ARIMA Models

Lesson 3.2

Introduction

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.

Stationarity

A stationary time series is one whose properties do not depend on the time at which the series is observed. 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.

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

Which of the following plots do you think are stationary?

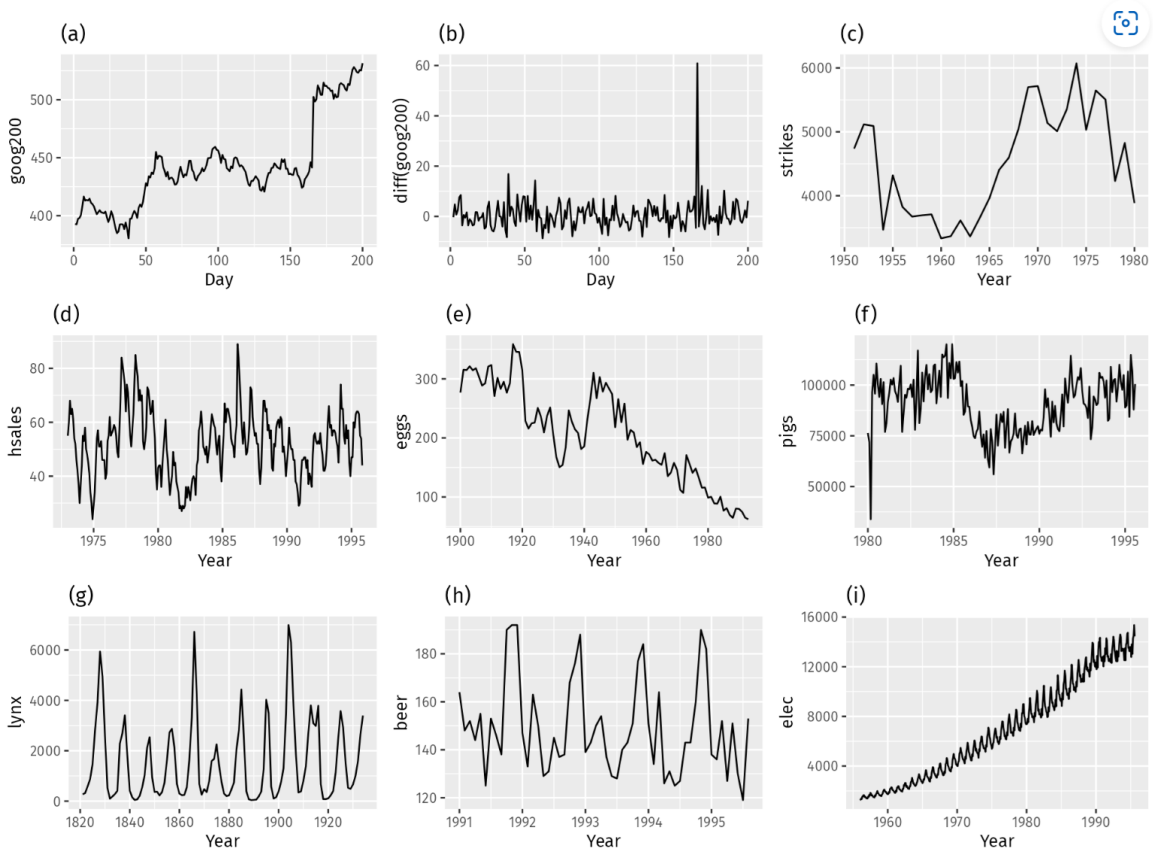


Figure 1: Which of these series are stationary?

Observe 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 stabilize the variance of a time series. Differencing can help stabilize the mean of a time series by removing changes in the level of a time series, and therefore eliminating (or reducing) trend and seasonality.

Another way to determine stationarity of a time series is to generate the ACF plot. 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.

```
library(fpp2)
library(tidyverse)
library(patchwork)
```

```
g1 <- ggAcf(goog200)
g2 <- ggAcf(diff(goog200))
g1 + g2
```

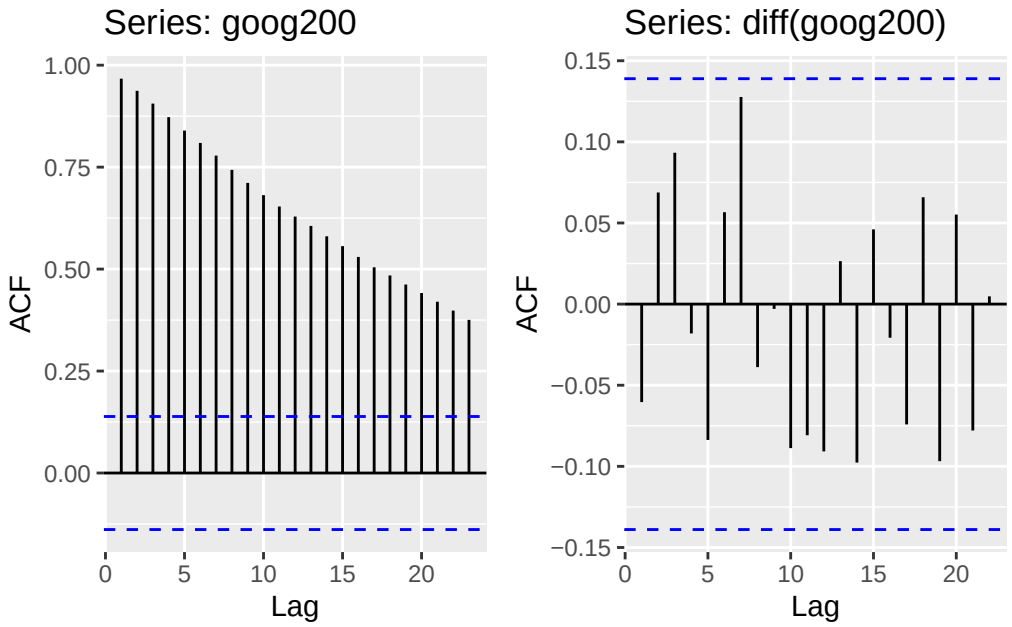


Figure 2: The ACF plots of the original Google stock price and the differnced series

The ACF of the differenced Google stock price looks just like that of a white noise series. There are no autocorrelations lying outside the 95% limits. To formally test if there is no more significant autocorrelations, we use the Ljung-Box test.

```
Box.test(diff(goog200), lag=10, type="Ljung-Box")
```

```
    Box-Ljung test

data:  diff(goog200)
X-squared = 11.031, df = 10, p-value = 0.3551
```

The above result 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

The differenced series which is the change between consecutive observations in the original series is written as

$$y_t^{'} = y_t - y_{t-1},\ t = 1, 2, \cdots, T$$

The differenced series will have only $T-1$ values, since it is not possible to calculate a difference $y_t^{'}$ for the first observation.

When the differenced series is white noise, the model for the original series can be written as

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

where ϵ_t denotes white noise. Rearranging this leads to the **random wal** model

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

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.

Second-order differencing

Occasionally the differenced data will not appear to be stationary and it may be necessary to difference the data a second time to obtain a stationary series:

$$\begin{aligned} y_t^{"} &= y_t^{'} - y_{t-1}^{'} \\ &= [y_t - y_{t-1}] - [y_{t-1} - y_{t-2}] \\ &= y_t - 2y_{t-1} + y_{t-2} \end{aligned}$$

In this case, $y_t^{"}$ will have $T-2$ values. Then, we would model the "change in the changes" of the original data. 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 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

$$y_t = y_{t-m} + \epsilon_t$$

Forecasts from this model are equal to the last observation from the relevant season.

