Chapter 5: The forecaster's toolbox

Ankit Gupta

27/12/2023

In this chapter, we discuss some general tools that are useful for many different forecasting situations. We will describe some benchmark forecasting methods, procedures for checking whether a forecasting method has adequately utilised the available information, techniques for computing prediction intervals, and methods for evaluating forecast accuracy.

Each of the tools discussed in this chapter will be used repeatedly in subsequent chapters as we develop and explore a range of forecasting methods.

5.1 A tidy forecasting workflow

The process of producing forecasts can be split up into a few fundamental steps:

(1) Preparing data

(2) Data Visualisation

(3) Specifying a model

(4) Model estimation

(5) Accuracy and performance evaluation

(6) Producing forecasts

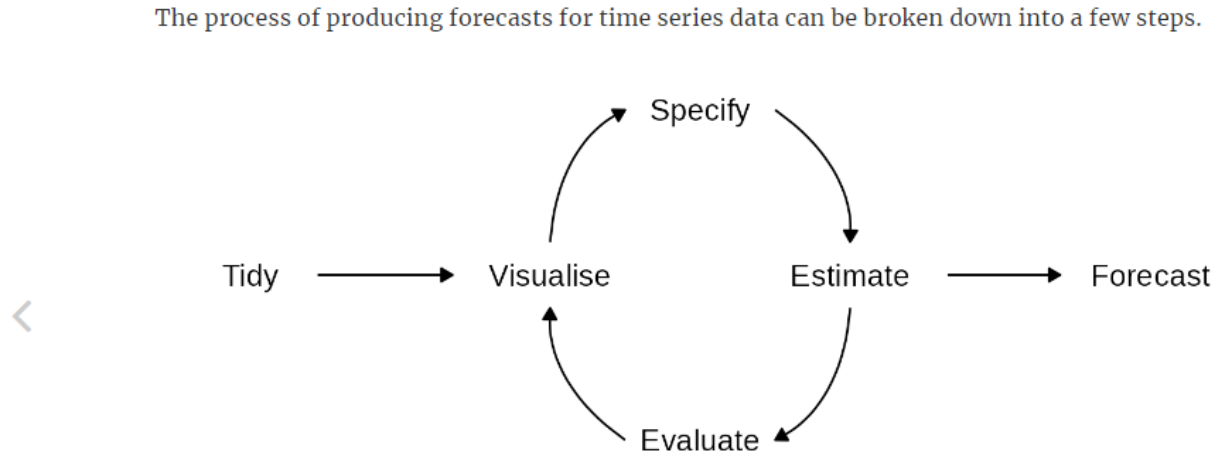


Figure 1: Process Workflow

Example

```
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
```

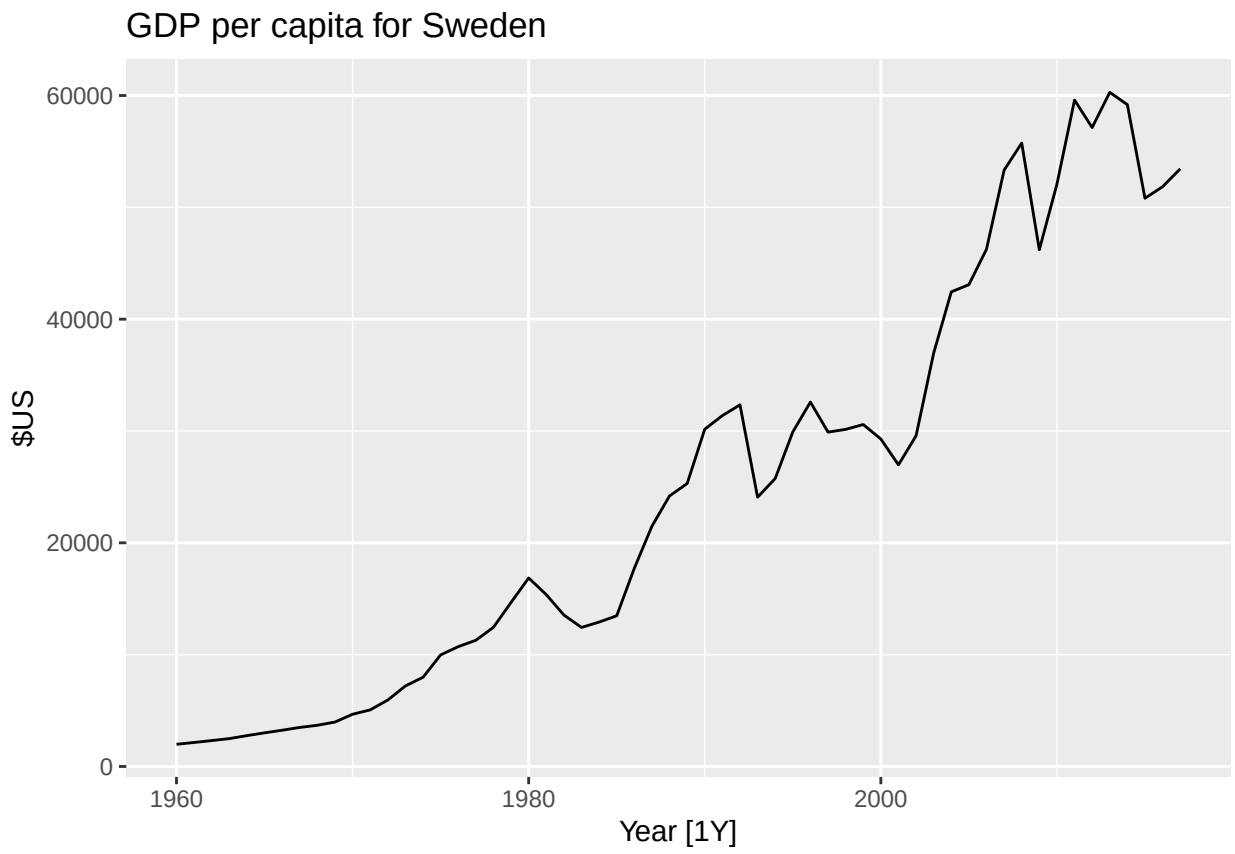
```
## -- Conflicts ------------------------------------------------ fpp3_conflicts --
## x lubridate::date()    masks base::date()
## x dplyr::filter()      masks stats::filter()
## x tsibble::intersect() masks base::intersect()
## x tsibble::interval()  masks lubridate::interval()
## x dplyr::lag()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

```
gdppc <- global_economy |>
  mutate(GDP_per_capita=GDP/Population) |>
  select(Year, Country, GDP, Population, GDP_per_capita)
gdppc
```

```
## # A tsibble: 15,150 x 5 [1Y]
## # Key:       Country [263]
##     Year Country             GDP Population GDP_per_capita
##    <dbl> <fct>               <dbl>      <dbl>          <dbl>
##  1  1960 Afghanistan  537777811.    8996351           59.8
##  2  1961 Afghanistan  548888896.    9166764           59.9
##  3  1962 Afghanistan  546666678.    9345868           58.5
##  4  1963 Afghanistan  751111191.    9533954           78.8
##  5  1964 Afghanistan  800000044.    9731361           82.2
##  6  1965 Afghanistan 1006666638.    9938414          101.
##  7  1966 Afghanistan 1399999967.   10152331          138.
##  8  1967 Afghanistan 1673333418.   10372630          161.
##  9  1968 Afghanistan 1373333367.   10604346          130.
## 10  1969 Afghanistan 1408888922.   10854428          130.
## # i 15,140 more rows
```

Data Visualisation

```
gdppc |>
  filter(Country=="Sweden") |>
  autoplot(GDP_per_capita)+
  labs(title = "GDP per capita for Sweden", y= "$US")
```



It is having an upward trend. Since it is yearly data, so it does not have the seasonal component.

Model estimation

Here, we use a linear model using the time trend. I am not saying that this is the best model for time series. It is just for demo.

```
fit <- gdppc |>
  model(trend_model=TSLM(GDP_per_capita ~ trend()))
```

```
## Warning: 7 errors (1 unique) encountered for trend_model
## [7] 0 (non-NA) cases
```

```
fit
```

```
## # A mable: 263 x 2
## # Key:     Country [263]
##    Country             trend_model
##    <fct>                   <model>
##  1 Afghanistan              <TSLM>
##  2 Albania                  <TSLM>
##  3 Algeria                  <TSLM>
##  4 American Samoa           <TSLM>
##  5 Andorra                  <TSLM>
##  6 Angola                   <TSLM>
##  7 Antigua and Barbuda      <TSLM>
##  8 Arab World               <TSLM>
##  9 Argentina                <TSLM>
```

```
## 10 Armenia                   <TSLM>
## # i 253 more rows
```

Here, w edenote the model notationally as:

$$y_t = \beta_0 + \beta_1 t + \epsilon_t$$

where ϵ_t $N(0, \sigma^2)$

Here, TSML is a linear model for time series. Here y variable is GPP_per_capita and x variable(explanatory variable) is time variable "t" which is showing by trend(). trend() (a "special" function specifying a linear trend when it is used within TSLM().

A "mable" as shown in the above table, is a model table where each cell corresponds to a fitted model.

Evaluation

We are skipping it for now

Producing forecasts

```
fit |> forecast(h="3 years")
```

```
## # A fable: 789 x 5 [1Y]
## # Key:     Country, .model [263]
##    Country        .model       Year   GDP_per_capita  .mean
##    <fct>          <chr>       <dbl>           <dist>  <dbl>
##  1 Afghanistan    trend_model  2018     N(526, 9653)   526.
##  2 Afghanistan    trend_model  2019     N(534, 9689)   534.
##  3 Afghanistan    trend_model  2020     N(542, 9727)   542.
##  4 Albania        trend_model  2018  N(4716, 476419)  4716.
##  5 Albania        trend_model  2019  N(4867, 481086)  4867.
##  6 Albania        trend_model  2020  N(5018, 486012)  5018.
##  7 Algeria        trend_model  2018  N(4410, 643094)  4410.
##  8 Algeria        trend_model  2019  N(4489, 645311)  4489.
##  9 Algeria        trend_model  2020  N(4568, 647602)  4568.
## 10 American Samoa trend_model  2018 N(12491, 652926) 12491.
## # i 779 more rows
```

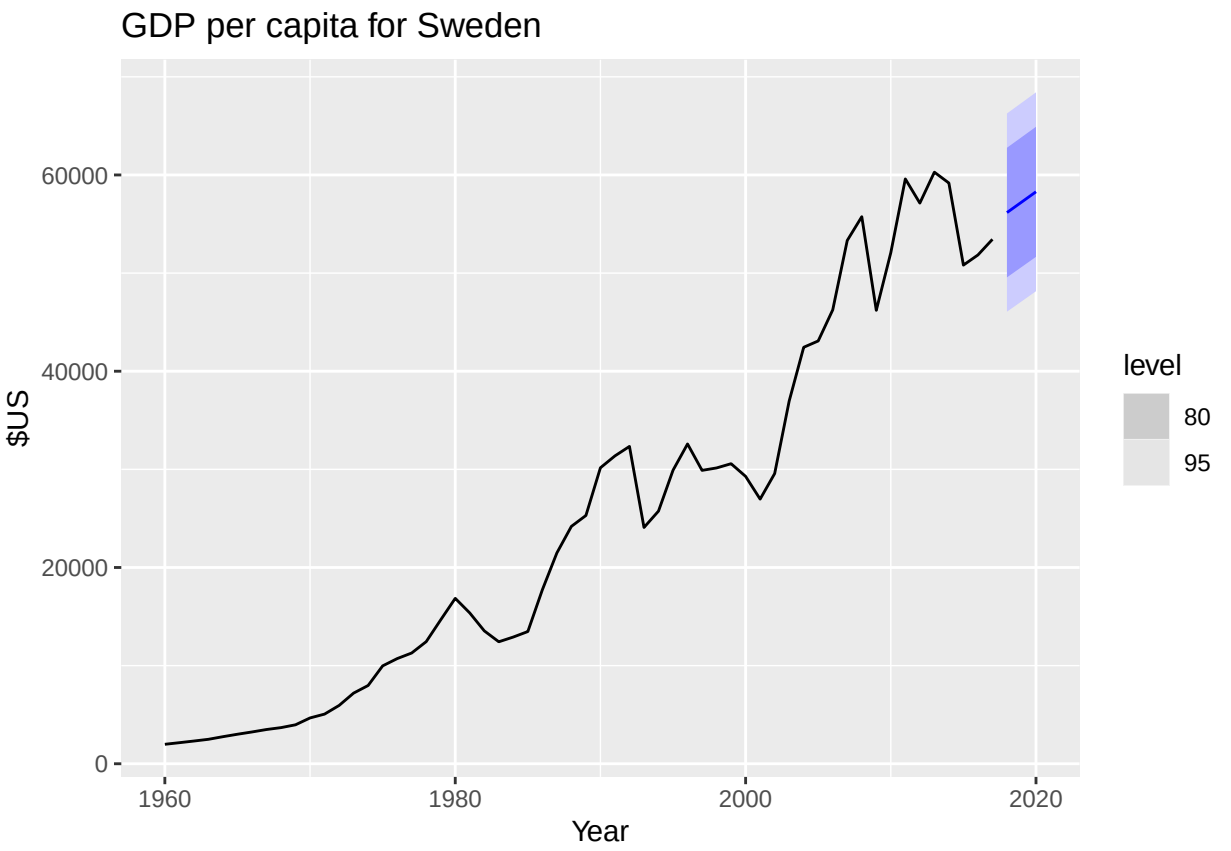
A "fable" is a forecast table with point forecasts and distributions. It is similar to a forecast. It tells something about future that is not true but it is informative about the future and very similar to what forecast is.

h=3 years means it is forecasting for next 3 years.

Our forecast is a whole distribution and a point forecast is something which we select from a distribution. N(.,.) represents the normal distribution. The GDP_per_capita column contains the forecast distribution, while the .mean column contains the point forecast. The point forecast is the mean (or average) of the forecast distribution.

Visualizing Forecasts

