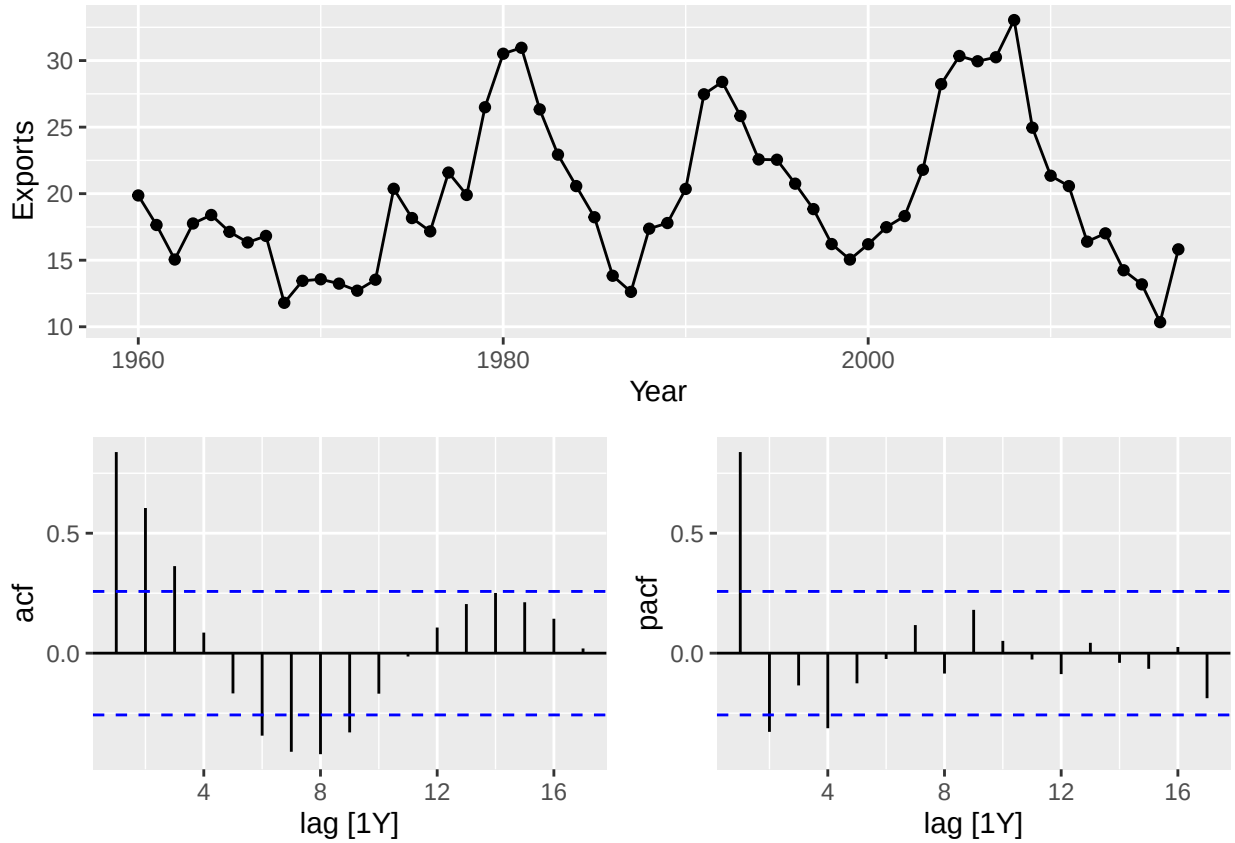
```
library(fpp3)
egypt <- global_economy |>
  filter(Code =="EGY")
fit <- egypt |>
  model(
    ar4 = ARIMA(Exports ~ pdq(p=4,d=0,q=0)),
    auto=ARIMA(Exports)
  )
glance(fit)
```

```
## # A tibble: 2 x 9
##   Country          .model sigma2 log_lik   AIC  AICc   BIC ar_roots  ma_roots
##   <fct>            <chr>   <dbl>   <dbl> <dbl> <dbl> <dbl> <list>    <list>
## 1 Egypt, Arab Rep. ar4      7.88   -141.  293.  295.  305. <cpl [4]> <cpl [0]>
## 2 Egypt, Arab Rep. auto     8.05   -142.  293.  294.  303. <cpl [2]> <cpl [1]>
```

here AICc for auto is little bit smaller than AR(4). So our manually chosen model is close to our automaically chosen model.

Alternatively plot pcf:

```
global_economy |>
  filter(Code =="EGY") |>
  gg_tsdisplay(Exports, plot_type = "partial")
```



here, acf follows a sinusoidal pattern so it's a cyclic data because it can't be seasonal because it is annual data.

ACF and PACF Interpretation

AR(1)

- $\rho_k = \phi_1^k$ for $k = 1, 2, ...;$
- $\alpha_1 = \phi_1$ and $\alpha_k = 0$ for $k = 2, 3, ...;$

So we have an AR(1) model when

- autocorrelations exponentially decay
- there is a single significant partial autocorrelation.

AR(p)

- ACF dies out in an exceptional or damped sine-wave manner.
- PACF has all zero spikes beyond the p^{th} spike.

So we have an AR(p) model when

- the ACF is exponentially decaying or sinusoidal

- there is a significant spike at lag p in PACF but None beyond p.

If we have got a pure MA model then it will be reverse.

MA(1)

$\rho_1 = \theta_1/(1+\theta_1^2)$ and $\rho_k = 0$ for $k = 2, 3, ...;$

$\alpha_k = -(-\theta_1)^k/(1+\theta_1^2+...+\theta_1^{2k})$

So we have an MA(1) model when

- the PACF is exponentially decaying and
- there is a single significant spike in ACF.

MA(q)

- PACF dies out in an exceptional or damped sine-wave manner.
- ACF has all zero spikes beyond the q^{th} spike.

So we have an MA(q) model when

- the PACF is exponentially decaying or sinusoidal
- there is a significant spike at lag q in ACF but None beyond q.

9.6 Estimation and order selection

Maximum Likelihood Estimation

- Having identified the model order, we need to estimate the parameters $c, \phi_1, ..., \phi_p, \theta_1, ..., \theta_q$
- MLE is very similar to least squares estimation obtained by minimizing

$$\Sigma_{t=1}^{T} e_t^2$$

- The ARIMA() function allows CLS(Conditional Least Squares) or MLE estimation. Default is MLE estimate.
- Non-linear optimization must be used in either case with some conditioning as well.
- Different software will give different estimates because one can use CLS and one can also use MLE etc.

Information Criteria

Akaike's Information Criterian (AIC): $AIC = -2\log L + 2(p+q+k+1)$

where L is the likelihood of the data, $k = 1$ if $c \neq 0$ and $k = 0$ if $c = 0$

Corrected AIC:

$AICc = AIC + \frac{2(p+q+k+1)(p+q+k+2)}{T-p-q-k-2}$

Bayesian Information Criterian

$BIC = AIC + [\log(T) - 2](p+q+k+1)$

Good Models are obtained by minimizing either the AIC,AICc or BIC. Our preference is to use the AICc.

Summary

Maximum likelihood estimation

Once the model order has been identified (i.e., the values of p, d and q),we need to estimate the parameters $c, \phi_1, ..., \phi_p, \theta_1, ..., \theta_q$.

When fable estimates the ARIMA model, it uses maximum likelihood estimation (MLE). This technique finds the values of the parameters which maximise the probability of obtaining the data that we have observed. For ARIMA models, MLE is similar to the least squares estimates that would be obtained by minimising $\Sigma_{t=1}^{T} \varepsilon_t^2$.

(For the regression models considered in Chapter 7, MLE gives exactly the same parameter estimates as least squares estimation.) Note that ARIMA models are much more complicated to estimate than regression models, and different software will give slightly different answers as they use different methods of estimation, and different optimisation algorithms.

In practice, the fable package will report the value of the log likelihood of the data; that is, the logarithm of the probability of the observed data coming from the estimated model. For given values of p, d and q,ARIMA() will try to maximise the log likelihood when finding parameter estimates.

Information Criteria

Akaike's Information Criterion (AIC), which was useful in selecting predictors for regression (see Section 7.5), is also useful for determining the order of an ARIMA model.

It can be written as

$$AIC = -2 \log L + 2(p + q + k + 1)$$

where L is the likelihood of the data, $k = 1$ if $c \neq 0$ and $k = 0$ if $c = 0$. Note that the last term in parentheses is the number of parameters in the model (including σ^2, the variance of the residuals).

It is important to note that these information criteria tend not to be good guides to selecting the appropriate order of differencing (d) of a model, but only for selecting the values of p and q. This is because the differencing changes the data on which the likelihood is computed, making the AIC values between models with different orders of differencing not comparable. So we need to use some other approach to choose d, and then we can use the AICc to select p and q.

9.7 ARIMA modelling in fable

Traditional modelling procedure for ARIMA models

First, we will see that how ARIMA modelling has been done for the last 40-50 years.

- Plot the data. Identify any unusual observations.
- If necessary, transform the data (using a Box-Cox transformation) to stabilize the variance that leads to a simpler model.
- If the data are non-stationary: take first differences of the data until the data are stationary.
- Examine the ACF/PACF: Is an AR(p) or MA(q) model appropriate ? You can't choose a mixed model with both p and q greater than zero. You have to choose either autoregressive model or moving model.
- Try your chosen model(s), and use the AICc to search for a better model. Either increase or decrease p or q to choose a better model and that's where we can try mized model.
- Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.
- Once the residuals look like white noise, calculate forecasts.

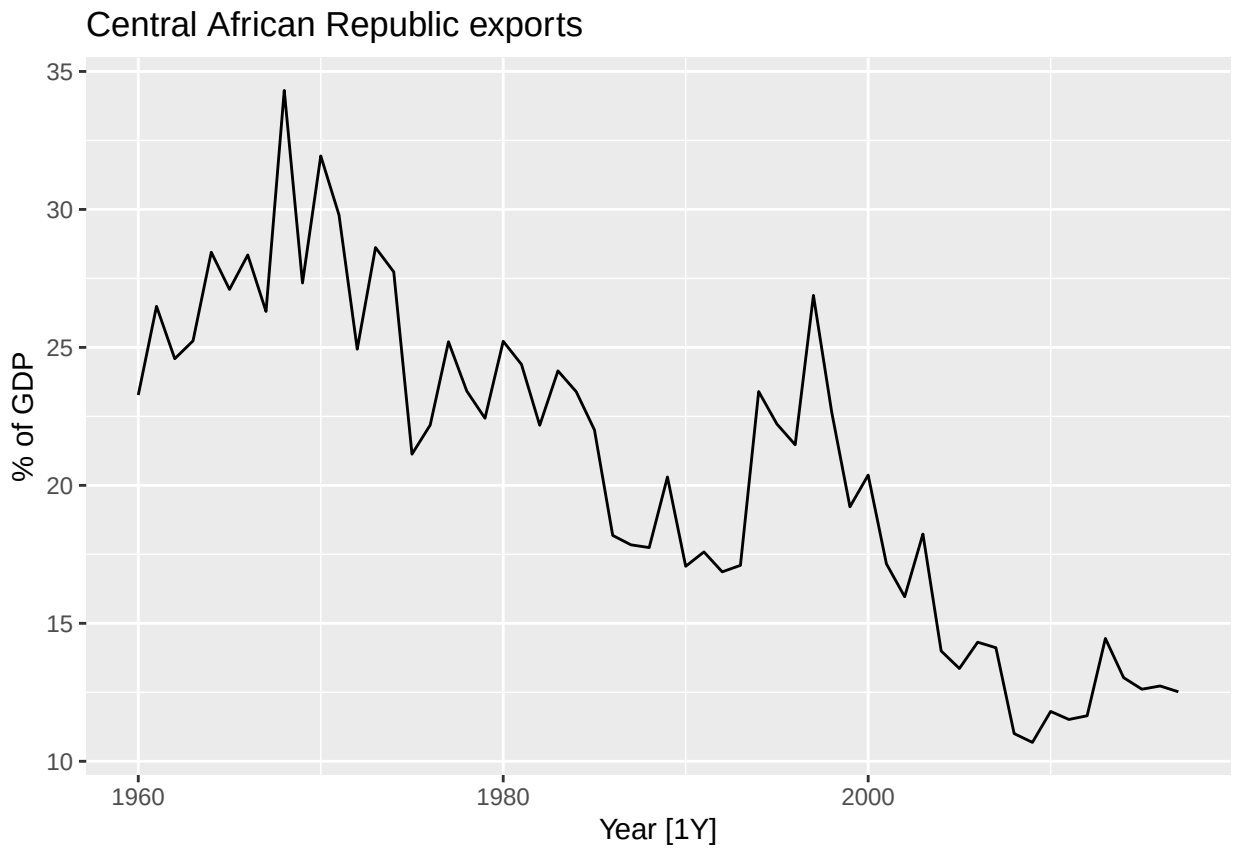
By using fable, we can remove some of the above steps.

Automatic modelling procedure with ARIMA()

- Plot the data. Identify any unusual observations.
- If necessary, transform the data (using a Box-Cox transformation) to stabilize the variance that leads to a simpler model.
- Use ARIMA to automatically select a model.
- Check the underlying assumptions and residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model but most of the time it works well.
- Once the residuals look like white noise, calculate forecasts.

Example: Central African Republic Exports

```
global_economy |>
  filter(Code == "CAF")|>
  autoplot(Exports) +
  labs(title = "Central African Republic exports", y="% of GDP")
```



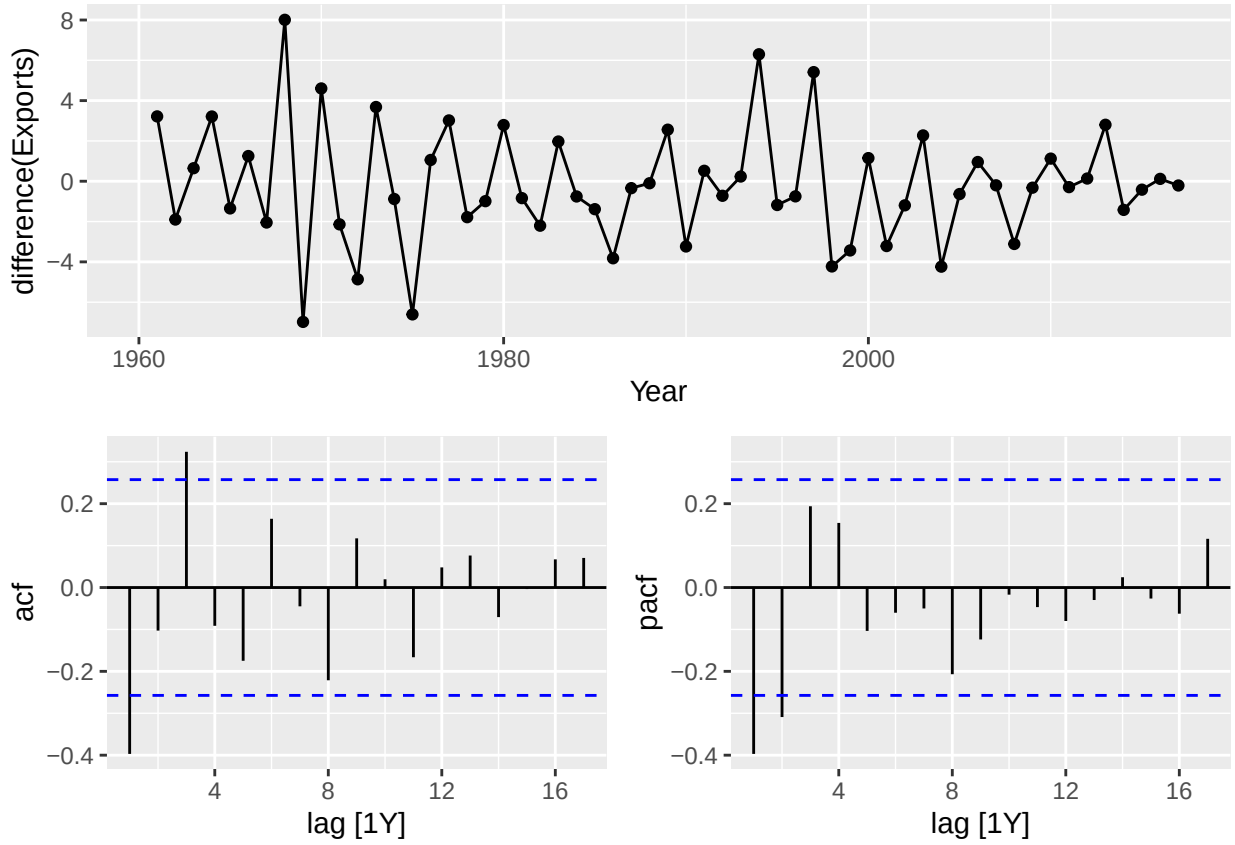
Here, in the data, some downward trending is present but it has some ups and downs over the time and we want to find a model for that.

Using the manual process, we leave the transformation part because we don't need to fix variance. Now we need to know how many differences we need to know to make it stationary.

```
global_economy |>
  filter(Code=="CAF") |>
  gg_tsdisplay(difference(Exports),plot_type = "partial")
```

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
```



By looking at the ACF and pcf plot, we can say that differenced data looks fine and it is stationary.

From the ACF plot, we find the order of MA model, so here last significant spike is for 3. So, q=3 and so MA(3) and so p=0 and d=1 make ARIMA(0,1,3). On the other side if we see the PACF plot, we get the last significant spike at p=2 which suggests ARIMA(2,1,0).

So, manually we get 2 possible models i.e. ARIMA(0,1,3) and ARIMA(2,1,0).

Now try these 2 models

```
caf_fit <- global_economy |>
  filter(Code == "CAF") |>
  model(
  arima210 = ARIMA(Exports ~ pdq(2,1,0)),
  arima013 = ARIMA(Exports ~ pdq(0,1,3)),
  stepwise = ARIMA(Exports),
  search = ARIMA(Exports,stepwise = FALSE),
  )
```

```
caf_fit |> pivot_longer(!Country,
                        names_to = "Model name",
```

```
                        values_to="Orders"
                        )
```

```
## # A mable: 4 x 3
## # Key:     Country, Model name [4]
##   Country                  `Model name`         Orders
##   <fct>                    <chr>               <model>
## 1 Central African Republic arima210     <ARIMA(2,1,0)>
## 2 Central African Republic arima013     <ARIMA(0,1,3)>
## 3 Central African Republic stepwise     <ARIMA(2,1,2)>
## 4 Central African Republic search       <ARIMA(3,1,0)>
```

stepwise and search are automatically chosen models. stepwise gives ARIMA(2,1,2) (use non-zero values of p and q) and search gives ARIMA(3,1,0) (it is simply AR(3)).

Now which model we should have to use.

```
glance(caf_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.
```

Here search model is the best and other models are little bit worse than this.

How does ARIMA() work ?

A non-seasonal ARIMA process

$\phi(B)(1-B)^d y_t = c + \theta(B)\epsilon_t$

Need to select appropriate orders d, p, q and whether to include the intercept c.

Hyndman and Khandkar (JSS,2008) algorithm:

- Select no. of differences d via KPSS test.
- Select p, q and c by minimizing $AICc$. Of course there are infinite number of p, q and whether or not "c" is in equation. So, we can't look at all of the models, so it uses a stepwise search to traverse the model space.
- Use stepwise search to traverse model space.

$AICc = -2\log(L) + 2(p+q+k+1)[1 + \frac{p+q+k+2}{T-p-q-k-2}]$.

Where L is the maximised likelihood fitted to the *differenced* data, $k = 1$ if $c \neq 0$ and $k = 0$ otherwise.

Stepwise procedure starts with:

Step 1: Select current model (with smallest AICc) from:

ARIMA(2,d,2);ARIMA(0,d,0);ARIMA(1,d,0)(AR-1);ARIMA(0,d,1)(MA-1)

Step 2: Consider variations of current model:

- Vary one of p,q from current model by ± 1;

- p,q both vary from current model by ± 1;
- Include/exclude "c" from current model.

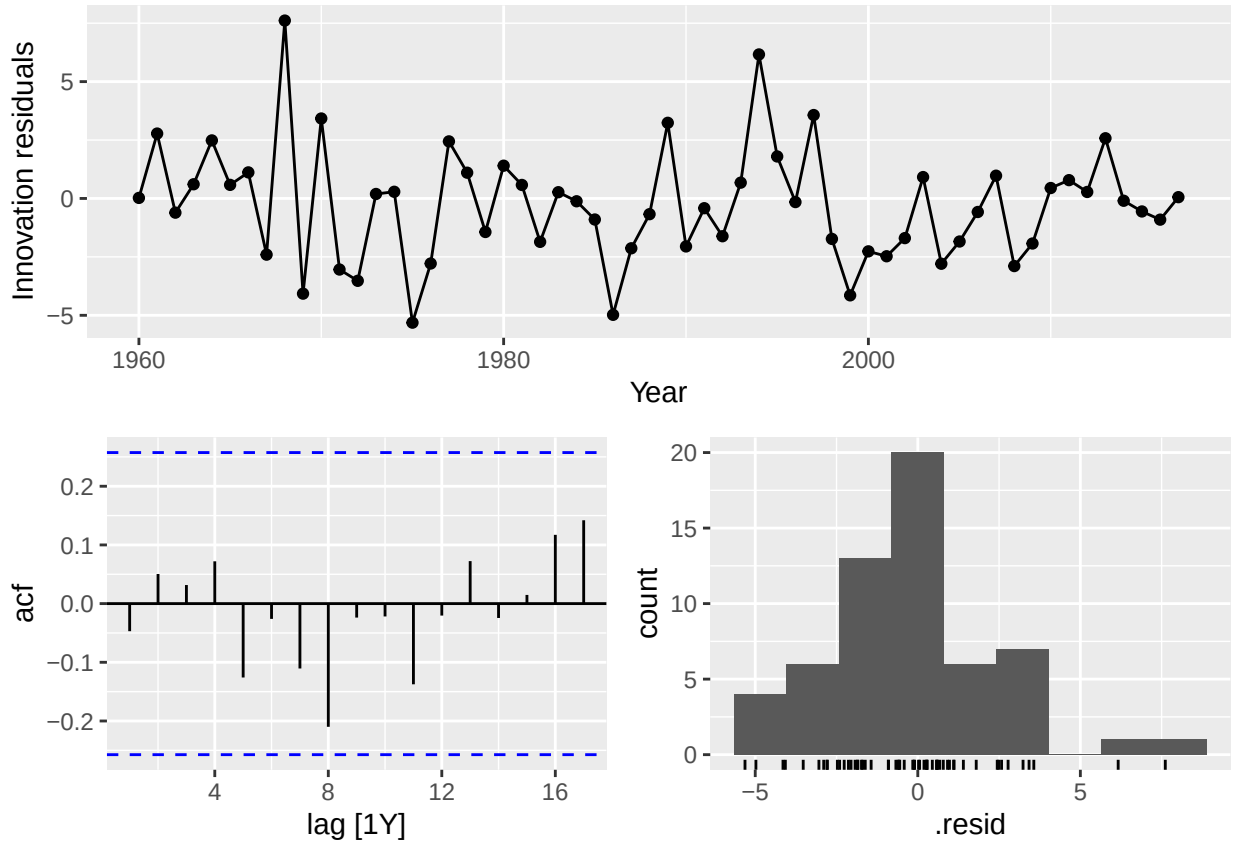
Model with lowest AICc becomes current model.

Repeat Step 2 until no lower AICc can be found

So, here we search the neighbor models and don't see p and q which are already seen and so it's like a greedy search. This stepwise model does not guaranteed to give best model but it hopefully will give a good model.

if stepwise=False then it will look for all the possible models upto the maximum order of p and q in the stepwise fashion which we have seen previously. For example, if upper order is 6 then it will search all possible values of p and q for which $p + q < 7$