The following code chunk illustrates the use transformation and differencing to achieve stationarity. The transformation and differencing have made the series look relatively stationary.

```
cbind("Sales ($million)" = a10,
      "Monthly log sales" = log(a10),
      "Annual change in log sales" = diff(log(a10),12)) %>%
  autoplot(facets=TRUE) +
    xlab("Year") + ylab("") +
    ggtitle("Antidiabetic drug sales")
```

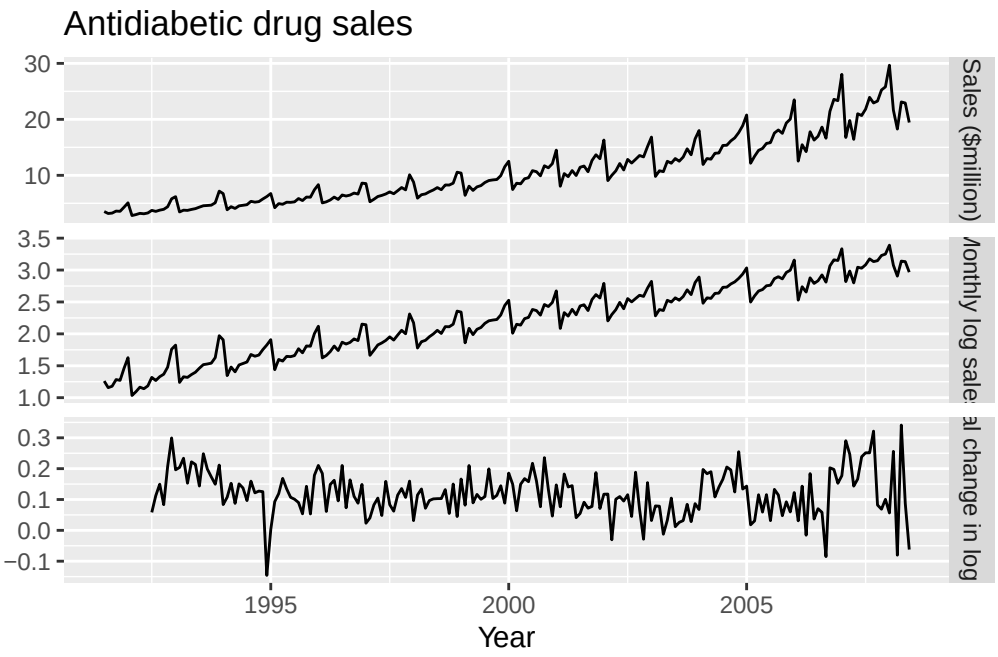


Figure 3: Logs and seasonal differences of A10 (antidiabetic) sales data

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. The following code chunk illustrates this idea.

```
cbind("Billion kWh" = usmelec,
      "Logs" = log(usmelec),
      "Seasonally\n differenced logs" =
        diff(log(usmelec),12),
      "Doubly\n differenced logs" =
        diff(diff(log(usmelec),12),1)) %>%
  autoplot(facets=TRUE) +
    xlab("Year") + ylab("") +
    ggtitle("Monthly US net electricity generation")
```

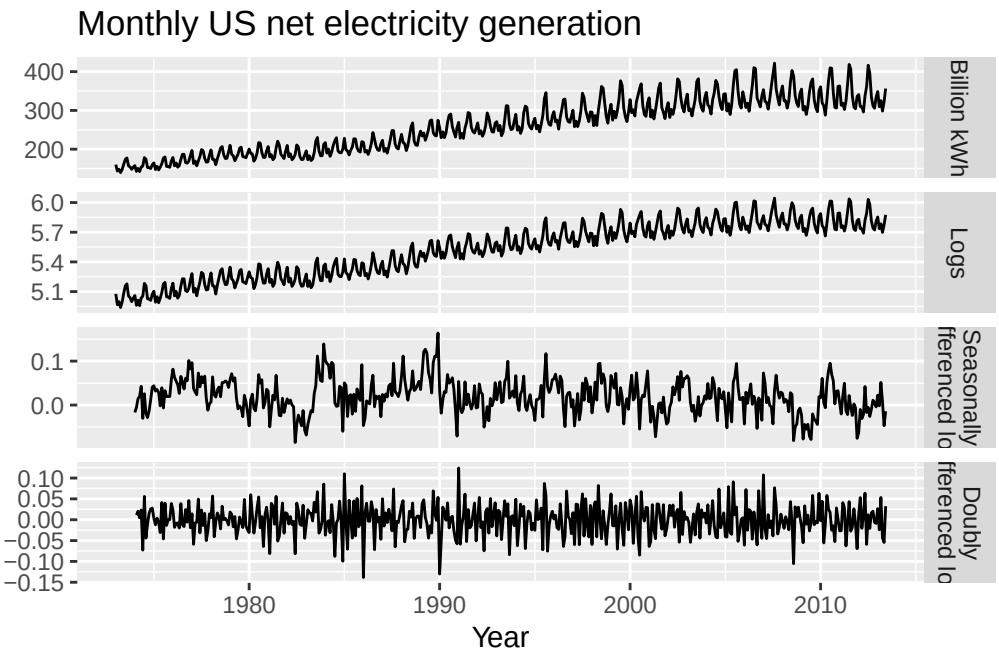


Figure 4: US net electricity generation (billion kWh)

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).

There is a degree of subjectivity in selecting which differences to apply. The seasonally differenced data in the Figure3 do not show substantially different behaviour from the seasonally differenced data in Figure 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.

If $y_t^{'} = y_t - y_{t-m}$ denotes the seasonally differenced series, then the twice-differenced series is

$$\begin{aligned} y_t^{"} &= y_t^{'} - y_{t-1}^{'} \\ &= [y_t - y_{t-m}] - [y_{t-1} - y_{t-m-1}] \\ &= y_t - y_{t-1} - y_{t-m} + y_{t-m-1} \end{aligned}$$

When both seasonal and first differences are applied, it makes no difference which is 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.

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

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, Phillips, Schmidt, & Shin, 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 *ur.kpss()* function from the **urca** package.

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

```
library(urca)
goog %>%
  ur.kpss() %>%
  summary()
```

```
######################
# KPSS Unit Root Test #
######################

Test is of type: mu with 7 lags.

Value of test-statistic is: 10.7223

Critical value for a significance level of:
                10pct  5pct 2.5pct  1pct
critical values 0.347 0.463  0.574 0.739
```

The test statistic is much bigger than the 1% critical value, indicating that the null hypothesis is rejected. That is, the data are not stationary. We can difference the data, and apply the test again.

```
goog %>%
  diff() %>%
  ur.kpss() %>%
  summary()
```

```
#######################
# KPSS Unit Root Test #
#######################

Test is of type: mu with 7 lags.

Value of test-statistic is: 0.0324

Critical value for a significance level of:
                10pct  5pct 2.5pct  1pct
critical values 0.347 0.463  0.574 0.739
```

This time, the test statistic is tiny, and well within the range we would expect for stationary data. So we can conclude that the differenced data are stationary.