```
fit |>
  forecast(h= "3 years") |>
  filter(Country == "Sweden") |>
  autoplot(gdppc)+labs(title = "GDP per capita for Sweden", y="$US")
```



Summary

Data preparation (tidy)

The first step in forecasting is to prepare data in the correct format. This process may involve loading in data, identifying missing values, filtering the time series, and other pre-processing tasks. The functionality provided by tsibble and other packages in the tidyverse substantially simplifies this step.

Many models have different data requirements; some require the series to be in time order, others require no missing values. Checking your data is an essential step to understanding its features and should always be done before models are estimated.

We will model GDP per capita over time; so first, we must compute the relevant variable.

Plot the data (visualise)

As we have seen in Chapter 2, visualisation is an essential step in understanding the data. Looking at your data allows you to identify common patterns, and subsequently specify an appropriate model.

Define a model (specify)

There are many different time series models that can be used for forecasting, and much of this book is dedicated to describing various models. Specifying an appropriate model for the data is essential for producing appropriate forecasts.

Models in fable are specified using model functions, which each use a formula (y ~ x) interface. The response variable(s) are specified on the left of the formula, and the structure of the model is written on the right.

Train the model (estimate)

Once an appropriate model is specified, we next train the model on some data. One or more model specifications can be estimated using the model() function.

Check model performance (evaluate)

Once a model has been fitted, it is important to check how well it has performed on the data. There are several diagnostic tools available to check model behaviour, and also accuracy measures that allow one model to be compared against another. Sections 5.8 and 5.9 go into further details.

Produce forecasts (forecast)

With an appropriate model specified, estimated and checked, it is time to produce the forecasts using forecast(). The easiest way to use this function is by specifying the number of future observations to forecast. For example, forecasts for the next 10 observations can be generated using h = 10. We can also use natural language; e.g., h = "2 years" can be used to predict two years into the future.

In other situations, it may be more convenient to provide a dataset of future time periods to forecast. This is commonly required when your model uses additional information from the data, such as exogenous regressors. Additional data required by the model can be included in the dataset of observations to forecast.

5.2 Some simple forecasting methods

All these methods are benchmark methods.

(1) *Mean(y): Average method:*

- Forecast of all future values is equal to mean of historical data $y_1, y_2, ..., y_T$ So, it is s simple method. We take the simple average of all historical data and it becomes the point forecast for all the future values.
- Forecasts: $\hat{y}_{T+h|T} = \overline{y} = \frac{(y_1+y_2+...y_T)}{T}$
- It is not a great forecast but it is a benchmark for which we compare the other methods.

(2) *Naive(y): Naive method:*

- Forecasts equal to last observed value.
- Forecasts: $\hat{y}_{T+h|T} = y_T$
- Consequence of efficient market hypothesis. It means if we have a financial data and we have a efficient market then the efficient market hypothesis says that this is the best forecast that is the stock price or exchange rate.

(3) *SNaive(y~lag(m)): Seasonal Naive method:*

- Forecasts equal to the last value from the same season. It means if we have a quarterly data and we are trying to predict quarter 1 then we take the last observed values from quarter 1 and similarly if we have to predict quarter 2 then we have to take the last observed value from quarter 2. It means last full year data is replicated into the future.
- Forecasts: $\hat{y}_{T+h|T} = y_{T+h-m(k+1)}$ where $m = seasonal\ period$ and k is the integer part of $\frac{h-1}{m}$.

where m=the seasonal period, and k is the integer part of (h−1)/m (i.e., the number of complete years in the forecast period prior to time T+h).

For example, with monthly data, the forecast for all future February values is equal to the last observed February value. With quarterly data, the forecast of all future Q2 values is equal to the last observed Q2 value (where Q2 means the second quarter). Similar rules apply for other months and quarters, and for other seasonal periods.

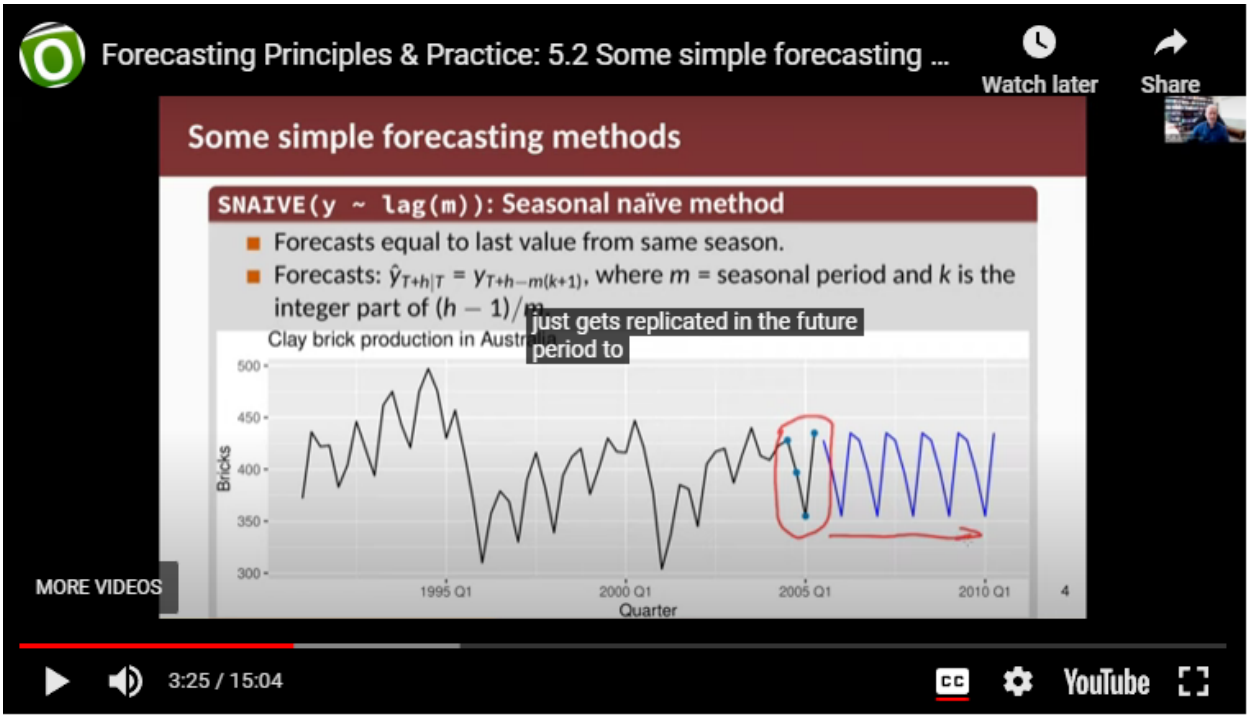


Figure 2: Last year data is replicated in future to give future point estimate

(4) *RW(y~drift()): Drift method:*

- forecasts equal to the last value plus average change.
- Forecasts:

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\Sigma_{t=2}^{T}(y_t - y_{t-1})$$

$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}(y_T - y_1)$$

* It is equivalent to extrapolating a line drawn between first and last observations.

- Here the summation is from t=2 to t=T so total period is T-2+1=T-1 and so we divide it by T-1 to get the average and "h" is number of steps ahead (forecast period) and so we multiplied it by "h".

[Extrapolation line which connect first and last observation is extended fot future forecast] (img2.png)

Now, we see how to implement these methods in R.

Model fitting

```
brick_fit <- aus_production |>
  filter(!is.na(Bricks)) |>
  model(
    Seasonal_naive = SNAIVE(Bricks),
    Naive=NAIVE(Bricks),
    Drift=RW(Bricks~drift()),
```

```
    Mean=MEAN(Bricks)
  )
brick_fit
```

```
## # A mable: 1 x 4
##   Seasonal_naive   Naive         Drift    Mean
##           <model> <model>       <model> <model>
## 1       <SNAIVE> <NAIVE> <RW w/ drift>  <MEAN>
```

Here, drift model uses the RW() function which stands for the random walk. In the Naive model, we can also replace NAIVE() with RW(), we get the same result.

Producing forecasts

```
brick_fc <- brick_fit |>
  forecast(h="5 years")
```

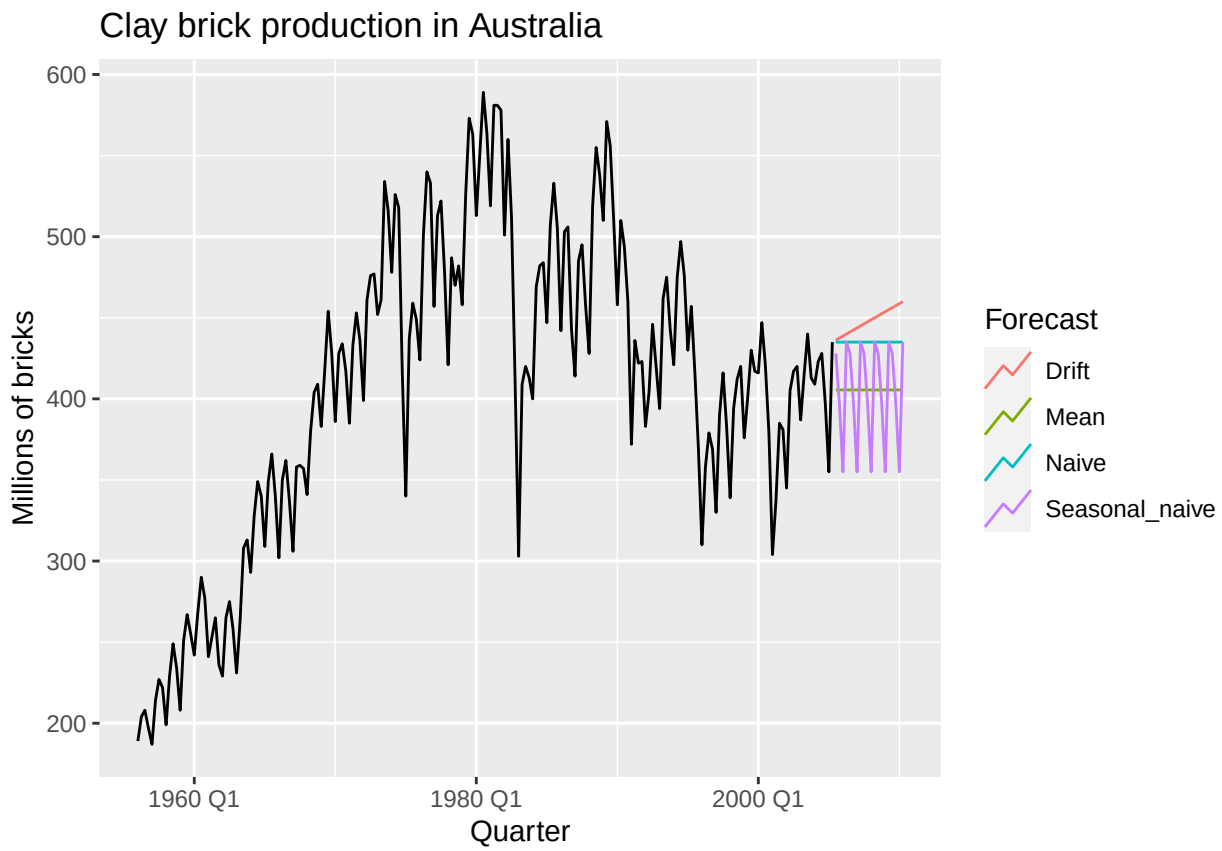
```
brick_fc
```

```
## # A fable: 80 x 4 [1Q]
## # Key:     .model [4]
##    .model          Quarter       Bricks .mean
##    <chr>             <qtr>       <dist> <dbl>
##  1 Seasonal_naive 2005 Q3 N(428, 2336)   428
##  2 Seasonal_naive 2005 Q4 N(397, 2336)   397
##  3 Seasonal_naive 2006 Q1 N(355, 2336)   355
##  4 Seasonal_naive 2006 Q2 N(435, 2336)   435
##  5 Seasonal_naive 2006 Q3 N(428, 4672)   428
##  6 Seasonal_naive 2006 Q4 N(397, 4672)   397
##  7 Seasonal_naive 2007 Q1 N(355, 4672)   355
##  8 Seasonal_naive 2007 Q2 N(435, 4672)   435
##  9 Seasonal_naive 2007 Q3 N(428, 7008)   428
## 10 Seasonal_naive 2007 Q4 N(397, 7008)   397
## # i 70 more rows
```

Visualizing Forecasts

```
brick_fc |>
  autoplot(aus_production, level = NULL)+
  labs(title = "Clay brick production in Australia",
       y="Millions of bricks")+
  guides(colour=guide_legend(title="Forecast"))
```

```
## Warning: Removed 20 rows containing missing values (`geom_line()`).
```



Here, by specifying "level=NULL", we mean don't plot the prediction intervals. By default it plots 80%,95% prediction intervals. Another argument is historical data i.e. aus_production here to plot the historical data as well.

Here, in this case the plausible forecast is probably the seasonal naive forecast out all 4 benchmarking methods.

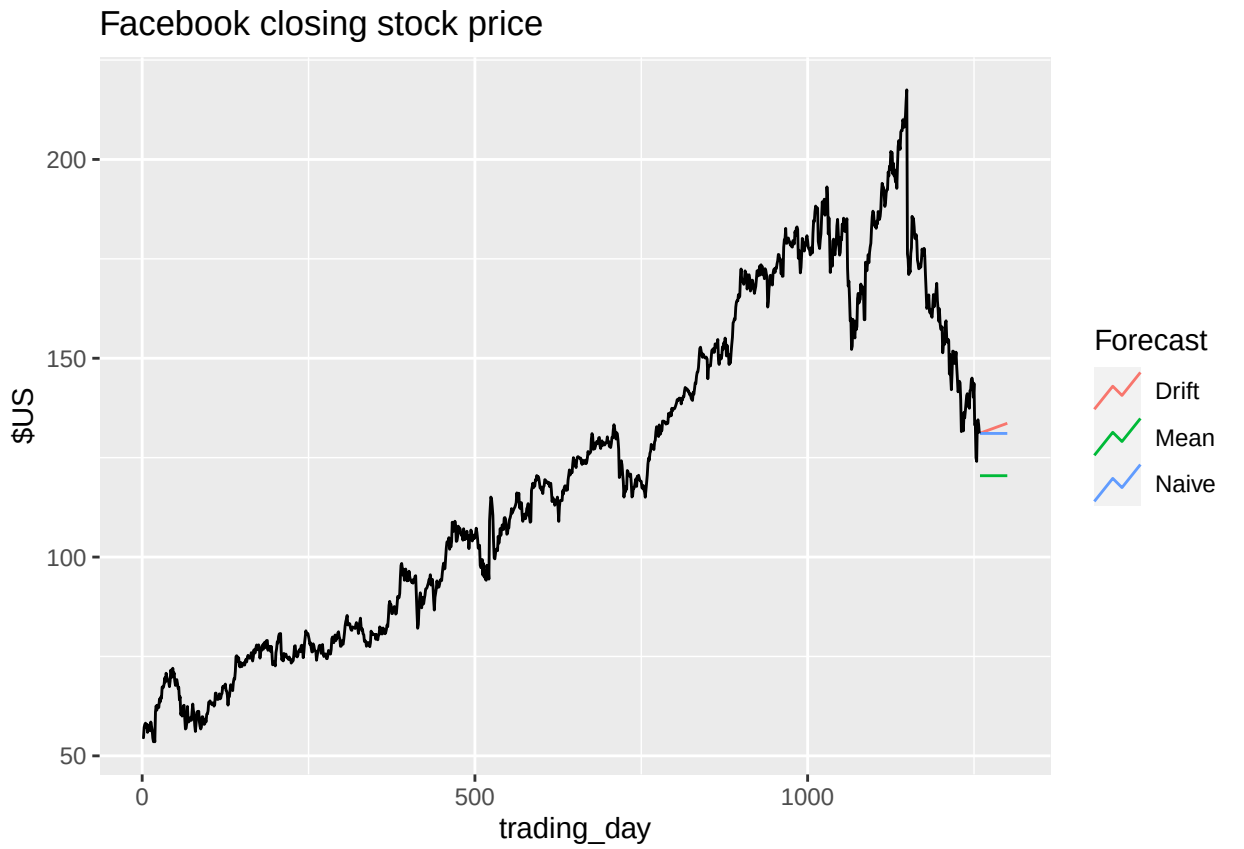
We are interested in more sophisticated forecast which is better than all 4 benchmarking forecasts.

Facebook closing stock price

```
# Extract training data
fb_stock <- gafa_stock |>
  filter(Symbol=="FB") |>
  mutate(trading_day=row_number()) |>
  update_tsibble(index = trading_day,regular = TRUE)