```
caf_fit |>
  select(search) |>
  gg_tsresiduals()
```



Here residuals looks okay. They look like white noise and there is no obvious pattern in the time plot. All of the ACF spikes inside the critical value. Residuals close to normal.

Portmanteau test of residuals for ARIMA models

It is for the group of acf spikes whether it is significant or not and it is like lunjue box test.

It is slightly different for ARIMA models than other models. We can get a more accurate Portmanteau test if we adjust the degrees of freedom to take account of the number of parameters in the model.

With ARIMA models, more accurate Portmanteau test obtained if degrees of freedom are adjusted to take account of number of parameters in the model.

- Use $l - K$ degrees of freedom, where $K = p + q =$ number of AR and MA parameters in the model.
- use dof(degree of freedom) argument in ljung_box().

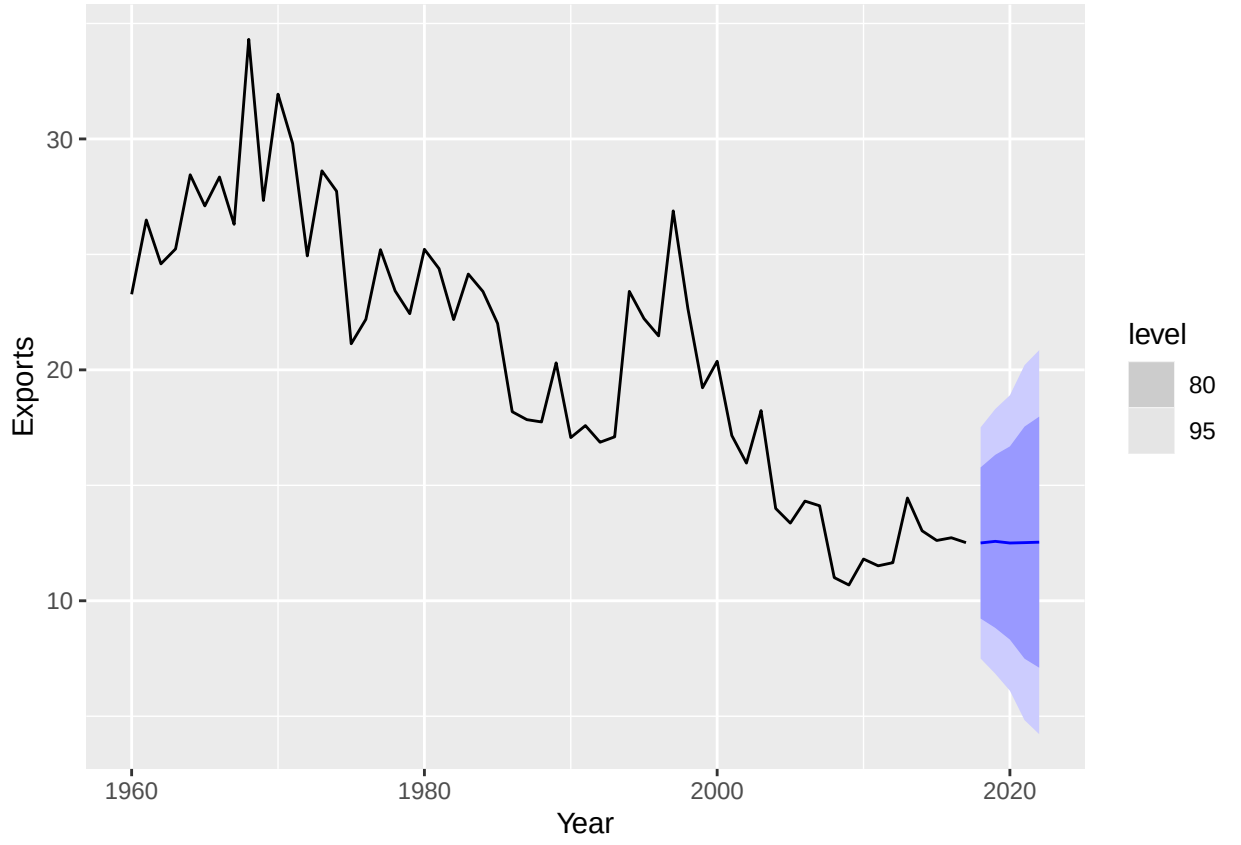
```
augment(caf_fit) |>
  filter(.model == "search") |>
  features(.innov,ljung_box,lag=10,dof=3)
```

```
## # A tibble: 1 x 4
##   Country                  .model lb_stat lb_pvalue
##   <fct>                    <chr>    <dbl>     <dbl>
## 1 Central African Republic search    5.75     0.569
```

Here, maximum number of lag is 10 and this is non-seasonal dataset, so we usually use lag=10. Here dof=3 because previously for search model, we have used ARIMA(3,1,0) and so p+q=3.

Here, p-value is very large so we say there is not any significant difference between what we are seeing here and white noise and so we can go ahead and use this model for forecasting.

```
caf_fit |>
  forecast(h=5)|>
  filter(.model == "search") |>
  autoplot(global_economy)
```



It is not a very interesting forecast and they are very close to the last observed value of the series. It is similar to any of the 4 models which we have seen.

——————————- Summary ——————————

##How does ARIMA() work?

The ARIMA() function in the fable package uses a variation of the Hyndman-Khandakar algorithm (Hyndman & Khandakar, 2008), which combines unit root tests, minimisation of the AICc and MLE to obtain an ARIMA model. The arguments to ARIMA() provide for many variations on the algorithm. What is described here is the default behaviour.

Hyndman-Khandakar algorithm for automatic ARIMA modelling

1. The number of differences $0 \leq d \leq 2$ is determined using repeated KPSS tests.
2. The values of p and q are then chosen by minimising the AICc after differencing the data d times. Rather than considering every possible combination of p and q , the algorithm uses a stepwise search to traverse the model space.

a. 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. If $d \leq 1$, an additional model is also fitted:

- ARIMA (0,d,0) without a constant.

b. The best model (with the smallest AICc value) fitted in step (a) is set to be the "current model".

c. Variations on the current model are considered:

- vary p and/or q from the current model by ± 1;
- include/exclude c from the current model.

The best model considered so far (either the current model or one of these variations) becomes the new current model.

d. Repeat Step 2(c) until no lower AICc can be found.

Figure 9.11 illustrates diagrammatically how the Hyndman-Khandakar algorithm traverses the space of the ARMA orders, through an example. The grid covers combinations of ARMA(p,q) orders starting from the top-left corner with an ARMA(0,0), with the AR order increasing down the vertical axis, and the MA order increasing across the horizontal axis.

The orange cells show the initial set of models considered by the algorithm. In this example, the ARMA(2,2) model has the lowest AICc value amongst these models. This is called the "current model" and is shown by the black circle. The algorithm then searches over neighbouring models as shown by the blue arrows. If a better model is found then this becomes the new "current model". In this example, the new "current model" is the ARMA(3,3) model. The algorithm continues in this fashion until no better model can be found. In this example the model returned is an ARMA(4,2) model.

The default procedure will switch to a new "current model" as soon as a better model is identified, without going through all the neighbouring models. The full neighbourhood search is done when greedy=FALSE.

The default procedure also uses some approximations to speed up the search. These approximations can be avoided with the argument approximation=FALSE. It is possible that the minimum AICc model will not be found due to these approximations, or because of the use of the stepwise procedure. A much larger set of models will be searched if the argument stepwise=FALSE is used. See the help file for a full description of the arguments.

Modelling procedure

When fitting an ARIMA model to a set of (non-seasonal) time series data, the following procedure provides a useful general approach.

1.Plot the data and identify any unusual observations.

2. If necessary, transform the data (using a Box-Cox transformation) to stabilise the variance.
3. If the data are non-stationary, take first differences of the data until the data are stationary.
4. Examine the ACF/PACF: Is an ARIMA(p,d,0) or ARIMA(0,d,q) model appropriate?
5. Try your chosen model(s), and use the AICc to search for a better model.
6. Check the residuals from your chosen model by plotting the ACF of the residuals, and doing a portmanteau test of the residuals. If they do not look like white noise, try a modified model.

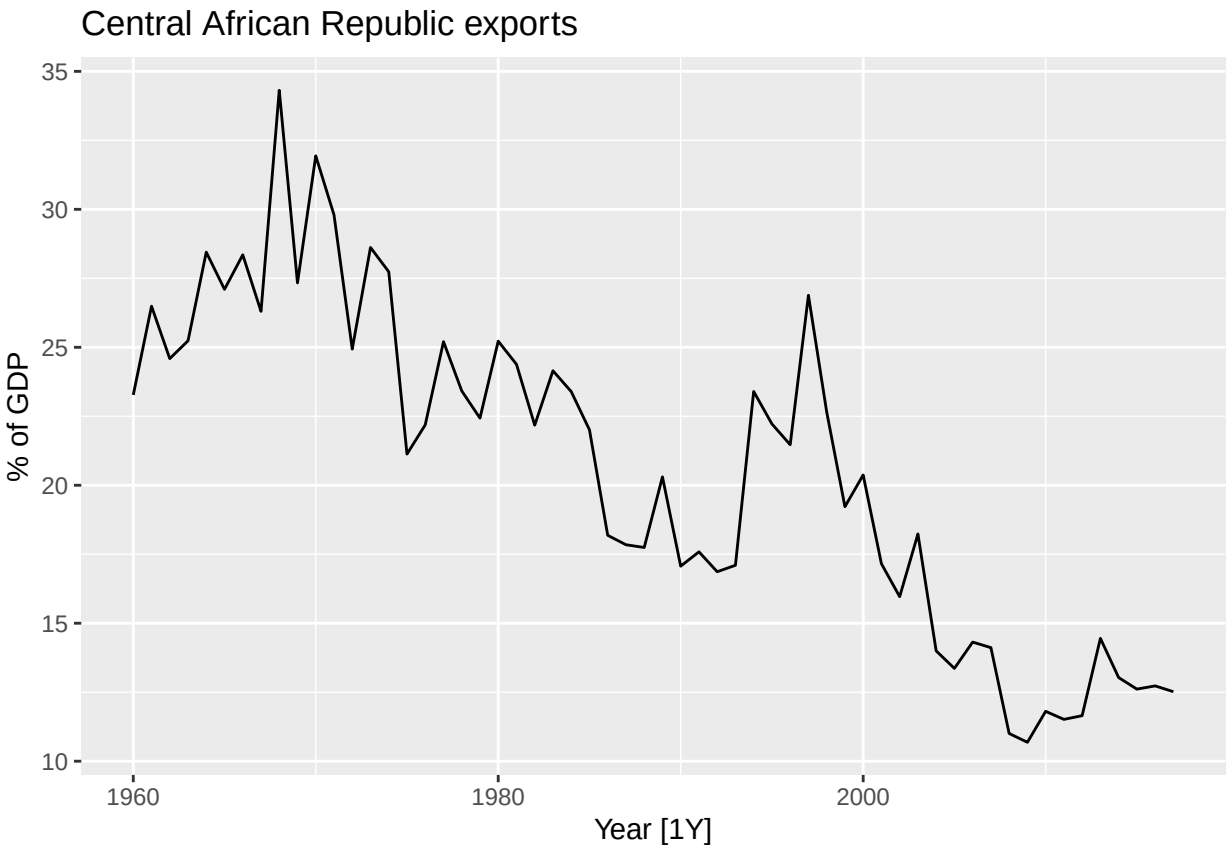
7.Once the residuals look like white noise, calculate forecasts.

The Hyndman-Khandakar algorithm only takes care of steps 3–5. So even if you use it, you will still need to take care of the other steps yourself.

Portmanteau tests of residuals for ARIMA models

With ARIMA models, more accurate portmanteau tests are obtained if the degrees of freedom of the test statistic are adjusted to take account of the number of parameters in the model. Specifically, we use −K degrees of freedom in the test, where K is the number of AR and MA parameters in the model. So for the non-seasonal models that we have considered so far, K=p+q. The value of K is passed to the ljung_box function via the argument dof, as shown in the example below.

```
global_economy |>
  filter(Code == "CAF") |>
  autoplot(Exports) +
  labs(title="Central African Republic exports",
       y="% of GDP")
```



1. The time plot shows some non-stationarity, with an overall decline. The improvement in 1994 was due to a new government which overthrew the military junta and had some initial success, before unrest caused further economic decline.

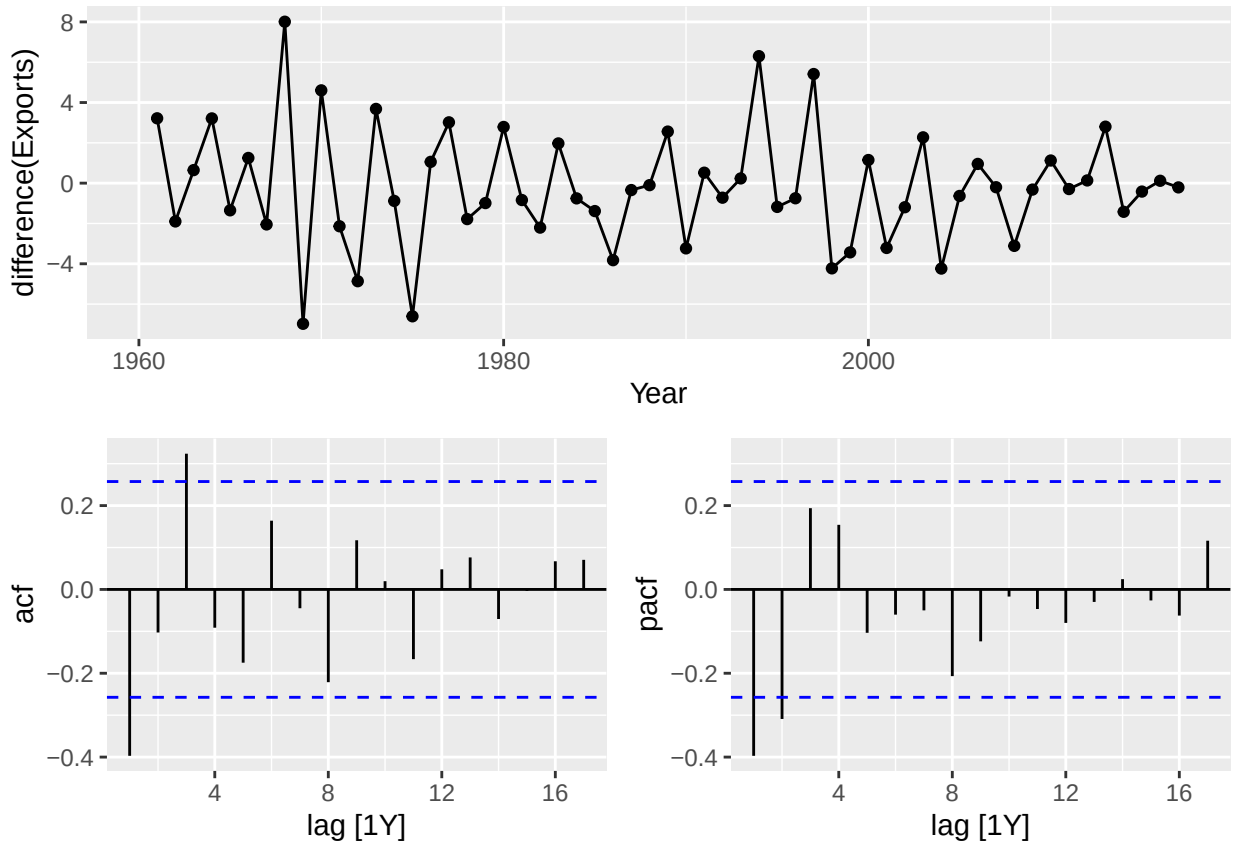
2.There is no evidence of changing variance, so we will not do a Box-Cox transformation.

3.To address the non-stationarity, we will take a first difference of the data. The differenced data are shown in Figure 9.14.

```
global_economy |>
  filter(Code == "CAF") |>
  gg_tsdisplay(difference(Exports), plot_type='partial')
```

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```

```
## Warning: Removed 1 rows containing missing values (`geom_point()`).
```



These now appear to be stationary.

4.The PACF shown in Figure 9.14 is suggestive of an AR(2) model; so an initial candidate model is an ARIMA(2,1,0). The ACF suggests an MA(3) model; so an alternative candidate is an ARIMA(0,1,3).

5. We fit both an ARIMA(2,1,0) and an ARIMA(0,1,3) model along with two automated model selections, one using the default stepwise procedure, and one working harder to search a larger model space.

```
caf_fit <- global_economy |>
  filter(Code == "CAF") |>
  model(arima210 = ARIMA(Exports ~ pdq(2,1,0)),
        arima013 = ARIMA(Exports ~ pdq(0,1,3)),
        stepwise = ARIMA(Exports),
        search = ARIMA(Exports, stepwise=FALSE))

caf_fit |> pivot_longer(!Country, names_to = "Model name",
                        values_to = "Orders")
```

```
## # A mable: 4 x 3
## # Key:     Country, Model name [4]
##   Country                  `Model name`         Orders
##   <fct>                    <chr>              <model>
## 1 Central African Republic arima210     <ARIMA(2,1,0)>
## 2 Central African Republic arima013     <ARIMA(0,1,3)>
## 3 Central African Republic stepwise     <ARIMA(2,1,2)>
## 4 Central African Republic search       <ARIMA(3,1,0)>
```

```
#> # A mable: 4 x 3
#> # Key:     Country, Model name [4]
#>   Country                  `Model name`         Orders
#>   <fct>                    <chr>               <model>
#> 1 Central African Republic arima210     <ARIMA(2,1,0)>
#> 2 Central African Republic arima013     <ARIMA(0,1,3)>
#> 3 Central African Republic stepwise     <ARIMA(2,1,2)>
#> 4 Central African Republic search       <ARIMA(3,1,0)>
glance(caf_fit) |> arrange(AICc) |> select(.model:BIC)
```

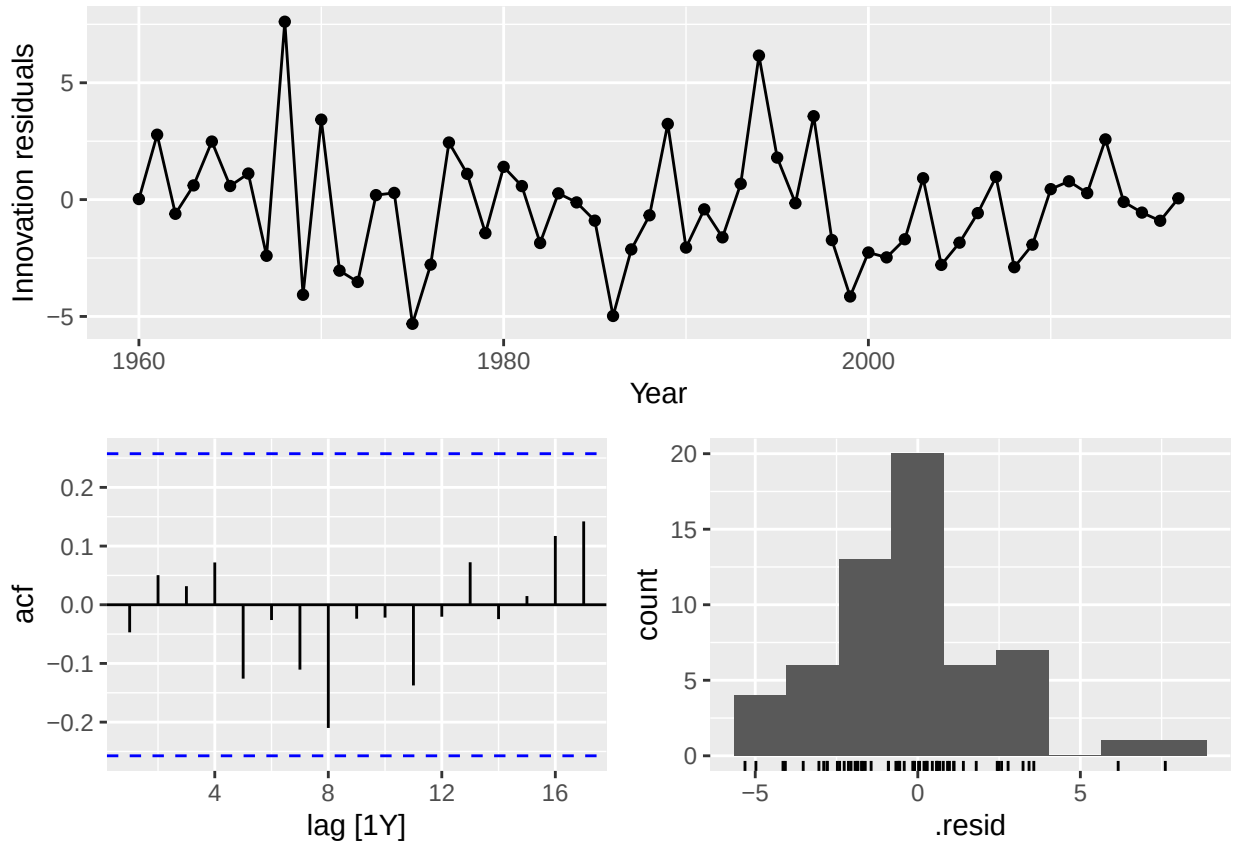
```
## # A tibble: 4 x 6
##   .model   sigma2 log_lik   AIC  AICc   BIC
##   <chr>     <dbl>   <dbl> <dbl> <dbl> <dbl>
## 1 search     6.52   -133.  274.  275.  282.
## 2 arima210   6.71   -134.  275.  275.  281.
## 3 arima013   6.54   -133.  274.  275.  282.
## 4 stepwise   6.42   -132.  274.  275.  284.
```

```
#> # A tibble: 4 × 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.
```

The four models have almost identical AICc values. Of the models fitted, the full search has found that an ARIMA(3,1,0) gives the lowest AICc value, closely followed by the ARIMA(2,1,0) and ARIMA(0,1,3) — the latter two being the models that we guessed from the ACF and PACF plots. The automated stepwise selection has identified an ARIMA(2,1,2) model, which has the highest AICc value of the four models.

6. The ACF plot of the residuals from the ARIMA(3,1,0) model shows that all autocorrelations are within the threshold limits, indicating that the residuals are behaving like white noise.