This process of using a sequence of KPSS tests to determine the appropriate number of first differences is carried out by the function *ndiffs()*.

```
ndiffs(goog)
```

```
[1] 1
```

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

A similar function for determining whether seasonal differencing is required is *nsdiffs()* 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 *nsdiffs()* to the logged US monthly electricity data.

```
usmelec %>%
  log() %>%
  nsdiffs()
```

```
[1] 1
```

```
usmelec %>%
  log() %>%
  diff(lag=12) %>%
  ndiffs()
```

```
[1] 1
```

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

Backshift operator

The backward shift operator B is a useful notational device when working with time series lags:

$$By_t = y\_t - 1$$

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

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

For monthly data, if we wish to consider "the same month last year," the notation is

$$B^{12} y_t = y_{t-12}$$

The backward shift operator is convenient for describing the process of differencing. A first difference can be written as

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

Note that a first difference is represented by $(1-B)$. Similarly, if second-order differences have to be computed, then:

$$y_t^{"} = y_t - 2y_{t-1} + y_{t-2} = (1 - 2B + B^2)y_t = (1 - B)^2 y_t$$

In general, a dth-order difference can be written as

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

Backshift notation is particularly useful when combining differences, as the operator can be treated using ordinary algebraic rules. In particular, terms involving
B can be multiplied together.

For example, a seasonal difference followed by a first difference can be written as

$$\begin{aligned}(1 - B)(1 - B^m)y_t &= (1 - B - B^m + B^{m+1})y_t \\ &= y_t - y_{t-1} - y_{t-m} + y_{t-m+1}\end{aligned}$$

the same result obtained earlier.

Autoregressive models

In a multiple regression model, 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.

Thus, an autoregressive model of order p can be written as

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

where ϵ_t is white noise. 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 series below show series from an $AR(1)$ model and an $AR(2)$ model. Changing the parameters $\phi_1, \phi_2, \cdots, \phi_p$ 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.

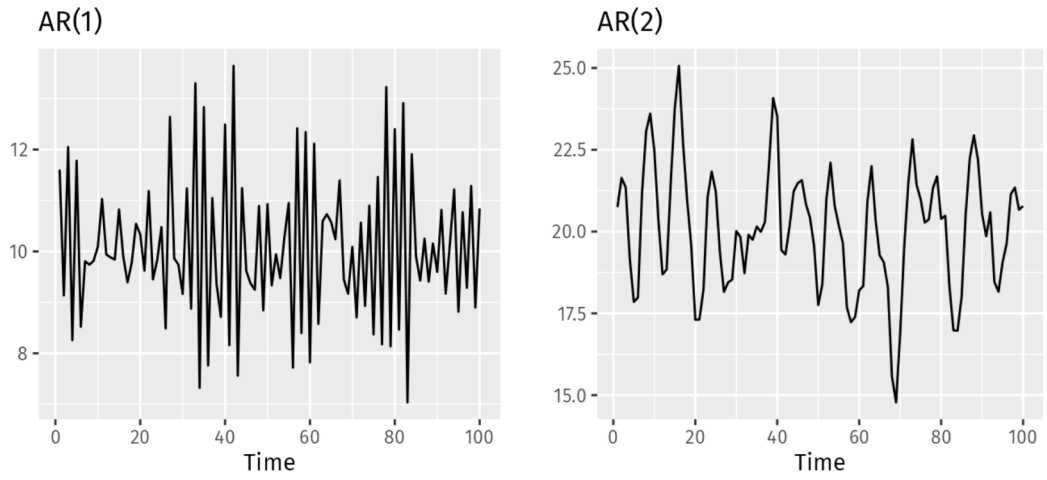


Figure 5: Series from AR(1) and AR(2) models

In the above figures, the $AR(1)$ model is $y_t = 18 - 0.8y_{t-1} + \epsilon_t$ and the $AR(2)$ model is $y_t = 8 + 1.3y_{t-1} - 0.7y_{t-2} + \epsilon_t$ with $\epsilon_t \sim N(), 1)$.

For an $AR(1)$ model:

- when $\phi_1 = 0$, y_t is equivalent to white noise
- when $\phi_1 = 1$ and $c = 0$, then y_t is equivalent to a rabdom walk
- when $\phi_1 = 1$ and $c \neq 0$, then y_t is equivalent to a random walk with a drift
- when $\phi_1 < 0$, y_t tends to oscillate aroud the mean

We normally restrict autoregressive models to stationary data, in which case some constraints on the values of the parameters are required.

- For an $AR(1)$ model: $-1 < \phi_1 < 1$
- For an $AR(2)$ model: $-1 < \phi_2 < 1, \phi_1 + \phi_2 < 1, \phi_2 - \phi_1 < 1$

When $p \geq 3$, the restrictions are much more complicated. R takes care of these restrictions when estimating a model.

Moving average models

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.

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

where ϵ_t is white noise. 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. However, moving average models should not be confused with the moving average smoothing discussed in the previous lessons. A moving average model is used for forecasting future values, while moving average smoothing is used for estimating the trend-cycle of past values.

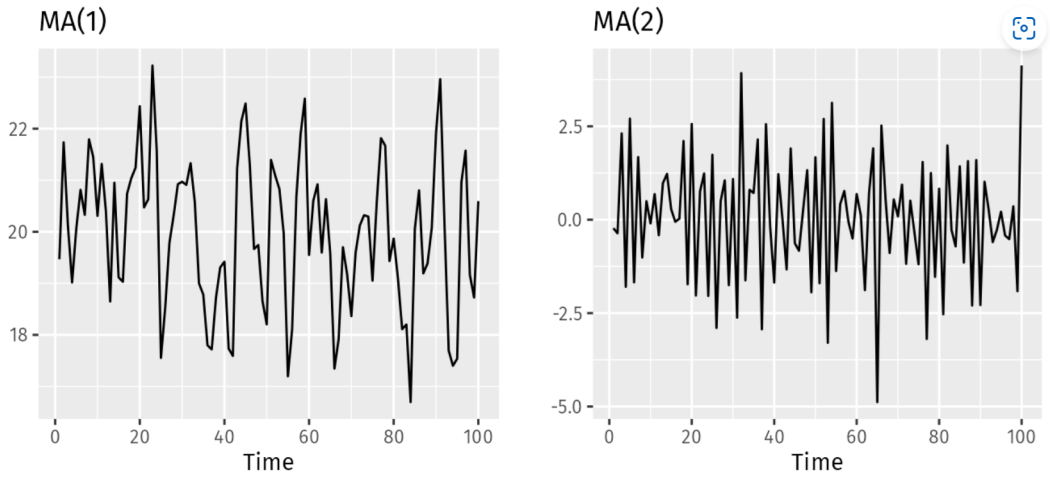


Figure 6: Series from MA(1) and MA(2) models

The equation of the $MA(1)$ model is $y_t = 20 + \epsilon_t + 0.8\epsilon_{t-1}$ and the $MA(2)$ model is $y_t = \epsilon_t - \epsilon_{t-1} + 0.8\epsilon_{t-2}$. In both cases, $\epsilon_t \sim N(0, 1)$.

