Leibniz Institute for Financial Research SAFE
Theodor-W.-Adorno-Platz 3 I 60629 Frankfurt am Main I Germany

LEIBNIZ INSTITUTE SAFE Sustainable Architecture for Finance in Europe

Time Series Analysis in R

Ina Krapp
SAFE Research Datacenter*

Last Update: April 16, 2024

R is a programming language often used for data analysis. In this tutorial, we will use it and RStudio to conduct time series analysis. RStudio is a user interface which makes working with R much more convenient. This tutorial is intended to be run as a RMarkdown file in RStudio, which allows you to execute the code yourself. To run it, download the RMarkdown version here and open it in a recent version of RStudio.

Contents

*krapp@safe-frankfurt.de

1 Contact

If you encounter any difficulties or just want general information, do not hesitate to contact us.

SAFE Research Datacenter: datacenter@safe-frankfurt.de

More information about the SAFE Research Datacenter and further guides can be found here.

2 Prerequisite

R can be downloaded here.
We'll also use RStudio. You can get it here.

3 Definition of a time series

A time series is a dataset in which one row contains a measurement value and another one contains the point in time when it was measured. Yearly, monthly or daily measurements are common, but every unit of time could be used.

4 Loading a time series into R

The tsibbledata package contains a time series analysis on production we will analyse. We can make it visible in the environment (upper right panel) by giving it a name:

```
australian_production = aus_production
```

We can look at it in the panel now (by clicking on its name) or use R to show the first rows of the table:

```
head(australian_production)
```

```
## # A tsibble: 6 x 7 [1Q]
##   Quarter  Beer Tobacco Bricks Cement Electricity   Gas
##     <qtr> <dbl>   <dbl>  <dbl>  <dbl>       <dbl> <dbl>
## 1 1956 Q1   284    5225    189    465        3923     5
## 2 1956 Q2   213    5178    204    532        4436     6
## 3 1956 Q3   227    5297    208    561        4806     7
## 4 1956 Q4   308    5681    197    570        4418     6
## 5 1957 Q1   262    5577    187    529        4339     5
## 6 1957 Q2   228    5651    214    604        4811     7
```

As we see, this is a table with 7 variables: A time variable (Quarter) and 6 different goods whose production values are measured.
In R, time series are stored as tsibble. This is a special type of data frame. You can verify this is a tsibble by making sure it has the class 'tbl_ts'.

```
class(australian_production)
```

```
## [1] "tbl_ts"     "tbl_df"     "tbl"        "data.frame"
```

5 Turning a dataset into a tsibble:

You can turn a dataset into a time series. To do that, you need to specify an index. The index is the column in which you find the time period. Look at this very small dataset with two columns and two rows:

```
testdata = data.frame(time = c(1,2), value = c(30, 40))
testdata
class(testdata)
```

```
##   time value
## 1    1    30
## 2    2    40
```

```
## [1] "data.frame"
```

So far, it is a simple dataframe. But its column 'time' can be used as index to turn it into a time series:

```
      teststibble = as_tsibble(testdata, index = time)
      class(teststibble)

```

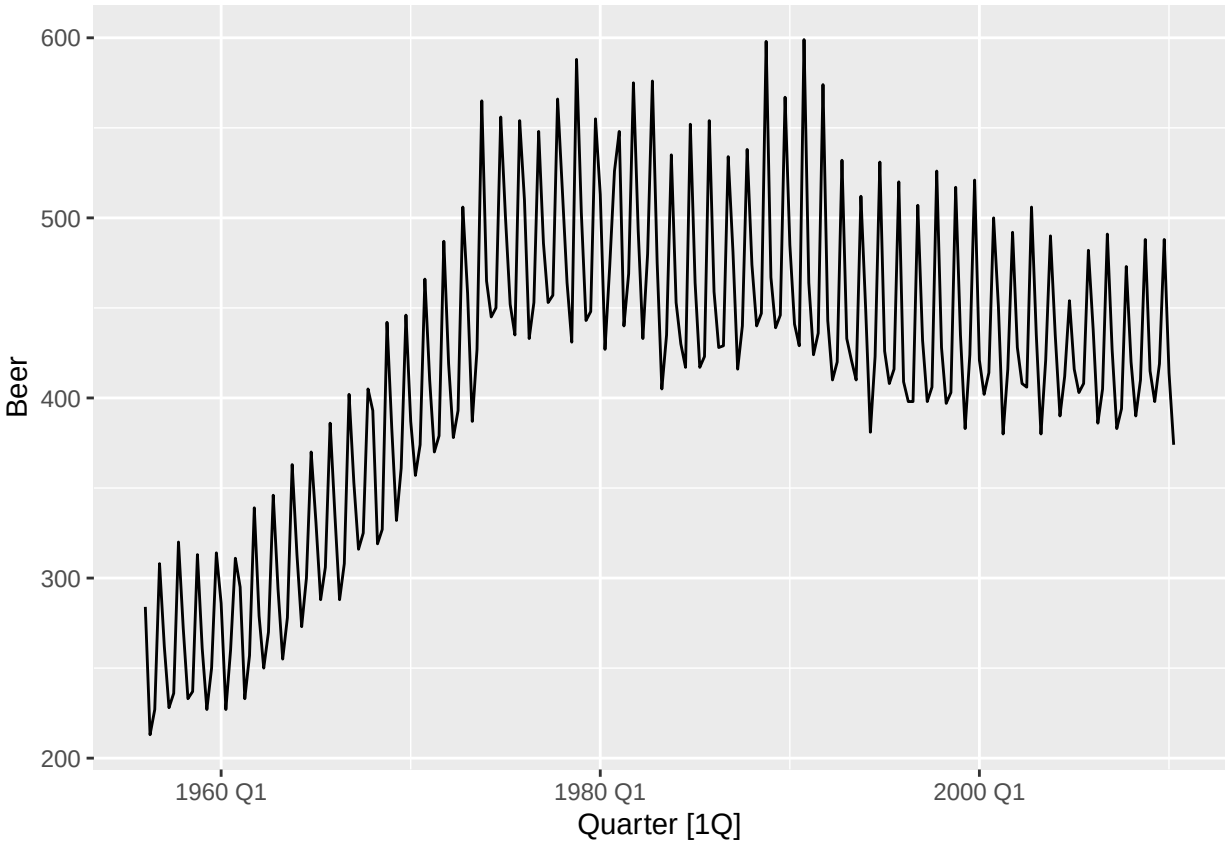
```
## [1] "tbl_ts"     "tbl_df"     "tbl"        "data.frame"
```

6 Visualizing time series

We will look more closely into the data of australian production now. To get an overview over the data, visualizing it is very helpful.