```
caf_fit |>
  select(search) |>
  gg_tsresiduals()
```



A portmanteau test (setting K=3) returns a large p-value, also suggesting that the residuals are white noise.

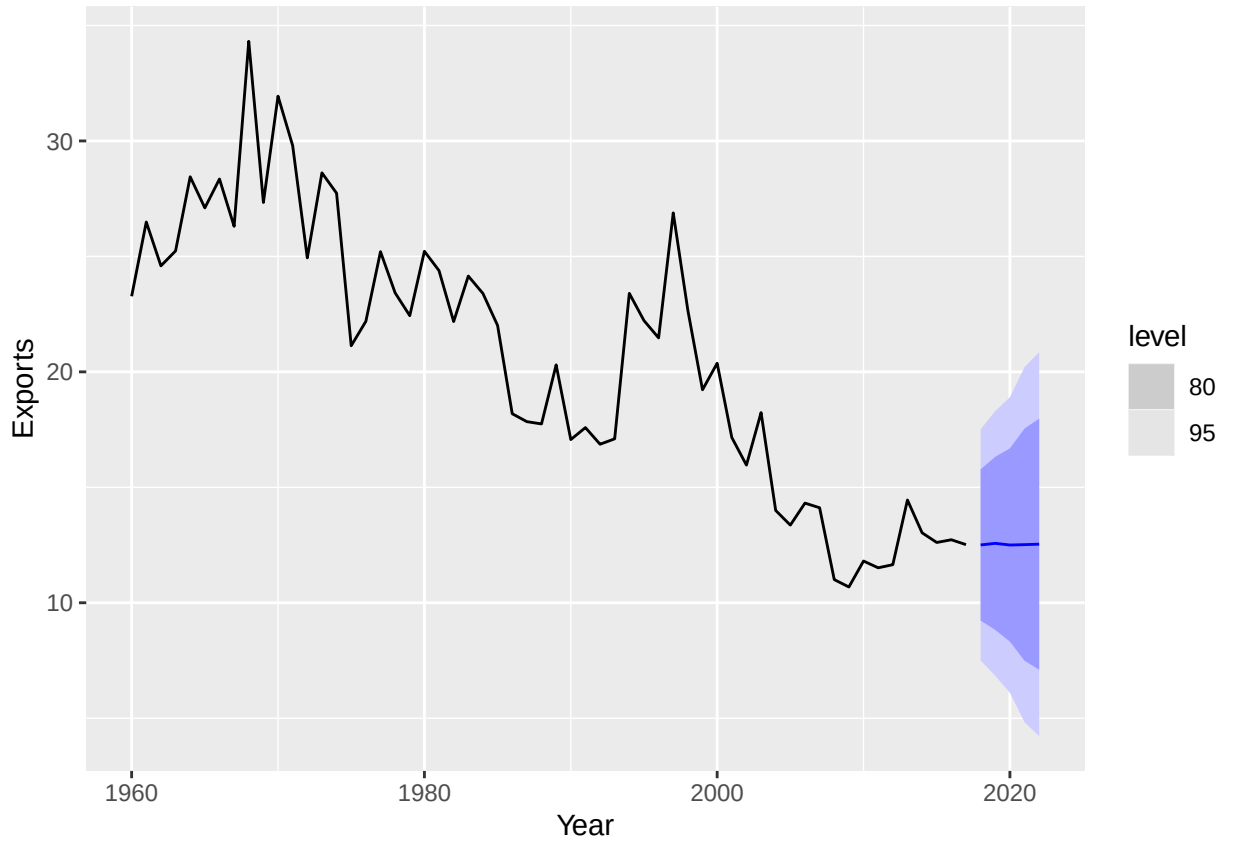
```
augment(caf_fit) |>
  filter(.model=='search') |>
  features(.innov, ljung_box, lag = 10, dof = 3)
```

```
## # A tibble: 1 x 4
##   Country                  .model lb_stat lb_pvalue
##   <fct>                    <chr>    <dbl>     <dbl>
## 1 Central African Republic search    5.75     0.569
```

```
#> # A tibble: 1 × 4
#>   Country                  .model lb_stat lb_pvalue
#>   <fct>                    <chr>    <dbl>     <dbl>
#> 1 Central African Republic search    5.75     0.569
```

7. Forecasts from the chosen model are shown in Figure 9.16.

```
caf_fit |>
  forecast(h=5) |>
  filter(.model=='search') |>
  autoplot(global_economy)
```



Note that the mean forecasts look very similar to what we would get with a random walk (equivalent to an ARIMA(0,1,0)). The extra work to include AR and MA terms has made little difference to the point forecasts in this example, although the prediction intervals are much narrower than for a random walk model.

Understanding constants in R

A non-seasonal ARIMA model can be written as

$$(1-\phi_1 B - \cdots - \phi_p B^p)(1-B)^d y_t = c + (1+\theta_1 B + \cdots + \theta_q B^q)\varepsilon_t, \quad (9.3)$$

$$(1-\phi_1 B - \cdots - \phi_p B^p)(1-B)^d (y_t - \mu t^d/d!) = (1+\theta_1 B + \cdots + \theta_q B^q)\varepsilon_t, \quad (9.4)$$

where $c = \mu(1-\phi_1 - \cdots - \phi_p)$ and μ is the mean of $(1-B)^d y_t$. The fable package uses the parameterisation of Equation (9.3) while most other R implementations use Equation (9.4).

Thus, 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 $d-1$.) When $d=0$, we have the special case that μ is the mean of y_t.

By default, the ARIMA() function 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 a quadratic or higher order trend is particularly dangerous when forecasting.

The constant can be specified by including 0 or 1 in the model formula (like the intercept in lm()). For example, to automatically select an ARIMA model with a constant, you could use ARIMA(y ~ 1 + …). Similarly, a constant can be excluded with ARIMA(y ~ 0 + …).

Plotting the characteristic roots

(This is a more advanced section and can be skipped if desired.)

We can re-write Equation (9.3) as

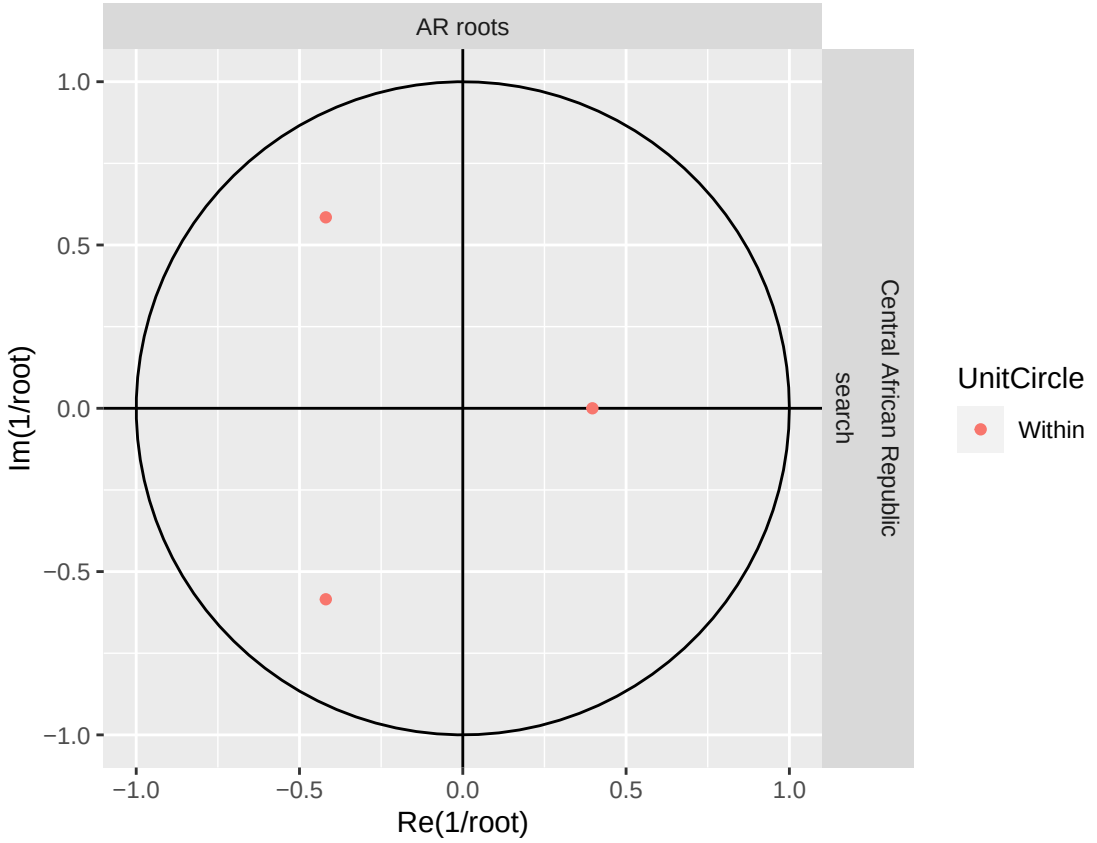
$\phi(B)(1-B)^d y_t = c + \theta(B)\varepsilon_t$

where $\phi(B) = (1 - \phi_1 B - \cdots - \phi_p B^p)$ is a p^{th} order polynomial in B and $\theta(B) = (1 + \theta_1 B + \cdots + \theta_q B_q)$ is a q^{th} order polynomial in B.

The stationarity conditions for the model are that the p complex roots of $\phi(B)$ lie outside the unit circle, and the invertibility conditions are that the q complex roots of $\theta(B)$ lie outside the unit circle. So we can see whether the model is close to invertibility or stationarity by a plot of the roots in relation to the complex unit circle.

It is easier to plot the inverse roots instead, as they should all lie within the unit circle. This is easily done in R. For the ARIMA(3,1,0) model fitted to the Central African Republic Exports, we obtain Figure 9.17.

```
gg_arma(caf_fit |> select(Country, search))
```



The three orange dots in the plot correspond to the roots of the polynomials (B). They are all inside the unit circle, as we would expect because fable ensures the fitted model is both stationary and invertible. Any roots close to the unit circle may be numerically unstable, and the corresponding model will not be good for forecasting.

The ARIMA() function will never return a model with inverse roots outside the unit circle. Models automatically selected by the ARIMA() function will not contain roots close to the unit circle either. Consequently, it is sometimes possible to find a model with better AICc value than ARIMA() will return, but such models will be potentially problematic.

9.8 Forecasting

Point Forecasts

Steps:

Suppose you have identified the model and you have written down in backshift notation.

- Rearrange ARIMA equation so y_t is on LHS
- Rewrite equation by replacing t by T+h.
- On RHS, replace future observations by their forecasts, future errors by zero, and past errors by corresponding residuals.

Start with $h = 1$. Repeat for $h = 2, 3, ...$ So, it is a iterative process.

ARIMA(3,1,1) Forecasts

Step 1:

$(1 - \hat{\phi}_1 B - \hat{\phi}_2 B^2 - \hat{\phi}_3 B^3)(1 - B)y_t = (1 + \hat{\theta}_1 B)\epsilon_t,$

After expanding this equation and rearranging the terms, we get,

$y_t = (1 + \hat{\phi}_1)y_{t-1} - (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} - (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} - \hat{\phi}_3 y_{t-4} + \epsilon_t + \hat{\theta}_1 \epsilon_{t-1}$

Now, let's generate one step ahead forecast

Step 2:

Put $t = T + 1$

$y_{T+1} = (1 + \hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \epsilon_{T+1} + \hat{\theta}_1 \epsilon_T$

Now, replace future errors by zero i.e. $\epsilon_{T+1} = 0$ because expected value is zero. Also, replace in-sample errors ϵ_T by the estimated residuals i.e. e_T, so we get forecast one step ahead as,

Step 3:

So, point forecast for $h = 2$

$\hat{y}_{T+1|T} = (1 + \hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \hat{\theta}_1 e_T$

Now, similary two step ahead,

$y_{T+2} = (1 + \hat{\phi}_1)y_{T+1} - (\hat{\phi}_1 - \hat{\phi}_2)y_T - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-1} - \hat{\phi}_3 y_{T-2} + \epsilon_{T+2} + \hat{\theta}_1 \epsilon_{T+1}$

So, point forecast for $h = 2$

Since expected value of y_{T+1} is $\hat{y}_{T+1} = y_{T+1|T}$ and expected value of future error is zero and MA part is zero, so we get.

$\hat{y}_{T+2|T} = y_{T+2} = (1 + \hat{\phi}_1)\hat{y}_{T+1|T} - (\hat{\phi}_1 - \hat{\phi}_2)y_T - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-1} - \hat{\phi}_3 y_{T-2}$

And we do this over and over multiple times more steps ahead.

Prediction Interval

Assuming $\epsilon_t \sim N(0, \sigma^2)$ (errors are normally distributed)

95% Prediction Interval $\hat{y}_{T+h|T} \pm 1.96\sqrt{v_{T+h|T}}$

Where $v_{T+h|T}$ is estimated forecast variance.

- Now, for one step ahead, we can estimate for any model, the one step ahead forecast within sample for any model is same as residual variance i.e. $v_{T+1|T} = \hat{\sigma}^2$ for all ARIMA models.
- Multi-step prediction interval for ARIMA(0,0,q):

$y_t = \epsilon_t + \Sigma_{i=1}^{q} \hat{\theta}_i \epsilon_{t-i}$.

$v_{T|T+h} = \hat{\sigma}^2[1 + \Sigma_{i=1}^{h-1} \hat{\theta}_i^{\,2}]$, for $h = 2, 3, ..$

- Other models beyond this book.
- Prediction intervals increase in size with *forecast horizon*
- Prediction intervals can be difficult to calculate by hand.
- Calculations assume that residuals are *uncorrelated* and *normally distributed*
- Prediction intervals tend to be too narrow:

(i) the *uncertainty in the parameter estimates* has not been accounted for.

(ii) *model uncertainty* has not been accounted for.

(iii) the ARIMA model assumes *historical patterns will not change* during the forecast period.

(iv) the ARIMA model assumes *uncorrelated future errors.*

The differencing has a role to play here. If $d = 0$ then the prediction intervals will increase and the flatten out so they will become parallel to the horizontal axis and increase after a point. If $d > 0$ they will increase forever.

———————————- Summary ———————————-

Point forecasts

Although we have calculated forecasts from the ARIMA models in our examples, we have not yet explained how they are obtained. Point forecasts can be calculated using the following three steps.

1. Expand the ARIMA equation so that y_t is on the left hand side and all other terms are on the right.
2. Rewrite the equation by replacing t with T+h.
3. On the right hand side of the equation, replace future observations with their forecasts, future errors with zero, and past errors with the corresponding residuals.

Beginning with $h = 1$, these steps are then repeated for $h = 2, 3, ...$ until all forecasts have been calculated.

The procedure is most easily understood via an example. We will illustrate it using a ARIMA(3,1,1) model which can be written as follows:

$(1 - \hat{\phi}_1 B - \hat{\phi}_2 B^2 - \hat{\phi}_3 B^3)(1 - B)y_t = (1 + \hat{\theta}_1 B)\varepsilon_t$.

Then we expand the left hand side to obtain

$[1 - (1 + \hat{\phi}_1)B + (\hat{\phi}_1 - \hat{\phi}_2)B^2 + (\hat{\phi}_2 - \hat{\phi}_3)B^3 + \hat{\phi}_3 B^4]y_t = (1 + \hat{\theta}_1 B)\varepsilon_t$,

and applying the backshift operator gives

$y_t - (1 + \hat{\phi}_1)y_{t-1} + (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} + (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} + \hat{\phi}_3 y_{t-4} = \varepsilon_t + \hat{\theta}_1 \varepsilon_{t-1}$.

Finally, we move all terms other than y_t to the right hand side:

$y_t = (1 + \hat{\phi}_1)y_{t-1} - (\hat{\phi}_1 - \hat{\phi}_2)y_{t-2} - (\hat{\phi}_2 - \hat{\phi}_3)y_{t-3} - \hat{\phi}_3 y_{t-4} + \varepsilon_t + \hat{\theta}_1 \varepsilon_{t-1}$. (9.5)

This completes the first step. While the equation now looks like an ARIMA(4,0,1), it is still the same ARIMA(3,1,1) model we started with. It cannot be considered an ARIMA(4,0,1) because the coefficients do not satisfy the stationarity conditions.

For the second step, we replace t with T+1 in (9.5):

$$y_{T+1} = (1+\hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \varepsilon_T + 1 + \hat{\theta}_1\varepsilon_T.$$

Assuming we have observations up to time T, all values on the right hand side are known except for ε_{T+1}, which we replace with zero, and ε_T, which we replace with the last observed residual e_T :

$$\hat{y}_{T+1|T} = (1+\hat{\phi}_1)y_T - (\hat{\phi}_1 - \hat{\phi}_2)y_{T-1} - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-2} - \hat{\phi}_3 y_{T-3} + \hat{\theta}_1 e_T.$$

A forecast of y_{T+2} is obtained by replacing t with $T+2$ in (9.5) . All values on the right hand side will be known at time$T$ except y_{T+1} which we replace with $\hat{y}_{T+1|T}$, and ε_{T+2} and ε_{T+1}, both of which we replace with zero:

$$\hat{y}_{T+2|T} = (1+\hat{\phi}_1)\hat{y}_{T+1|T} - (\hat{\phi}_1 - \hat{\phi}_2)y_T - (\hat{\phi}_2 - \hat{\phi}_3)y_{T-1} - \hat{\phi}_3 y_{T-2}.$$

The process continues in this manner for all future time periods. In this way, any number of point forecasts can be obtained.

Prediction intervals

The calculation of ARIMA prediction intervals is more difficult, and the details are largely beyond the scope of this book. We will only give some simple examples.

The first prediction interval is easy to calculate. If $\hat{\sigma}$ is the standard deviation of the residuals, then a 95% prediction interval is given by $\hat{y}_{T+1|T} \pm 1.96\hat{\sigma}$. This result is true for all ARIMA models regardless of their parameters and orders.

Multi-step prediction intervals for ARIMA(0,0,q) models are relatively easy to calculate. We can write the model as

$$y_t = \varepsilon_t + \Sigma_{i=1}^{q}\theta_i\varepsilon_{t-i}.$$

Then, the estimated forecast variance can be written as

$$\hat{\sigma}_h^2 = \hat{\sigma}^2[1 + \Sigma_{i=1}^{h-1}\hat{\theta}_i^2], \text{ for } h = 2, 3, ...,$$

where $\hat{\theta}_i = 0$ for i>q, and a 95% prediction interval is given by $\hat{y}_{T+h|T} \pm 1.96\hat{\sigma}_h$.

In Section 9.4, we showed that an AR(1) model can be written as an MA(∞) model. Using this equivalence, the above result for MA(q) models can also be used to obtain prediction intervals for AR(1) models.

More general results, and other special cases of multi-step prediction intervals for an ARIMA(p,d,q) model, are given in more advanced textbooks such as Brockwell & Davis (2016).

The prediction intervals for ARIMA models are based on assumptions that the residuals are uncorrelated and normally distributed. If either of these assumptions does not hold, then the prediction intervals may be incorrect. For this reason, always plot the ACF and histogram of the residuals to check the assumptions before producing prediction intervals.

If the residuals are uncorrelated but not normally distributed, then bootstrapped intervals can be obtained instead, as discussed in Section 5.5. This is easily achieved by simply adding bootstrap=TRUE in the forecast() function.

In general, prediction intervals from ARIMA models increase as the forecast horizon increases. For stationary models (i.e., with d=0) they will converge, so that prediction intervals for long horizons are all essentially the same. For d 1, the prediction intervals will continue to grow into the future.

As with most prediction interval calculations, ARIMA-based intervals tend to be too narrow. This occurs because only the variation in the errors has been accounted for. There is also variation in the parameter estimates, and in the model order, that has not been included in the calculation. In addition, the calculation assumes that the historical patterns that have been modelled will continue into the forecast period.

9.9 Seasonal ARIMA(SARIMA) models

They behave similarly to non-seasonal ARIMA models. The difference is we have another part of the model which describes the seasonal nature of the series.

Till now we have discussed about ARIMA(p,d,q) model, where p,d,q describes the non-seasonal part of the model. Now, we are going to add a section with uppercase P, D, Q to describe the seasonal part of the model and a little "m" at the end i.e. $(P, D, Q)_m$ which says how many observations per year so for example, 12 for monthly data.

So, here, lowercase p,d,q says non-seasonal part of the model and uppercase P,D,Q says seasonal part of the model.

Example

$ARIMA(1,1,1)(1,1,1)_4$ model (without constant):

m=4 means it is quarterly data. If I write equation in backshift notation:

$(1-\phi_1 B)(1-\Phi_1 B^4)(1-B)(1-B^4)y_t = (1+\theta_1 B)(1+\Theta_1 B^4)\epsilon_t$

Where

$(1-\phi_1 B)$ is non-seasonal AR(1)

$(1-\Phi_1 B^4)$ is seasonal AR(1)

$(1-B)$ is non-seasonal difference

$(1-B^4)$ is seasonal difference

$(1-\theta_1 B)$ is non-seasonal MA(1)

$(1-\Theta_1 B^4)$ is seasonal MA(1)

All the factors can be multiplied out and the general model written as follows:

$y_t = (1+\phi_1)y_{t-1} - \phi_1 y_{t-2} + (1+\Phi_1)y_{t-4} - (1+\phi_1+\Phi_1+\phi_1\Phi_1)y_{t-5} + (\phi_1+\phi_1\Phi_1)y_{t-6} - \Phi_1 y_{t-8} + (\Phi_1+\phi_1\Phi_1)y_{t-9} - \phi_1\Phi_1 y_{t-10} + \epsilon_t + \theta_1\epsilon_{t-1} + \Theta_1\epsilon_{t-4} + \theta_1\Theta_1\epsilon_{t-5}$

It is very complex so it is better to write in backshift notation.

Just as with non-seasonal model, we can look at the acf and guess an appropriate moving average model or we can look at the pacf and guess an appropriate AR model. We can do the same with these seasonal models. except that we only look at the spikes that are multiples of the period. So, if it is monthly data, we want to look at the spikes at the multiple of 12 i.e. 12,24,36,... If it is quarterly data, we look at the spikes at lags 4,8,12,16,... If we have look at the last significant spike in the acf on the seasonal lags, it gives you the seasonal MA(q) order and if we have a autoregressive model, we see the last significant spike in the seasonal lags in the pacf. So,

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF or ACF.

$ARIMA(0,0,0)(0,0,1)_{12}$ will show a spike at lag 12 in the acf but no other significant spikes. The PACF will show exponentially decay in the seasonal lags; that is, at lags 12,24,36,...

$ARIMA(0,0,0)(1,0,0)_{12}$ will show exponential decay in the seasonal lags of the ACF. A single significant spike at lag 12 in the PACF.

How does ARIMA() work for seasonal models ?

A seasonal ARIMA process

$\Phi(B)\phi(B)(1-B)^d(1-B)^D y_t = c + \Theta(B)\theta(B)\epsilon_t$

Need to select appropriate orders: d,D,p,q,P,Q and whether to include the intercept c.

Hyndman and Khandakar (JSS,2008) algorithm:

- Select no. of differences d via KPSS test and D using seasonal strength. If seasonality is strong then we take the seasonal difference otherwise we won't.
- Select p,q,P,Q and c by minimizing AICc.
- Use stepwise search to traverse model space.

Example: US leisure employment

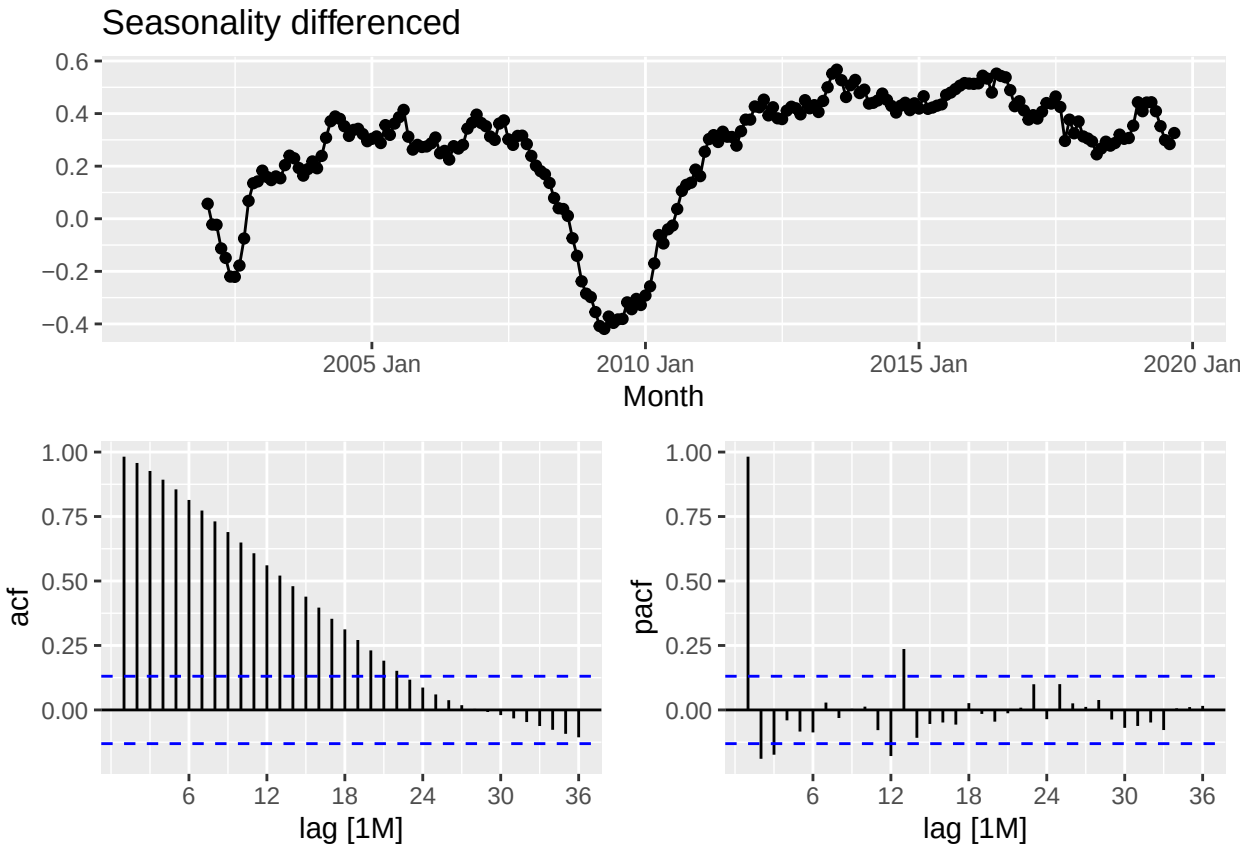
```
leisure <- us_employment |>
  filter(Title=="Leisure and Hospitality", year(Month)>2000) |>
  mutate(Employed = Employed/1000) |>
  select(Month,Employed)
autoplot(leisure,Employed)+
  labs(title = "US employment: leisure and hospitality",y="People (millions)")
```

Here we have a very strong seasonality and a bit of trend, so we are going to need a seasonal ARIMA model.