Figure 6 shows some data from an MA(1) model and an MA(2) model. Changing the parameters $\theta_1, \theta_2, \cdots, \theta_q$ 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 $MA(\infty)$ model. For example, using repeated substitution, we can demonstrate this for an AR(1) model:

$$\begin{aligned}
y_t &= \phi_1 y_{t-1} + \epsilon_t \\
&= \phi_1(\phi_1 y_{t-2} + \epsilon_{t-1}) + \epsilon_t \\
&= \phi_1^2 y_{t-2} + \phi_1 \epsilon_{t-1} + \epsilon_t \\
&= \phi_1^3 y_{t-3} + \phi_1^2 \epsilon_{t-2} + \phi_1 \epsilon_{t-1} + \epsilon_t \\
&\text{etc.}
\end{aligned}$$

Provided $-1 < \phi_1 < 1$, the value ϕ_1^k will will get smaller as k gets larger. So eventually we obtain

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

an $MA(\infty)$ model.

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 $AR(\infty)$ 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, $y_t = \epsilon_t + \theta_1 \epsilon_{t-1}$. Its $AR(\infty)$ representation, the most recent error can be written as a linear function of current and past observations:

$$\epsilon_t = \sum_{j=0}^{\infty} (-\theta)^j y_{t-j}$$

where $|\theta| > 1$, the weights increase as lags increase, so the more distant the observations the greater their influence on the current error. When $|\theta| = 1$, 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 $|\theta| < 1$, so the most recent observations have higher weight than observations from the more distant past. Thus, the process is invertible when $|\theta| < 1$.

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: $-1 < \theta_2 < 1, \theta_2 + \theta_1 > -1, \theta_1 - \theta_2 < 1$.

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

ARIMA Models

If we combine differencing with autoregression and moving average model, we obtain the ARIMA (non-seasonal) model. ARIMA stands fro Autoregressive Integrated Moving Average. The full model can be written as

$$y_t^{'} = c + \phi_1 y_{t-1}^{'} + \cdots + \phi_p y_{t-p}^{'} + \theta_1 \epsilon_{t-1} + \cdots + \theta_q \epsilon_{t-q} + \epsilon_t$$

where $y_t{}'$ is the differenced series. The predictors on the right hand side include both lagged values of y_t and the lagged errors. We call this an $ARIMA(p, d, q)$, where

p = order of the autoregressive part,

d = degree of differencing involved

q = order of the moving average part

The same stationarity and invertibility conditions that are used for autoregressive and moving average models also apply to an ARIMA model.

Many of the models that we have already discussed are special cases of teh ARIMA model, as shown below

Model	ARIMA(p,d,q)
White noise	ARIMA(0,0,0) with no constant
Random walk	ARIMA(0,1,0) with no constant
Random walk with drift	ARIMA(0,1,0) with a constant
AR(p)	ARIMA(p,0,0)
MA(q)	ARIMA(0,0,q)

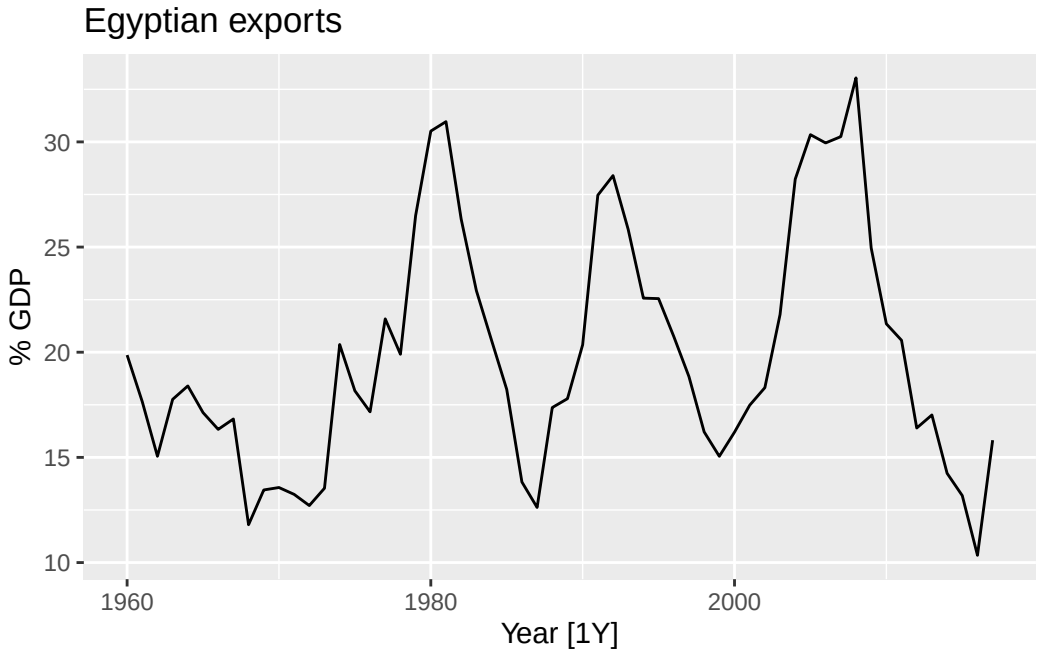
Once we start combining components in this way to form more complicated models, it is much easier to work with the backshift notation. For example, the ARIMA(p,d,q) can be written in backshift notation as follows:

$$\underbrace{(1 - \phi_1 B - \cdots - \phi_p B)}_{AR(p)} \quad \underbrace{(1 - B)^d y_t}_{\text{d differences}} = \underbrace{c + (1 + \theta_1 B + \cdots + \theta_q B^q)\epsilon_t}_{MA(q)}$$

Selecting appropriate values for p, d, and q can be difficult. However, the *ARIMA()* function from the **fable** package will do it for us automatically.

An example

```
library(fpp3)
global_economy %>%
  filter(Code == "EGY") %>%
  autoplot(Exports) +
  labs(y = "% GDP",
       title = "Egyptian exports")
```



The following code chunk automatically selects an ARIMA model.

```
fit <- global_economy %>%
  filter(Code=="EGY") %>%
  model(ARIMA(Exports))

report(fit)
```

```
Series: Exports
Model: ARIMA(2,0,1) w/ mean

Coefficients:
```