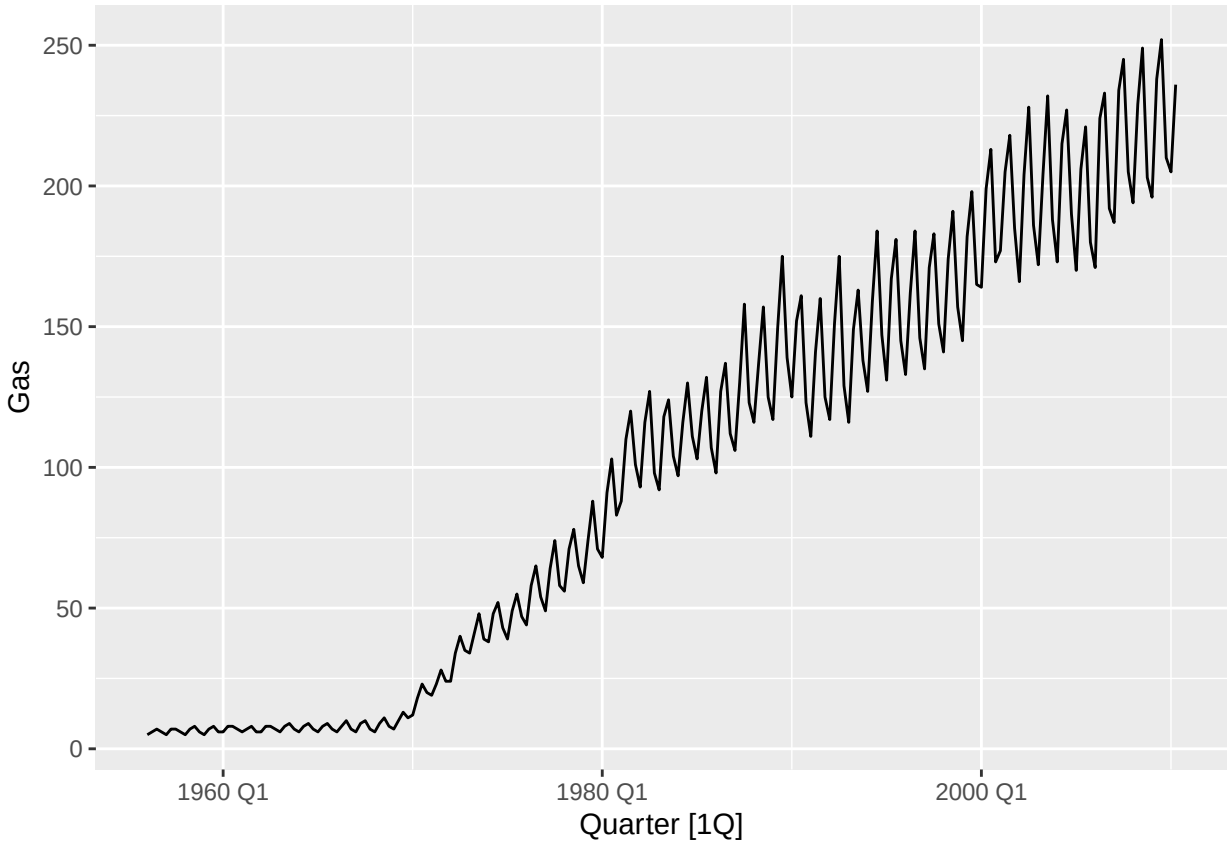
```
autoplot(australian_production)
```

```
## Plot variable not specified, automatically selected `.vars = Beer`
```



The autoplot-function automatically displays the time on the x-axis and the first other numeric variable on the y-axis. The first variable in this dataset is beer, so it is depicted above.
If we want to look at another variable, we have to specify it. Let's take a look at the production of natural gas:

```
1    autoplot(australian_production, .vars = Gas)
```



As we can see, the natural gas production was initially at a relatively low level and started to raise around 1970. It has always fluctuated, but as production increased, the fluctuations became stronger.

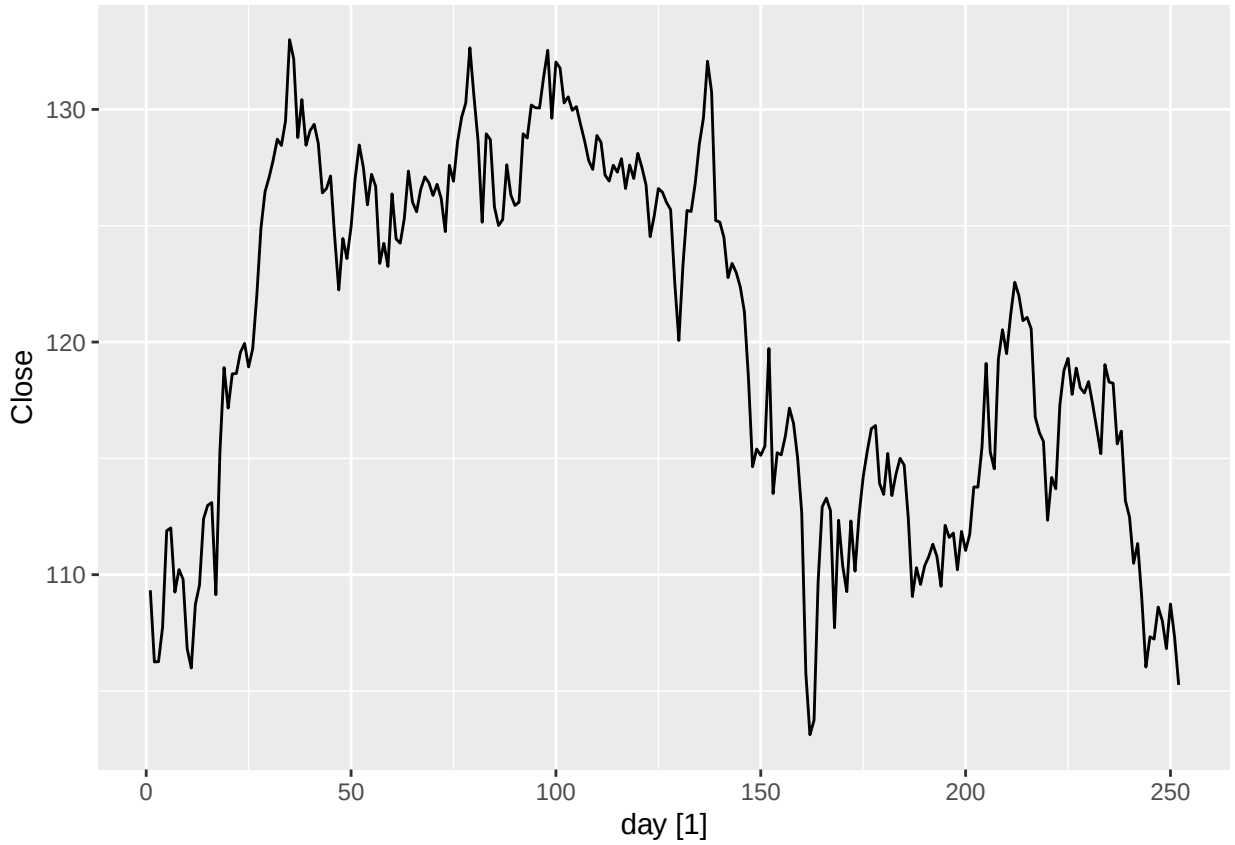
7 A second example: Stock prices

Other time series are more difficult to interpret directly. Stock prices, for example, are very volatile. Data on the apple stock can be loaded with this code:

```
  apple_stock = gafa_stock %>%
# Filter by stock and year
filter(Symbol == "AAPL", year(Date) == 2015) %>%
# Select the variables we are interested in
select(Symbol, Date, Close)%>%
mutate(day = row_number()) %>% # Re-index based on trading days
update_tsibble(index = day, regular = TRUE)
```