```
leisure |>
  gg_tsdisplay(difference(Employed,12),plot_type = "partial",lag=36)+
  labs(title="Seasonality differenced",y="")
```

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```

```
## Warning: Removed 12 rows containing missing values (`geom_point()`).
```

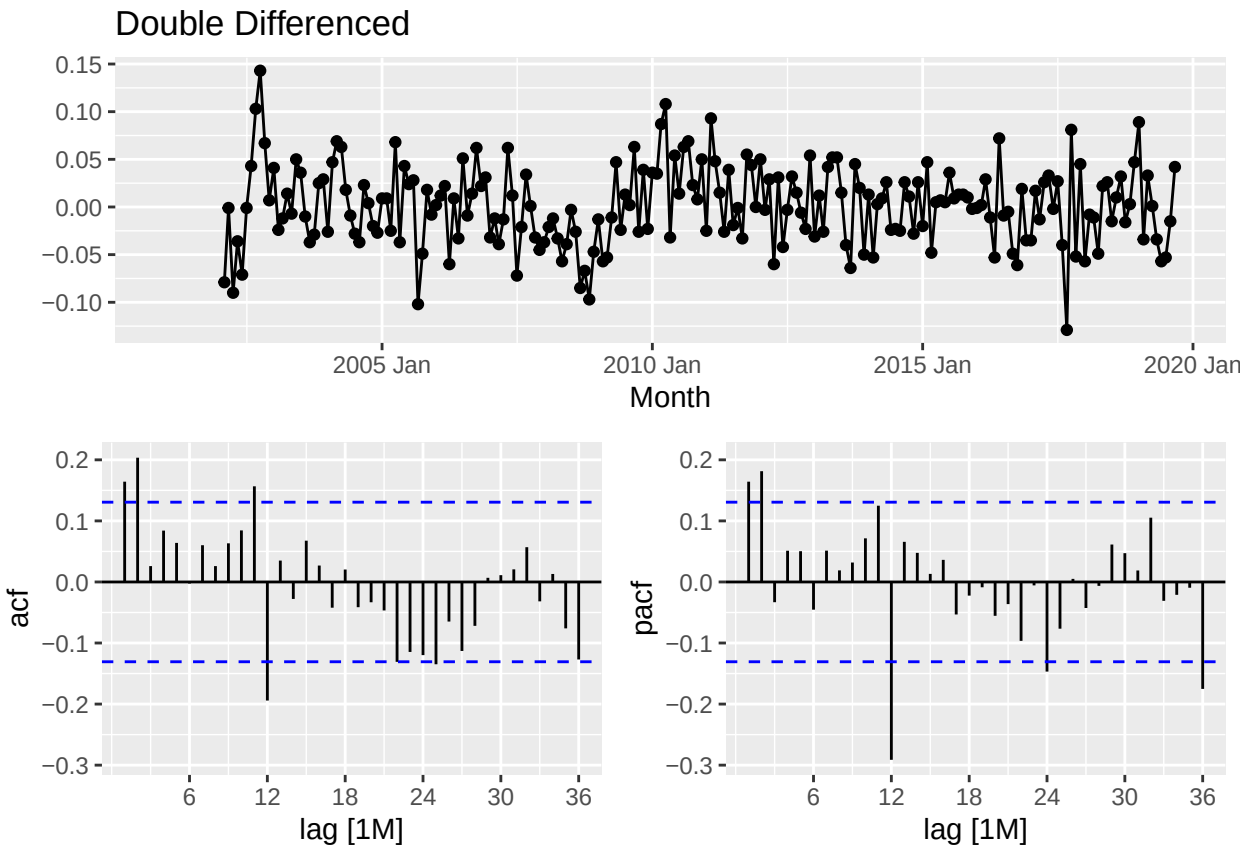


Here, we have done differencing at lag 12 because that's a seasonal lag. So we take a difference at lag 12. We can see the ACF and PACF plot and it's clearly non-stationary, it's wandering around. So although the seasonal differences removed most of the seasonality in the data that still has some non-stationary behavior, so we need to do another differences as well to make it stationary. So, if we do the second order differencing (that is add one more non-seasonal (lag 1) difference)

```
leisure |>
  gg_tsdisplay(difference(Employed,12) |> difference(),
               plot_type = "partial",lag=36)+
  labs(title = "Double Differenced", y ="")
```

```
## Warning: Removed 13 rows containing missing values (`geom_line()`).
```

```
## Warning: Removed 13 rows containing missing values (`geom_point()`).
```



Now, it looks stationary. Although in acf and pacf, there are some significant spikes, we don't have the long decay on the positive side which is typical non-stationary data.

Since, we have done two differencing, so $d = 1, D = 1$ and seasonal lags are at 12,24,36 and all of these lags are significant on the pacf side and on acf side, only one significant spike at lag 12. So possibly we will pick seasonal MA(1) and for non-seasonal lags, upto lag 12, we have 2 spikes in both acf and pacf plots, so we have a choice: i.e. either non-seasonal MA(2) or non-seasonal AR(2), either would be okay.

So, these are the possibilities:

```
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, approx=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]>
```

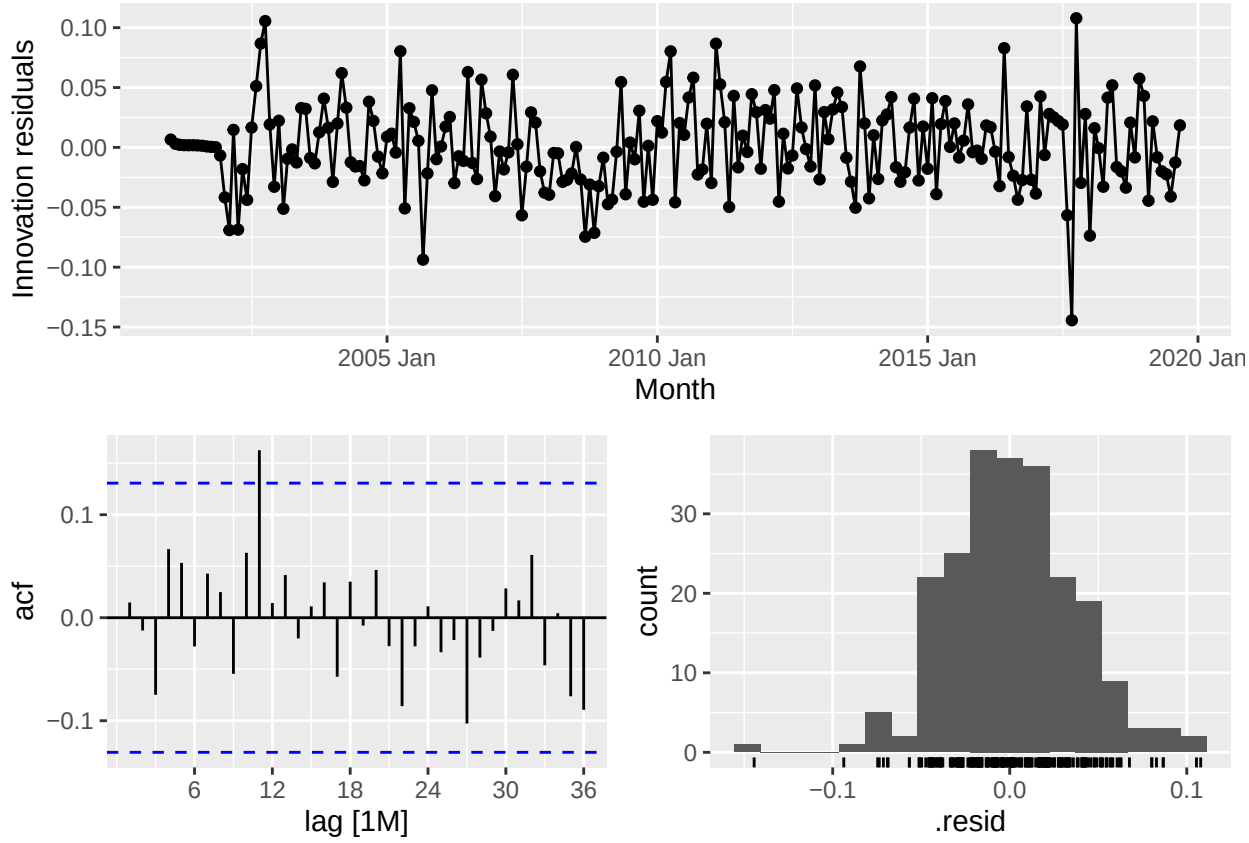
```
glance(fit) |>
  arrange(AICc) |>
  select(.model:BIC)
```

```
## # 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.
```

Here, automatically chosen model is doing slightly better.

Now, we look at the residuals.

```
fit |>
  select(auto) |>
  gg_tsresiduals(lag=36)
```



Firsy thing to notice that we have a little spike in acf plot at lag 11 but it is one lag out of 36, so it is okay. Now, see how ljung_box test goes.

```
augment(fit) |>
  filter(.model == "auto") |>
  features(.innov, 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
```

Here, I will take 24 lags so 2 years and I am going to use degrees of freedom = 4 because auto model has 4 parameters 2+1+1=4. Here, p-value is good, so it is a good model. Now, we do some forecasting.

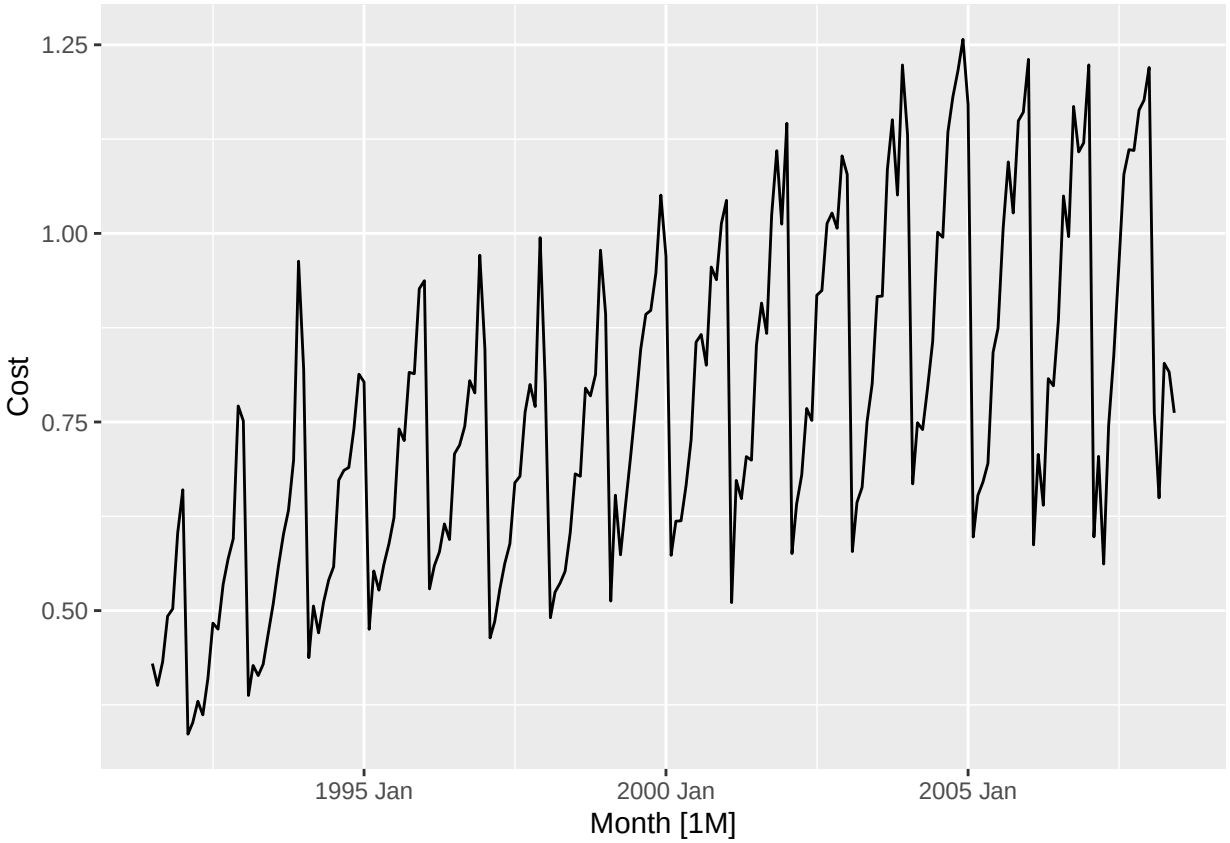
```
forecast(fit, h=36) |>
  filter(.model == "auto") |>
  autoplot(leisure) +
  labs(title = "US employment: leisure and hospitality", y = "People (in millions)")
```

It looks great. You might wonder where the trend comes from. Thats because of two lots of differencing, so double differencing induce that. Here, d=1 and D=1 makes two differencing.

Example: Corticosteriod drug sales

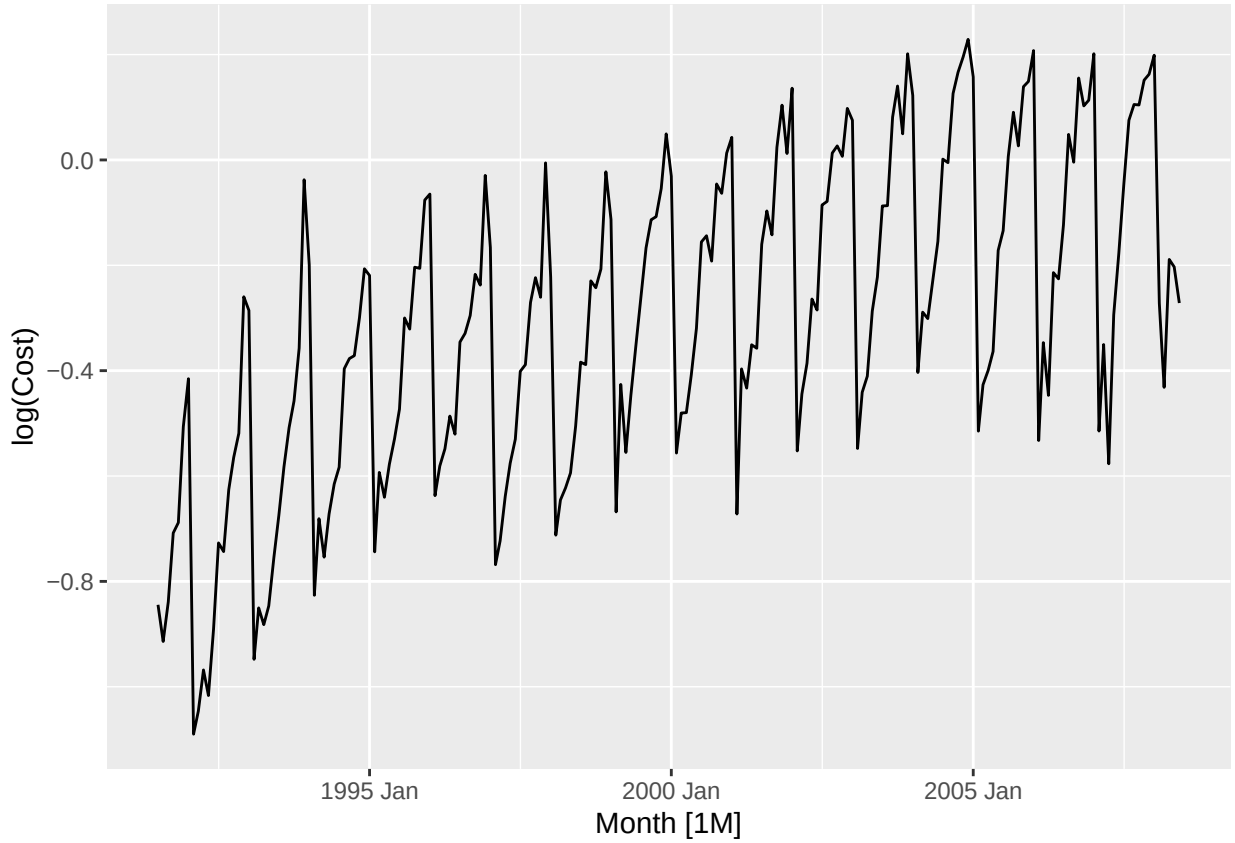
```
h02 <- PBS |>
  filter(ATC2 =="H02") |>
  summarise(Cost = sum(Cost)/1e6)
```

```
h02 |> autoplot(Cost)
```



here we have little differences/variation of peaks initially and at the end. So taking log makes it even better.

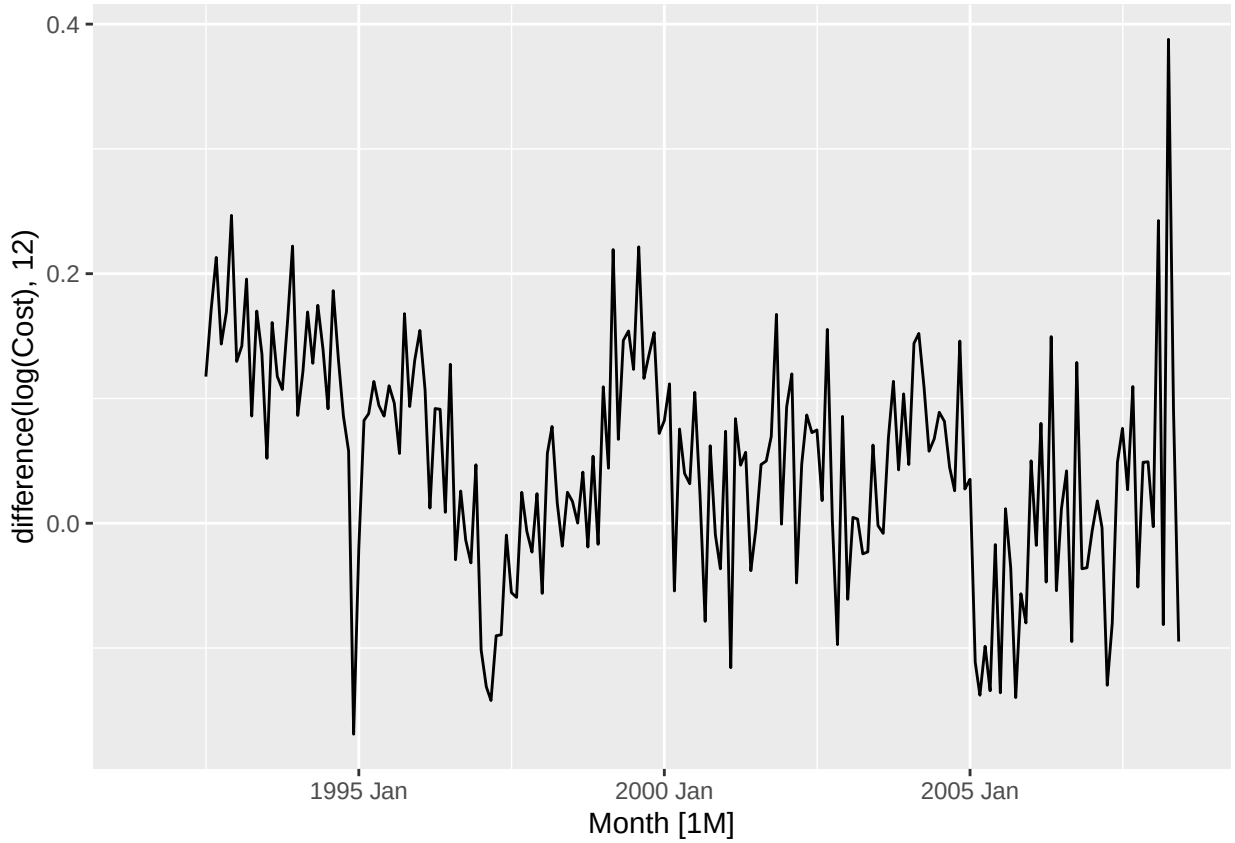
```
h02 |> autoplot(log(Cost))
```



it is slightly even and better now. it's box-cox transformation. Since it is seasonal so we take a difference of 12.