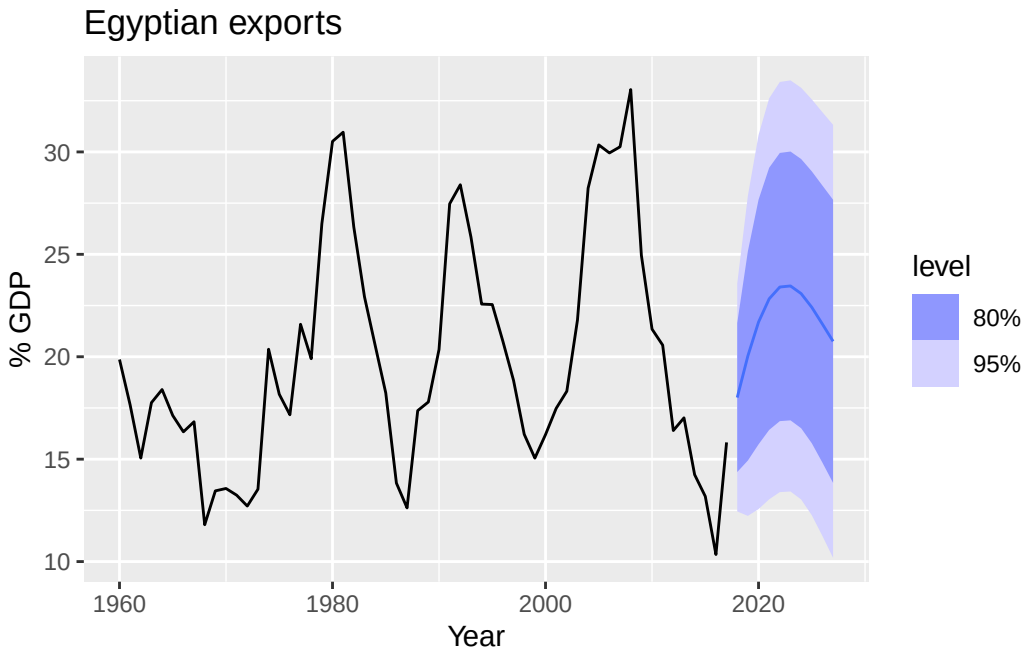
```
         ar1      ar2      ma1  constant
      1.6764  -0.8034  -0.6896    2.5623
s.e.  0.1111   0.0928   0.1492    0.1161

sigma^2 estimated as 8.046:  log likelihood=-141.57
AIC=293.13   AICc=294.29   BIC=303.43
```

This is an ARIMA(2,0,1) model with equation given by

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

where ϵ_t is white noise wirth standard deviation of of $2.837 = \sqrt{8.046}$. Forecasts from this model is shown below.

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



The *ARIMA()* function is useful, but anything automatic can be a little dangerous, and it is worth understanding something of the behavior of the models even when you rely on an automatic procedure to choose the model for you.

The constant c has an important effect on the long-term forecasts obtained from these models.

- If $c = 0$ and $d = 0$, the long-term forecasts will go to zero.
- If $c = 0$ and $d = 1$, the long-term forecasts will go to a non-zero constant.
- If $c = 0$ and $d = 2$, the long-term forecasts will follow a straight line.
- If $c \neq 0$ and $d = 0$, the long-term forecasts will go to the mean of the data.
- If $c \neq 0$ and $d = 1$, the long-term forecasts will follow a straight line.
- If $c \neq 0$ and $d = 1$, the long-term forecasts will follow a quadratic trend. (This is not recommended, and fable will not permit it.)

The value of d also has an effect on the prediction intervals — 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, so the prediction intervals will all be essentially the same.

This behavior is seen the forecasts from $ARIMA(2, 0, 1)$ of the Egyptian export data. The prediction intervals are almost the same width for the last few forecast horizons, and the final point forecasts are close to the mean of the data.

The value of p is important if the data show cycles. To obtain cyclic forecasts, it is necessary to have $p \geq 2$, along with some additional conditions on the parameters. For an AR(2) model, cyclic behavior occurs if $\phi_1^2 + 4\phi_2 < 0$ (as is the case for the Egyptian exports model).

ACF and PACF plots

It is usually not possible to tell, simply from a time plot, what values of p and q are appropriate for the data. However, it is sometimes possible to use the ACF plot, and the closely related PACF plot, to determine appropriate values for p and q.

Recall that an ACF plot shows the autocorrelations which measure the relationship between y_t and y_{t-k} for different values of k. Now if y_t and y_{t-1} are correlated, then y_{t-1} and y_{t-2} must also be correlated. However, y_t and y_{t-2} then might be correlated, simply because they are both connected to y_{t-1}, rather than because of any new information contained in y_{t-2} that could be used in forecasting y_t.

To overcome this problem, we can use partial autocorrelations. These measure the relationship between
y_t y_{t-k} after removing the effects of lags $1, 2, \cdots, k-1$. So the first partial autocorrelation is identical to the first autocorrelation, because there is nothing between them to remove. Each partial autocorrelation can be estimated as the last coefficient in an autoregressive model. Specifically, α_k, the k^{th} partial autocorrelation coefficient, is equal to the estimate of

ϕ_k in an $AR(k)$ model. In practice, there are more efficient algorithms for computing α_k than fitting all of these autoregressions, but they give the same results.

Below are the ACF and PACF plots for the Egyptian export data.

```
global_economy %>%
  filter(Code=="EGY") %>%
  ACF(Exports) %>%
  autoplot()
```

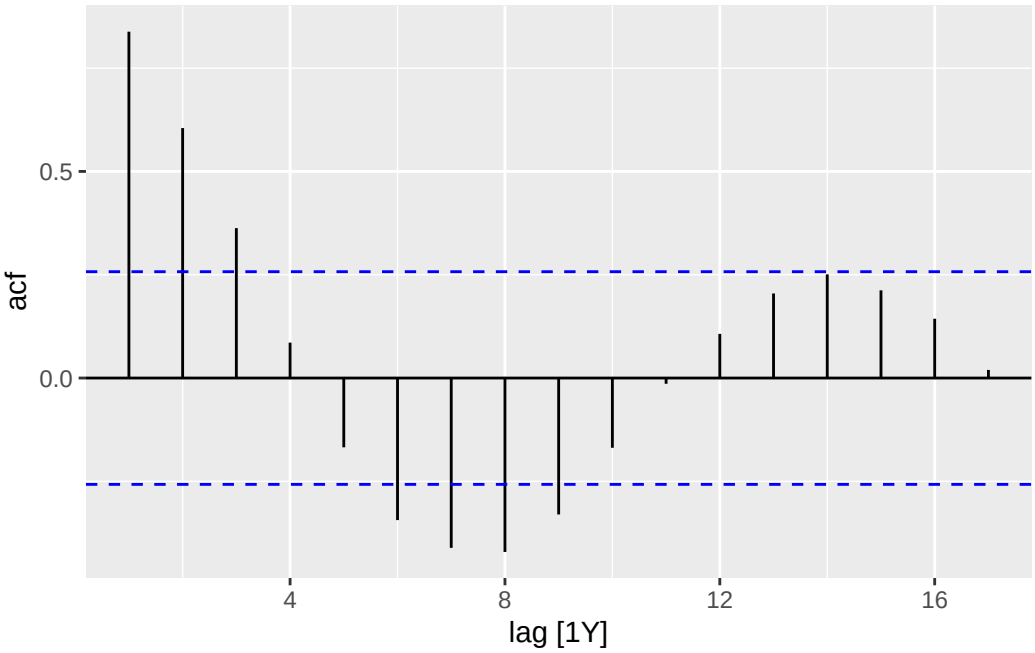


Figure 7: ACF Plot of Egyption Exports

```
global_economy %>%
  filter(Code=="EGY") %>%
  PACF(Exports) %>%
  autoplot()
```