# Specify, estimate and forecast
fb_stock |>
  model(
  Mean=MEAN(Close),
  Naive=NAIVE(Close),
  Drift=RW(Close ~ drift())
) |>
  forecast(h=42) |>
  autoplot(fb_stock,level=NULL)+
  labs(title = "Facebook closing stock price",y="$US")+
```

```
  guides(colour=guide_legend(title="Forecast"))
```



Here trading day represents when we got trading in the data. For example, if on monday, trading does not happened then it will not be showing. We have not fitted the seasonal naive because seasonality is not presented in the data. h=42 says that it forecasts for next 42 trading days ahead.

Here, MEAN() is not a very sensible model for stock prices because they wander around a lot. The NAIVE() and DRIFT() methods are not too bad here becasue they start from very recent observation.

Summary

Some forecasting methods are extremely simple and surprisingly effective. We will use four simple forecasting methods as benchmarks throughout this book.

To illustrate them, we will use quarterly Australian clay brick production between 1970 and 2004

```
bricks <- aus_production |>
  filter_index("1970 Q1" ~ "2004 Q4") |>
  select(Bricks)
bricks
```

```
## # A tsibble: 140 x 2 [1Q]
##    Bricks Quarter
##     <dbl>   <qtr>
##  1    386 1970 Q1
##  2    428 1970 Q2
##  3    434 1970 Q3
##  4    417 1970 Q4
##  5    385 1971 Q1
```

```
##  6    433 1971 Q2
##  7    453 1971 Q3
##  8    436 1971 Q4
##  9    399 1972 Q1
## 10    461 1972 Q2
## # i 130 more rows
```

The filter_index() function is a convenient shorthand for extracting a section of a time series.

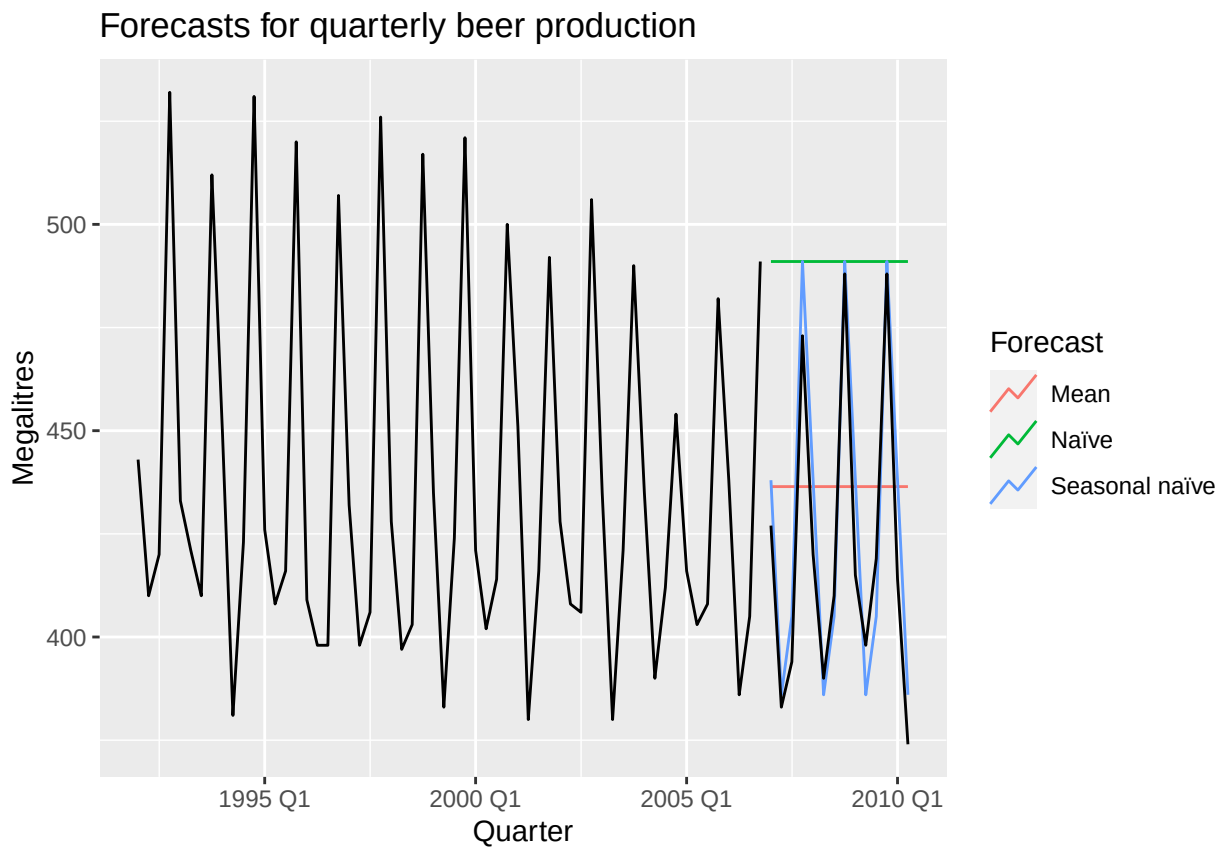
a naïve forecast is optimal when data follow a random walk (see Section 9.1), these are also called random walk forecasts and the RW() function can be used instead of NAIVE.

Example: Australian quarterly beer production

Figure 5.7 shows the first three methods applied to Australian quarterly beer production from 1992 to 2006, with the forecasts compared against actual values in the next 3.5 years.

```
# Set training data from 1992 to 2006
train <- aus_production |>
  filter_index("1992 Q1" ~ "2006 Q4")
# Fit the models
beer_fit <- train |>
  model(
    Mean = MEAN(Beer),
    `Naïve` = NAIVE(Beer),
    `Seasonal naïve` = SNAIVE(Beer)
  )
# Generate forecasts for 14 quarters
beer_fc <- beer_fit |> forecast(h = 14)
# Plot forecasts against actual values
beer_fc |>
  autoplot(train, level = NULL) +
  autolayer(
    filter_index(aus_production, "2007 Q1" ~ .),
    colour = "black"
  ) +
  labs(
    y = "Megalitres",
    title = "Forecasts for quarterly beer production"
  ) +
  guides(colour = guide_legend(title = "Forecast"))
```

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



In this case, only the seasonal naïve forecasts are close to the observed values from 2007 onwards.

Example: Google's daily closing stock price

In Figure 5.8, the non-seasonal methods are applied to Google's daily closing stock price in 2015, and used to forecast one month ahead. Because stock prices are not observed every day, we first set up a new time index based on the trading days rather than calendar days.

```
# Re-index based on trading days
google_stock <- gafa_stock |>
  filter(Symbol == "GOOG", year(Date) >= 2015) |>
  mutate(day = row_number()) |>
  update_tsibble(index = day, regular = TRUE)
# Filter the year of interest
google_2015 <- google_stock |> filter(year(Date) == 2015)
# Fit the models
google_fit <- google_2015 |>
  model(
    Mean = MEAN(Close),
    `Naïve` = NAIVE(Close),
    Drift = NAIVE(Close ~ drift())
  )
# Produce forecasts for the trading days in January 2016
google_jan_2016 <- google_stock |>
  filter(yearmonth(Date) == yearmonth("2016 Jan"))
google_fc <- google_fit |>
  forecast(new_data = google_jan_2016)
```

```r
# Plot the forecasts
google_fc |>
  autoplot(google_2015, level = NULL) +
  autolayer(google_jan_2016, Close, colour = "black") +
  labs(y = "$US",
       title = "Google daily closing stock prices",
       subtitle = "(Jan 2015 - Jan 2016)") +
  guides(colour = guide_legend(title = "Forecast"))
```

Sometimes one of these simple methods will be the best forecasting method available; but in many cases, these methods will serve as benchmarks rather than the method of choice. That is, any forecasting methods we develop will be compared to these simple methods to ensure that the new method is better than these simple alternatives. If not, the new method is not worth considering.

5.3 Fitted values and residuals

- $\hat{y}_{t|t-1}$ is the forecast of y_t based on the observations $y_1, ..., y_{t-1}$. It is expected value of the future value of y_t given everything all the information upto the period before $t-1$.
- We call these values as fitted values.
- Sometimes we drop the subscript: $\hat{y}_t \equiv \hat{y}_{t|t-1}$
- We call these values as fitted values and not forecasts because Often not true forecasts since many of the methods we use to generate these, we actually estimate the parameters are estimated using all the information on all data.
- *For example:*

(i) $\hat{y}_t = \overline{y}$ for average method.

(ii) $\hat{y}_t = y_{t-1} + \frac{y_T - y_1}{T-1}$ for drift method.

Examples

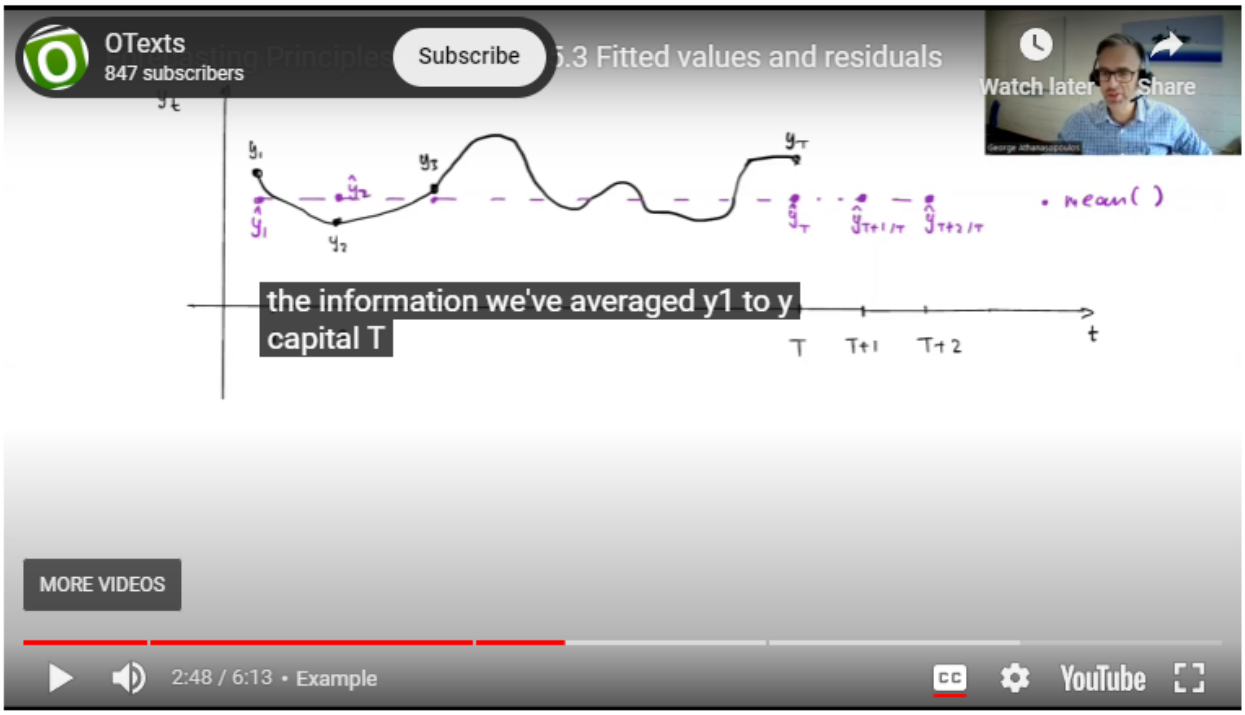


Figure 3: Mean Method

Here, we take the average from y_1 to y_T and all the fitted values lie on this average line. This is not really a true forecast.

Here, the fitted values are $\hat{y_{2|1}}$,$\hat{y_{3|2}}$, $\hat{y_{4|3}}$ etc.

Residuals in Forecasting

It is the difference between the observed value and its fitted value:

$$e_t = y_t - \hat{y}_{t|t-1}$$

Assumptions:

- Residuals e_t are uncorrelated. If they are not then then information left in residuals that should be used in computing forecasts.
- e_t have mean zero. If they don't then forecasts are biased.

Useful Properties (For distribution and prediction intervals):

- e_t have constant variance hence they are homoscedestic.
- e_t are normally distributed.

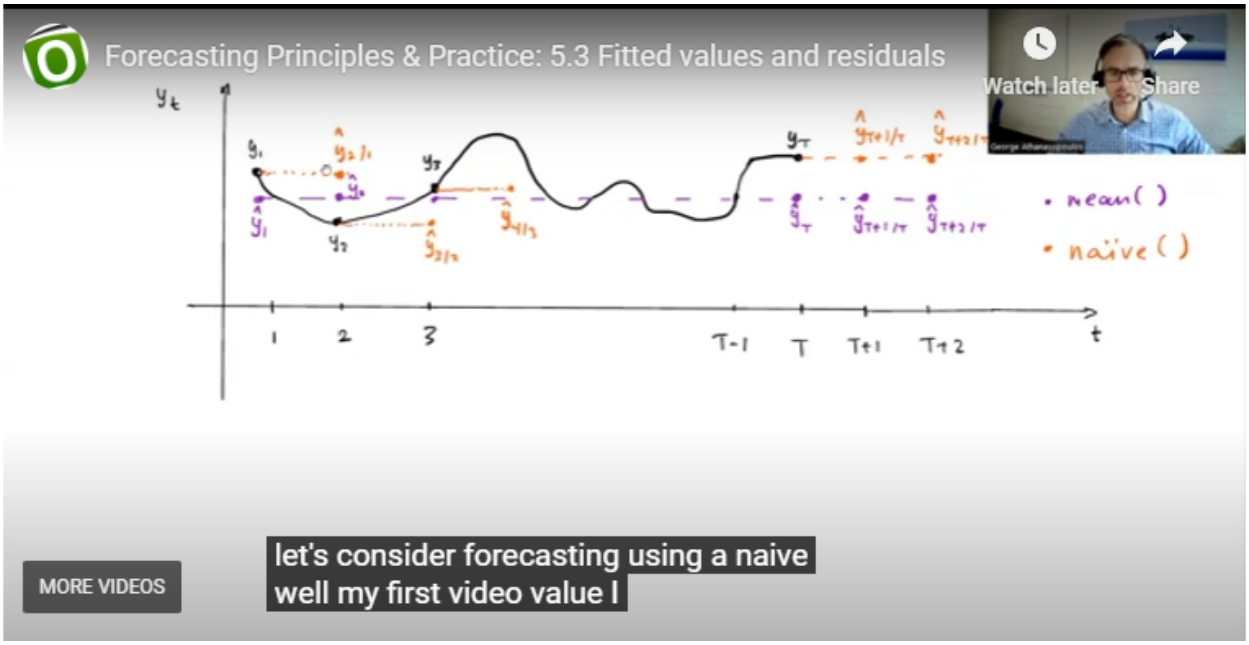


Figure 4: Naive Method

The fitted values and residuals from a model can be obtained using the augment() function. In the beer production example in Section 5.2, we saved the fitted models as beer_fit. So we can simply apply `augment()` to this object to compute the fitted values and residuals for all models.