```
h02 |> autoplot(log(Cost) |> difference(12))
```

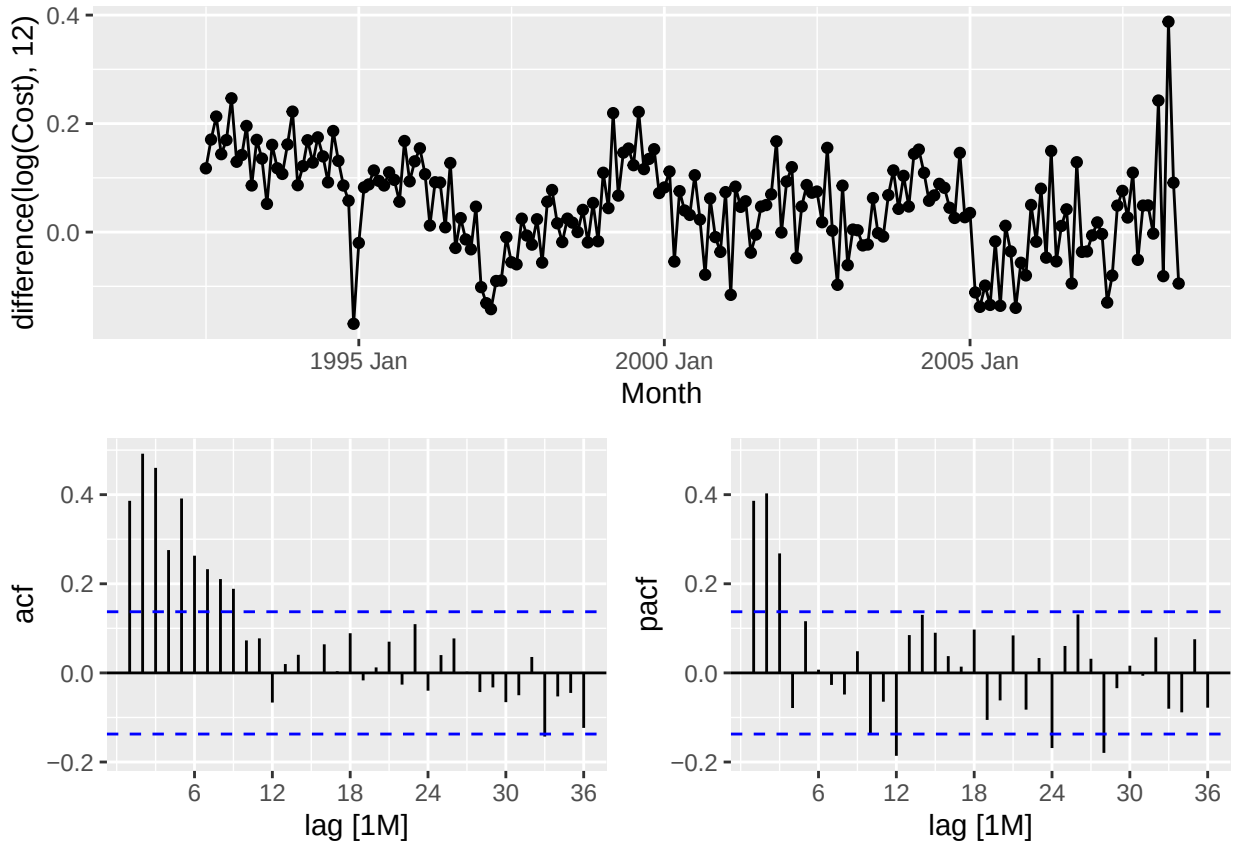
```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```



It is now closer to stationary. We will not go for next differencing.

```
h02 |> gg_tsdisplay(difference(log(Cost),12), lag_max = 36, plot_type = "partial")
```

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
## Warning: Removed 12 rows containing missing values (`geom_point()`).
```



For seasonal lags, we have not found any significant spike in acf plot but in pacf we significant spikes are ar 12, 24,... So, seasonal side, it is AR(2)

ffor seasonal lags, we found 3 lags in pacf but in acf, it is complicated and we don't want a model with too many parameters at this stage, so we take, AR(3) on pacf side only.

So, we get:

- Choose D=1 and d=0
- Spikes in PACF at lags 12 and 24 suggests seasonal AR(2) term.
- Spikes in pacf suggests possible non-seasonal AR(3) term.
- Initial candidate model: $ARIMA(3,0,0)(2,1,0)_{12}$

We did little change in values and tries various models with same d and D and $ARIMA(3,0,1)(0,1,2)_{12}$ gives better AICc. So fit this model.

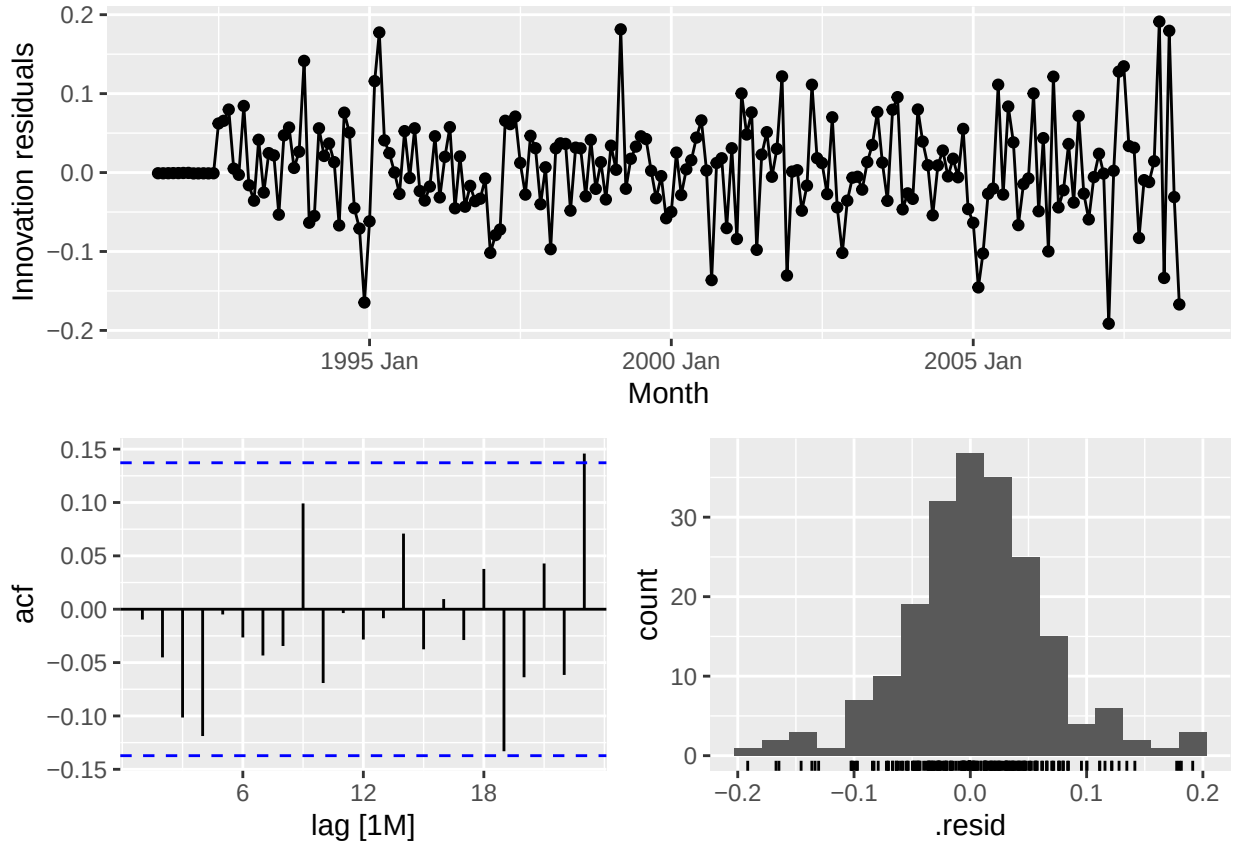
```
fit <- h02 |>
  model(best=ARIMA(log(Cost) ~ 0 + pdq(3,0,1) + PDQ(0,1,2)))
report(fit)
```

```
## Series: Cost
## Model: ARIMA(3,0,1)(0,1,2)[12]
## Transformation: log(Cost)
##
## Coefficients:
##          ar1     ar2     ar3     ma1     sma1     sma2
##      -0.1603  0.5481  0.5678  0.3827  -0.5222  -0.1768
```

```
## s.e.   0.1636  0.0878  0.0942  0.1895   0.0861   0.0872
##
## sigma^2 estimated as 0.004278:  log likelihood=250.04
## AIC=-486.08   AICc=-485.48   BIC=-463.28
```

"0" means we are not adding constant here.

```
gg_tsresiduals(fit)
```




```
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
```

Here, p-value is less tan 0.05. It is something about data is not captured in the model. There's information left in the data. Significant correlations left in the residuals that are not capturing in the model. I still could do some forecasting probably that are not too bad. But let's see of we can find something better.

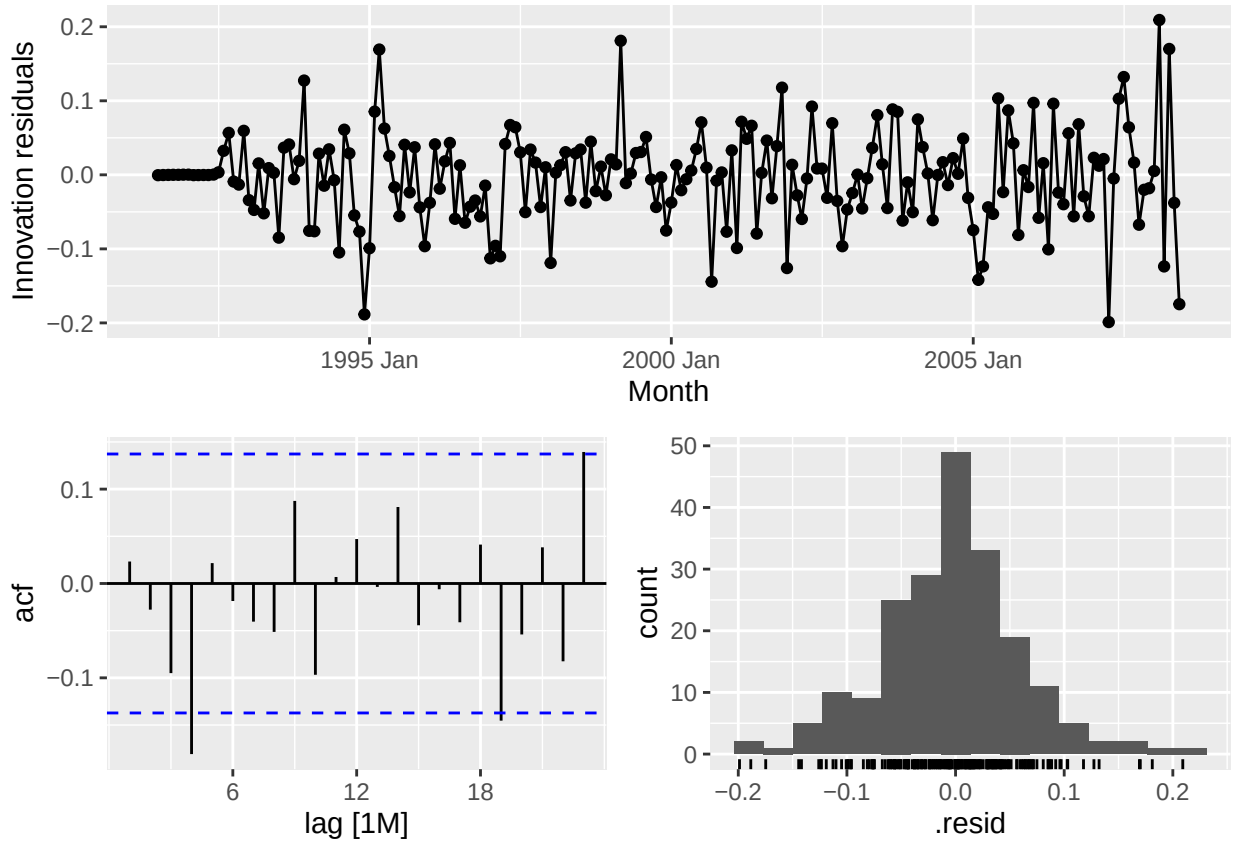
```
fit <- h02 |> model(
  auto = ARIMA(log(Cost))
)
report(fit)
```

```
## 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=245.39
## AIC=-482.78   AICc=-482.56   BIC=-469.77
```

Here, we can see (2,1,0)(0,1,1) is good model. Notice here, second order differencing is done. So, d=1 and D=1.

```
gg_tsresiduals(fit)
```



Now, look at the residuals, it is little worse than what we had before.

```
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
```

This model has 3 parameters so dof=3. nOW P-VALUE is worse now. So, this model is not doing well in terms of capturing the information in the dataset.

Now, let's use the power of computation and make it work really really hard and look for quite big models. So, we can fit quite large models as:

```
fit <- h02 |>
  model(best = ARIMA(log(Cost),
                     stepwise = FALSE, approximation = FALSE,
                     order_constraint = P+q+P+Q<=9))
```

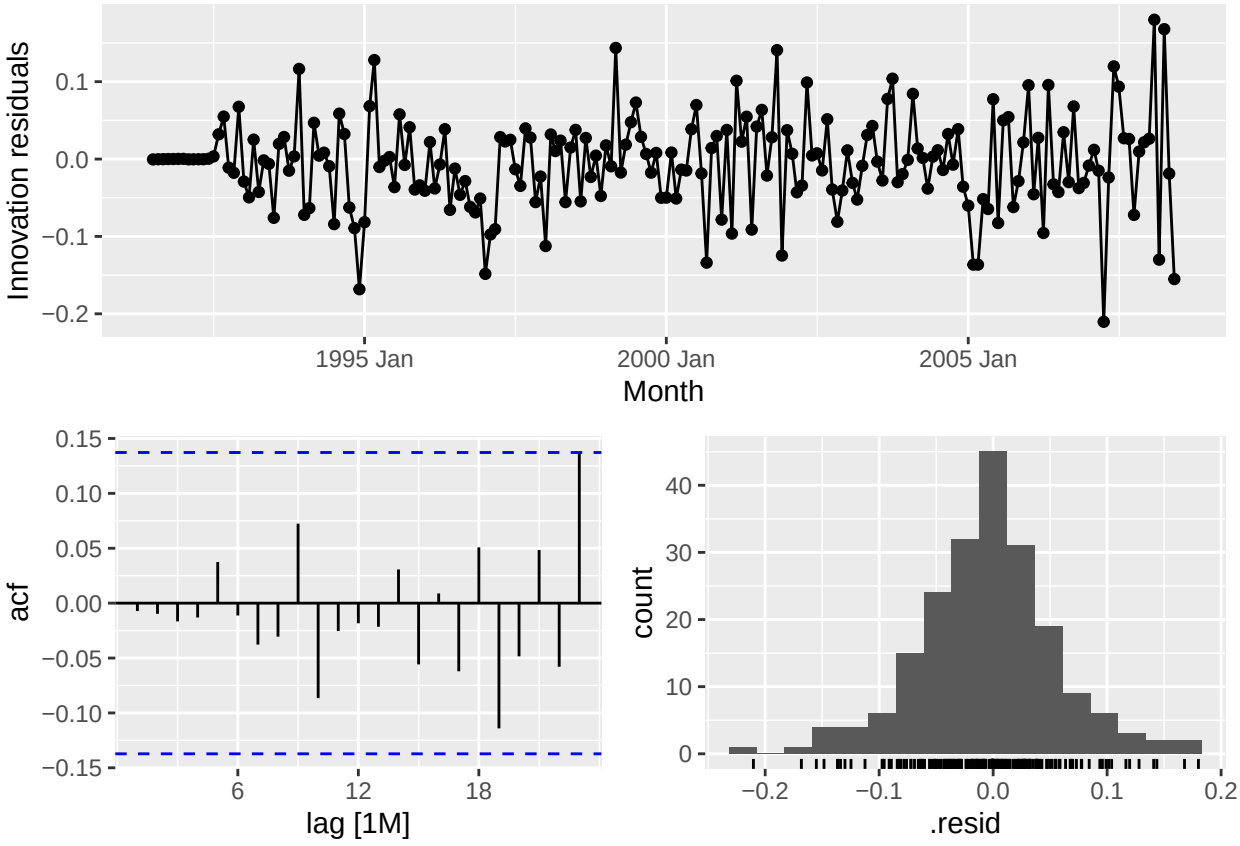
```
## Warning: Model specification induces a quadratic or higher order polynomial trend.
## This is generally discouraged, consider removing the constant or reducing the number of differences.
```

```
report(fit)
```

```
## Series: Cost
## Model: ARIMA(4,1,1)(2,1,2)[12]
## Transformation: log(Cost)
##
## Coefficients:
##           ar1     ar2     ar3      ar4      ma1    sar1     sar2     sma1
##       -0.0425  0.2098  0.2017  -0.2273  -0.7424  0.6213  -0.3832  -1.2019
## s.e.   0.2167  0.1813  0.1144   0.0810   0.2074  0.2421   0.1185   0.2491
##          sma2
##       0.4959
## s.e.  0.2135
##
## sigma^2 estimated as 0.004049:  log likelihood=254.31
## AIC=-488.63   AICc=-487.4   BIC=-456.1
```

Here, we have used 9 parameters which is the edge of the parameter space. Here, AICc statistics is better than the other one that was selected automatically.

```
gg_tsresiduals(fit)
```



They don't look too bad.

```
augment(fit) |>
  features(.innov, ljung_box,lag=36,dof=9)
```

```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 best      36.5     0.106
```

Here, p-value is better than 0.05. So one model now passes the test.

Now see how well these models work with the training and test set.

Training Data: July 1991 to June 2006

Test Data: July 2006 to June 2008

```
fit <- h02 |>
  filter_index(~ "2006 Jun") |>
  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)
```

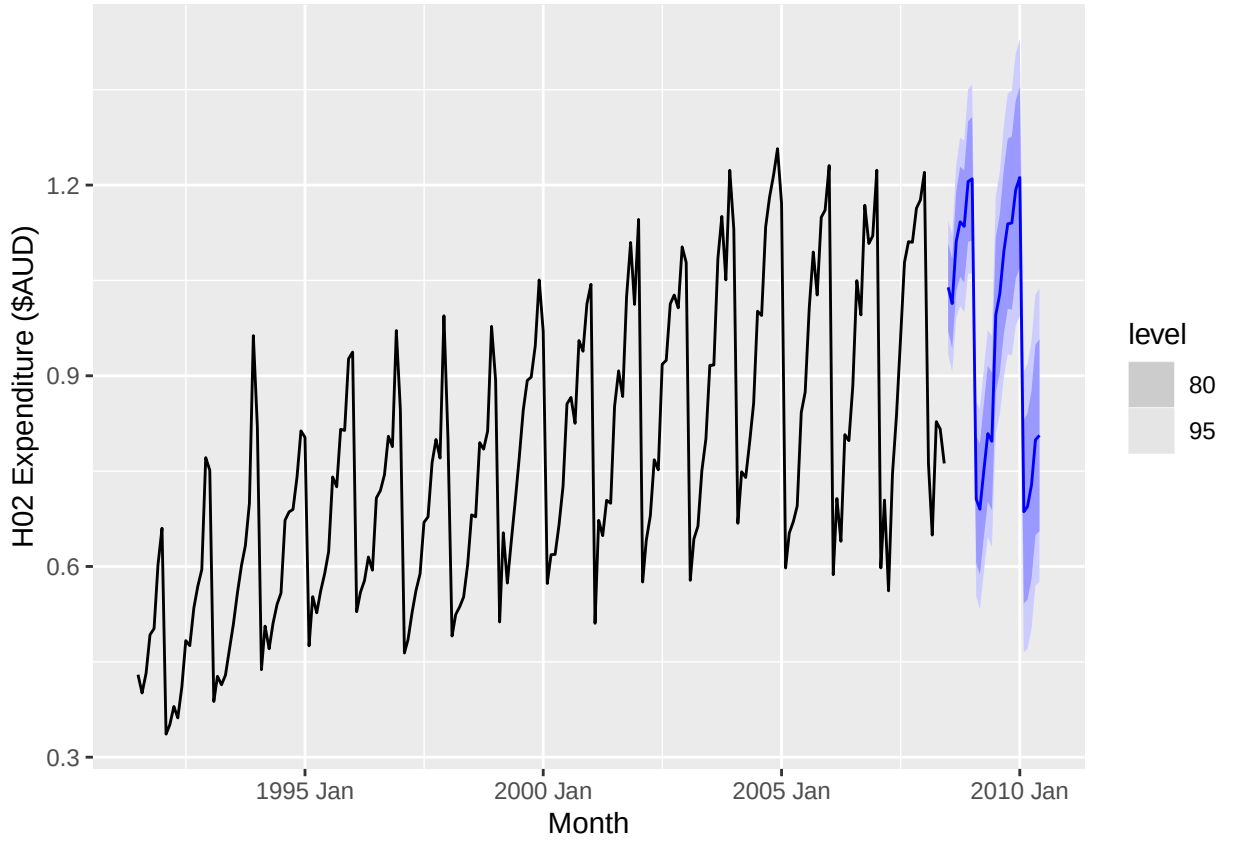
```
## # A tibble: 4 x 10
##   .model             .type       ME   RMSE    MAE    MPE  MAPE  MASE RMSSE   ACF1
##   <chr>              <chr>    <dbl>  <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl>  <dbl>
## 1 ARIMA(log(Cost) ~ Test  -0.0141  0.0668 0.0533 -1.92   6.36 0.891 0.929 -0.197
## 2 ARIMA(log(Cost) ~ Test   0.00583 0.0666 0.0491  0.462  5.74 0.822 0.927 -0.246
## 3 ARIMA(log(Cost) ~ Test  -0.0129  0.0653 0.0512 -1.80   6.16 0.856 0.908 -0.168
## 4 ARIMA(log(Cost) ~ Test  -0.0128  0.0651 0.0514 -1.80   6.19 0.860 0.906 -0.163
```

I am not using AICc for this comparison of models. I am using root mean square on the test set. Therefore it does not matter how many we have done differencing, test set comparisions are still going to be valid. The only thing I can't do the AICc statistic on the training set if I am using different lots of differencing.

Here, we have to make a decision which model we have to use. I am going to use ARIMA(3,0,1)(0,1,2)[12] because it is best one on AICc on the whole dataset and second best on the test set. So it does pretty well everywhere.

- Models with lowest AICc values tend to give slightly better results than the other models.
- AICc comparisons must have the same orders of differencing. But RMSE test set comparisons can involve any models.
- Use the best model available, even if it does not pass all the tests.