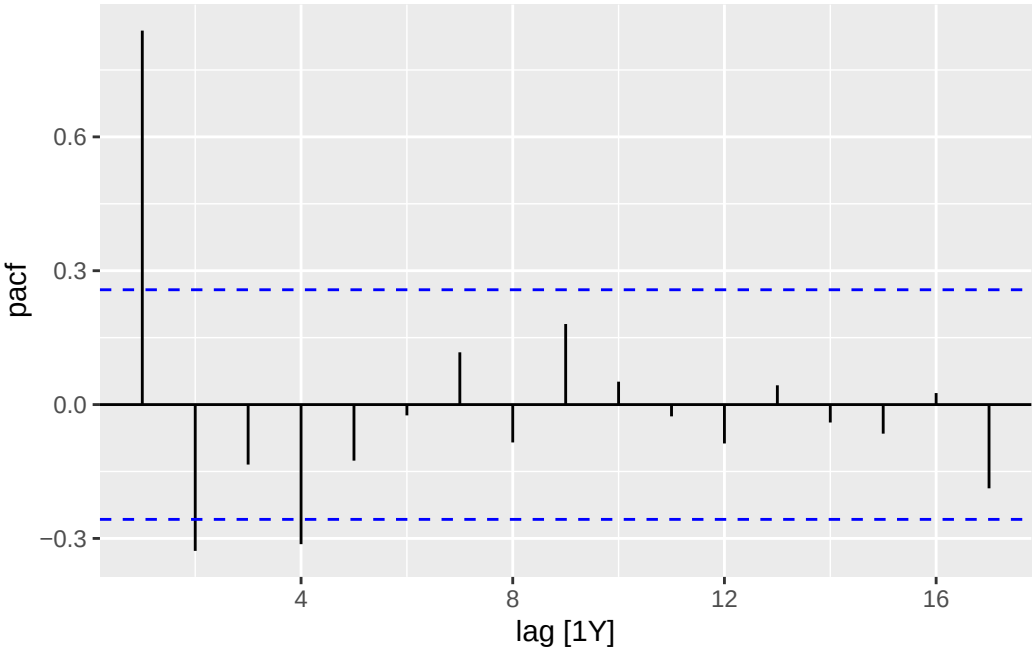
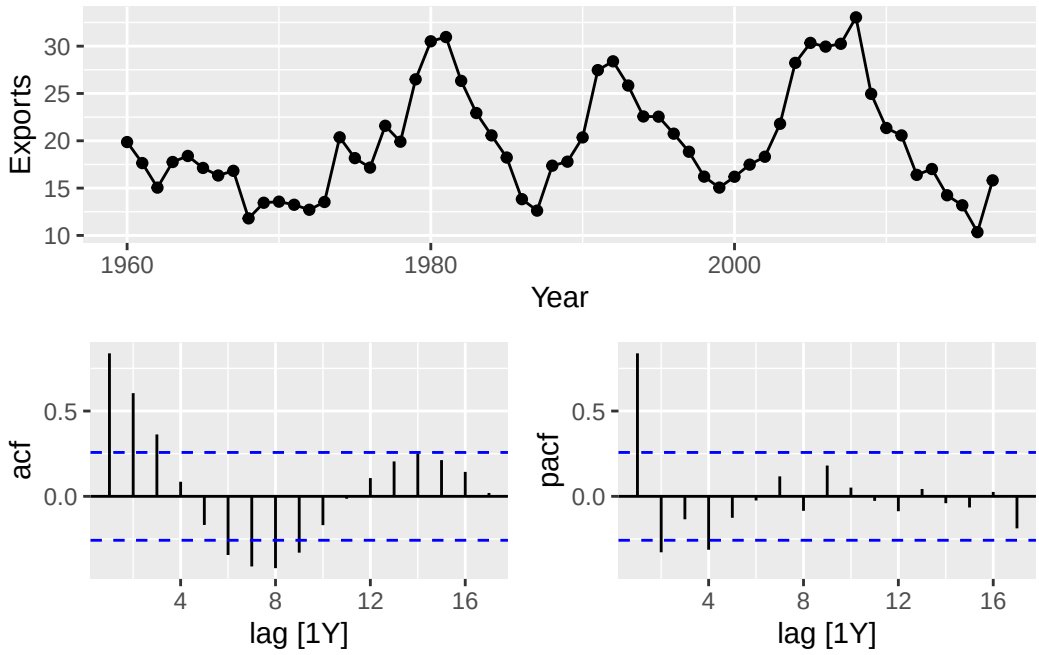


Figure 8: PACF Plot of Egyption Exports

The partial autocorrelations have the same critical values of $\pm 1.96/\sqrt{T}$ as for ordinary autocorrelations and these are the blue dashed lines in the ACF and PACF plots.

A convenient way to produce a time plot, ACF plot and PACF plot in one command is to use the *gg_tsdisplay()* function with *plot_type = "partial"*.

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



If the data are from an $ARIMA(p, d, 0)$ or $ARIMA(0, d, q)$ model, then the ACF and PACF plots can be helpful in determining the value of p or q.

The data may follow an $ARIMA(p, d, 0)$ model if the ACF and PACF plots of the differenced data show the following patterns:

- the ACF is exponentially decaying or sinusidal
- there is a significant spike at lag p in the PACF, but none beyond lag p

The data may follow a $ARIMA(0, d, q)$ model if the ACF and PACF plots of the differenced data show the following pattersn:

- the PACF is exponentially decaying or sinusidal
- there is a significant spike at lag q in the ACF, but none beyond lag q

In summary we have the following practical guidelines:

1. To identify the order of $AR(p)$ look at the PACF plot of the differenced series.
    - A sharp cutoff in the PACF, while the ACF tapers gradually, suggests an $AR(p)$ model.
    - The lag at which the PACF cuts off indicates the value of p.
2. To identify the order of $MA(q)$ look at the ACF plot.

- A sharp cutoff in the ACF, while the PACF tapers gradually, suggests an $MA(q)$ model.
- The lag at which the ACF cuts off suggests the value of q.

For example,

- If PACF cuts off at lag 2, while ACF gradually decreases, then an $AR(2)$ model might be appropriate.
- If ACF cuts off at lag 3, while PACF gradually decreases, then an $MA(3)$ model might fit.

Estimation and Order Selection

MAximum Likelihood Estimation

Once the model order has been identified (the values of p, d, and q), we proceed to estimating the parameters $c, \phi_1, \cdots, \phi_p, \theta_1, \cdots, \theta_q$. The **fable** package uses maximum likelihood estimation (MELE). 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 minimizing

$$\sum_{t=1}^{T} \epsilon_t^2.$$

Note that different softwares 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 maximize the log likelihood when finding parameter estimates.

Akaike Information Criterion

As a practical advise, we find other models in the vicinity of what we have identified based on the ACF and PACF plots or maybe using *ARIMA()* or *autoarima()* functions in R. Then we compare these models based on performance metrics such as the Akaike's Information Criterion (AIC) or the Bayesian Information Criterion (BIC).

The AIC is given by

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

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

For ARIMA models, the corrected AIC is preferred and is given by

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

while the BIC is written as

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

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

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.

ARIMA modelling in fable

The *ARIMA()* function in the **fable** package uses a variation of the Hyndman-Khandakar algorithm (Hyndman & Khandakar, 2008) which combines unit root tests, minimization of the *AICc* and MLE to obtain an ARIMA model. The arguments to the *ARIMA()* provide for many variations on the algorithm. The defaults are described below.

1. The number of difference $0 \leq d \leq 2$ is determined by using repeated KPSS tests.
2. The values of The values of p and q are then chosen by minimizing 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 of the current model are considered:

   - vary p and/or q from the current model ± 1
   - include or exclude a constant from the current model

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

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

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 stabilize 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 to identify initial values of p and q.
5. Try your chosen model(s), and use the AIC_c 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.

