Ch3 | The Forecasters Toolbox

Michael Rose

November 3, 2018

Chapter 3 | The Forecaster's Toolbox

In this chapter we will discuss:

- General tools useful for different forecasting situations
- benchmark forecasting methods
- transformations to make forecasting simpler
- methods for checking whether a forecasting method has utilized the available information
- techniques for computing prediction intervals

3.1 | Some simple forecasting methods

We will use the following four forecasting methods as benchmarks througout this book:

Average Method

All the forecasts of all future values are equal to the average of the historical data. Let the historical data be $y_1, ..., y_T$. Then

$$y_{T+h|T} = \bar{y} = (y_1 + ... + y_T)/T \text{ where } y_{T+h|T} \text{ is the estimate of } y_{t+h} \text{ based on the data } y_1, ..., y_T$$

```
# y contains the time series
# h is the forecast horizon

# mean(y, h)
```

Naive Method

We can simply set all forecasts to be the value of the last observation.

$$y_{T+h|T} = y_T$$

This method works well for many economic and financial time series

```
# naive(y, h)
# rwf(y, h)  equivalent alternative
```

Because a naive forecast is optimal when data follow a random walk, these are also called **Random Walk Forecasts**.

Seasonal Naive Method

When our data is highly seasonal, we can set our forecast to be equal to the last observed value from the same season of the year.

$y_{T+h|T} = y_{T+h-m(k+1)}$ where m is the seasonal period k is the integer part of $\frac{h-1}{m}$ (the number of complete years in the forecast period prior to time $T+h$)

```
# snaive(y, h)
```

Drift Method

A variation on the naive method is to allow the forecasts to increase or decrease over time, where the amount of change over time (called the **drift**) is set to be the average change seen in the historical data.

$$y_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1}) = y_T + h(\frac{y_T - y_1}{T-1})$$

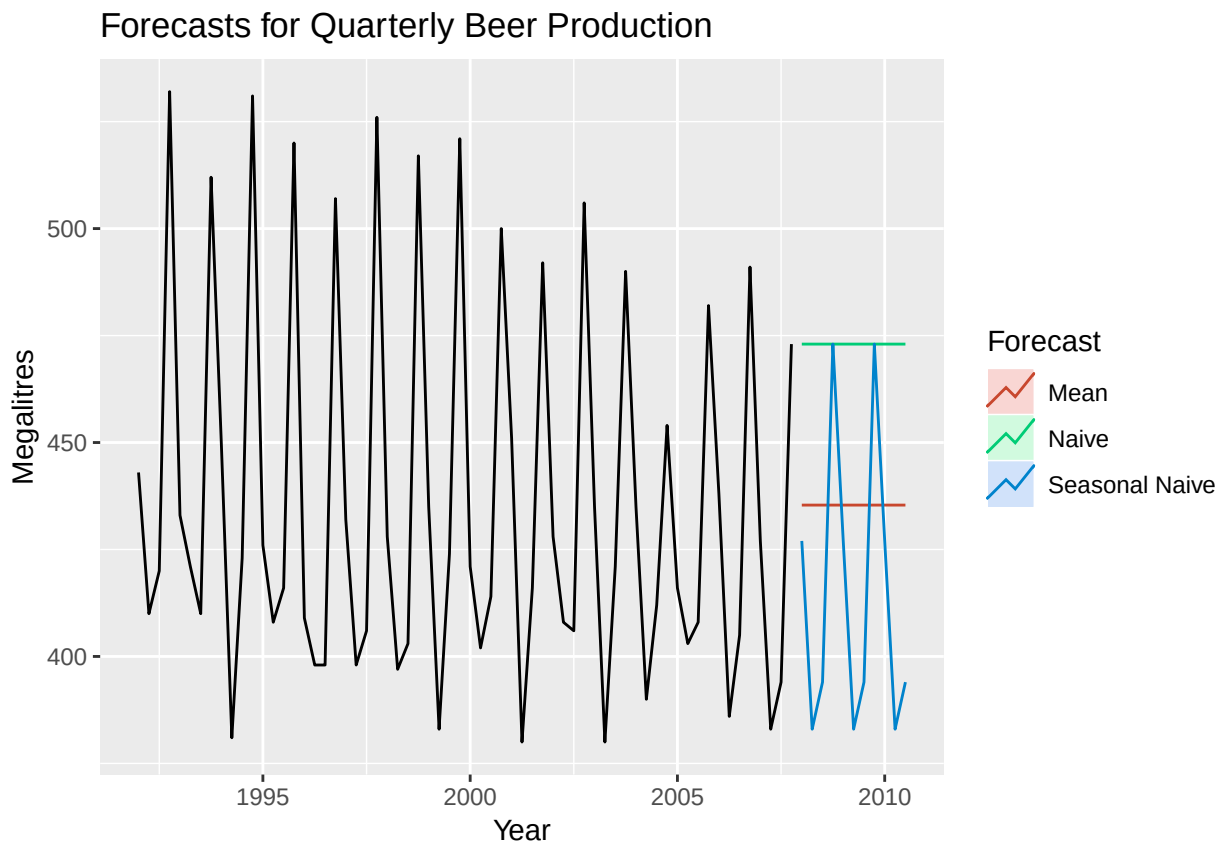
This is equivalent to drawing a line between the first and last observations, and extrapolating it into the future.

```
# rwf(y, h, drift = TRUE)
```