```
fit <- h02 |>
  model(ARIMA(Cost ~ 0 + pdq(3,0,1) + PDQ(0,1,2)))
fit |>
  forecast() |>
  autoplot(h02) +labs(y="H02 Expenditure ($AUD)")
```



It is pretty doing well.

Summary

So far, we have restricted our attention to non-seasonal data and non-seasonal ARIMA models. However, ARIMA models are also capable of modelling a wide range of seasonal data.

A seasonal ARIMA model is formed by including additional seasonal terms in the ARIMA models we have seen so far.

We use uppercase notation for the seasonal parts of the model, and lowercase notation for the non-seasonal parts of the model.

The seasonal part of the model consists of terms that are similar to the non-seasonal components of the model, but involve backshifts of the seasonal period. For example, an ARIMA$(1,1,1)(1,1,1)_4$ model (without a constant) is for quarterly data (m=4), and can be written as

$(1-\phi_1 B)(1-\Phi_1 B^4)(1-B)(1-B^4)y_t = (1+\theta_1 B)(1+\Theta_1 B^4)\varepsilon_t$.

ACF/PACF

The seasonal part of an AR or MA model will be seen in the seasonal lags of the PACF and ACF. For example, an ARIMA$(0,0,0)(0,0,1)_{12}$ model will show:

a spike at lag 12 in the ACF but no other significant spikes; exponential decay in the seasonal lags of the PACF (i.e., at lags 12, 24, 36, …). Similarly, an ARIMA$(0,0,0)(1,0,0)_{12}$ model will show:

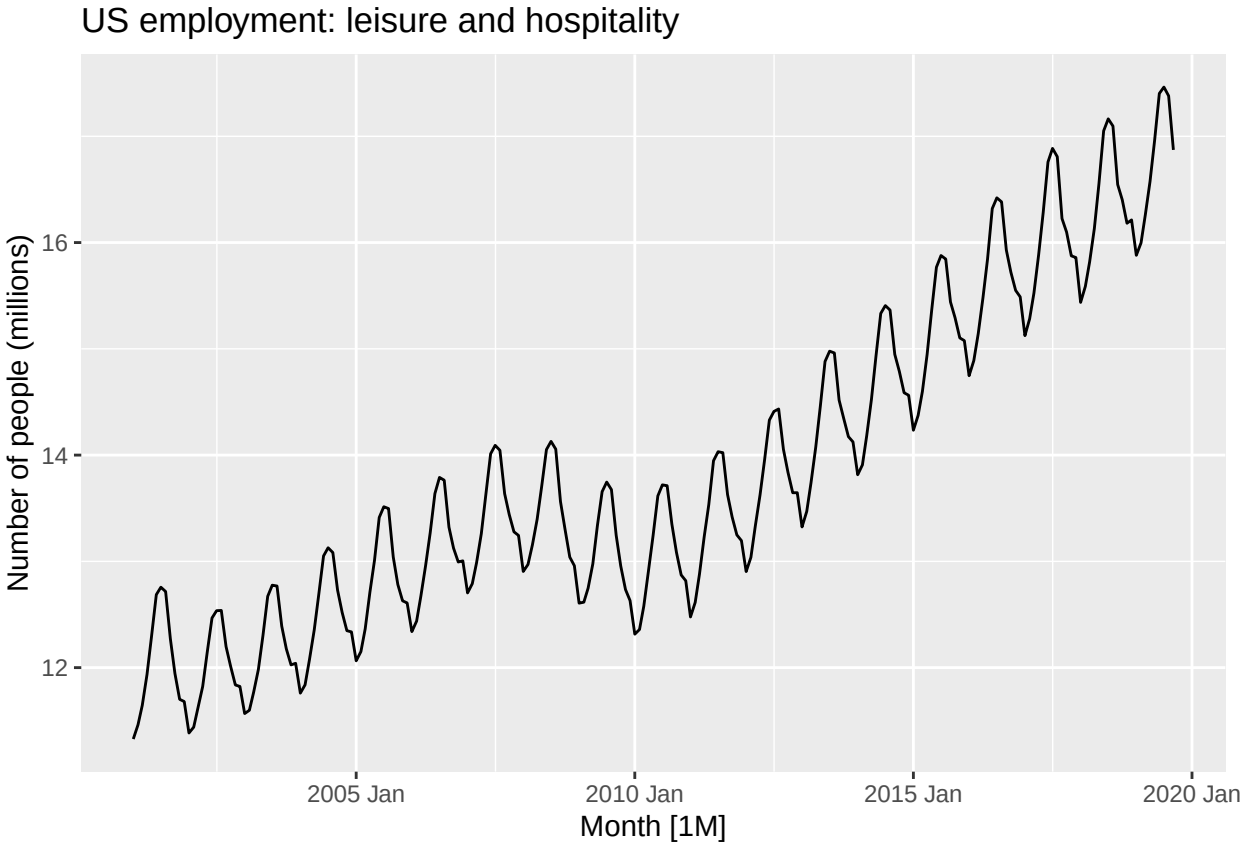
exponential decay in the seasonal lags of the ACF; a single significant spike at lag 12 in the PACF. In considering the appropriate seasonal orders for a seasonal ARIMA model, restrict attention to the seasonal lags.

The modelling procedure is almost the same as for non-seasonal data, except that we need to select seasonal AR and MA terms as well as the non-seasonal components of the model. The process is best illustrated via examples.

Example: Monthly US leisure and hospitality employment

We will describe seasonal ARIMA modelling using monthly US employment data for leisure and hospitality jobs from January 2001 to September 2019, shown in Figure 9.18.

```
leisure <- us_employment |>
  filter(Title == "Leisure and Hospitality",
         year(Month) > 2000) |>
  mutate(Employed = Employed/1000) |>
  select(Month, Employed)
autoplot(leisure, Employed) +
  labs(title = "US employment: leisure and hospitality",
       y="Number of people (millions)")
```

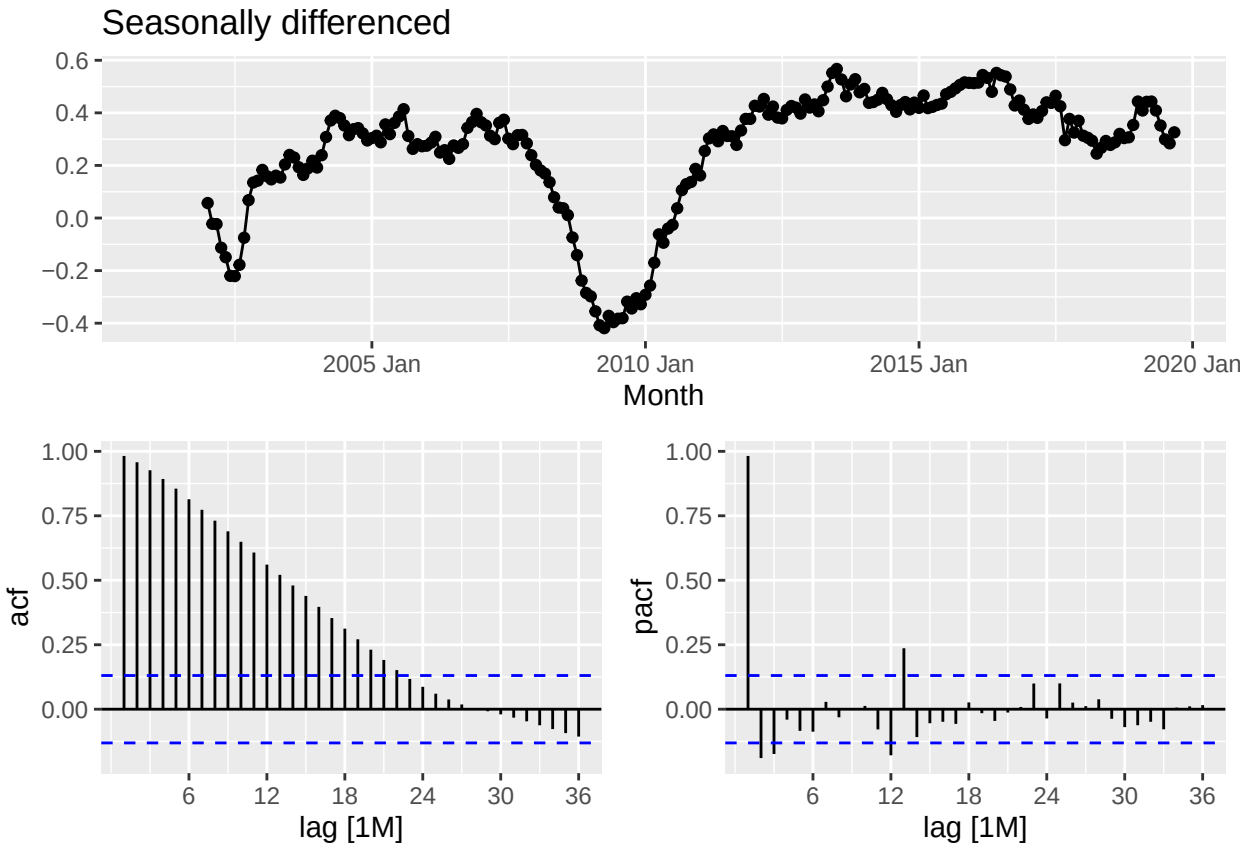


The data are clearly non-stationary, with strong seasonality and a nonlinear trend, so we will first take a seasonal difference. The seasonally differenced data are shown in Figure 9.19.

```
leisure |>
  gg_tsdisplay(difference(Employed, 12),
               plot_type='partial', lag=36) +
  labs(title="Seasonally differenced", y="")
```

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```

```
## Warning: Removed 12 rows containing missing values (`geom_point()`).
```

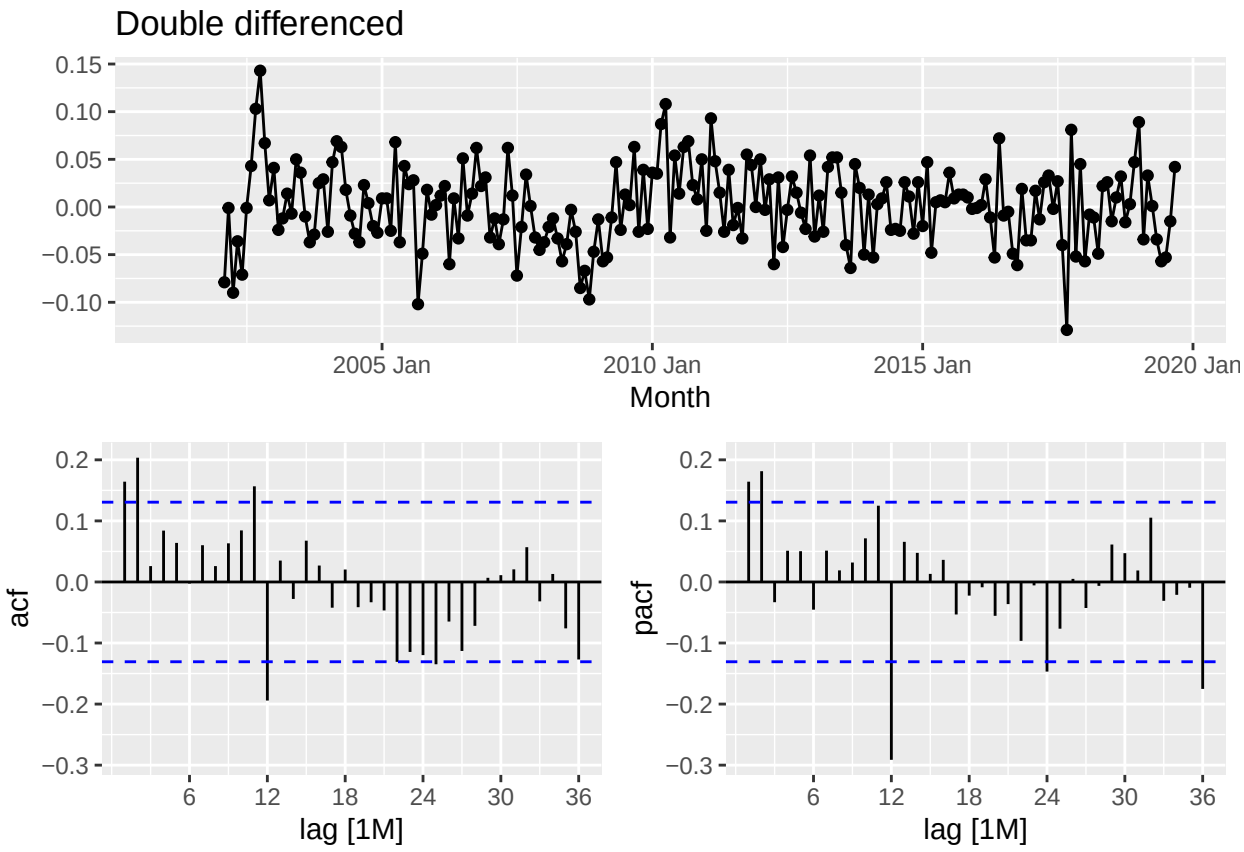


These are also clearly non-stationary, so we take a further first difference in Figure 9.20.

```
leisure |>
  gg_tsdisplay(difference(Employed, 12) |> difference(),
               plot_type='partial', lag=36) +
  labs(title = "Double differenced", y="")
```

```
## Warning: Removed 13 rows containing missing values (`geom_line()`).
```

```
## Warning: Removed 13 rows containing missing values (`geom_point()`).
```



Our aim now is to find an appropriate ARIMA model based on the ACF and PACF shown in Figure 9.20. The significant spike at lag 2 in the ACF suggests a non-seasonal MA(2) component. The significant spike at lag 12 in the ACF suggests a seasonal MA(1) component. Consequently, we begin with an ARIMA$(0,1,2)(0,1,1)_{12}$ model, indicating a first difference, a seasonal difference, and non-seasonal MA(2) and seasonal MA(1) component. If we had started with the PACF, we may have selected an ARIMA$(2,1,0)(0,1,1)_{12}$ model — using the PACF to select the non-seasonal part of the model and the ACF to select the seasonal part of the model. We will also include an automatically selected model. By setting stepwise=FALSE and approximation=FALSE, we are making R work extra hard to find a good model. This takes much longer, but with only one series to model, the extra time taken is not a problem.

```
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, approx = 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]>
```

```
#> # 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]>
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.
```

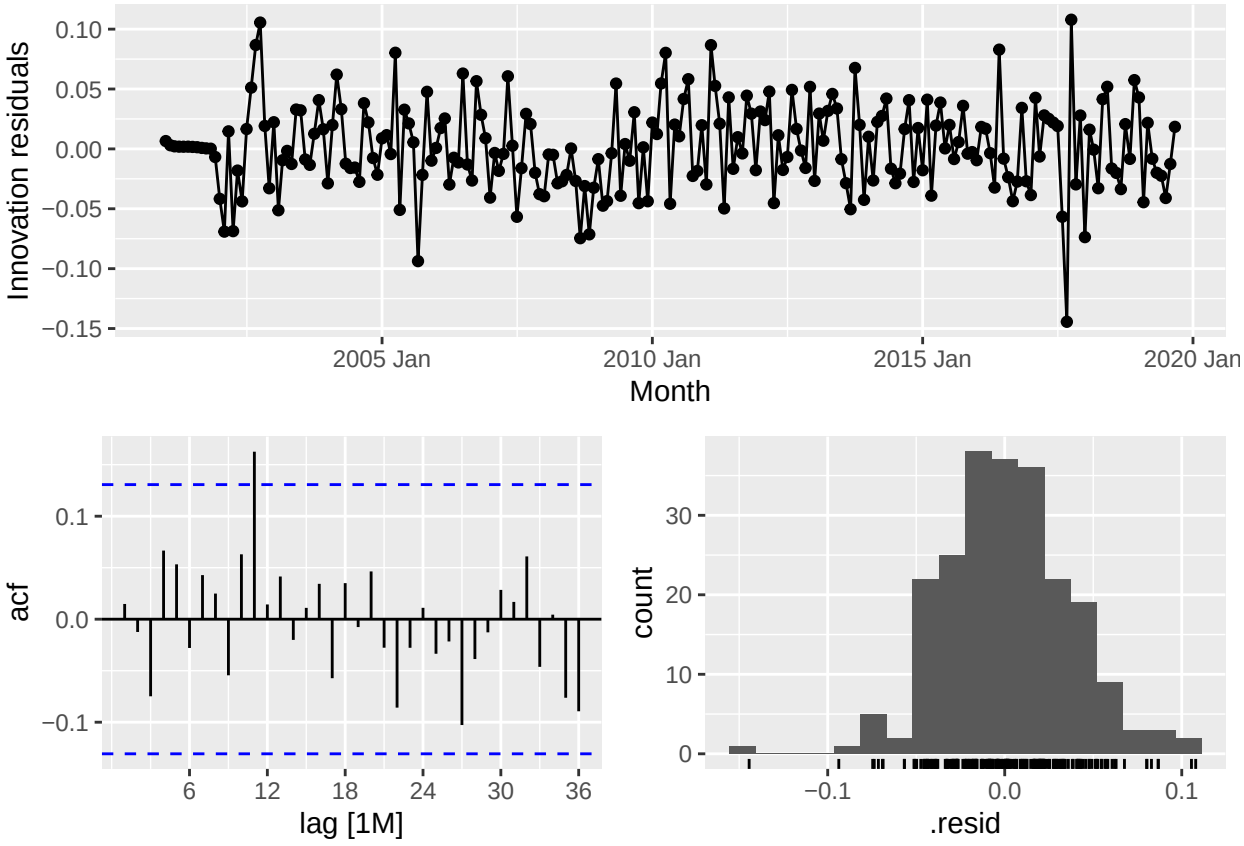
```
#> # A tibble: 3 × 6
#>   .model       sigma2 log_lik   AIC  AICc   BIC
#>   <chr>         <dbl>   <dbl> <dbl> <dbl> <dbl>
#> 1 auto        0.00142    395. -780. -780. -763.
#> 2 arima210011 0.00145    392. -776. -776. -763.
#> 3 arima012011 0.00146    391. -775. -775. -761.
```

The ARIMA() function uses unitroot_nsdiffs() to determine D (the number of seasonal differences to use), and unitroot_ndiffs() to determine d (the number of ordinary differences to use), when these are not specified. The selection of the other model parameters (p,q,P and Q) are all determined by minimizing the AICc, as with non-seasonal ARIMA models.

The three fitted models have similar AICc values, with the automatically selected model being a little better. Our second "guess" of ARIMA$(2,1,0)(0,1,1)_{12}$ turned out to be very close to the automatically selected model of ARIMA$(2,1,0)(1,1,1)_{12}$.

The residuals for the best model are shown in Figure 9.21.

```
fit |> select(auto) |> gg_tsresiduals(lag=36)
```



One small but significant spike (at lag 11) out of 36 is still consistent with white noise. To be sure, we use a Ljung-Box test, being careful to set the degrees of freedom to match the number of parameters in the model.