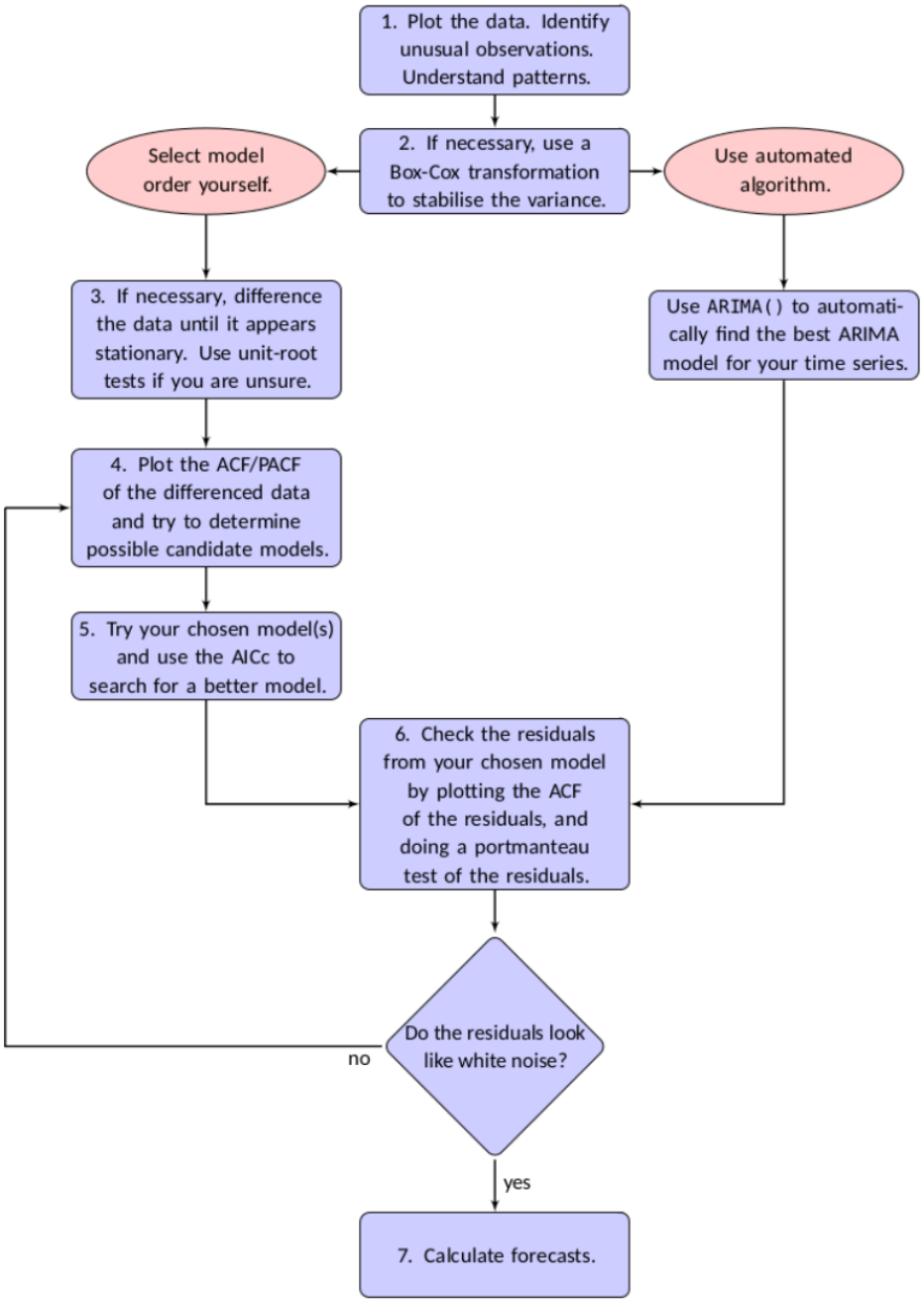
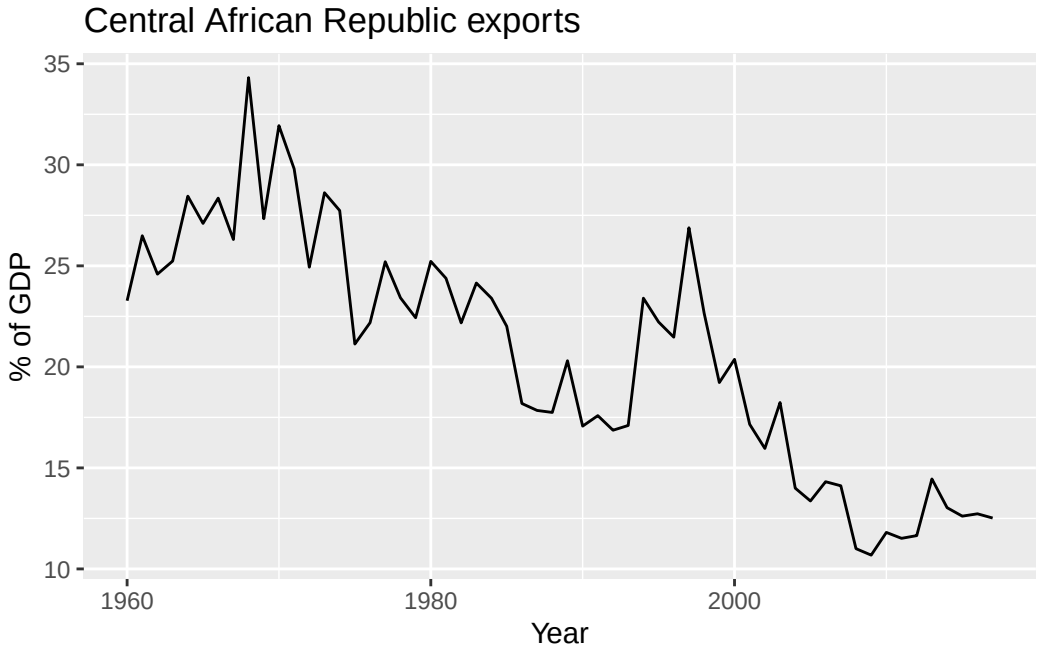