```
augment(beer_fit)
```

```
## # A tsibble: 180 x 6 [1Q]
## # Key:       .model [3]
##    .model Quarter  Beer .fitted .resid .innov
##    <chr>    <qtr> <dbl>   <dbl>  <dbl>  <dbl>
##  1 Mean   1992 Q1   443    436.   6.55   6.55
##  2 Mean   1992 Q2   410    436. -26.4  -26.4
##  3 Mean   1992 Q3   420    436. -16.4  -16.4
##  4 Mean   1992 Q4   532    436.  95.6   95.6
##  5 Mean   1993 Q1   433    436.  -3.45  -3.45
##  6 Mean   1993 Q2   421    436. -15.4  -15.4
##  7 Mean   1993 Q3   410    436. -26.4  -26.4
##  8 Mean   1993 Q4   512    436.  75.6   75.6
##  9 Mean   1994 Q1   449    436.  12.6   12.6
## 10 Mean   1994 Q2   381    436. -55.4  -55.4
## # i 170 more rows
```

There are three new columns added to the original data:

- fitted contains the fitted values;
- resid contains the residuals;
- innov contains the "innovation residuals" which, in this case, are identical to the regular residuals.

Residuals are useful in checking whether a model has adequately captured the information in the data. For this purpose, we use innovation residuals.

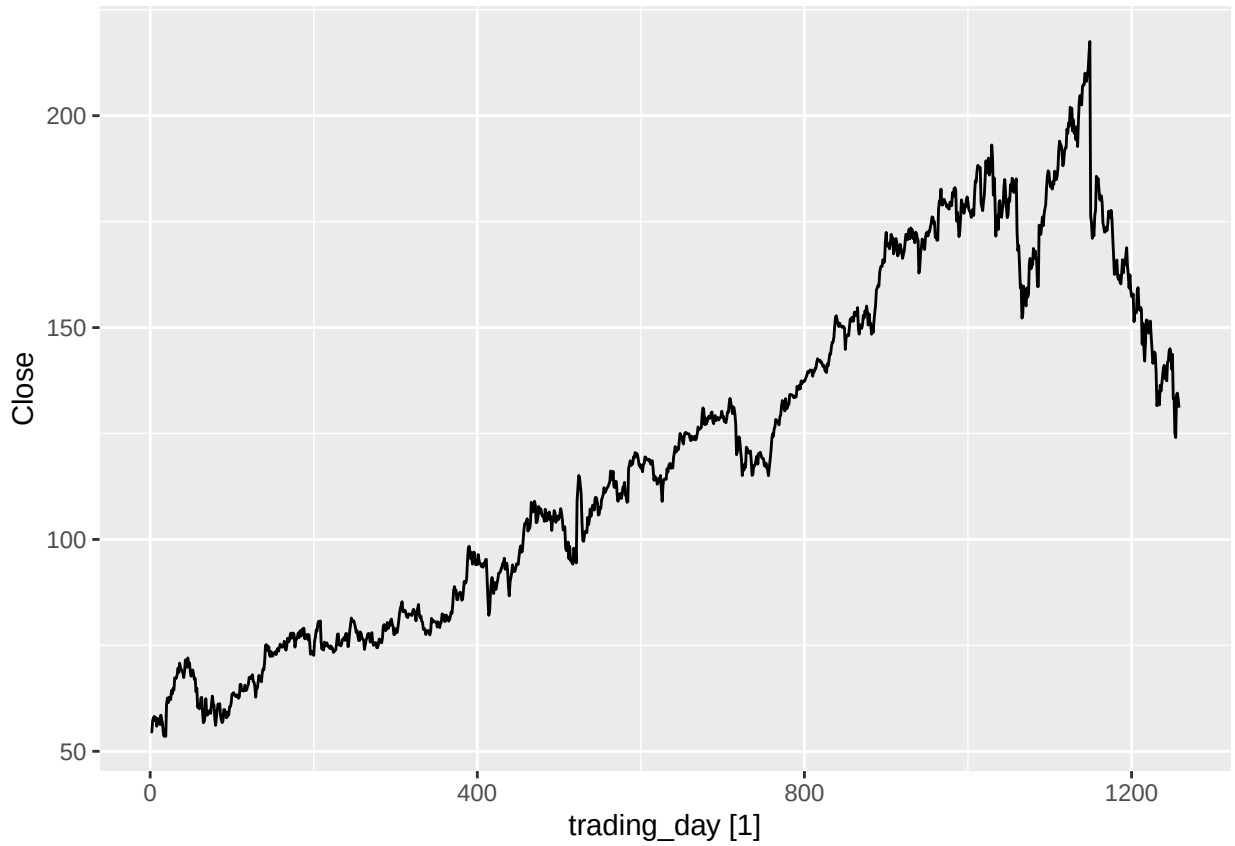
If patterns are observable in the innovation residuals, the model can probably be improved. We will look at

some tools for exploring patterns in residuals in the next section.

5.4 Residual diagnostics

Example: Facebook closing stock price

```
fb_stock |> autoplot(Close)
```



Here, we are going to fit a model and check the residuals and look at the residuals of the model and check whether they satisfy the assumptions which we made.

```
fit <- fb_stock |> model(NAIVE(Close))
augment(fit)
```

```
## # A tsibble: 1,258 x 7 [1]
## # Key:       Symbol, .model [1]
##    Symbol .model       trading_day Close .fitted .resid .innov
##    <chr>  <chr>              <int> <dbl>   <dbl>  <dbl>  <dbl>
##  1 FB     NAIVE(Close)           1  54.7    NA    NA     NA
##  2 FB     NAIVE(Close)           2  54.6    54.7 -0.150 -0.150
##  3 FB     NAIVE(Close)           3  57.2    54.6  2.64   2.64
##  4 FB     NAIVE(Close)           4  57.9    57.2  0.720  0.720
##  5 FB     NAIVE(Close)           5  58.2    57.9  0.310  0.310
##  6 FB     NAIVE(Close)           6  57.2    58.2 -1.01  -1.01
##  7 FB     NAIVE(Close)           7  57.9    57.2  0.720  0.720
##  8 FB     NAIVE(Close)           8  55.9    57.9 -2.03  -2.03
##  9 FB     NAIVE(Close)           9  57.7    55.9  1.83   1.83
```

```
## 10 FB     NAIVE(Close)          10  57.6    57.7 -0.140 -0.140
## # i 1,248 more rows
```

Here we are using the NAIVE model. It is very common model to use for the stock price data. So, here we got the fitted and residuals values and innov values.

When we are testing then we use the .innov values column. Here, we have residual and innov column have the same values but but for different for certain types of models or we have done a transformation of the data before we fit a model. In this case we have not done any transformation and we are using a straight forward additive model so two columns are same.

Here first value is missing because we can't get the forecast for the forecast using the NAIVE method because we don't know what is happening before that observation.

Here, fitted value means $\hat{y}_{t|t-1} = y_{t-1}$ and residual means $e_t == y_t - \hat{y}_{t|t-1} = y_t - y_{t-1}$. We can check in the above table.

Now first we plot the observations.

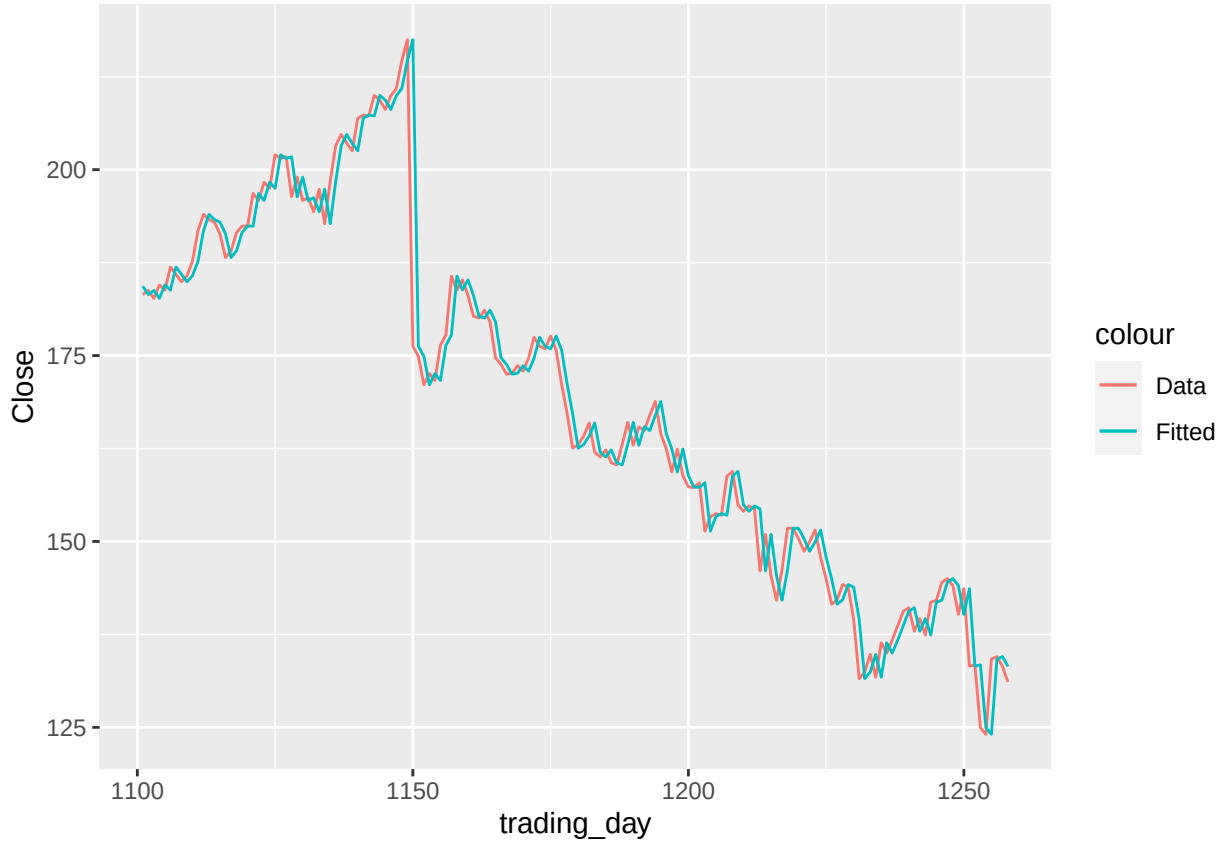
```
augment(fit) |>
  ggplot(aes(x=trading_day))+
  geom_line(aes(y=Close, colour="Data"))+
  geom_line(aes(y=.fitted, colour="Fitted"))
```

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

They are very very close because they does not change day to day and so the observation yesterday is quite similar to the observation today and so these two plots are very similar.

If we take the vertical differences.

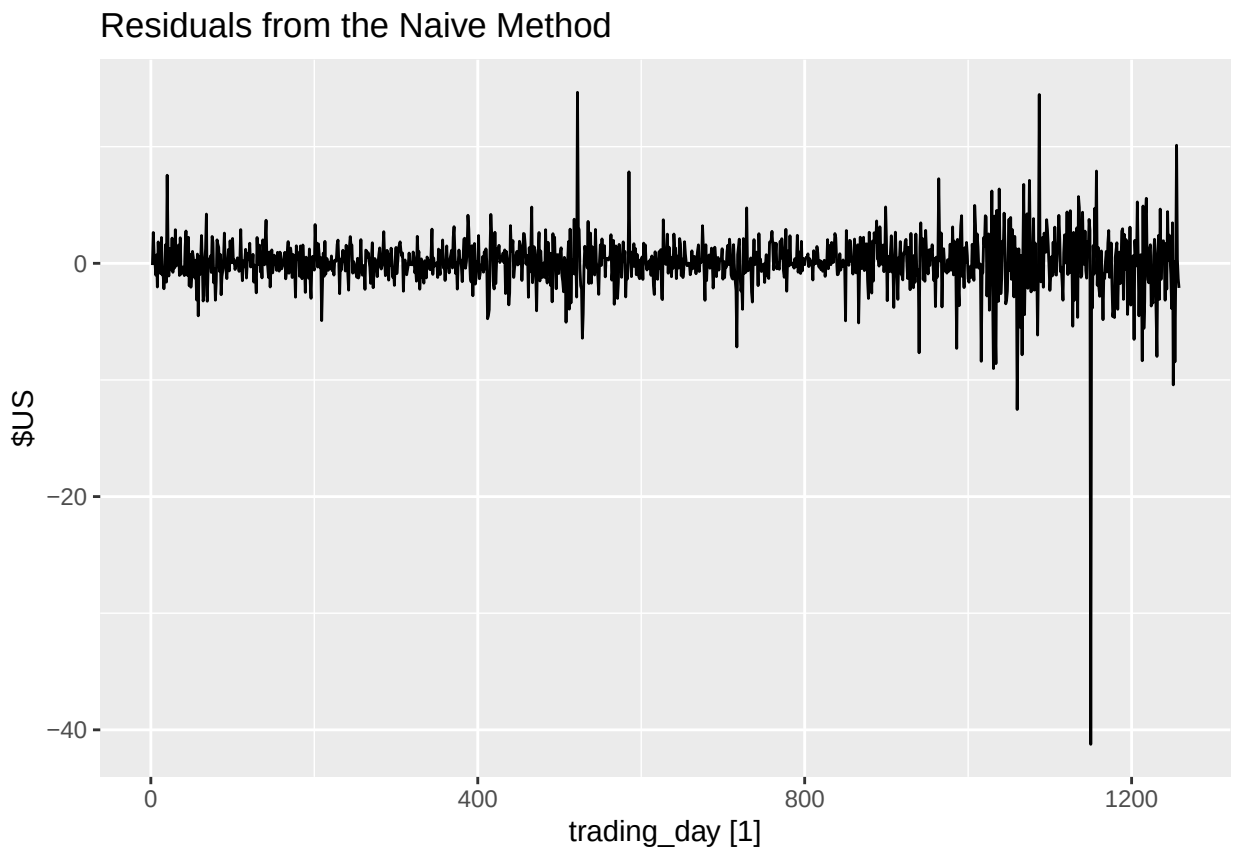
```
augment(fit) |>
  filter(trading_day>1100) |>
  ggplot(aes(x=trading_day))+
  geom_line(aes(y=Close, colour="Data"))+
  geom_line(aes(y=.fitted, colour="Fitted"))
```



Now, check the residuals.