This code also creates a new variable, 'day', which simply assigns a number to each trading day. This is necessary because the stocks in this dataset are not traded on weekends, but a time series is not allowed to contain such gaps.
We can take a look at the developement of the apple stock in the year 2015 with the following code:

```
  # Plot the apple stock value:
  autoplot(apple_stock, .vars = Close)
```



8 The ARIMA model

To analyse time series, the ARIMA model is widely used. It contains of several parts which we'll look at piece by piece.
Consider a simple regression formula:
$y = x * \beta + \epsilon$
Such a formula models the relationship between an explanatory variable x and an explained variable y. It predicts y based on the values of x and β. $\hat{y} = x * \beta$
A common way to analyse a time series is to use a variant of this: An autoregressive model (AR model). This is the 'AR' in ARIMA. An autoregressive model uses past values of the y-variable as explanatory variables. Like in ordinary linear regression, there is an error ϵ, which can not be predicted, but will be 0 on average. The simplest form is: $y_t = y_{t-1} * \beta + \epsilon_t$
This model can be used for forecasting. If t is a future time period and t-1 is the current time period, then the prediction for the future is: $\hat{y}_t = y_{t-1} * \beta$
Like the ordinary linear regression model, it can easily be expanded. For example, more past time periods can be added: $\hat{y}_t = y_{t-1} * \beta_1 + y_{t-2} * \beta_2$
Future error values can not be observed, but past error values are known and can be included: $y_t = y_{t-1} * \beta_1 + y_{t-2} * \beta_2 + \rho_1 \epsilon_{t-1} + \rho_2 \epsilon_{t-2}$
This model with the inclusion of past errors is called 'autoregressive moving average' (ARMA). It works very well for stationary time series (stationary time series have a constant mean and variance). But the gas data is obviously not stationary: Its mean and its variance both increase over time.
The apple stock price data also is not looking stationary. One can test this formally with the Kwiatkowski-Phillips-Schmidt-Shin-test. If it gives a value of 0.01, the data is not stationary. If it gives a value of 0.1, the data is most likely stationary. In R, it can be used like this:

```
1    apple_stock %>%
```

```
  features(Close, unitroot_kpss)
```

```
## # A tibble: 1 x 3
##   Symbol kpss_stat kpss_pvalue
##   <chr>      <dbl>       <dbl>
## 1 AAPL        1.47        0.01
```

As can be seen, the apple stock data is most likely not stationary. When the data is not stationary, it can be differentiated. A dataset is differentiated by substracting the value of the previous time period from each value.

```
    # Differentiate the data
apple_stock = apple_stock%>%  mutate(diff_close = difference(Close))
head(apple_stock)
```

```
## # A tsibble: 6 x 5 [1]
## # Key:       Symbol [1]
##   Symbol Date       Close   day diff_close
##   <chr>  <date>     <dbl> <int>      <dbl>
## 1 AAPL   2015-01-02  109.     1    NA
## 2 AAPL   2015-01-05  106.     2    -3.08
## 3 AAPL   2015-01-06  106.     3     0.0100
## 4 AAPL   2015-01-07  108.     4     1.49
## 5 AAPL   2015-01-08  112.     5     4.14
## 6 AAPL   2015-01-09  112.     6     0.120
```

If a time series is differentiated before it is modeled by an ARMA model, the time series is called 'Integrated' (since Integration is the opposite of Differentiation). The model for the integrated time series then is called 'ARIMA', where the 'I' stands for 'Integrated'.
In the next step, we will create an ARIMA model for apple stock prices and use it to predict the future values of the stock.

9 Create an ARIMA model and predict stock prices:

The function 'ARIMA' can be used to pick an ARIMA model for a specific time series. It uses the kpss test to test if the data has to be differentiated. It also checks if past values or past errors can help predict future values.
Since we only used data from 2015 so far, we can create a forecast for January 2016 and compare it to the real values of the apple stock in this month. This code loads the apple stock data from January 2016.

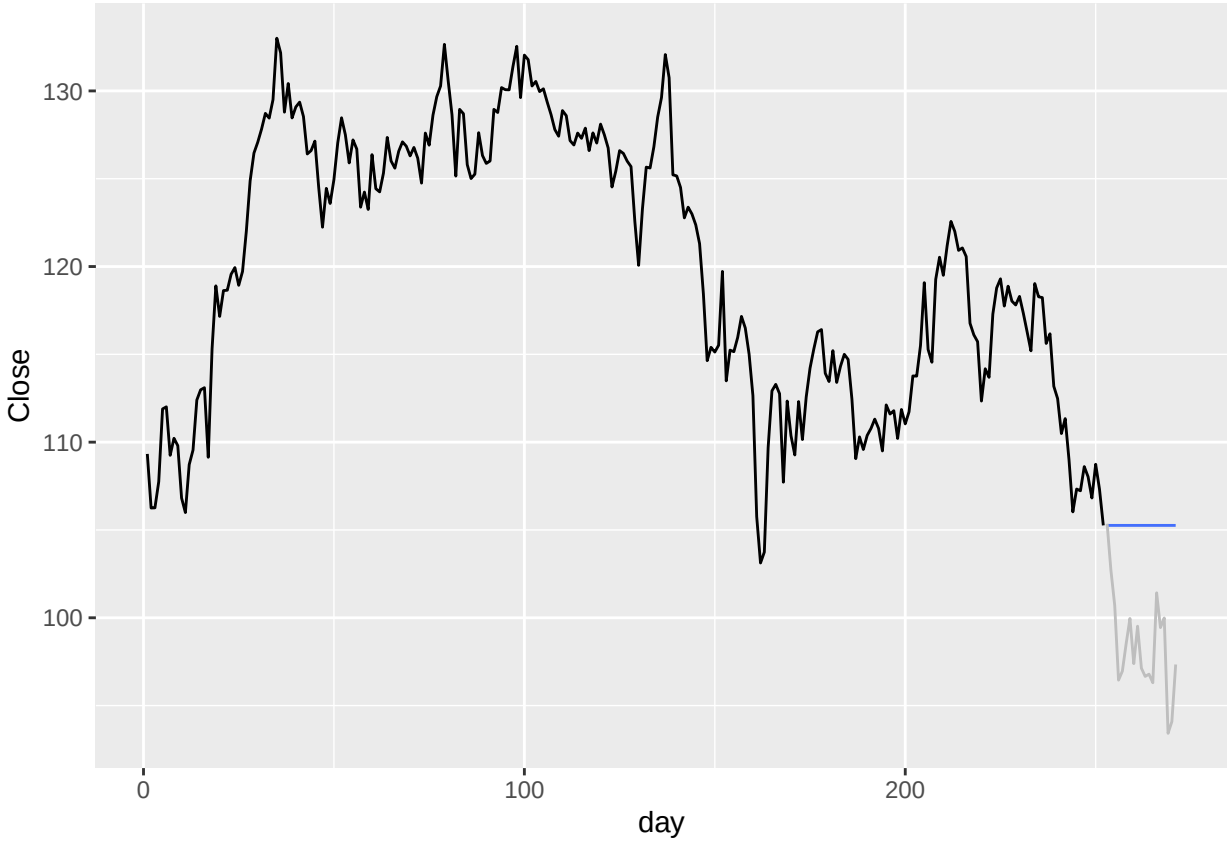
```
    # Get the values of the apple stock for January 2016
apple_jan_2016 = gafa_stock %>%
  filter(yearmonth(Date) == yearmonth("2016 Jan"), Symbol == "AAPL")%>%
  # Select the variables we are interested in
  select(Symbol, Date, Close)%>%
  mutate(day = row_number()+252) %>% # Re-index based on trading days
  update_tsibble(index = day, regular = TRUE)%>%  mutate(diff_close =
   difference(Close)) # Differentiate for january
```