Figure 9: General process for forecasting using ARIMA model

Example

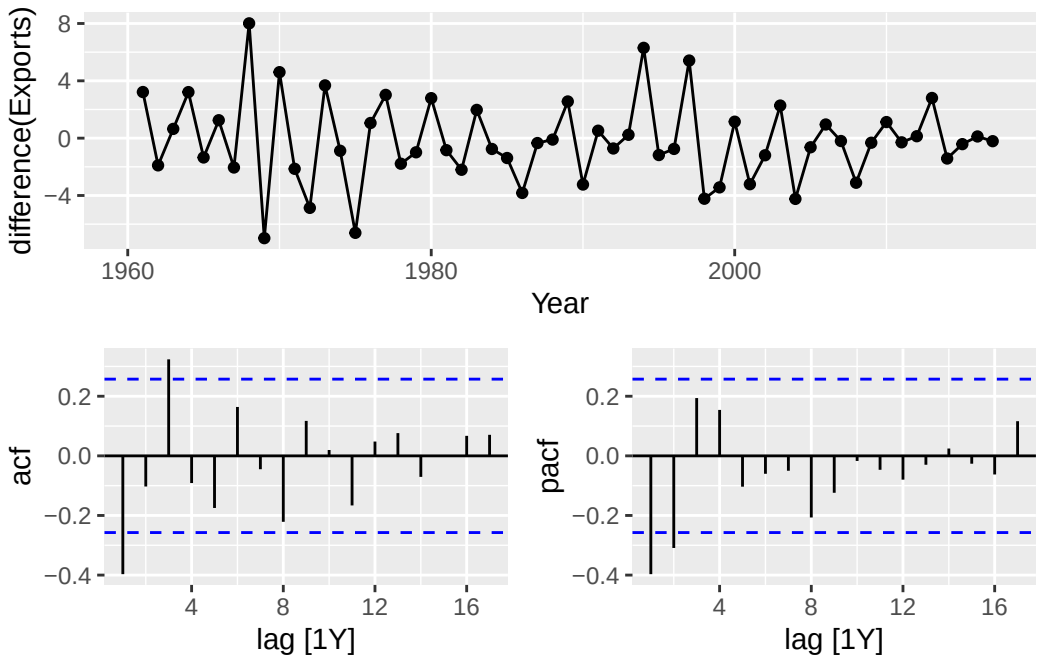
```
global_economy %>%
  filter(Code == "CAF") %>%
  autoplot(Exports) +
  labs(x = "Year",
       y = "% of GDP",
       title = "Central African Republic exports")
```



Highlights:

1. The time plot shows some non-stationarity, with an overall 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 below.

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



Highlights:

4. The differenced series now appear to be stationary.
5. The PACF shown 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).
6. 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.

```
arima.fit <- global_economy %>%
  filter(Code == "CAF") %>%
  model(arima210 = ARIMA(Exports ~ pdq(2,1,0)),
        arima013 = ARIMA(Exports ~ pdq(0,1,3)),
        arima_step = ARIMA(Exports),
        arima_search = ARIMA(Exports, stepwise = FALSE))

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

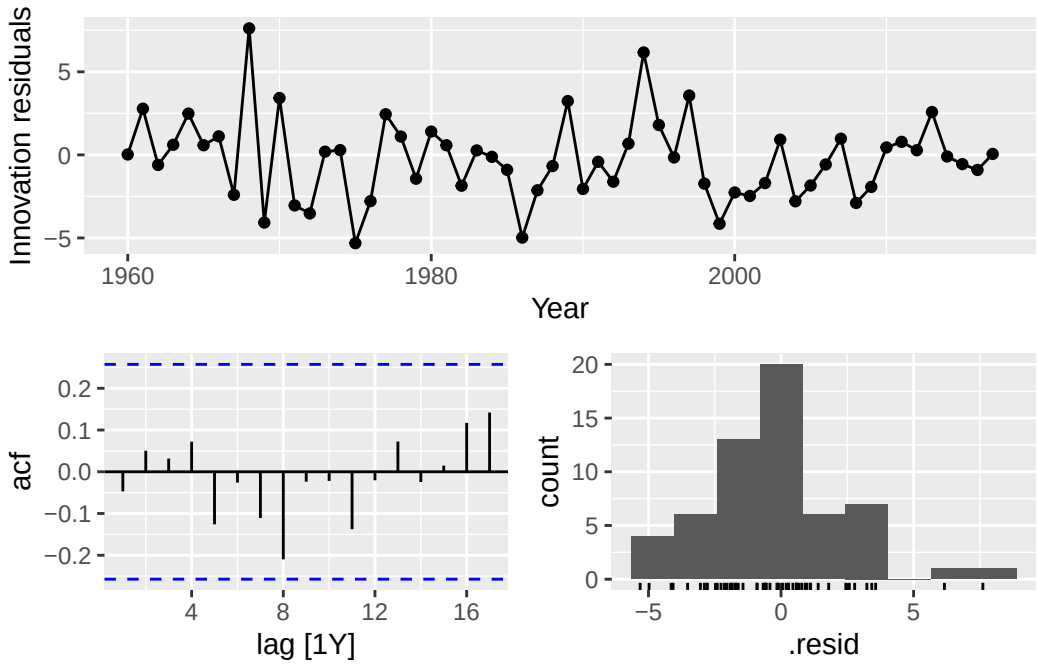
```
# A tibble: 4 x 6
  .model       sigma2 log_lik   AIC  AICc   BIC
  <chr>         <dbl>   <dbl> <dbl> <dbl> <dbl>
1 arima_search   6.52   -133.  274.  275.  282.
2 arima210       6.71   -134.  275.  275.  281.
3 arima013       6.54   -133.  274.  275.  282.
4 arima_step     6.42   -132.  274.  275.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

Highlights:

7. 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.

8. 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.

```
arima.fit %>%
  select(arima_search) %>%
  gg_tsresiduals()
```



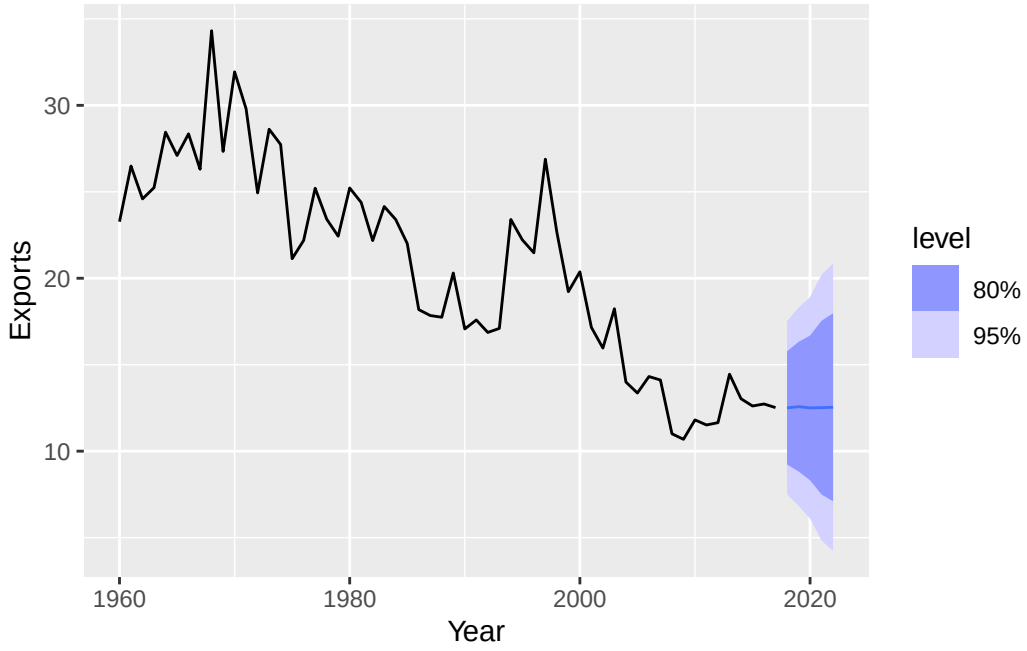
A portmanteau test (setting $K = 3$) returns a large p-value, also suggesting that the residuals are white noise.

```
augment(arima.fit) %>%
  filter(.model =="arima_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 arima_search    5.75     0.569
```

Forecasts from the chosen model are shown below.

```
arima.fit %>%
  forecast(h = 5) %>%
  filter(.model =="arima_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.