```
augment(fit) |>
  autoplot(.resid)+
  labs(y="$US",
       title="Residuals from the Naive Method")
```

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

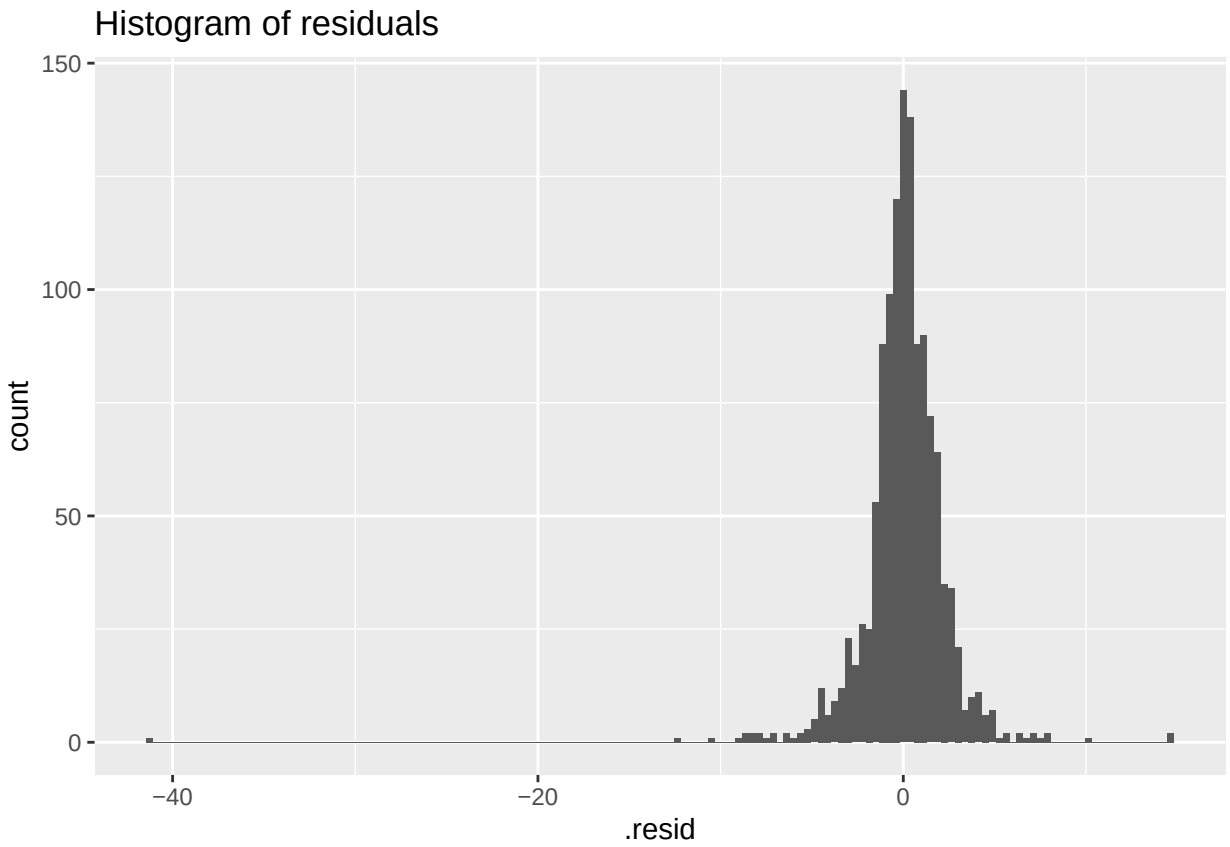


Here, we are getting spikes for large values in trading days and all these things suggest that our assumptions are not right here.

Now, make the histogram from the residuals.

```
augment(fit) |>
  ggplot(aes(x=.resid)) +
  geom_histogram(bins=150)+
  labs(title = "Histogram of residuals")
```

```
## Warning: Removed 1 rows containing non-finite values (`stat_bin()`).
```

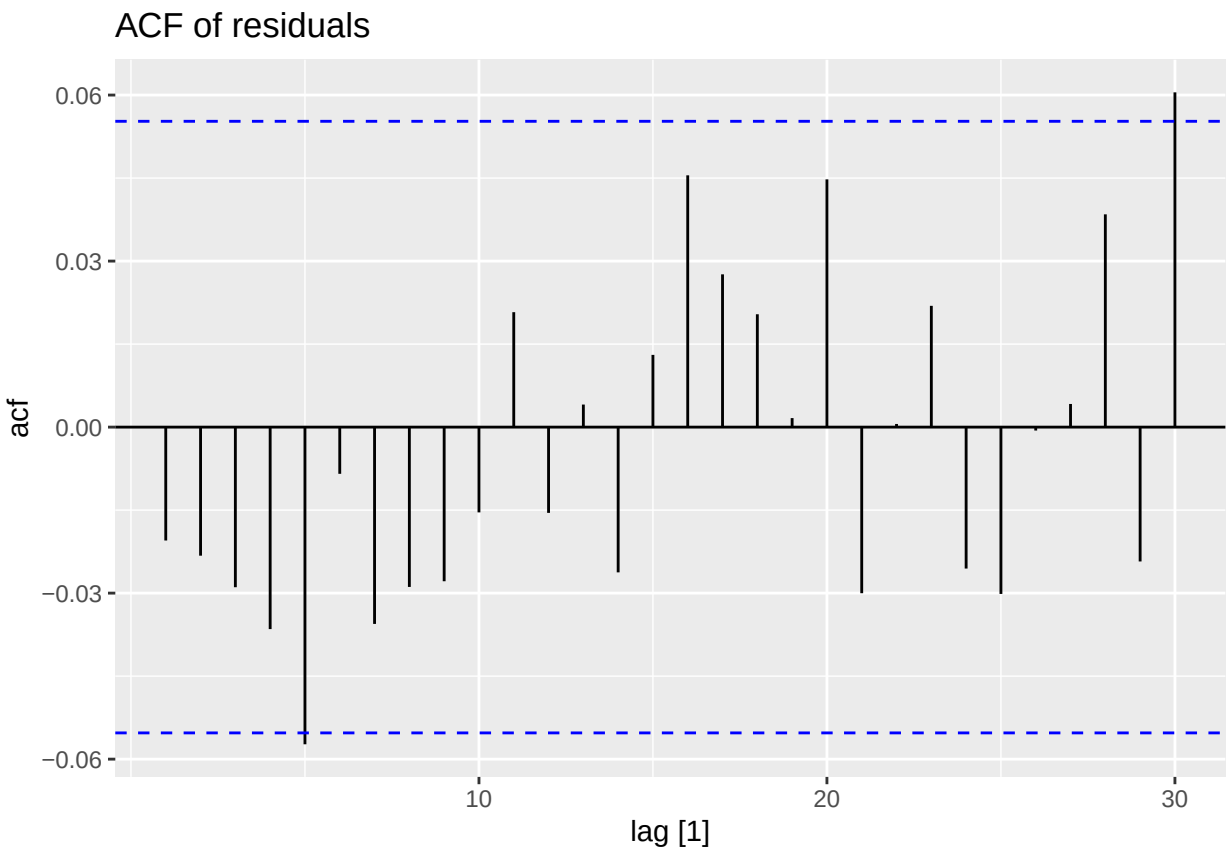


bins is the number of bars in the histogram. if we don't give the value to it then it takes the default value.

These residuals looks normal but it is not normal because it is having outliers and also it is little more peaked than what we should we expect.

Now, we look at the acf of the residuals.

```
augment(fit) |>
  ACF(.resid) |>
  autoplot()+labs(title = "ACF of residuals")
```

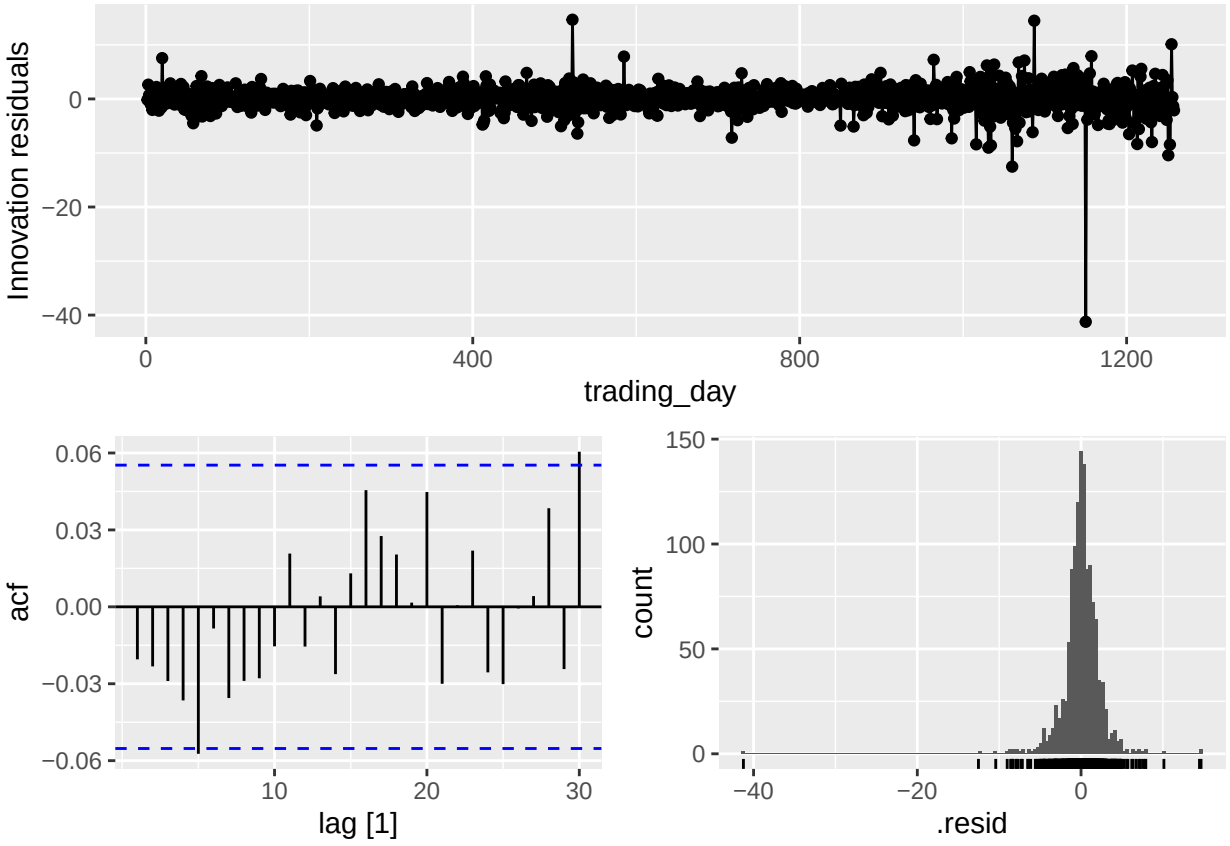


Here, we get two spikes above the limit out of 30 spikes which is not unusual.

Now, we can plot all the graphs as:

```
gg_tsresiduals(fit)
```

```
## Warning: Removed 1 row containing missing values (`geom_line()`).
## Warning: Removed 1 rows containing missing values (`geom_point()`).
## Warning: Removed 1 rows containing non-finite values (`stat_bin()`).
```



ACF of residuals

- We assume that residuals are white noise (uncorrelated, mean zero and constant variance). If they are not then there is information left in the residuals that should be used in computing forecasts and make the forecasts better.
- So a standard residual diagnostic is to check the ACF of the residuals of a forecasting method. Using autocorrelation, we are also checking whether lag 1 is significant or lag 2 is significant or lag 3 is significant etc.
- We `expect` these to look like white noise.

Portmanteau tests

The word Portmanteau refers to a suitcase which carries everything together. So this test combines a lot of things. So this test do all the correlations simultaneously.

r_k = autocorrelation of residuals at lag k

Consider a whole set of r_k values, and develop a test to see whether the set is significantly different from a zero set.

(1) Box-Pierce test

- It is developed by George Box (same in box-cot transformation) and his PhD student Pierce.
- The idea is if we squared all the correlations up to some upper limit say lag "l". If we multiply it by the number of observations then we get a thing that has a chi-squared distribution.

- We can compare it against the probabilities that we would expect from the chi-squared distribution to see whether the value of Q for our data set is bigger than what we would expect by chance. If it is a small Q then the sort of things which we expect by chance and so we can say well the test is insignificant. If it is a large Q bigger than what we would expect by chance then we say there is a significant autocorrelation is going on here. So Summary is:

$$Q = T\Sigma_{k=1}^{l} r_k^2$$

where l is the max lag being considered and T is the number of observations.

- If each r_k close to zero then Q will be *small*.
- If some r_k values large (positive or negative), Q will be *large*.

After this test, Box with his another PhD student comes with new test which gives little more accurate result and that's the one we will use.

Ljung-Box test

$$Q^* = T(T+2)\Sigma_{k=1}^{l}(T-k)^{-1} r_k^2$$

where l is the max lag being considered and T is the number of observations.

- My preference (based on experiments): $l = 10$ for non-seasonal data, $l = 2m$ for seasonal data (where m is seasonal period, so for quarterly data, m=4 and for monthly data, m=12)
- Better performance, especially in small samples.
- Here, $(T+k)^{-1}$ is a weight and this test has value which also has a small chi-squared distribution.
- If the data or residuals are white noise (WN) then Q^* has χ^2 distribution with l degrees of freedom and we compute the probabilities of getting a value as large under the assumption of white noise residuals.
- lag=l