Creating a forecast in R is composed of three steps: First, a model is created. We use an ARIMA model, but there are many different types of models available.
Second, based on the model, a forecast is created. We add the data from january 2016 as new data so the model predicts values for January.
The third step is to plot the forecast. Again, we can use the autoplot function which automatically adds the forecasted values. The function 'autolayer' adds the real data from january 2016 in a grey color for comparison.

```
    # Fit an ARIMA model for the apple stock:
apple_fit = apple_stock %>%
  model(Arima = ARIMA(Close))

# Create the forecast
apple_fc = apple_fit %>%
forecast(new_data = apple_jan_2016)

# Plot the forecast
apple_fc %>%
  autoplot(apple_stock, level = NULL) +
  autolayer(apple_jan_2016, Close, colour = "grey")
```



To learn more about the ARIMA model which was used to create this forecast, we can simply type the name of the fitted model.

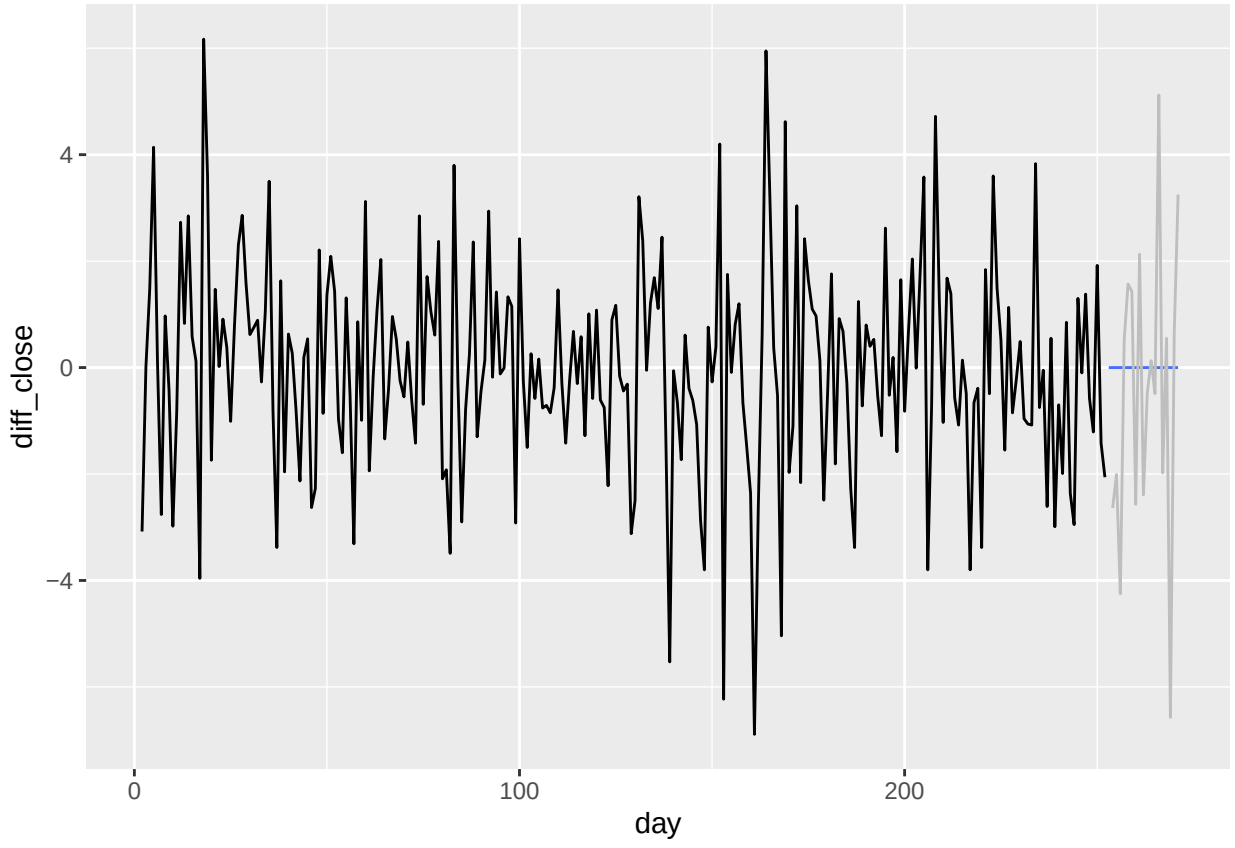
```
    apple_fit
```

```
## # A mable: 1 x 2
## # Key:     Symbol [1]
##   Symbol          Arima
##   <chr>         <model>
## 1 AAPL   <ARIMA(0,1,0)>
```