```
augment(fit) |>
  filter(.model == "auto") |>
  features(.innov, ljung_box, lag=24, dof=4)
```

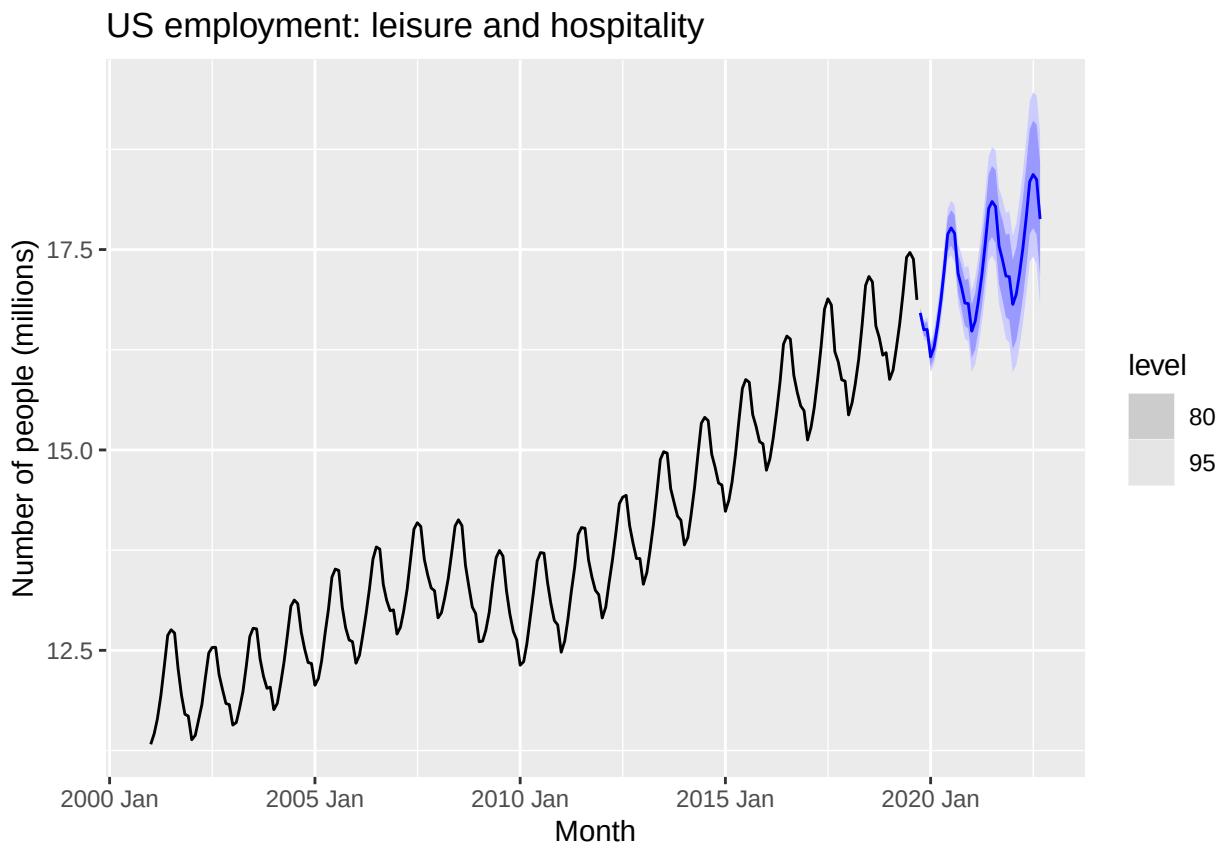
```
## # A tibble: 1 x 3
##   .model lb_stat lb_pvalue
##   <chr>    <dbl>     <dbl>
## 1 auto      16.6     0.680
```

```
#> # A tibble: 1 × 3
#>   .model lb_stat lb_pvalue
#>   <chr>    <dbl>     <dbl>
#> 1 auto      16.6     0.680
```

The large p-value confims that the residuals are similar to white noise.

Thus, we now have a seasonal ARIMA model that passes the required checks and is ready for forecasting. Forecasts from the model for the next three years are shown in Figure 9.22. The forecasts have captured the seasonal pattern very well, and the increasing trend extends the recent pattern. The trend in the forecasts is induced by the double differencing.

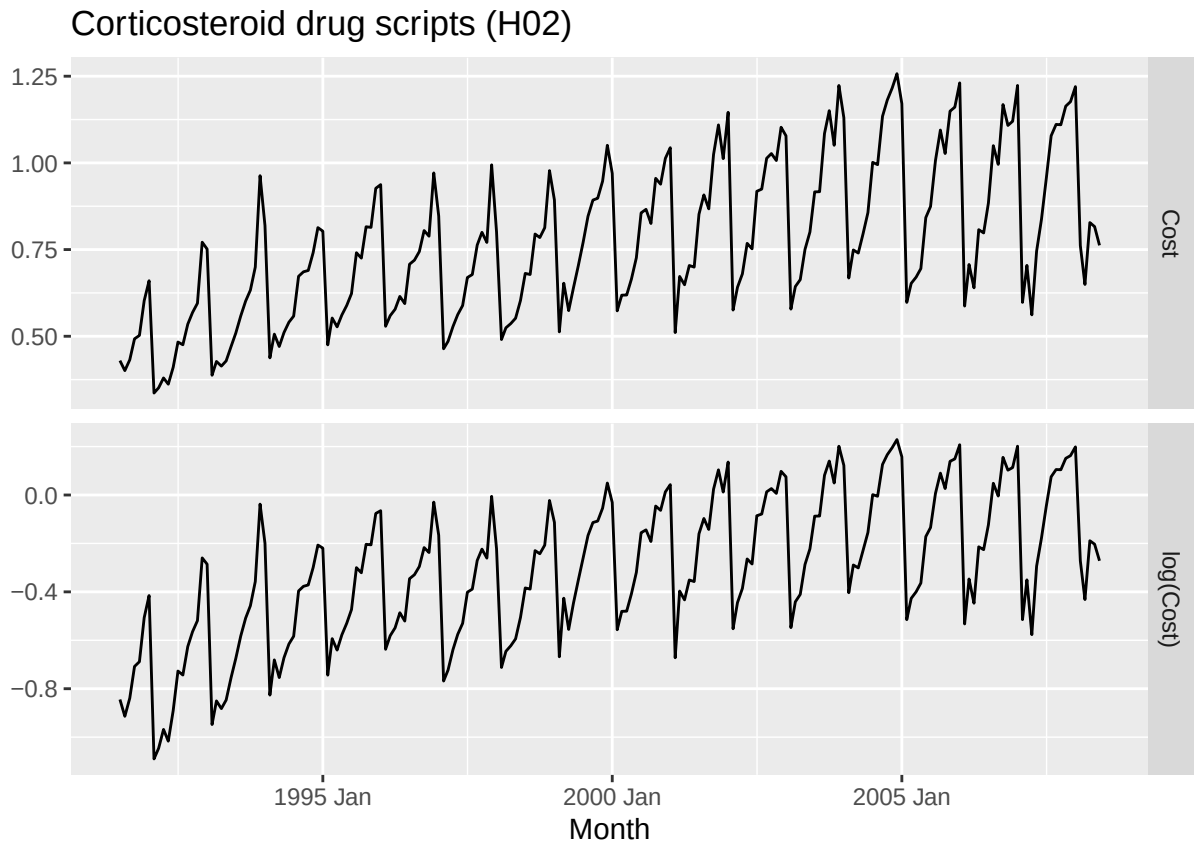
```
forecast(fit, h=36) |>
  filter(.model=='auto') |>
  autoplot(leisure) +
  labs(title = "US employment: leisure and hospitality",
       y="Number of people (millions)")
```



Example: Corticosteroid drug sales in Australia

For our second example, we will try to forecast monthly corticosteroid drug sales in Australia. These are known as H02 drugs under the Anatomical Therapeutic Chemical classification scheme.

```
h02 <- PBS |>
  filter(ATC2 == "H02") |>
  summarise(Cost = sum(Cost)/1e6)
h02 |>
  mutate(log(Cost)) |>
  pivot_longer(-Month) |>
  ggplot(aes(x = Month, y = value)) +
  geom_line() +
  facet_grid(name ~ ., scales = "free_y") +
  labs(y="", title="Corticosteroid drug scripts (H02)")
```



Data from July 1991 to June 2008 are plotted in Figure 9.23. There is a small increase in the variance with the level, so we take logarithms to stabilise the variance.

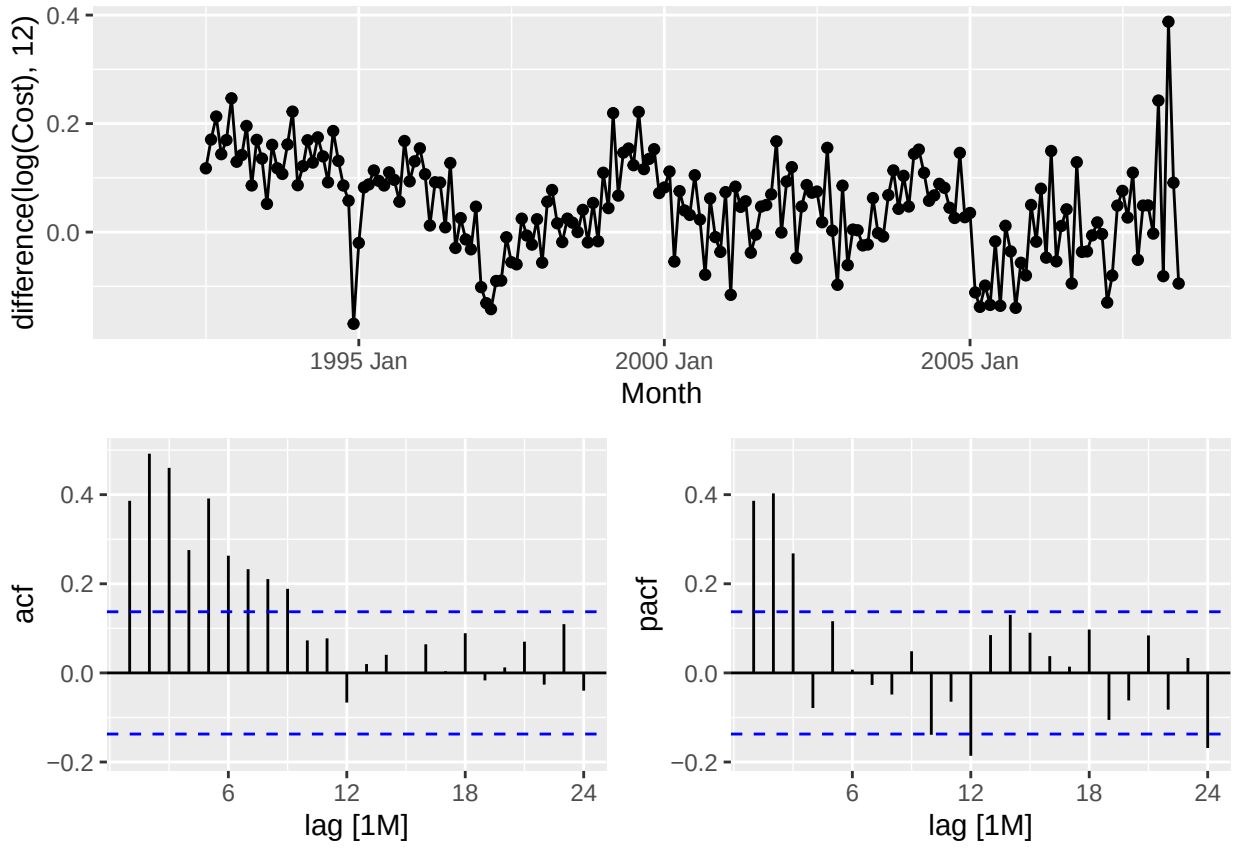
The data are strongly seasonal and obviously non-stationary, so seasonal differencing will be used. The seasonally differenced data are shown in Figure 9.24. It is not clear at this point whether we should do another difference or not. We decide not to, but the choice is not obvious.

The last few observations appear to be different (more variable) from the earlier data. This may be due to the fact that data are sometimes revised when earlier sales are reported late.

```
h02 |> gg_tsdisplay(difference(log(Cost), 12),
                    plot_type='partial', lag_max = 24)
```

```
## Warning: Removed 12 rows containing missing values (`geom_line()`).
```

```
## Warning: Removed 12 rows containing missing values (`geom_point()`).
```

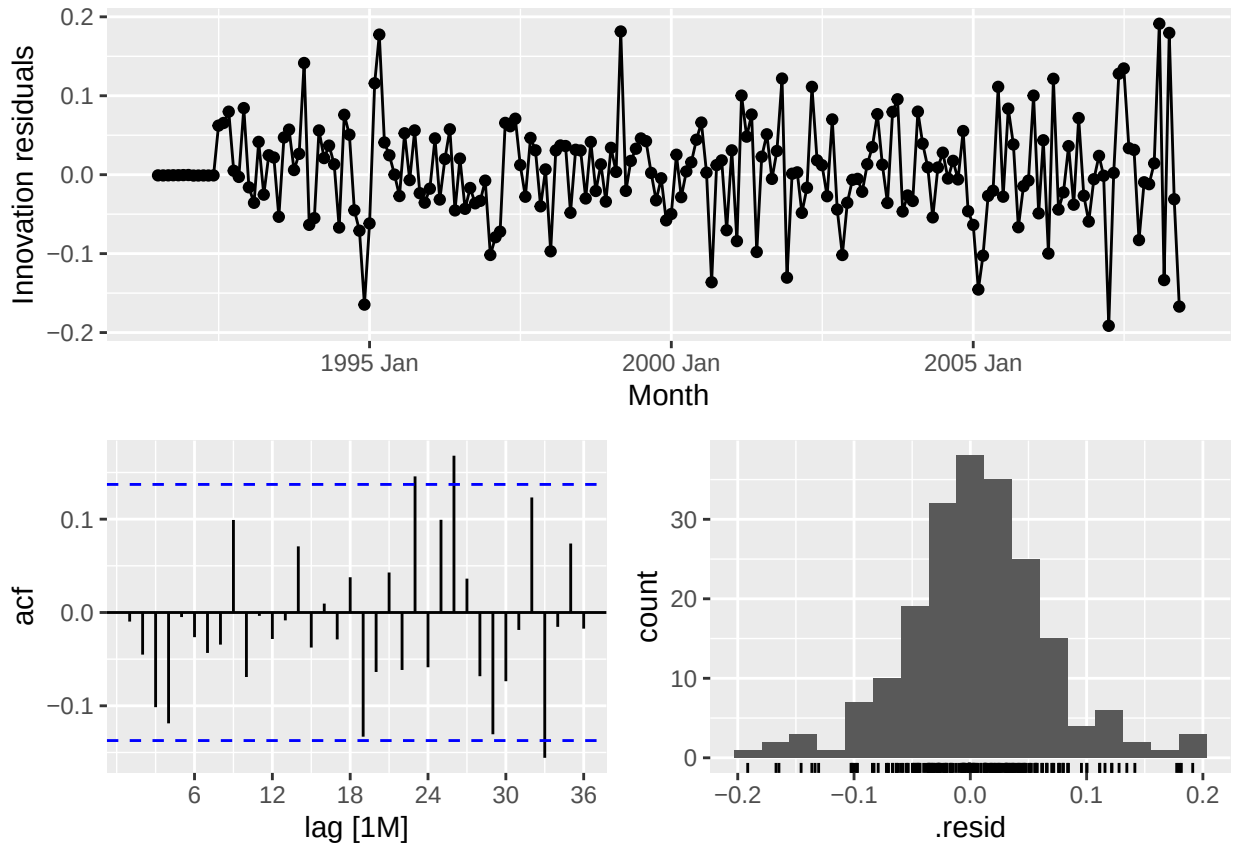


In the plots of the seasonally differenced data, there are spikes in the PACF at lags 12 and 24, but nothing at seasonal lags in the ACF. This may be suggestive of a seasonal AR(2) term. In the non-seasonal lags, there are three significant spikes in the PACF, suggesting a possible AR(3) term. The pattern in the ACF is not indicative of any simple model.

Consequently, this initial analysis suggests that a possible model for these data is an ARIMA$(3,0,0)(2,1,0)_{12}$. We fit this model, along with some variations on it, and compute the AICc values shown in Table 9.2.

Of these models, the best is the ARIMA$(3,0,1)(0,1,2)_{12}$ model (i.e., it has the smallest AICc value). The innovation residuals from this model are shown in Figure 9.25.

```
fit <- h02 |>
  model(ARIMA(log(Cost) ~ 0 + pdq(3,0,1) + PDQ(0,1,2)))
fit |> gg_tsresiduals(lag_max=36)
```




```
augment(fit) |>
  features(.innov, ljung_box, lag = 36, dof = 6)
```

```
## # A tibble: 1 x 3
##   .model                                                lb_stat lb_pvalue
##   <chr>                                                   <dbl>     <dbl>
## 1 ARIMA(log(Cost) ~ 0 + pdq(3, 0, 1) + PDQ(0, 1, 2))    50.7    0.0104
```

```
#> # A tibble: 1 × 3
#>   .model                                                lb_stat lb_pvalue
#>   <chr>                                                   <dbl>     <dbl>
#> 1 ARIMA(log(Cost) ~ 0 + pdq(3, 0, 1) + PDQ(0, 1, 2))    50.7    0.0104
```

There are a few significant spikes in the ACF, and the model fails the Ljung-Box test. The model can still be used for forecasting, but the prediction intervals may not be accurate due to the correlated residuals.

Next we will try using the automatic ARIMA algorithm. Running ARIMA() with all arguments left at their default values led to an ARIMA $(2,1,0)(0,1,1)_{12}$ model. Running ARIMA() with stepwise=FALSE and approximation=FALSE gives an ARIMA$(2,1,3)(0,1,1)_{12}$ model. However, both models still fail the Ljung-Box test for 36 lags. Sometimes it is just not possible to find a model that passes all of the tests.

Test set evaluation:

We will compare some of the models fitted so far using a test set consisting of the last two years of data. Thus, we fit the models using data from July 1991 to June 2006, and forecast the script sales for July 2006 – June 2008. The results are summarised in Table 9.3.

The models chosen manually are close to the best model over this test set based on the RMSE values, while

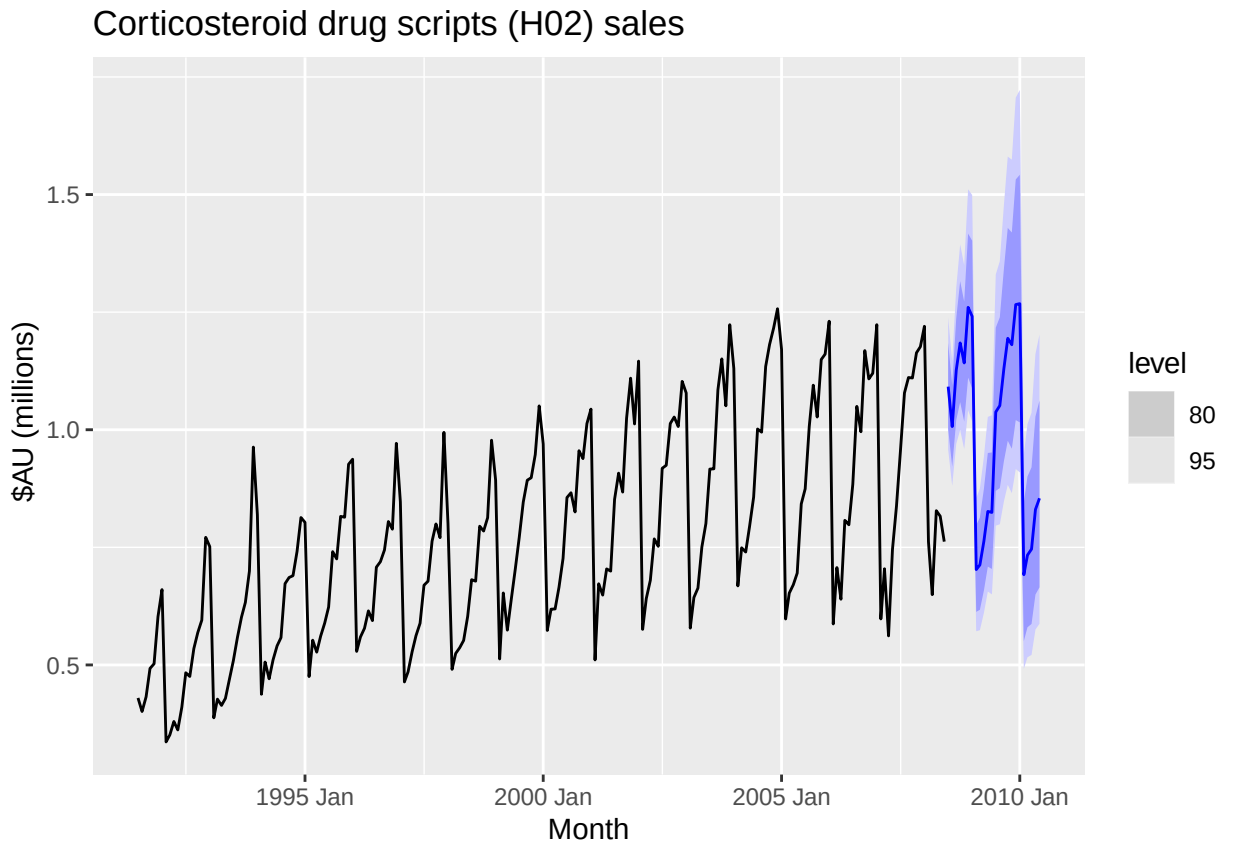
those models chosen automatically with ARIMA() are not far behind.

When models are compared using AICc values, it is important that all models have the same orders of differencing. However, when comparing models using a test set, it does not matter how the forecasts were produced — the comparisons are always valid. Consequently, in the table above, we can include some models with only seasonal differencing and some models with both first and seasonal differencing, while in the earlier table containing AICc values, we only compared models with seasonal differencing but no first differencing.

None of the models considered here pass all of the residual tests. In practice, we would normally use the best model we could find, even if it did not pass all of the tests.

Forecasts from the ARIMA$(3,0,1)(0,1,2)_{12}$ model (which has the second lowest RMSE value on the test set, and the best AICc value amongst models with only seasonal differencing) are shown in Figure 9.26.

```
h02 |>
  model(ARIMA(log(Cost) ~ 0 + pdq(3,0,1) + PDQ(0,1,2))) |>
  forecast() |>
  autoplot(h02) +
  labs(y=" $AU (millions)",
       title="Corticosteroid drug scripts (H02) sales")
```



9.10 ARIMA vs ETS

ARIMA vs ETS

- Myth that ARIMA models are more general than exponential smoothing. There are some ARIMA models that are equivalent to exponential smoothing models but there are models in both classes are

not available in the other class.

- Linear exponential smoothing models (that is the one with no multiplicative components and that's are 6 models with no multiplicative seasonality or no multiplicative trend) all special cases of ARIMA models. But any exponential smoothing model that has a multiplicative component whether it is in error or seasonal part, it does not have an ARIMA equivalent model. On the other hand there are lots of ARIMA models that have no exponential smoothing counterparts including all of the stationary models and all of the models where p and q are large numbers.
- Non-linear exponential smoothing models have no equivalent ARIMA counterparts.
- Many ARIMA models have no exponential smoothing counterparts.
- ETS models all non-stationary. Models with seasonality or non-damped trend (or both) have two unit-roots; all other models have one unit root.

Example: Australian Population

```
aus_economy <- global_economy  |>
  filter(Code == "AUS") |>
  mutate(Population=Population/1e6)
aus_economy |>
  slice(-n()) |>
  stretch_tsibble(.init = 10) |>
  model(ets=ETS(Population), arima= ARIMA(Population)) |>
  forecast(h=1) |>
  accuracy(aus_economy) |>
  select(.model,ME:RMSSE)
```

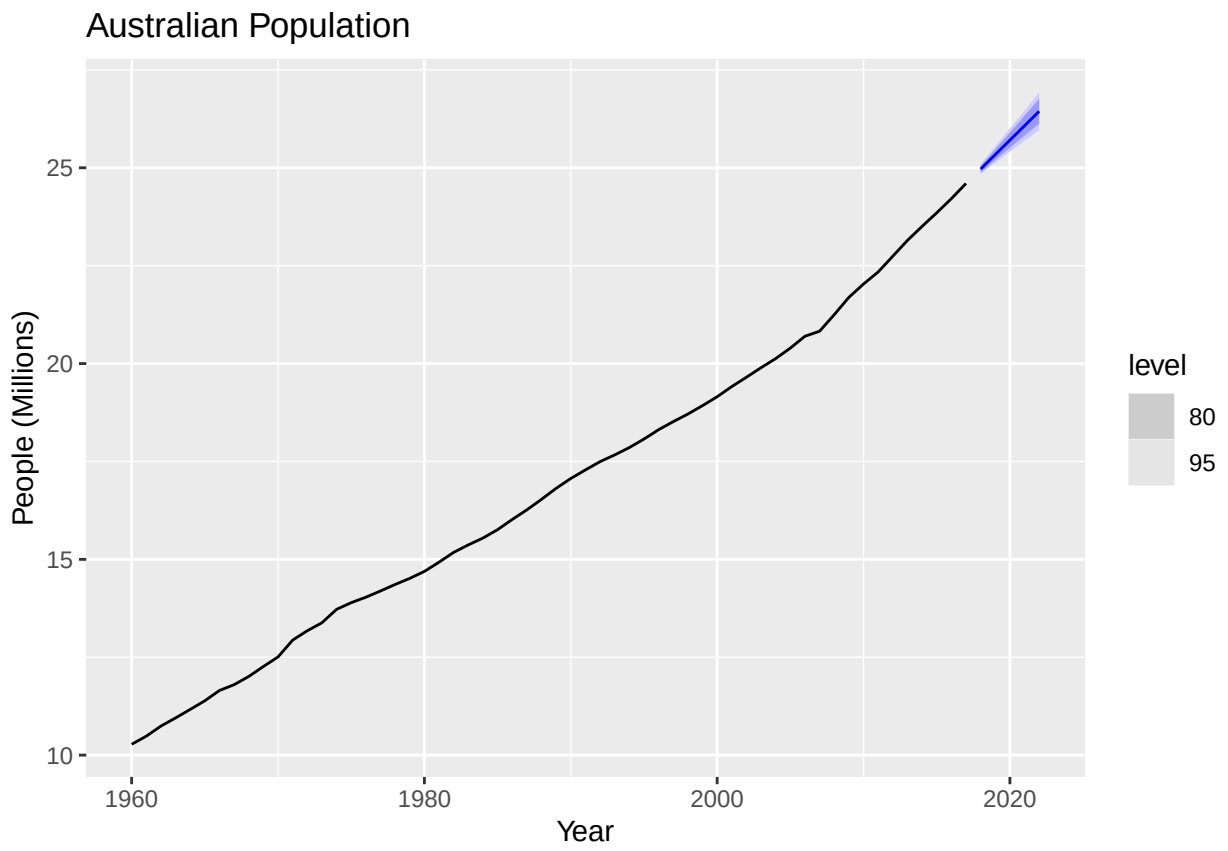
```
## # A tibble: 2 x 8
##   .model     ME   RMSE    MAE   MPE  MAPE  MASE RMSSE
##   <chr>   <dbl>  <dbl>  <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 arima  0.0420 0.194  0.0789 0.277 0.509 0.317 0.746
## 2 ets    0.0202 0.0774 0.0543 0.112 0.327 0.218 0.298
```

Here we have used a time series cross validation approach, so we take the data and stretch it with initial dataset of length 10, so we need at least 10 observations to form a model and of course then rest of the data gets added in the stretched_tsibble and for each model, we fit ets model automatically and arima model automatically. And we forecast them one step ahead. This is the only way we can compare ARIMA and ETS models because we can't compare on AIC because the likelihood of ETS model is calculated in different way from the ARIMA model so resulting AIC or AICc statistic will not be comparable.

Here, ETS model has done better than ARIMA model because RMSE is less for ETS model.

If we take the ETS model and then apply it on the whole dataset and forecast it. It will perform very well as.