```
augment(fit) |>
  features(.resid, ljung_box,lag=10)
```

```
## # A tibble: 1 x 4
##   Symbol .model       lb_stat lb_pvalue
##   <chr>  <chr>          <dbl>     <dbl>
## 1 FB     NAIVE(Close)    12.1     0.276
```

Since it is non-seasonal data so we are using lag=10.

- Here, FB is the key.
- Here, "lb_stat" is Q^* and "lb_pvalue" is a probability of gettting a Q^* as bis as this if the residuals were really white noise and so we reject the hypothesis of white noise if this p-value is less than 0.05 (a arbitrary threshold) that represents very unlikely to occur by chance but here p-value is 0.276 so very likely to occur by chance so these residuals are not easily distinguished from white noise so we accept the white noise hypothesis.

So, when we fit a model and we are ready to do some forecasting then it is a good idea to check your residuals to make sure it satisfies the assumptions. In this case, autocorrelation is satisfied but not the normality of the residuals and that will affect the size of the prediction intervals much more than the point forecasts.

- Summary

A good forecasting method will yield innovation residuals with the following properties:

(1) The innovation residuals are uncorrelated. If there are correlations between innovation residuals, then there is information left in the residuals which should be used in computing forecasts.

(2) The innovation residuals have zero mean. If they have a mean other than zero, then the forecasts are biased.

Any forecasting method that does not satisfy these properties can be improved. However, that does not mean that forecasting methods that satisfy these properties cannot be improved. It is possible to have several different forecasting methods for the same data set, all of which satisfy these properties. Checking these properties is important in order to see whether a method is using all of the available information, but it is not a good way to select a forecasting method.

If either of these properties is not satisfied, then the forecasting method can be modified to give better forecasts. Adjusting for bias is easy: if the residuals have mean m, then simply add m to all forecasts and the bias problem is solved. Fixing the correlation problem is harder, and we will not address it until Chapter 10.

In addition to these essential properties, it is useful (but not necessary) for the residuals to also have the following two properties.

(3) The innovation residuals have constant variance. This is known as "homoscedasticity".

(4) The innovation residuals are normally distributed.

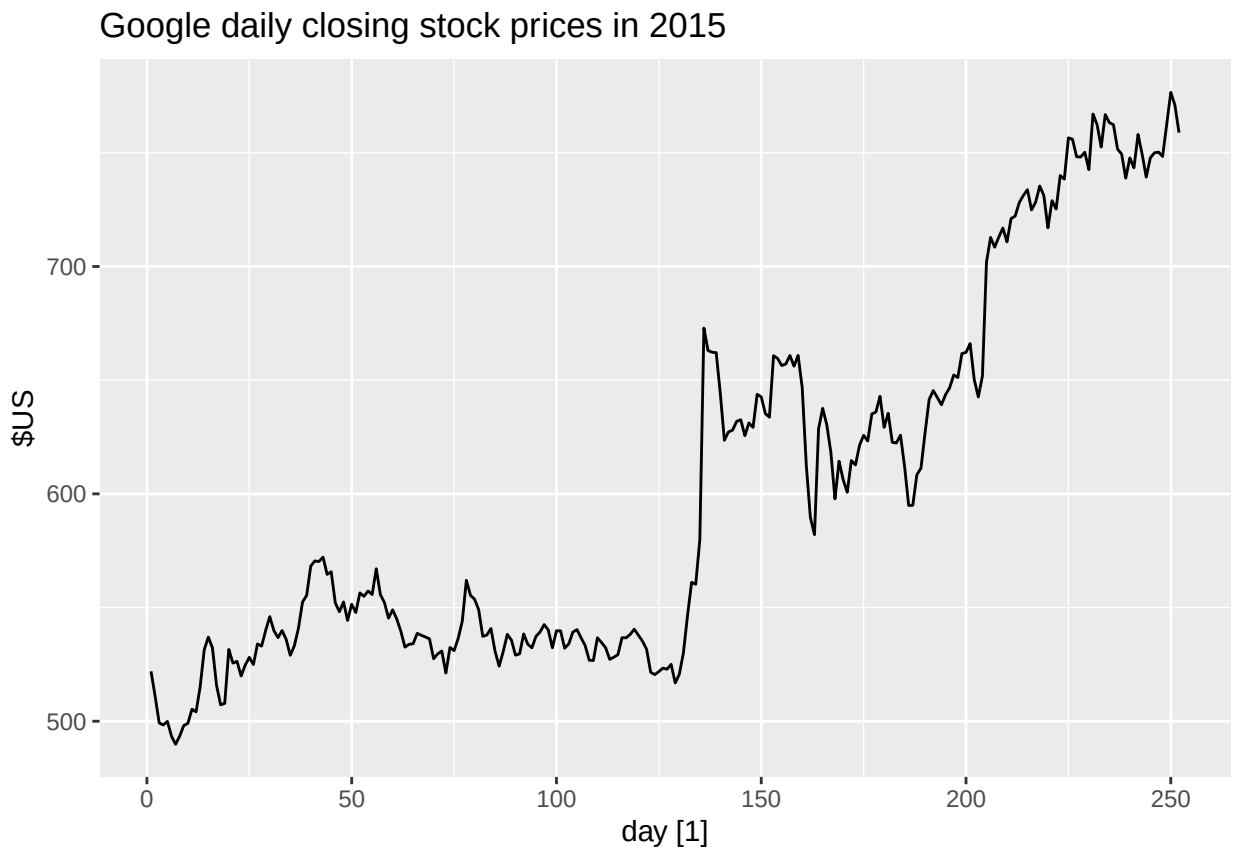
These two properties make the calculation of prediction intervals easier (see Section 5.5 for an example). However, a forecasting method that does not satisfy these properties cannot necessarily be improved. Sometimes applying a Box-Cox transformation may assist with these properties, but otherwise there is usually little that you can do to ensure that your innovation residuals have constant variance and a normal distribution. Instead, an alternative approach to obtaining prediction intervals is necessary. We will show how to deal with non-normal innovation residuals in Section 5.5.

Example: Forecasting Google daily closing stock prices

We will continue with the Google daily closing stock price example from Section 5.2. For stock market prices and indexes, the best forecasting method is often the naïve method. That is, each forecast is simply equal to the last observed value, or $\hat{y}_t = y_{t-1}$. Hence, the residuals are simply equal to the difference between consecutive observations:

The following graph shows the Google daily closing stock price for trading days during 2015. The large jump corresponds to 17 July 2015 when the price jumped 16% due to unexpectedly strong second quarter results.

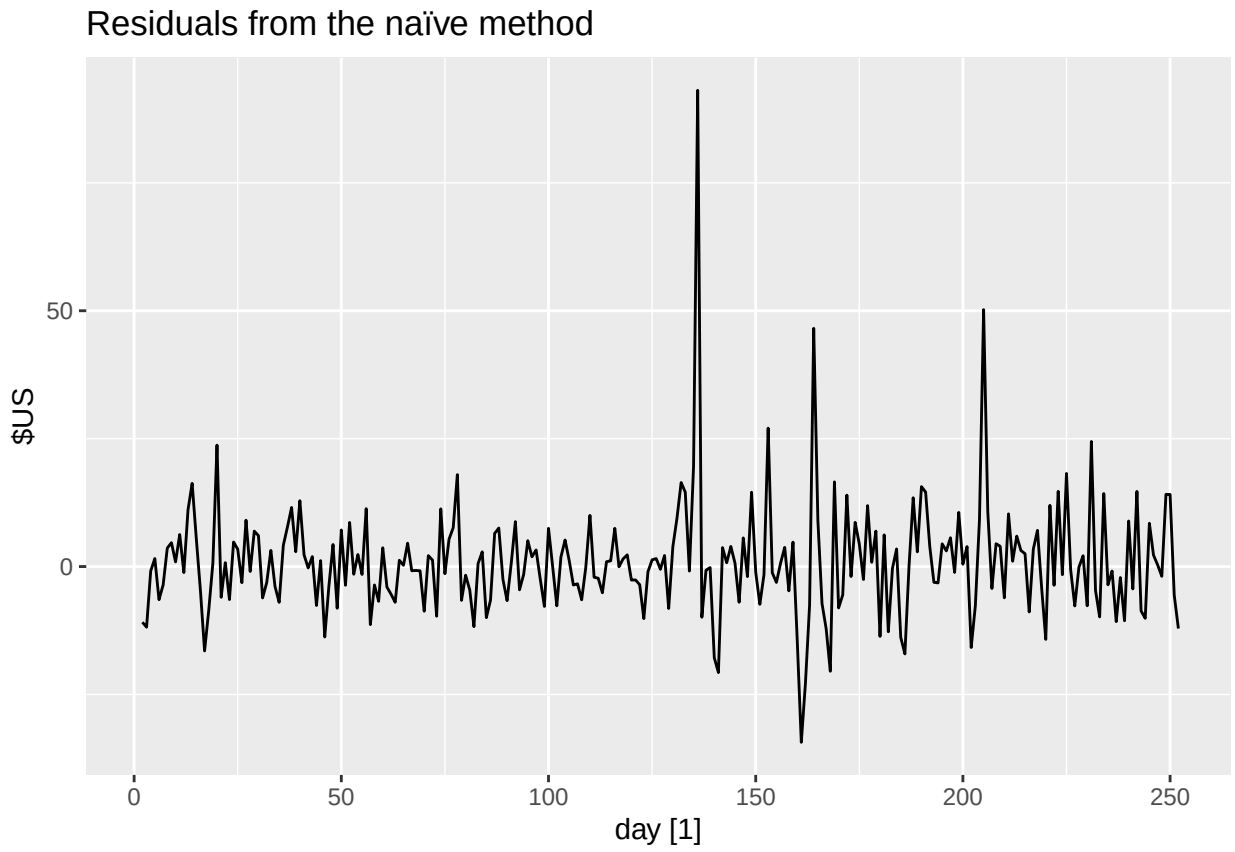
```
autoplot(google_2015, Close) +
  labs(y = "$US",
       title = "Google daily closing stock prices in 2015")
```



The residuals obtained from forecasting this series using the naïve method are shown in Figure 5.10. The large positive residual is a result of the unexpected price jump in July.

```
aug <- google_2015 |>
  model(NAIVE(Close)) |>
  augment()
autoplot(aug, .innov) +
  labs(y = "$US",
       title = "Residuals from the naïve method")
```

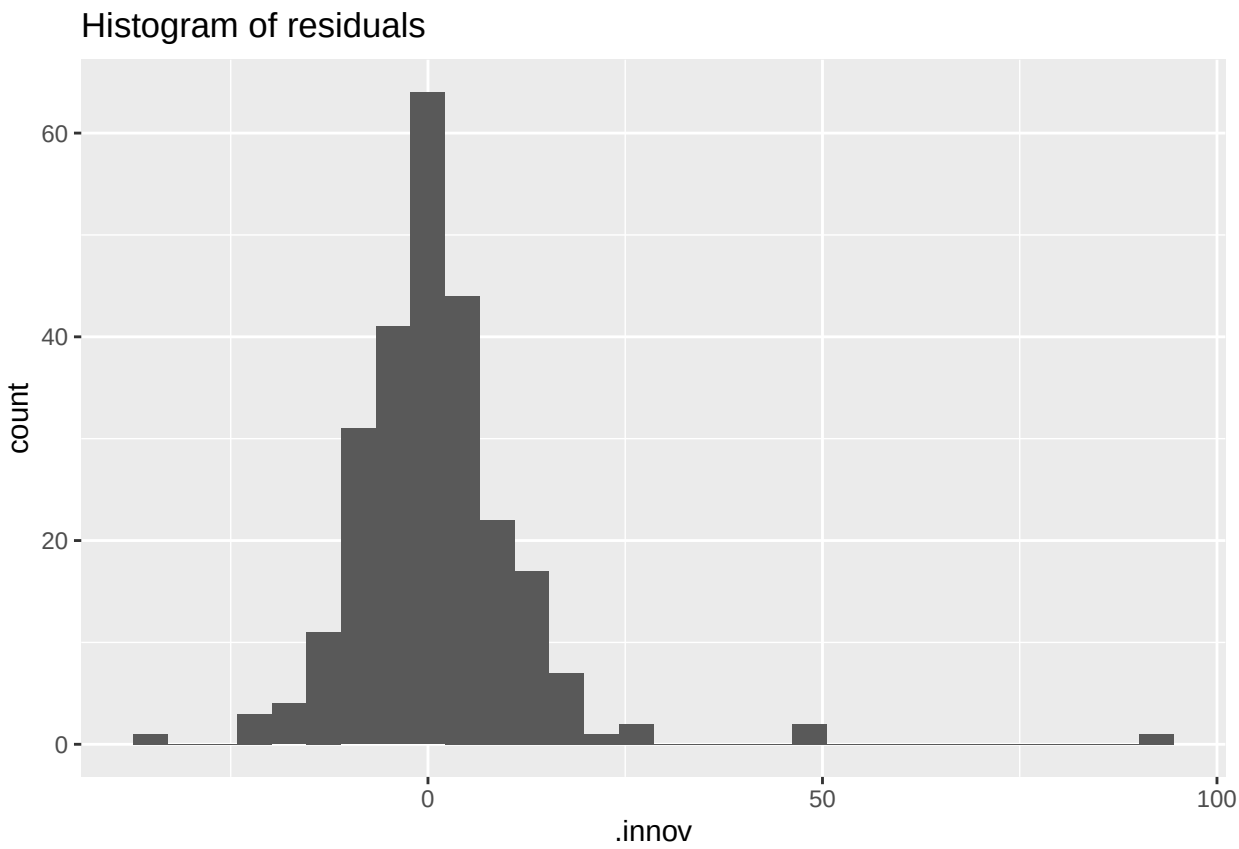
```
## Warning: Removed 1 row containing missing values (`geom_line()`).
```



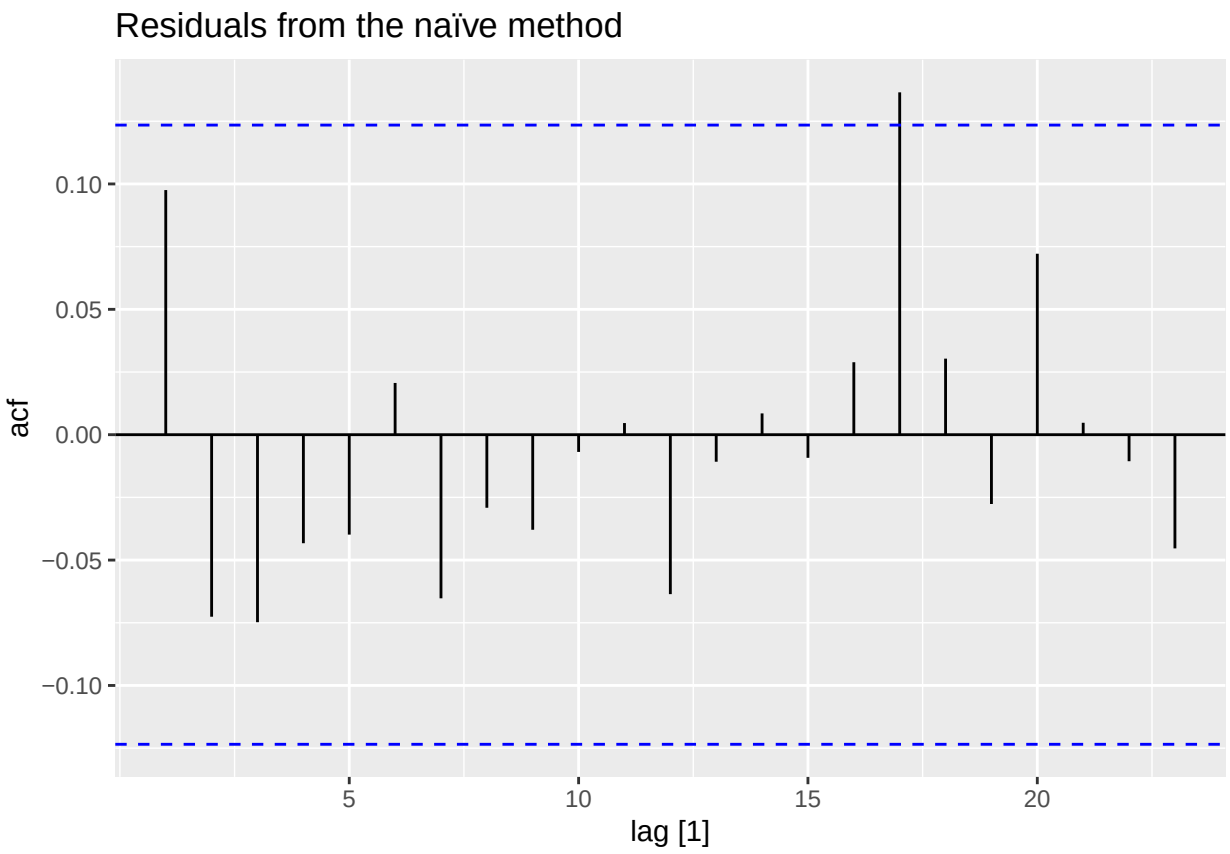

```
aug |>
  ggplot(aes(x = .innov)) +
  geom_histogram() +
  labs(title = "Histogram of residuals")
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

```
## Warning: Removed 1 rows containing non-finite values (`stat_bin()`).
```




```
aug |>
  ACF(.innov) |>
  autoplot() +
  labs(title = "Residuals from the naïve method")
```



These graphs show that the naïve method produces forecasts that appear to account for all available information. The mean of the residuals is close to zero and there is no significant correlation in the residuals series. The time plot of the residuals shows that the variation of the residuals stays much the same across the historical data, apart from the one outlier, and therefore the residual variance can be treated as constant. This can also be seen on the histogram of the residuals. The histogram suggests that the residuals may not be normal — the right tail seems a little too long, even when we ignore the outlier. Consequently, forecasts from this method will probably be quite good, but prediction intervals that are computed assuming a normal distribution may be inaccurate.