This output is interpreted as follows: The model fitted by the algorithm has no AR component, is integrated once, and has no MA component. In other words, the algorithm differentiated the data, like the kpss test we performed above suggested we should. The model does not include any past values or past errors (the AR and MA component are both zero). The algorithm is not guaranteed to be accurate, and you can also create a model by hand, but for stock market data, it is a common result that past values don't tell much about future values.
To see why this is the case. we can take a look at the differentiated data. Instead of the absolute value of the apple stock, it represents the change from day to day. So we can see if the value increased or decreased compared to the day before.
Again, we have the three steps to create a forecast: Fit the model, create the forecast and plot it and the real January data.

```
    # Fit the model
apple_fit = apple_stock %>%
  model(Arima = ARIMA(diff_close) )
# Create the forecast
apple_fc = apple_fit %>%
  forecast(new_data = apple_jan_2016)
# Plot the forecast
apple_fc %>%
  autoplot(apple_stock, level = NULL) +
  autolayer(apple_jan_2016, diff_close, colour = "grey")
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
## Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
```



As we can see, the model predicted the value of the apple stock to remain constant. Above,

we see the change in value, and the predicted change in value is 0. To see why the model gives this prediction, we can again look at the components of the model.

```
apple_fit
```

```
## # A mable: 1 x 2
## # Key:     Symbol [1]
##   Symbol          Arima
##   <chr>         <model>
## 1 AAPL   <ARIMA(0,0,0)>
```

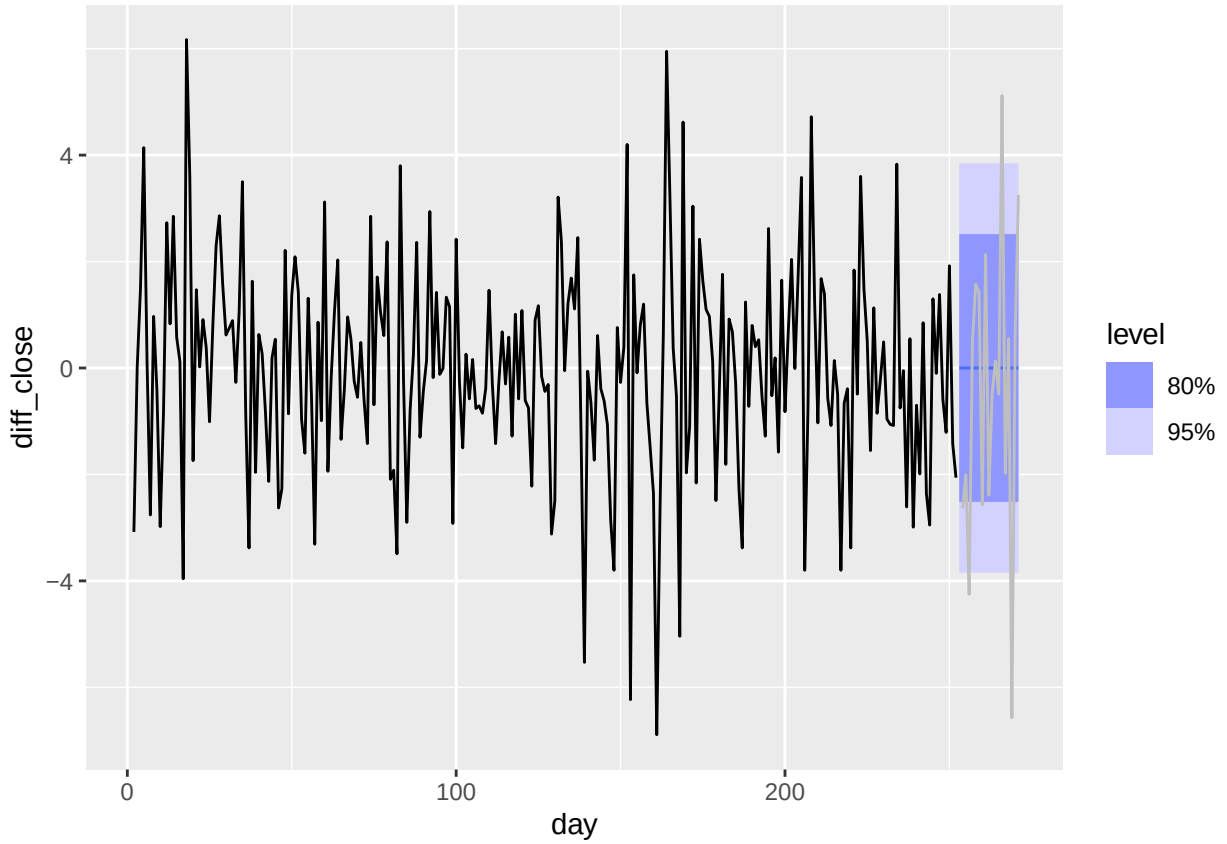
This special ARIMA model is called 'White Noise'. Expressed as a formula: $y_t = \epsilon_t$ Since the error term is 0 on average, the model predicts no change in the value of the apple stock. Of course, as we can see from the real January data, the value changed a lot. But the algorithm detected no predictable pattern in these changes. So even though the forecast is not very accurate, it is as accurate as it can be with this type of model. Next, we will look at how this uncertainty can be mesured and visualized.

10 Showing confidence intervals

Unless the time series is perfectly predictable, forecasts are not 100% accurate. Probably even more important than the future value that the model considers to be most likely is the confidence interval.The future value is likely to be higher or lower than the model predicts, but it is unlikely to be outside of the confidence interval.
The confidence interval can be computed for different probabilities. By default, it is computed in R for 80% and 95%. With 80% probability, a future value will be in the 80%-interval, and with 95%, it will be in the 95%-interval. The confidence interval is automatically shown unless you add 'level = NULL' in the autoplot function.

```
# Fit the model
apple_fit = apple_stock %>%
  model(Arima = ARIMA(diff_close) )
# Create the forecast
apple_fc = apple_fit %>%
  forecast(new_data = apple_jan_2016)
# Plot the forecast
apple_fc %>%
  autoplot(apple_stock) +
  autolayer(apple_jan_2016, diff_close, colour = "grey")
```

```
## Warning: Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
## Removed 1 row containing missing values or values outside the scale range
## (`geom_line()`).
```