```
aus_economy |>
  model(ETS(Population)) |>
  forecast(h = "5 years") |>
  autoplot(aus_economy) +
  labs(title = "Australian Population", y = "People (Millions)")
```



Example (Seasonal case): Australian Cement Production

```
#Cement <- aus_production |>
#  select(Cement) |>
#  filter_index("1998 Q1" ~ .)
#train <- Cement |> filter_index(. ~ "2007 Q4" )
#fit <- train |>
#  model(
#    arima = ARIMA(Cement),
#    ets= ETS(Cement)
#  )
```

```
#fit |>
#  select(arima) |>
#  report()
```

This is how ARIMA model looks like.

Now, comes to ETS model.

```
#fit |>
#  select(ets) |>
#  report()
```

Here gamma parameter is very low so seasonality is not changing much period to period.

Here, we can't compare the AICc statistics. The only way to choose between them is test set.

```
#gg_tsresiduals(fit |> select(arima), lag_max= 16)
```

It looks great and well behaved for arima model and for ets model, it also looks good as:

```
#gg_tsresiduals(fit |> select(ets), lag_max= 16)
```

Now, do lljung_box test.

```
#fit |>
#  select(arima) |>
#  augment() |>
#  features(.innov, ljung_box,lag=16,dof=6)
```

Here, p-value is good so no problem. Residuals are not significantly different from white noise.

```
#fit |>
#  select(ets) |>
#  augment() |>
#  features(.innov, ljung_box,lag=16)
```

Here, ets model also passes the test.

So, we can't tell the difference between these two models based on the in-sample analysis of the residuals.

If we compare them out of sample on two and half years of data and we see that arima model actually does slightly better here.

```
#fit |>
#  forecast(h="2 years 6 months") |>
#  accuracy(cement) |>
#  select(-ME,-MPE,-ACF1)
```

For ARIMA, rmse is less.

```
#fit |>
#  select(arima)  |>
#  forecast(h= "3 years") |>
#  autoplot(cement) +
#  labs(title = "Cement Production in Australia", y = "Tonnes ('000)")
```

It's actually not a very good model because actual values lies at edge of the prediction interval which suggests that prediction interbals are too narrow here.

Summary

It is a commonly held myth that ARIMA models are more general than exponential smoothing. While linear exponential smoothing models are all special cases of ARIMA models, the non-linear exponential smoothing models have no equivalent ARIMA counterparts. On the other hand, there are also many ARIMA models that have no exponential smoothing counterparts. In particular, all ETS models are non-stationary, while some ARIMA models are stationary. Figure 9.27 shows the overlap between the two model classes.

Check Screenshot – 214

The ETS models with seasonality or non-damped trend or both have two unit roots (i.e., they need two levels of differencing to make them stationary). All other ETS models have one unit root (they need one level of differencing to make them stationary).

Table 9.4 gives the equivalence relationships for the two classes of models. For the seasonal models, the ARIMA parameters have a large number of restrictions.

Check screenshot – 215

The AICc is useful for selecting between models in the same class. For example, we can use it to select an ARIMA model between candidate ARIMA models20 or an ETS model between candidate ETS models. However, it cannot be used to compare between ETS and ARIMA models because they are in different model classes, and the likelihood is computed in different ways. The examples below demonstrate selecting between these classes of models.

Comparing ARIMA() and ETS() on non-seasonal data

We can use time series cross-validation to compare ARIMA and ETS models. Let's consider the Australian population from the global_economy dataset, as introduced in Section 8.2.

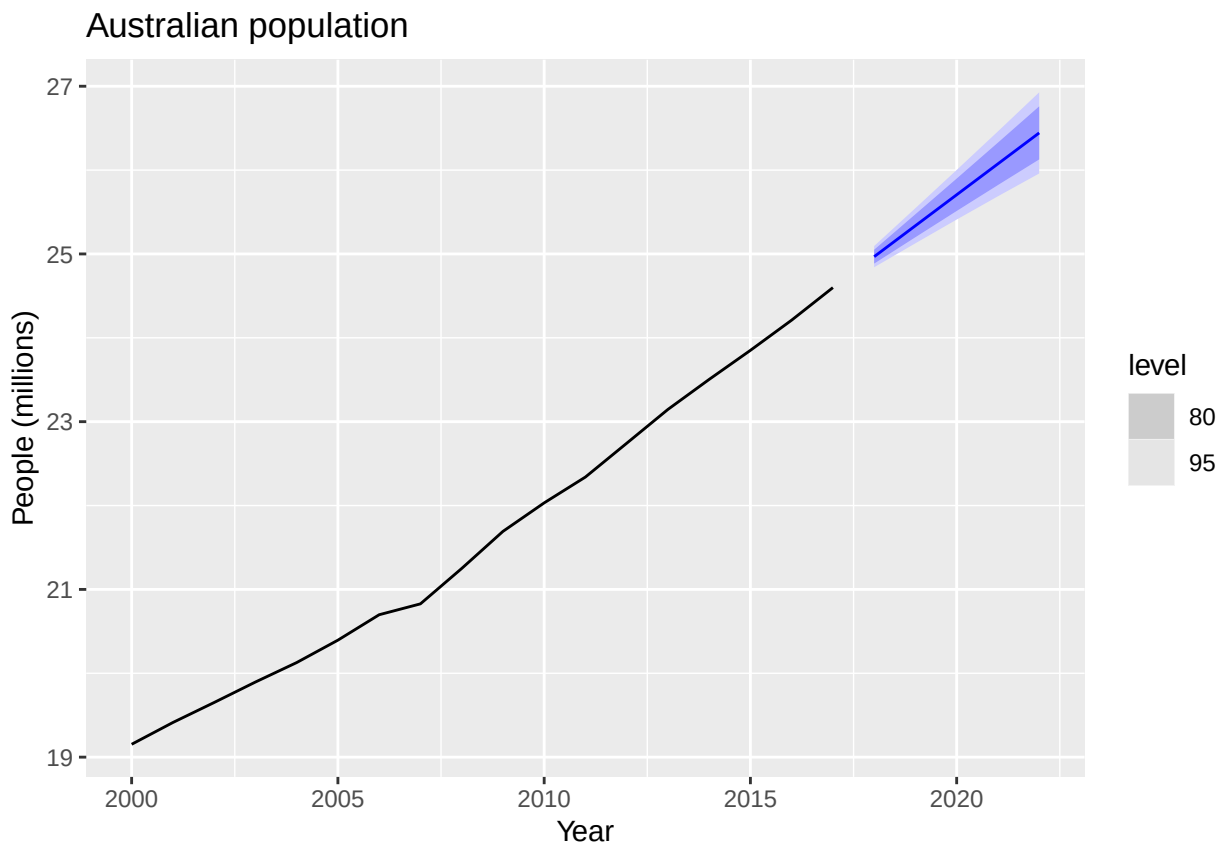
```
aus_economy <- global_economy |>
  filter(Code == "AUS") |>
  mutate(Population = Population/1e6)

aus_economy |>
  slice(-n()) |>
  stretch_tsibble(.init = 10) |>
  model(
    ETS(Population),
    ARIMA(Population)
  ) |>
  forecast(h = 1) |>
  accuracy(aus_economy) |>
  select(.model, RMSE:MAPE)
```

```
## # A tibble: 2 x 5
##   .model              RMSE    MAE   MPE  MAPE
##   <chr>              <dbl>  <dbl> <dbl> <dbl>
## 1 ARIMA(Population) 0.194  0.0789 0.277 0.509
## 2 ETS(Population)   0.0774 0.0543 0.112 0.327
```

```
#> # A tibble: 2 × 5
#>   .model              RMSE    MAE   MPE  MAPE
#>   <chr>              <dbl>  <dbl> <dbl> <dbl>
#> 1 ARIMA(Population) 0.194  0.0789 0.277 0.509
#> 2 ETS(Population)   0.0774 0.0543 0.112 0.327
```

In this case the ETS model has higher accuracy on the cross-validated performance measures. Below we generate and plot forecasts for the next 5 years generated from an ETS model.

```
aus_economy |>
  model(ETS(Population)) |>
  forecast(h = "5 years") |>
  autoplot(aus_economy |> filter(Year >= 2000)) +
  labs(title = "Australian population",
       y = "People (millions)")
```



Comparing ARIMA() and ETS() on seasonal data

In this case we want to compare seasonal ARIMA and ETS models applied to the quarterly cement production data (from aus_production). Because the series is relatively long, we can afford to use a training and a test set rather than time series cross-validation. The advantage is that this is much faster. We create a training set from the beginning of 1988 to the end of 2007 and select an ARIMA and an ETS model using the ARIMA() and ETS() functions.

```
cement <- aus_production |>
  select(Cement) |>
  filter_index("1988 Q1" ~ .)
train <- cement |> filter_index(. ~ "2007 Q4")
```

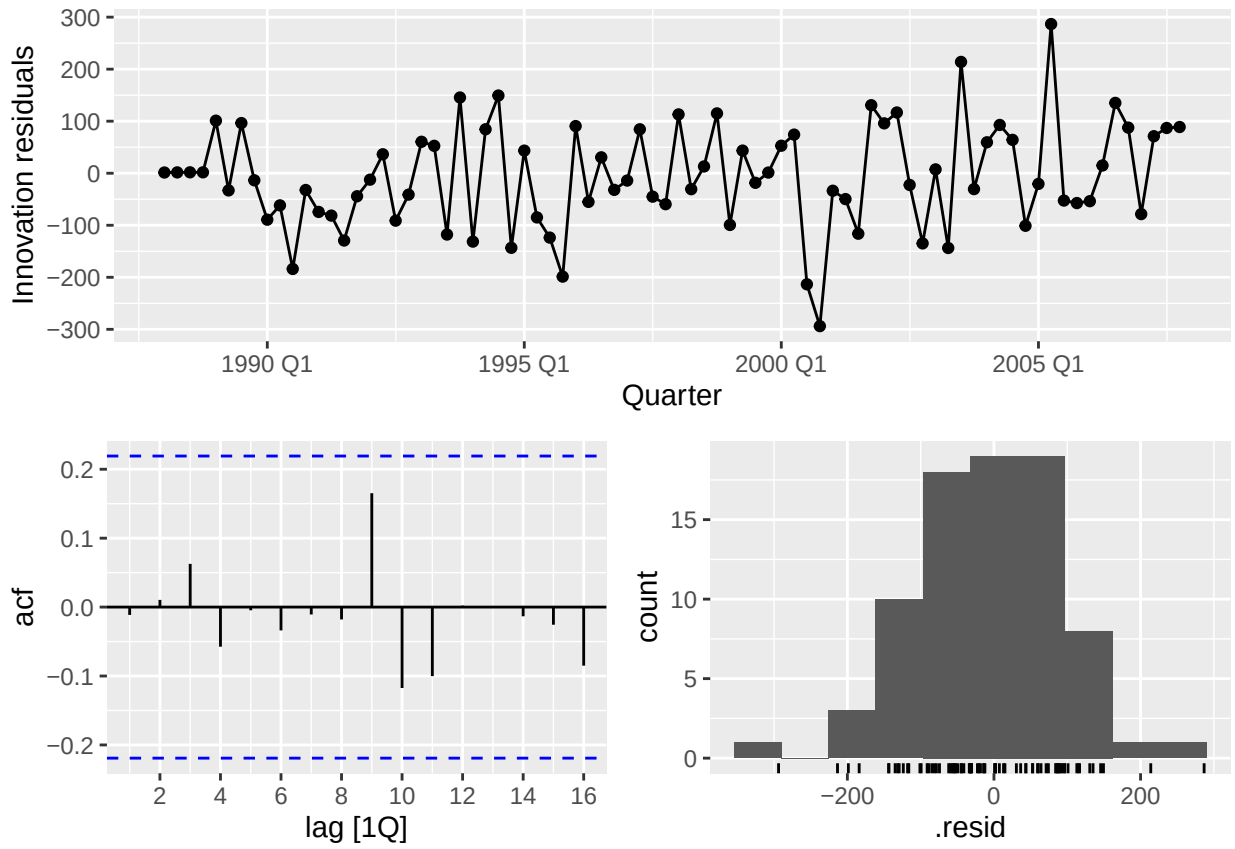
The output below shows the model selected and estimated by ARIMA(). The ARIMA model does well in capturing all the dynamics in the data as the residuals seem to be white noise.

```
fit_arima <- train |> model(ARIMA(Cement))
report(fit_arima)
```

```
## Series: Cement
## Model: ARIMA(1,0,1)(2,1,1)[4] w/ drift
##
## Coefficients:
##          ar1      ma1   sar1     sar2     sma1  constant
##       0.8886  -0.2366  0.081  -0.2345  -0.8979    5.3884
## s.e.  0.0842   0.1334  0.157   0.1392   0.1780    1.4844
##
```

```
## sigma^2 estimated as 11456:  log likelihood=-463.52
## AIC=941.03   AICc=942.68   BIC=957.35
```

```
#> Series: Cement
#> Model: ARIMA(1,0,1)(2,1,1)[4] w/ drift
#>
#> Coefficients:
#>          ar1      ma1   sar1     sar2     sma1  constant
#>       0.8886  -0.2366  0.081  -0.2345  -0.8979     5.388
#> s.e.  0.0842   0.1334  0.157   0.1392   0.1780     1.484
#>
#> sigma^2 estimated as 11456:  log likelihood=-463.5
#> AIC=941   AICc=942.7   BIC=957.4
fit_arima |> gg_tsresiduals(lag_max = 16)
```




```
augment(fit_arima) |>
  features(.innov, ljung_box, lag = 16, dof = 5)
```

```
## # A tibble: 1 x 3
##   .model        lb_stat lb_pvalue
##   <chr>           <dbl>     <dbl>
## 1 ARIMA(Cement)    6.37     0.847
```

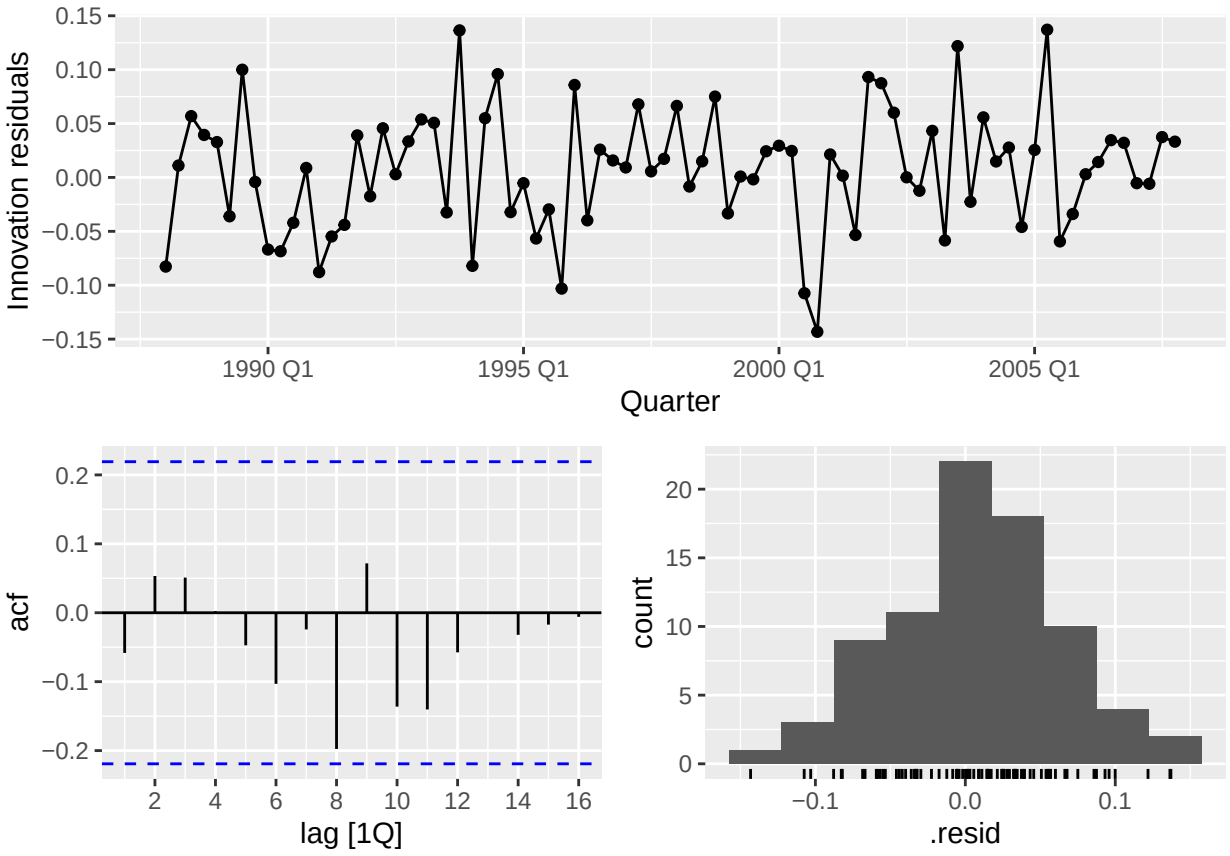
```
#> # A tibble: 1 × 3
#>   .model        lb_stat lb_pvalue
#>   <chr>           <dbl>     <dbl>
#> 1 ARIMA(Cement)    6.37     0.847
```

The output below also shows the ETS model selected and estimated by ETS(). This model also does well in capturing all the dynamics in the data, as the residuals similarly appear to be white noise.

```
fit_ets <- train |> model(ETS(Cement))
report(fit_ets)
```

```
## Series: Cement
## Model: ETS(M,N,M)
##   Smoothing parameters:
##     alpha = 0.7533714
##     gamma = 0.0001000093
##
##   Initial states:
##      l[0]     s[0]    s[-1]    s[-2]     s[-3]
##  1694.712 1.031179 1.045209 1.011424 0.9121874
##
##   sigma^2:  0.0034
##
##      AIC     AICc      BIC
## 1104.095 1105.650 1120.769
```

```
#> Series: Cement
#> Model: ETS(M,N,M)
#>   Smoothing parameters:
#>     alpha = 0.7534
#>     gamma = 1e-04
#>
#>   Initial states:
#>  l[0]  s[0] s[-1] s[-2]  s[-3]
#>  1695 1.031 1.045 1.011 0.9122
#>
#>   sigma^2:  0.0034
#>
#>  AIC AICc  BIC
#> 1104 1106 1121
fit_ets |>
  gg_tsresiduals(lag_max = 16)
```




```
augment(fit_ets) |>
  features(.innov, ljung_box, lag = 16)
```

```
## # A tibble: 1 x 3
##   .model      lb_stat lb_pvalue
##   <chr>         <dbl>     <dbl>
## 1 ETS(Cement)    10.0     0.865
```

```
#> # A tibble: 1 × 3
#>   .model      lb_stat lb_pvalue
#>   <chr>         <dbl>     <dbl>
#> 1 ETS(Cement)    10.0     0.865
```

The output below evaluates the forecasting performance of the two competing models over the test set. In this case the ARIMA model seems to be the slightly more accurate model based on the test set RMSE, MAPE and MASE.

```
# Generate forecasts and compare accuracy over the test set
bind_rows(
    fit_arima |> accuracy(),
    fit_ets |> accuracy(),
    fit_arima |> forecast(h = 10) |> accuracy(cement),
    fit_ets |> forecast(h = 10) |> accuracy(cement)
  ) |>
  select(-ME, -MPE, -ACF1)
```

```
## # A tibble: 4 x 7
##   .model        .type     RMSE   MAE  MAPE  MASE RMSSE
```

```
##   <chr>          <chr>    <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ARIMA(Cement) Training  100.  79.9  4.37 0.546 0.582
## 2 ETS(Cement)   Training  103.  80.0  4.41 0.547 0.596
## 3 ARIMA(Cement) Test      216. 186.   8.68 1.27  1.26
## 4 ETS(Cement)   Test      222. 191.   8.85 1.30  1.29
```

```
#> # A tibble: 4 × 7
#>   .model         .type     RMSE   MAE  MAPE  MASE RMSSE
#>   <chr>          <chr>    <dbl> <dbl> <dbl> <dbl> <dbl>
#> 1 ARIMA(Cement) Training  100.  79.9  4.37 0.546 0.582
#> 2 ETS(Cement)   Training  103.  80.0  4.41 0.547 0.596
#> 3 ARIMA(Cement) Test      216. 186.   8.68 1.27  1.26
#> 4 ETS(Cement)   Test      222. 191.   8.85 1.30  1.29
```

Below we generate and plot forecasts from the ARIMA model for the next 3 years.

```
cement |>
  model(ARIMA(Cement)) |>
  forecast(h="3 years") |>
  autoplot(cement) +
  labs(title = "Cement production in Australia",
       y = "Tonnes ('000)")
```