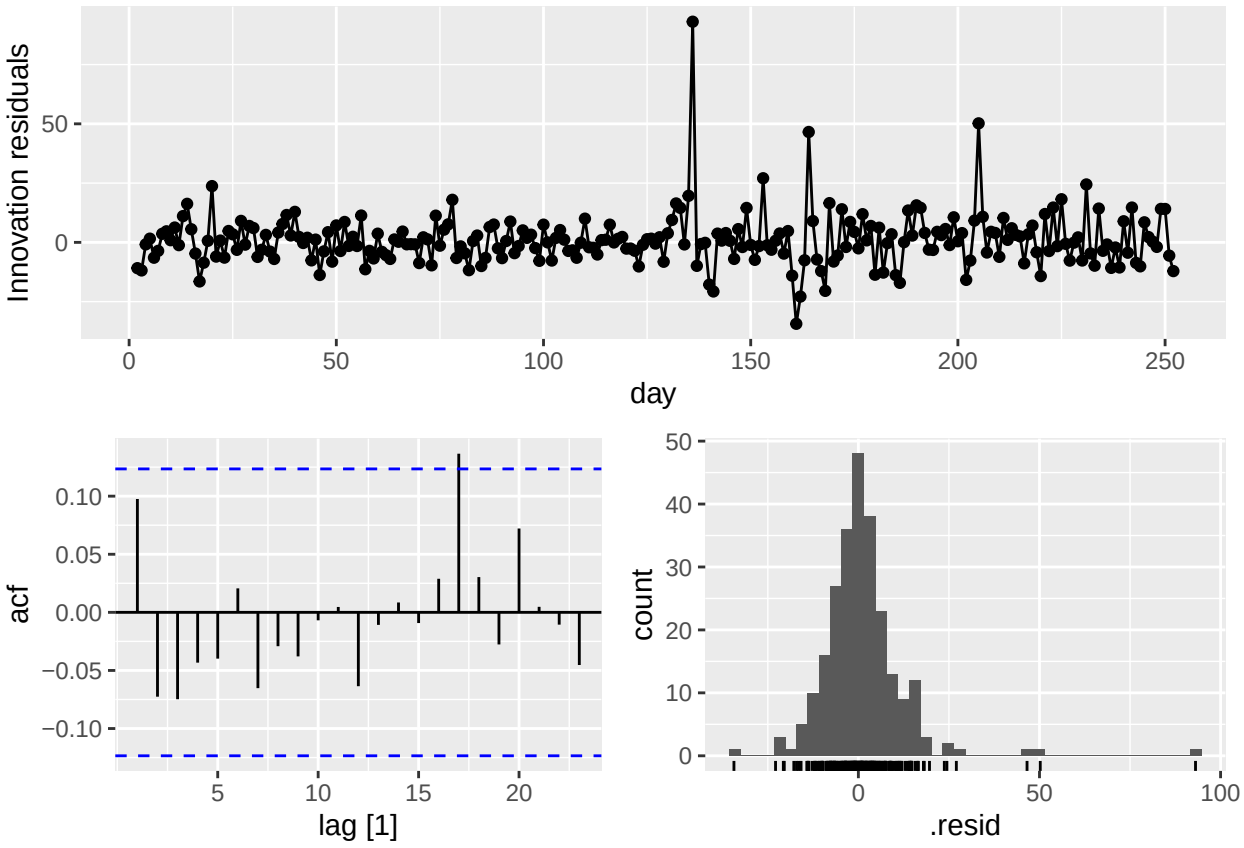
A convenient shortcut for producing these residual diagnostic graphs is the gg_tsresiduals() function, which will produce a time plot, ACF plot and histogram of the residuals.

```
google_2015 |>
  model(NAIVE(Close)) |>
  gg_tsresiduals()
```

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

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

```
## Warning: Removed 1 rows containing non-finite values (`stat_bin()`).
```



Portmanteau tests for autocorrelation

In addition to looking at the ACF plot, we can also do a more formal test for autocorrelation by considering a whole set of r_k values as a group, rather than treating each one separately.

Recall that r_k is the autocorrelation for lag k

When we look at the ACF plot to see whether each spike is within the required limits, we are implicitly carrying out multiple hypothesis tests, each one with a small probability of giving a false positive. When enough of these tests are done, it is likely that at least one will give a false positive, and so we may conclude that the residuals have some remaining autocorrelation, when in fact they do not.

In order to overcome this problem, we test whether the first l autocorrelations are significantly different from what would be expected from a white noise process. A test for a group of autocorrelations is called a portmanteau test, from a French word describing a suitcase or coat rack carrying several items of clothing.

One such test is the Box-Pierce test

If each r_k is close to zero, then Q will be small. If some r_k values are large (positive or negative), then Q will be large. We suggest using $l = 10$ for non-seasonal data and $l = 2m$ for seasonal data, where m is the period of seasonality. However, the test is not good when l is large, so if these values are larger than $T/5$, then use $l = T/5$

A related (and more accurate) test is the Ljung-Box test.

Again, large values of Q^* suggest that the autocorrelations do not come from a white noise series.

How large is too large? If the autocorrelations did come from a white noise series, then both Q and Q^* would have a χ^2 distribution with l degrees of freedom

```
aug |> features(.innov, box_pierce, lag = 10)
```

```
## # A tibble: 1 x 4
##   Symbol .model       bp_stat bp_pvalue
##   <chr>  <chr>          <dbl>     <dbl>
## 1 GOOG   NAIVE(Close)    7.74     0.654
```

```
aug |> features(.innov, ljung_box, lag = 10)
```

```
## # A tibble: 1 x 4
##   Symbol .model       lb_stat lb_pvalue
##   <chr>  <chr>          <dbl>     <dbl>
## 1 GOOG   NAIVE(Close)    7.91     0.637
```

For both Q and Q^*, the results are not significant (i.e., the p-values are relatively large). Thus, we can conclude that the residuals are not distinguishable from a white noise series.

An alternative simple approach that may be appropriate for forecasting the Google daily closing stock price is the drift method. The `tidy()` function shows the one estimated parameter, the drift coefficient, measuring the average daily change observed in the historical data.

```
fit <- google_2015 |> model(RW(Close ~ drift()))
tidy(fit)
```

```
## # A tibble: 1 x 7
##   Symbol .model              term  estimate std.error statistic p.value
##   <chr>  <chr>               <chr>    <dbl>     <dbl>     <dbl>   <dbl>
## 1 GOOG   RW(Close ~ drift()) b        0.944     0.705      1.34   0.182
```

Applying the Ljung-Box test, we obtain the following result.

```
augment(fit) |> features(.innov, ljung_box, lag=10)
```

```
## # A tibble: 1 x 4
##   Symbol .model              lb_stat lb_pvalue
##   <chr>  <chr>                 <dbl>     <dbl>
## 1 GOOG   RW(Close ~ drift())    7.91     0.637
```

As with the naïve method, the residuals from the drift method are indistinguishable from a white noise series.

5.5 Distributional forecasts and prediction intervals

So far we have talked about the point forecast, now we discuss about the probability distribution of the forecast.

- A point forecast $\hat{y}_{T+h|T}$ is (usually) the mean of the conditional probability distribution $y_{T+h}|y_1, y_2, ..., y_T$ but we are also interested in the whole probability distribution especially so that we can get the prediction intervals.
- Most time series models produce normally distributed forecasts provided the residuals of the model are normally distributed. If you have done the transformation before the modelling then they will be normally distributed on the transform scale and when we back transform them, we get a transform normal distribution.
- The forecast distribution describes the probability of observing any future value. So it gives a way to compute things like 95% prediction intervals which contain the true value with 95% probability.

Forecast Distributions:

Assuming residuals are normal, uncorrelated, $sd = \hat{\sigma}$ (standard deviation is constant that means it is homoscadestic) then we can derive the probability distributions for the 4 benchmark methods.

These assumptions are important because if these assumptions are not true then the following statements are not true is some way. But assuming that residuals are normal and there is no autocorrelation in them and there is constant variance then we can prove mathematically the following distributions:

(1) Mean: $y_{T+h|T} \sim N(\bar{y}, (1 + \frac{1}{T})\hat{\sigma}^2)$

It does not depend on "h" and so same prediction interval for all future values.

(2) Naive: $y_{T+h|T} \sim N(y_T, h\hat{\sigma}^2)$

Here variance of the forecast distribution increases linearly with h.

(3) Seasonal Naive: $y_{T+h|T} \sim N(y_{T+h-m(k+1)}, (k+1)\hat{\sigma}^2)$

It is close to linear increase with "h" because "k" is integer part of "h".

(4) Drift: $y_{T+h|T} \sim N(y_T + \frac{h}{T-1}(y_T - y_1), h\frac{T+h}{T}\hat{\sigma}^2)$

Here also, variance of the forecast distribution increases linearly with h.

where k is the integer part of $\frac{h-1}{m}$ and h is the forecast horizon.

Note that when $h = 1$ and T is large (means lots of data) then these all give the same approximate forecast variance: $\hat{\sigma}^2$ and this is true for all forecasting models not only the above 4 models.

Prediction Intervals

Once we know the forecast distribution then we can get the prediction intervals and if the forecast distribution is normal as it usually is for the models, then we can simply write down an expression as the point forecast plus/minus some number times the forecast variance and that number depends on the level means how wide or what probability coverage you want your prediction interval to have. If it is 95% then it is 1.96.

Remember that a prediction interval gives a region within which we expect a true value to lie with a specific probability. It's always possible that the true value lie outside the given range but hopefully the probability coverage that you think you have got it is actually what happens in practice. So,

- A prediction interval gives a region within which we expect y_{T+h} to lie with a specified probability.
- Assuming the forecast errors are normally distributed, then a 95% PI(prediction interval) is:

$$\hat{y}_{T+h|T} \pm 1.96\hat{\sigma}^2$$

where $\hat{\sigma}_h$ is the standard deviation of the h-step distribution.

- When $h = 1$ then $\hat{\sigma}_h$ can be estimated from the residuals.

When we use fable then these things are computed automatically and we don't have to compute by hand.

Example

```
aus_production |>
  filter(!is.na(Bricks))  |>
  model(Seasonal_Naive=SNAIVE(Bricks)) |>
  forecast(h="5 years")
```

```
## # A fable: 20 x 4 [1Q]
## # Key:     .model [1]
##    .model         Quarter        Bricks .mean
##    <chr>            <qtr>        <dist> <dbl>
##  1 Seasonal_Naive 2005 Q3  N(428, 2336)   428
##  2 Seasonal_Naive 2005 Q4  N(397, 2336)   397
##  3 Seasonal_Naive 2006 Q1  N(355, 2336)   355
##  4 Seasonal_Naive 2006 Q2  N(435, 2336)   435
##  5 Seasonal_Naive 2006 Q3  N(428, 4672)   428
##  6 Seasonal_Naive 2006 Q4  N(397, 4672)   397
##  7 Seasonal_Naive 2007 Q1  N(355, 4672)   355
##  8 Seasonal_Naive 2007 Q2  N(435, 4672)   435
##  9 Seasonal_Naive 2007 Q3  N(428, 7008)   428
## 10 Seasonal_Naive 2007 Q4  N(397, 7008)   397
## 11 Seasonal_Naive 2008 Q1  N(355, 7008)   355
## 12 Seasonal_Naive 2008 Q2  N(435, 7008)   435
## 13 Seasonal_Naive 2008 Q3  N(428, 9343)   428
## 14 Seasonal_Naive 2008 Q4  N(397, 9343)   397
## 15 Seasonal_Naive 2009 Q1  N(355, 9343)   355
## 16 Seasonal_Naive 2009 Q2  N(435, 9343)   435
## 17 Seasonal_Naive 2009 Q3 N(428, 11679)   428
## 18 Seasonal_Naive 2009 Q4 N(397, 11679)   397
## 19 Seasonal_Naive 2010 Q1 N(355, 11679)   355
## 20 Seasonal_Naive 2010 Q2 N(435, 11679)   435
```

Here, seasonal naive distribution for bricks is computed using the previous formulas.

If you want to calculate the prediction interval then you can use the hilo() function.

```
aus_production |>
  filter(!is.na(Bricks)) |>
  model(Seasonal_naive = SNAIVE(Bricks)) |>
  forecast(h= "5 years") |>
  hilo(level=95) |>
  mutate(lower = `95%`$lower, upper=`95%`$upper)
```

```
## # A tsibble: 20 x 7 [1Q]
## # Key:       .model [1]
##    .model         Quarter        Bricks .mean                  `95%` lower upper
##    <chr>            <qtr>        <dist> <dbl>                 <hilo> <dbl> <dbl>
##  1 Seasonal_naive 2005 Q3  N(428, 2336)   428 [333.2737, 522.7263]95  333.  523.
##  2 Seasonal_naive 2005 Q4  N(397, 2336)   397 [302.2737, 491.7263]95  302.  492.
##  3 Seasonal_naive 2006 Q1  N(355, 2336)   355 [260.2737, 449.7263]95  260.  450.
##  4 Seasonal_naive 2006 Q2  N(435, 2336)   435 [340.2737, 529.7263]95  340.  530.
##  5 Seasonal_naive 2006 Q3  N(428, 4672)   428 [294.0368, 561.9632]95  294.  562.
##  6 Seasonal_naive 2006 Q4  N(397, 4672)   397 [263.0368, 530.9632]95  263.  531.
##  7 Seasonal_naive 2007 Q1  N(355, 4672)   355 [221.0368, 488.9632]95  221.  489.
##  8 Seasonal_naive 2007 Q2  N(435, 4672)   435 [301.0368, 568.9632]95  301.  569.
##  9 Seasonal_naive 2007 Q3  N(428, 7008)   428 [263.9292, 592.0708]95  264.  592.
## 10 Seasonal_naive 2007 Q4  N(397, 7008)   397 [232.9292, 561.0708]95  233.  561.
## 11 Seasonal_naive 2008 Q1  N(355, 7008)   355 [190.9292, 519.0708]95  191.  519.
## 12 Seasonal_naive 2008 Q2  N(435, 7008)   435 [270.9292, 599.0708]95  271.  599.
## 13 Seasonal_naive 2008 Q3  N(428, 9343)   428 [238.5474, 617.4526]95  239.  617.
## 14 Seasonal_naive 2008 Q4  N(397, 9343)   397 [207.5474, 586.4526]95  208.  586.
## 15 Seasonal_naive 2009 Q1  N(355, 9343)   355 [165.5474, 544.4526]95  166.  544.
## 16 Seasonal_naive 2009 Q2  N(435, 9343)   435 [245.5474, 624.4526]95  246.  624.
```

```
## 17 Seasonal_naive 2009 Q3 N(428, 11679)   428 [216.1855, 639.8145]95  216.  640.
## 18 Seasonal_naive 2009 Q4 N(397, 11679)   397 [185.1855, 608.8145]95  185.  609.
## 19 Seasonal_naive 2010 Q1 N(355, 11679)   355 [143.1855, 566.8145]95  143.  567.
## 20 Seasonal_naive 2010 Q2 N(435, 11679)   435 [223.1855, 646.8145]95  223.  647.
```

Summary

- Point forecasts are often useless without a measure of uncertainty (such as prediction intervals).
- Prediction intervals require a stochastic model(with random errors, etc).
- For most models, prediction intervals get wider as the forecast horizon increases. (For mean() model, it is not true but for other models, it is true)
- Use `level` argument to control coverage.
- Check residual assumptions before believing them.
- Prediction intervals are usually too narrow due to unaccounted uncertainty.

Summary

Forecast distributions

we express the uncertainty in our forecasts using a probability distribution. It describes the probability of observing possible future values using the fitted model. The point forecast is the mean of this distribution. Most time series models produce normally distributed forecasts — that is, we assume that the distribution of possible future values follows a normal distribution.