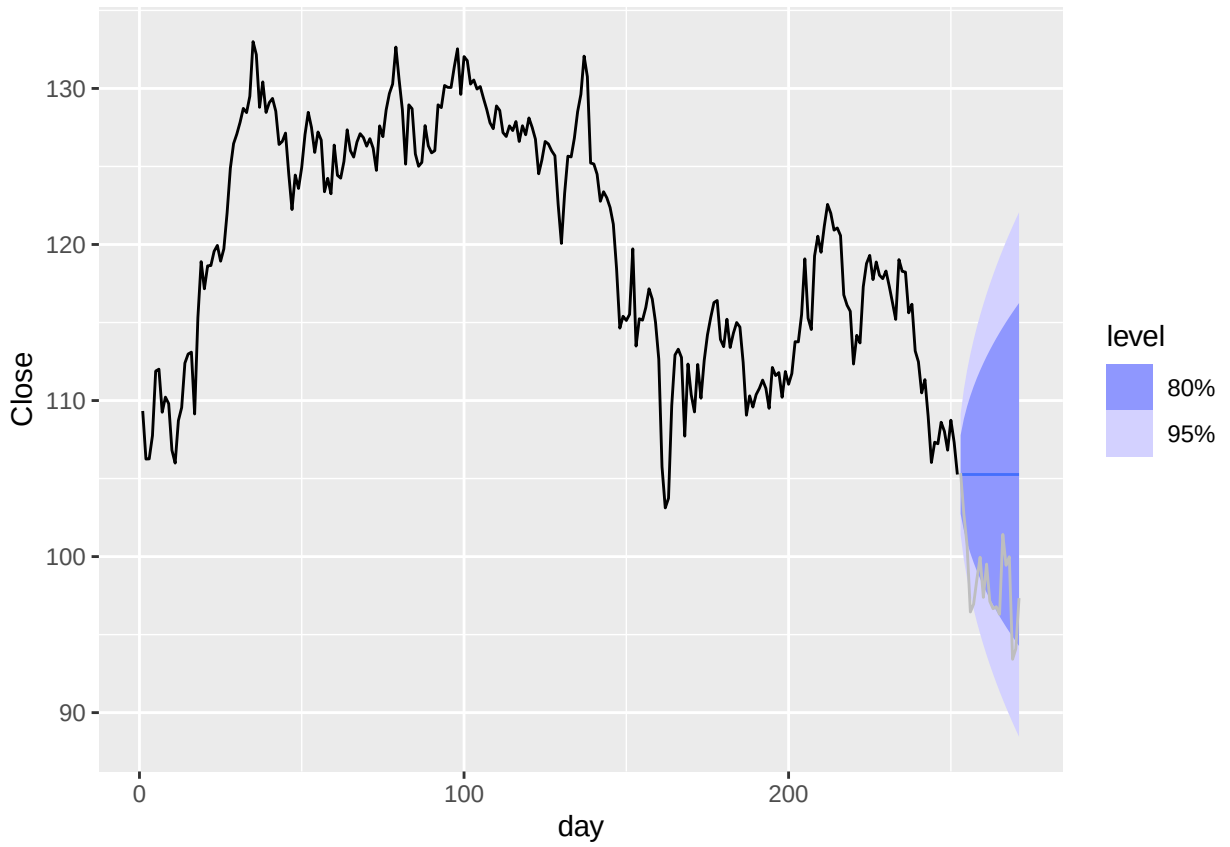


The confidence interval is constructed with the following insight: Keep in mind the model behind this forecast looks like this: $y_t = \epsilon_t$
We know that the model will not be accurate because it contains this error term. But we know the error term will have the value of 0 on average. Likewise, it has a standard deviation that can be calculated from the past data. Individual values may be above or below the confidence interval, but most values are within the calculated area. The random error term can give values very different from the mean, but the average size of the deviation from the mean can be calculated very well. In that sense, the prediction is fairly accurate.
We can also look at the confidence intervals for the value of the Apple stock:

```
# Fit an ARIMA model for the apple stock:
apple_fit = apple_stock %>%
  model(Arima = ARIMA(Close))

# Create the forecast
apple_fc = apple_fit %>%   # You can tell how far to the future the
forecast should go with 'new_data' or 'h'
forecast(new_data = apple_jan_2016)
  #  forecast(h = 30)

# Plot the forecast
apple_fc %>%
  autoplot(apple_stock) +
  autolayer(apple_jan_2016, Close, colour = "grey")
```

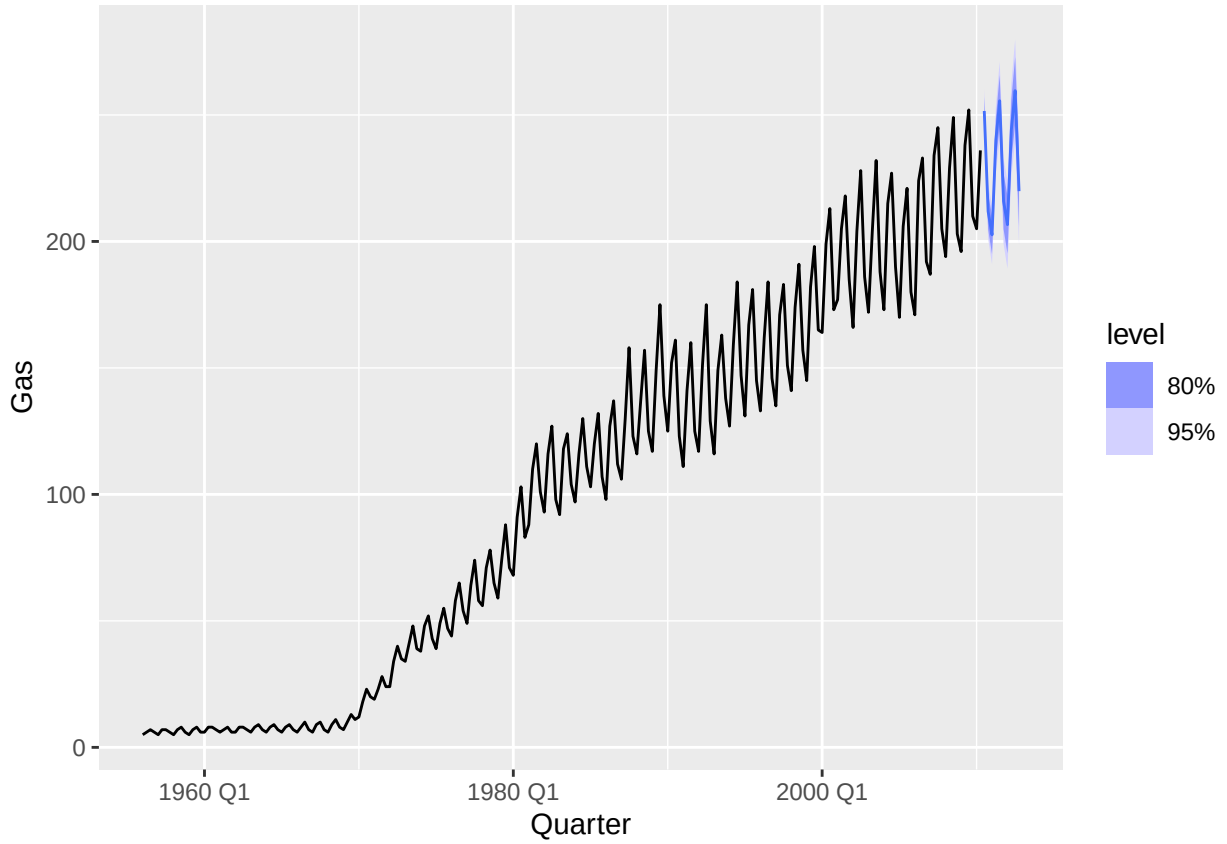


This is typical for stocks: They often follow a random walk. A random walk is a process where the errors sum up over time. The confidence interval is small in the first forecasted period, but increases quickly because the standard error grows. In the case of the apple stock, the real value is lower than the predicted value, but it is still within the 95% confidence interval.