Prediction intervals

A prediction interval gives an interval within which we expect y_t to lie with a specified probability.

Table 5.1: Multipliers to be used for prediction intervals.

Percentage	Multiplier
50	0.67
55	0.76
60	0.84
65	0.93
70	1.04
75	1.15
80	1.28
85	1.44
90	1.64
95	1.96
96	2.05
97	2.17
98	2.33
99	2.58

Figure 5: Prediction Interval

The value of prediction intervals is that they express the uncertainty in the forecasts. If we only produce point forecasts, there is no way of telling how accurate the forecasts are. However, if we also produce

prediction intervals, then it is clear how much uncertainty is associated with each forecast. For this reason, point forecasts can be of almost no value without the accompanying prediction intervals.

One-step prediction intervals

When forecasting one step ahead, the standard deviation of the forecast distribution can be estimated using the standard deviation of the residuals given by

$$\hat{\sigma} = \sqrt{\frac{1}{T-K-M}\Sigma_{t=1}^{T} e_t^2}$$

where K is the number of parameters estimated in the forecasting method, and
M is the number of missing values in the residuals. (For example, M=1 for a naive forecast, because we can't forecast the first observation.)

Multi-step prediction interval

A common feature of prediction intervals is that they usually increase in length as the forecast horizon increases. The further ahead we forecast, the more uncertainty is associated with the forecast, and thus the wider the prediction intervals. That is,
σ_h usually increases with h (although there are some non-linear forecasting methods which do not have this property).

To produce a prediction interval, it is necessary to have an estimate of σ_h. As already noted, for one-step forecasts (h=1), Equation (5.1) provides a good estimate of the forecast standard deviation σ_1. For multi-step forecasts, a more complicated method of calculation is required. These calculations assume that the residuals are uncorrelated.

Benchmark methods

For the four benchmark methods, it is possible to mathematically derive the forecast standard deviation under the assumption of uncorrelated residuals. If $\hat{\sigma}_h$ denotes the standard deviation of the h-step forecast distribution, and $\hat{\sigma}$ is the residual standard deviation given by (5.1), then we can use the expressions shown in Table 5.2. Note that when $h = 1$ and T is large, these all give the same approximate value $\hat{\sigma}$.

Table 5.2: Multi-step forecast standard deviation for the four benchmark methods, where σ is the residual standard deviation, m is the seasonal period, and k is the integer part of $(h-1)/m$ (i.e., the number of complete years in the forecast period prior to time $T+h$).

Benchmark method	h-step forecast standard deviation
Mean	$\hat{\sigma}_h = \hat{\sigma}\sqrt{1+1/T}$
Naïve	$\hat{\sigma}_h = \hat{\sigma}\sqrt{h}$
Seasonal naïve	$\hat{\sigma}_h = \hat{\sigma}\sqrt{k+1}$
Drift	$\hat{\sigma}_h = \hat{\sigma}\sqrt{h(1+h/(T-1))}$

Prediction intervals can easily be computed for you when using the `fable` package. For example, here is the output when using the naïve method for the Google stock price.

Figure 6: Benchmarks Method

Prediction intervals can easily be computed for you when using the fable package. For example, here is the output when using the naïve method for the Google stock price.

```
google_2015 |>
  model(NAIVE(Close)) |>
  forecast(h = 10) |>
  hilo()
```

```
## # A tsibble: 10 x 7 [1]
## # Key:       Symbol, .model [1]
##    Symbol .model         day        Close .mean                  `80%`
##    <chr>  <chr>        <dbl>       <dist> <dbl>                 <hilo>
##  1 GOOG   NAIVE(Close)   253  N(759, 125)  759. [744.5400, 773.2200]80
##  2 GOOG   NAIVE(Close)   254  N(759, 250)  759. [738.6001, 779.1599]80
##  3 GOOG   NAIVE(Close)   255  N(759, 376)  759. [734.0423, 783.7177]80
##  4 GOOG   NAIVE(Close)   256  N(759, 501)  759. [730.1999, 787.5601]80
##  5 GOOG   NAIVE(Close)   257  N(759, 626)  759. [726.8147, 790.9453]80
##  6 GOOG   NAIVE(Close)   258  N(759, 751)  759. [723.7543, 794.0058]80
##  7 GOOG   NAIVE(Close)   259  N(759, 876)  759. [720.9399, 796.8202]80
##  8 GOOG   NAIVE(Close)   260 N(759, 1002)  759. [718.3203, 799.4397]80
##  9 GOOG   NAIVE(Close)   261 N(759, 1127)  759. [715.8599, 801.9001]80
## 10 GOOG   NAIVE(Close)   262 N(759, 1252)  759. [713.5329, 804.2272]80
## # i 1 more variable: `95%` <hilo>
```

The hilo() function converts the forecast distributions into intervals. By default, 80% and 95% prediction intervals are returned, although other options are possible via the level argument.

When plotted, the prediction intervals are shown as shaded regions, with the strength of colour indicating the probability associated with the interval. Again, 80% and 95% intervals are shown by default, with other options available via the level argument.

```
google_2015 |>
  model(NAIVE(Close)) |>
  forecast(h = 10) |>
  autoplot(google_2015) +
  labs(title="Google daily closing stock price", y="$US" )
```

Prediction intervals from bootstrapped residuals

When a normal distribution for the residuals is an unreasonable assumption, one alternative is to use bootstrapping, which only assumes that the residuals are uncorrelated with constant variance. We will illustrate the procedure using a naïve forecasting method.

A one-step forecast error is defined as $e_t = y_t - \hat{y}_{t|t-1}$. For a naïve forecasting method, $\hat{y}_{t|t-1} = y_{t-1}$, so we can rewrite this as

$$y_t = y_{t-1} + e_t$$

Assuming future errors will be similar to past errors, when $t > T$ we can replace e_t by sampling from the collection of errors we have seen in the past (i.e., the residuals). So we can simulate the next observation of a time series using

$$y^*_{T+1} = y_T + e^*_{T+1}$$

where e^*_{T+1} is a randomly sampled error from the past, and y^*_{T+1} is the possible future value that would arise if that particular error value occurred. We use a * to indicate that this is not the observed y_{T+1} value, but one possible future that could occur. Adding the new simulated observation to our data set, we can repeat the process to obtain

$$y^*_{T+2} = y^*_{T+1} + e^*_{T+2}$$

Doing this repeatedly, we obtain many possible futures. To see some of them, we can use the generate() function.

```
fit <- google_2015 |>
  model(NAIVE(Close))
sim <- fit |> generate(h = 30, times = 5, bootstrap = TRUE)
sim
```

```
## # A tsibble: 150 x 6 [1]
## # Key:       Symbol, .model, .rep [5]
##    Symbol .model       .rep    day  .innov  .sim
##    <chr>  <chr>        <chr> <dbl>   <dbl> <dbl>
##  1 GOOG   NAIVE(Close) 1       253  -7.42   751.
##  2 GOOG   NAIVE(Close) 1       254  -9.81   742.
##  3 GOOG   NAIVE(Close) 1       255 -21.4    720.
##  4 GOOG   NAIVE(Close) 1       256   7.67   728.
##  5 GOOG   NAIVE(Close) 1       257   5.01   733.
##  6 GOOG   NAIVE(Close) 1       258  -8.60   724.
##  7 GOOG   NAIVE(Close) 1       259  -0.104  724.
##  8 GOOG   NAIVE(Close) 1       260 -21.7    703.
##  9 GOOG   NAIVE(Close) 1       261   0.556  703.
## 10 GOOG   NAIVE(Close) 1       262  -1.48   702.
## # i 140 more rows
```

Here we have generated five possible sample paths for the next 30 trading days. The `.rep` variable provides a new key for the tsibble. The plot below shows the five sample paths along with the historical data.

```
google_2015 |>
  ggplot(aes(x = day)) +
  geom_line(aes(y = Close)) +
  geom_line(aes(y = .sim, colour = as.factor(.rep)),
    data = sim) +
  labs(title="Google daily closing stock price", y="$US" ) +
  guides(colour = "none")
```

Then we can compute prediction intervals by calculating percentiles of the future sample paths for each forecast horizon. The result is called a bootstrapped prediction interval. The name "bootstrap" is a reference to pulling ourselves up by our bootstraps, because the process allows us to measure future uncertainty by only using the historical data.

This is all built into the forecast() function so you do not need to call generate() directly.

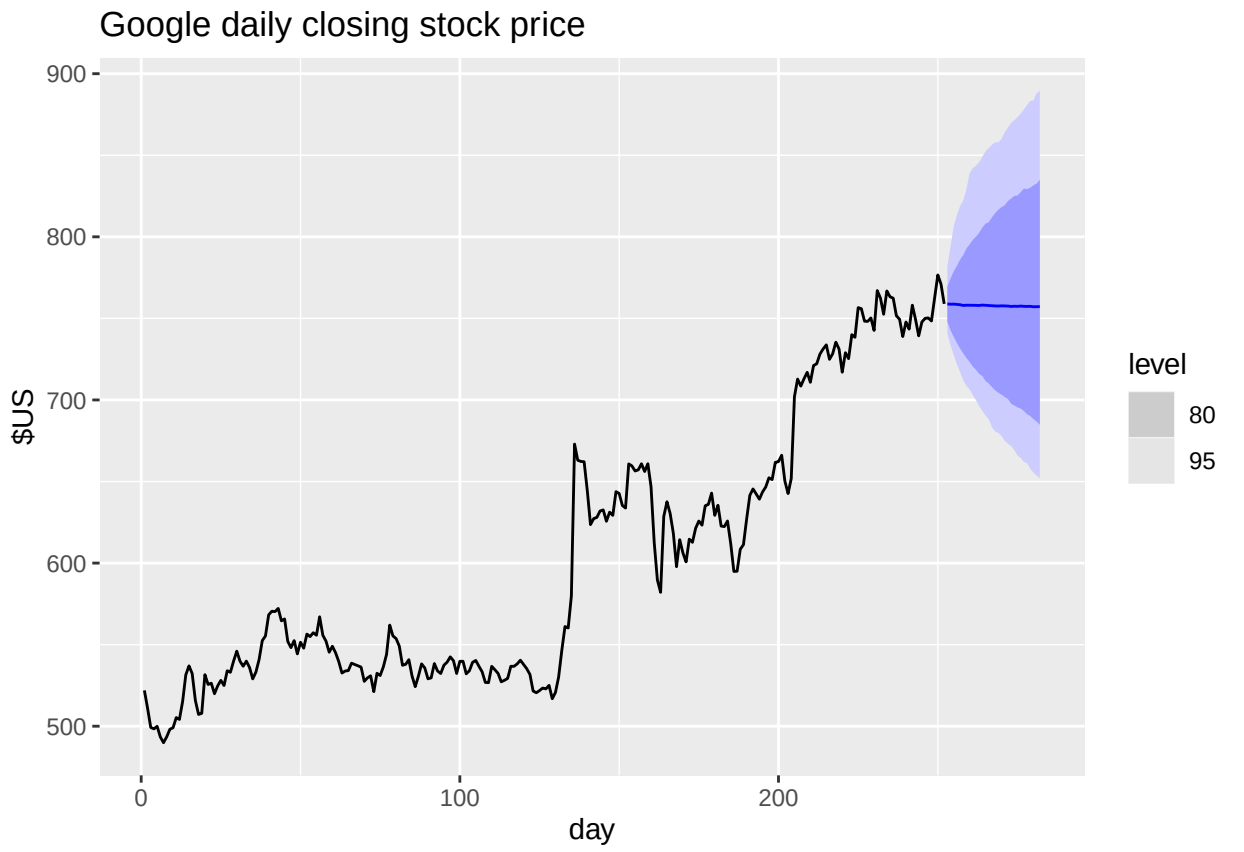
```
fc <- fit |> forecast(h = 30, bootstrap = TRUE)
fc
```

```
## # A fable: 30 x 5 [1]
## # Key:     Symbol, .model [1]
##    Symbol .model         day        Close .mean
##    <chr>  <chr>        <dbl>       <dist> <dbl>
##  1 GOOG   NAIVE(Close)   253 sample[5000]  759.
##  2 GOOG   NAIVE(Close)   254 sample[5000]  759.
##  3 GOOG   NAIVE(Close)   255 sample[5000]  759.
##  4 GOOG   NAIVE(Close)   256 sample[5000]  759.
##  5 GOOG   NAIVE(Close)   257 sample[5000]  758.
##  6 GOOG   NAIVE(Close)   258 sample[5000]  758.
##  7 GOOG   NAIVE(Close)   259 sample[5000]  758.
##  8 GOOG   NAIVE(Close)   260 sample[5000]  758.
##  9 GOOG   NAIVE(Close)   261 sample[5000]  758.
## 10 GOOG   NAIVE(Close)   262 sample[5000]  758.
## # i 20 more rows
```

Notice that the forecast distribution is now represented as a simulation with 5000 sample paths. Because there is no normality assumption, the prediction intervals are not symmetric. The `.mean` column is the mean

of the bootstrap samples, so it may be slightly different from the results obtained without a bootstrap.

```
autoplot(fc, google_2015) +
  labs(title="Google daily closing stock price", y="$US" )
```



The number of samples can be controlled using the times argument for forecast(). For example, intervals based on 1000 bootstrap samples can be sampled with:

```
google_2015 |>
  model(NAIVE(Close)) |>
  forecast(h = 10, bootstrap = TRUE, times = 1000) |>
  hilo()
```

```
## # A tsibble: 10 x 7 [1]
## # Key:       Symbol, .model [1]
##    Symbol .model         day        Close .mean                 `80%`
##    <chr>  <chr>        <dbl>       <dist> <dbl>                <hilo>
##  1 GOOG   NAIVE(Close)   253 sample[1000]  759. [747.9560, 769.0150]80
##  2 GOOG   NAIVE(Close)   254 sample[1000]  758. [741.9222, 774.2653]80
##  3 GOOG   NAIVE(Close)   255 sample[1000]  758. [737.0033, 778.7120]80
##  4 GOOG   NAIVE(Close)   256 sample[1000]  758. [733.8594, 783.7325]80
##  5 GOOG   NAIVE(Close)   257 sample[1000]  759. [729.2415, 787.7342]80
##  6 GOOG   NAIVE(Close)   258 sample[1000]  758. [726.7069, 790.5961]80
##  7 GOOG   NAIVE(Close)   259 sample[1000]  758. [724.6329, 793.6741]80
##  8 GOOG   NAIVE(Close)   260 sample[1000]  758. [721.7033, 796.5258]80
##  9 GOOG   NAIVE(Close)   261 sample[1000]  759. [720.4273, 798.2710]80
## 10 GOOG   NAIVE(Close)   262 sample[1000]  758. [716.6306, 801.1456]80
## # i 1 more variable: `95%` <hilo>
```

5.6 Forecasting using transformations

Mathematical Transformations:

- If the data show different variation at different levels of series, then a transformation can be useful.
- Denote original observations as $y_1, y_2, ..., y_n$ and transformed observations as $w_1, w_2, ..., w_n$ then the Box-cox transformation can be defined as:

$w_t = \log(y_t)\ if\ \lambda = 0$ and

$w_t = \frac{sign(y_t)|y_t|^{\lambda-1}}{\lambda} if \lambda \neq 0$