11 An ARIMA model for the Australian gas production

As we've seen, ARIMA models can unfortunately not create very precise forecasts for the apple stock. Now, we will see that they are much more precise for the Australian gas production. There are different ways to define for how far in the future we want to create a forecast. So far, we simply created one to match the January data we wanted to compare our forecast to. Now, instead, we pass the parameter h to tell the program how far in the future the forecast is supposed to go. h is always in the same unit as the time variable of the data, so since we have data for every Quarter, 'h = 10' creates a prediction for the next 10 Quarters.

```
# Fit the model
gas_production_fit = australian_production %>%
  model(Arima = ARIMA(Gas) )
# Create the forecast
gas_production_fc = gas_production_fit %>%
  forecast(h = 10)
# Plot the forecast
gas_production_fc %>%
  autoplot(australian_production)
```



The forecast using an ARIMA model now looks very different. This model is looking much more precise. Its confidence intervals are smaller, and its prediction shows the same fluctuations like the past years.
As we've seen, the Australian gas production data contains more regular patterns than the apple stock market data. These patterns are picked up by the model, so the prediction no longer is simply a mean value for all future periods. Instead, the model predicts that the seasonal pattern will continue in the future. There is still an error term in the model, so the prediction may not be 100% accurate. But the confidence intervals are a lot smaller than previously.
This model also is much more complex than the previous ones. Looking at it, we get the following information:

```
gas_production_fit
```

```
## # A mable: 1 x 1
##                                    Arima
##                                  <model>
## 1 <ARIMA(0,0,3)(0,1,0)[4] w/ drift>
```

Let's move through this information piece by piece:

1. The model is an ARIMA model, like before. The ARIMA part gives three past error values that the model considers. It does not use any gas production values from previous years, and is not integrated.

2. It is a seasonal model. The data is given for each Quarter, so there are four seasons in the model. The number of seasons is given in the angular brackets. The brackets to the left of it give information about the seasonal ARIMA. They follow the same convention like the standard ARIMA: First are the AR coefficients (0 here), then if the data was

differentiated (yes, once), finally the MA coefficient (0 again). The difference between the seasonal and non-seasonal ARIMA is the reference. Previously, when we differentiated the model, we subtracted the value of yesterday from the value of today. But since the gas production data is quarterly, 'ordinary' differentiation would mean to subtract a value measured in winter from a value measured in spring. Instead, seasonal differentiation subtracts the value in the spring of one year from the value in the spring of the next year.

3. Finally, this model contains a drift. That means a constant is added in each period. We can see the value of the constant and the coefficients using the report command:

```
report(gas_production_fit)
```

```
## Series: Gas
## Model: ARIMA(0,0,3)(0,1,0)[4] w/ drift
##
## Coefficients:
##          ma1     ma2     ma3  constant
##       0.7684  0.5258  0.6811    4.0231
## s.e.  0.0492  0.0712  0.0551    0.8661
##
## sigma^2 estimated as 18.7:  log likelihood=-616.36
## AIC=1242.73   AICc=1243.02   BIC=1259.56
```

As we can see, the model assumes that the gas production will raise by 4.0231 petajoules each time period on average, so more than 16 petajoules a year. The influence of previous errors (the 'ma' coefficients) is positive, so if the production was unusually low or high in a quarter, it can be expected to remain unusually low or high for the next three quarters.

12 Forecasting using other explanatory variables

So far, we've only done forecasting based on the data of a single variable. But it is possible to include more variables into the model. However, this has to be handeled carefully. For example, assume we want to create a forecast of electricity production in Australia. We suspect that gas production has an influence (since gas can be used to create electricity), so we want to include it as explanatory variable.
Then, we require a prediction for the gas production first. The forecast of the electricity will be conditional on the assumption that the gas production will develop in a certain way. The prediction of the ARIMA model for the gas production appeared very precise, so we will assume that the gas production will develop like anticipated by the forecast. We define the future data for the prediction of the next 10 quarters as the mean predicted by the forecast:

```
future_gas_production = new_data(australian_production, 10) %>%
mutate(Gas = gas_production_fc$.mean)
```

The rest of the process is largely the same we know already. First, we fit a model to predict electricity. We now explicitly define Gas production as explanatory variable.

```
electricity_fit = australian_production %>%
model(ARIMA(Electricity ~ Gas))
```

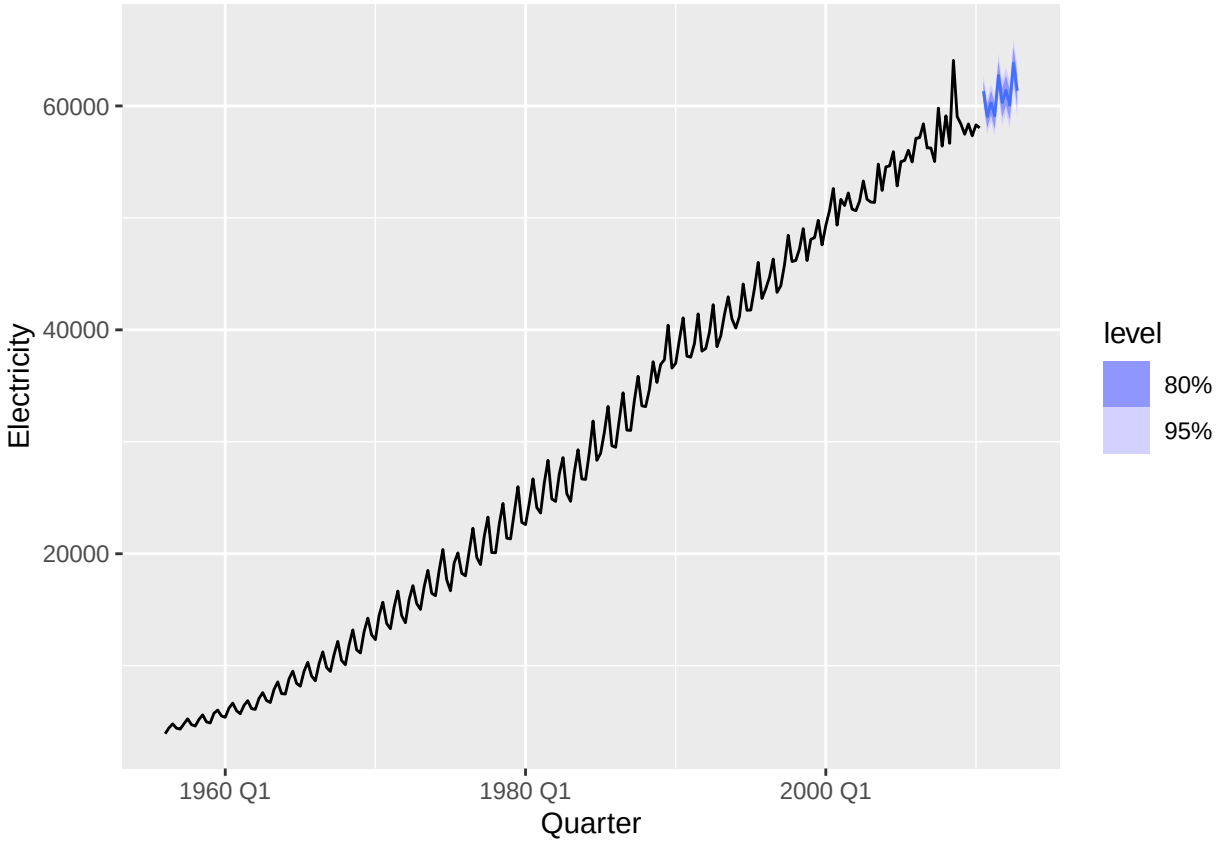
We can take a look at the model:

```
report(electricity_fit)
```

```
## Series: Electricity
## Model: LM w/ ARIMA(2,0,0)(1,1,2)[4] errors
##
## Coefficients:
##          ar1     ar2    sar1     sma1    sma2      Gas  intercept
##       0.5064  0.2389  0.8842  -1.6408  0.7204  30.7802   847.6066
## s.e.  0.0687  0.0701  0.0891   0.1094  0.0828  10.6951   125.6433
##
## sigma^2 estimated as 498454:  log likelihood=-1705.88
## AIC=3427.76   AICc=3428.46   BIC=3454.69
```

As we see, this is a Linear Model (LM). Gas production is estimated to have positive influence on electricity production. The model also contains ARIMA and Seasonal ARIMA components to predict the future development of the electricity production.
To create and plot the forecast, we give the prediction of the future gas production as new data. Then, we plot the forecast.

```
    # Create the forecast
electricity_fc = electricity_fit %>%
  forecast(new_data = future_gas_production)
# Plot the forecast
electricity_fc %>%
  autoplot(australian_production)
```

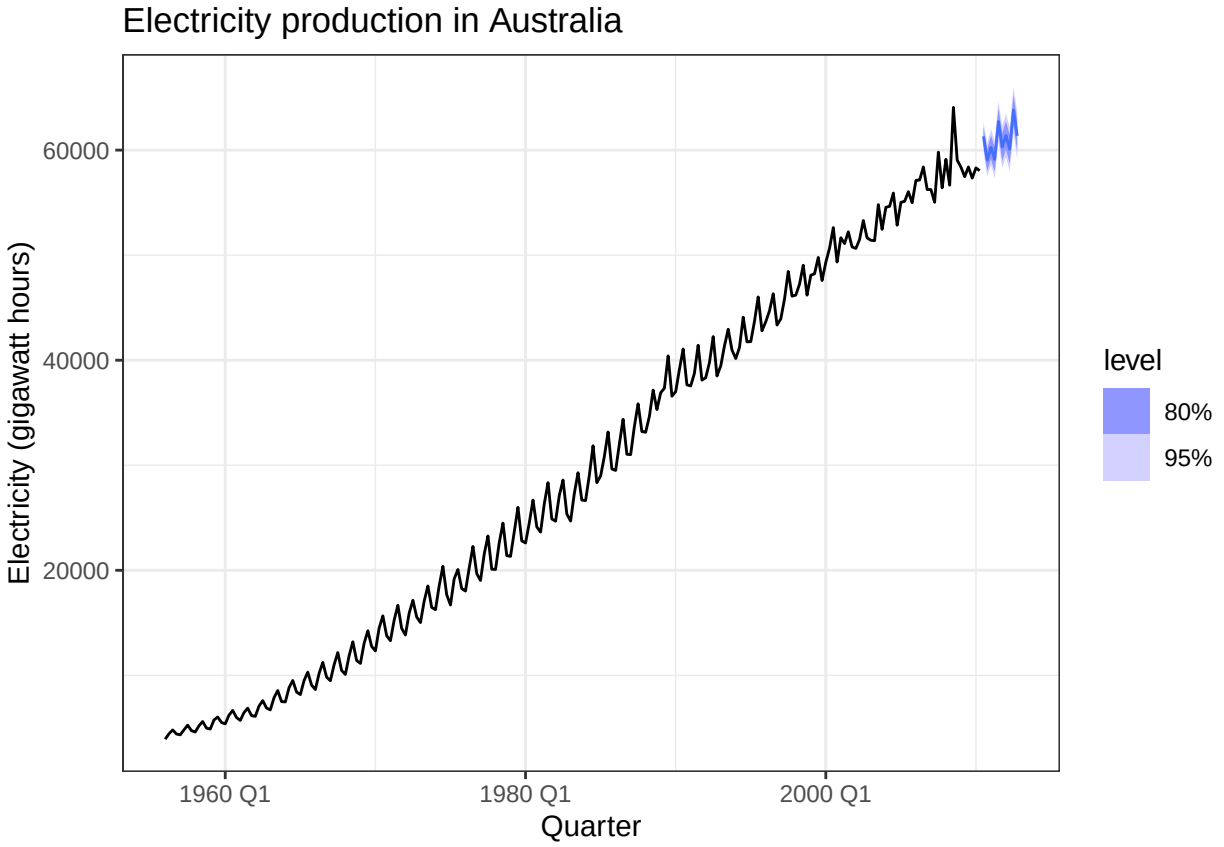


Electricity appears to be slightly more volatile than gas production.

13 Changing the appearance of a plot

This tutorial can not go into details about the appearance of plots in R. But I briefly want to show how to add a title and how to change the labeling of the x- and y-axis, so here is a modified plot of the electricity production. The code below also adds a theme, it changes the design of the graphic.

```
  electricity_fc %>%
autoplot(australian_production) +
labs(title = "Electricity production in Australia",x = "Quarter", y = "
 Electricity (gigawatt hours)" )+ theme_bw()
```



Finally, you can export plots with the following code:

```
  ggsave('my_plot.png')  # This saves the plot under the name 'my_plot',
you can give it any name you want.
```

```
## Saving 6.5 x 4.5 in image
```

This command always saves the last plot you created. pdf, jpg and a few other formats are also available, change the ending of the plot name depending on which one you want - 'my_plot.jpg' would save the plot as jpg and so on.

14 Sources

This tutorial is based on the third edition of 'Forecasting: Principles and Practice'. The book is available online: `https://otexts.com/fpp3/` It is written by the authors of the fable package that we used for forecasting here, and goes into much more detail than this tutorial could cover.