Examples

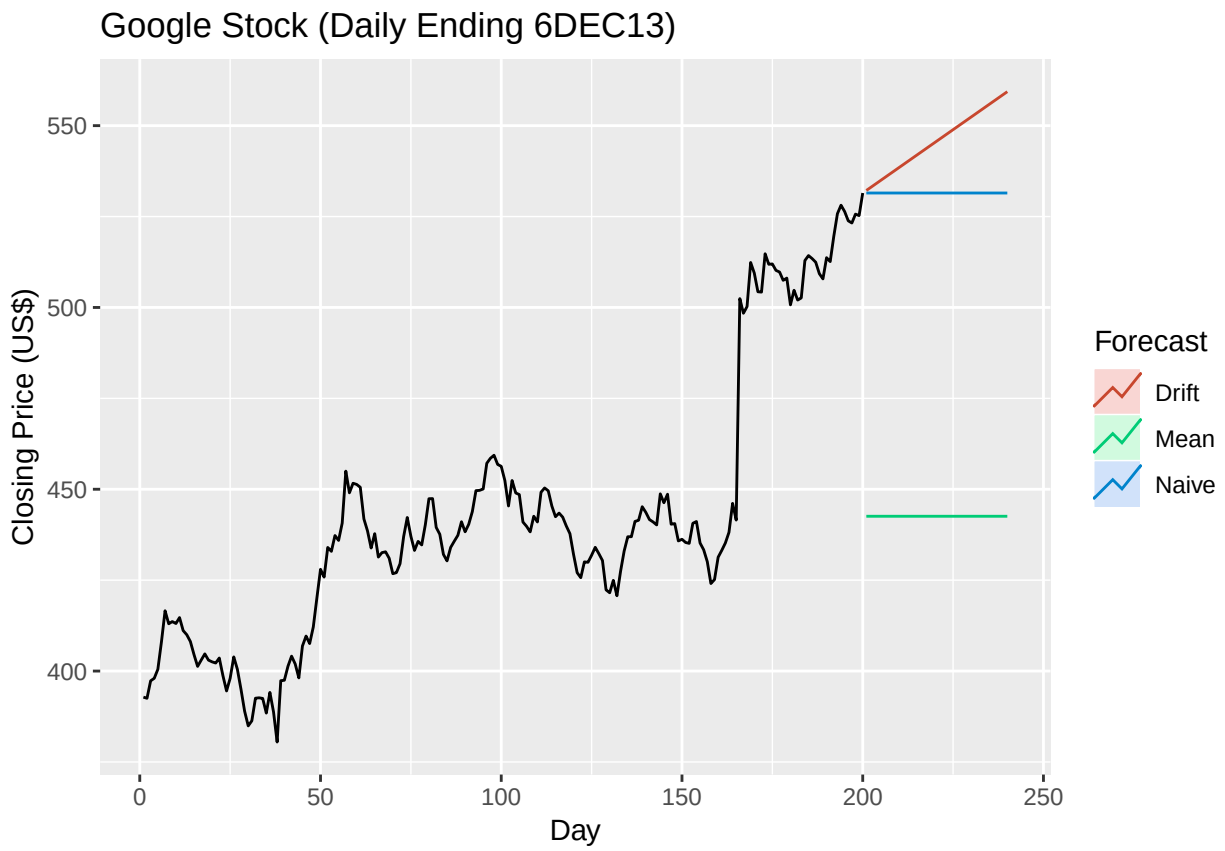
```
# set training data from 1992 to 2007
beer2 <- window(ausbeer, start = 1992, end = c(2007, 4))

# plot forecasts for first 3 methods
autoplot(beer2) +
  autolayer(meanf(beer2, h = 11), series = "Mean", PI = FALSE) +
  autolayer(naive(beer2, h=11), series = "Naive", PI=FALSE) +
  autolayer(snaive(beer2, h=11), series="Seasonal Naive", PI=FALSE) +
  ggtitle("Forecasts for Quarterly Beer Production") +
  xlab("Year") + ylab("Megalitres") +
  guides(colour = guide_legend(title = "Forecast"))
```



Here we apply the non-seasonal methods to a series of 200 days of Google daily closing stock price:

```
autoplot(goog200) +
  autolayer(meanf(goog200, h = 40), series = "Mean", PI=FALSE) +
  autolayer(rwf(goog200, h = 40), series = "Naive", PI=FALSE) +
  autolayer(rwf(goog200, h = 40, drift = TRUE), series = "Drift", PI=FALSE) +
  ggtitle("Google Stock (Daily Ending 6DEC13)") +
  xlab("Day") + ylab("Closing Price (US$)") +
  guides(colour = guide_legend(title = "Forecast"))
```



These methods are generally considered benchmarks, but they may be the most effective in some cases. When we develop a new method, they must be better than these benchmarks – else they aren't worth considering.

3.2 | Transformations and Adjustments

In this section we deal with 4 kinds of adjustments:

- Calendar Adjustments
- Population Adjustments
- Inflation Adjustments
- Mathematical Transformations

We want to simplify the patterns in the historical data by removing known sources of variation or by making the pattern more consistent across the whole data set.

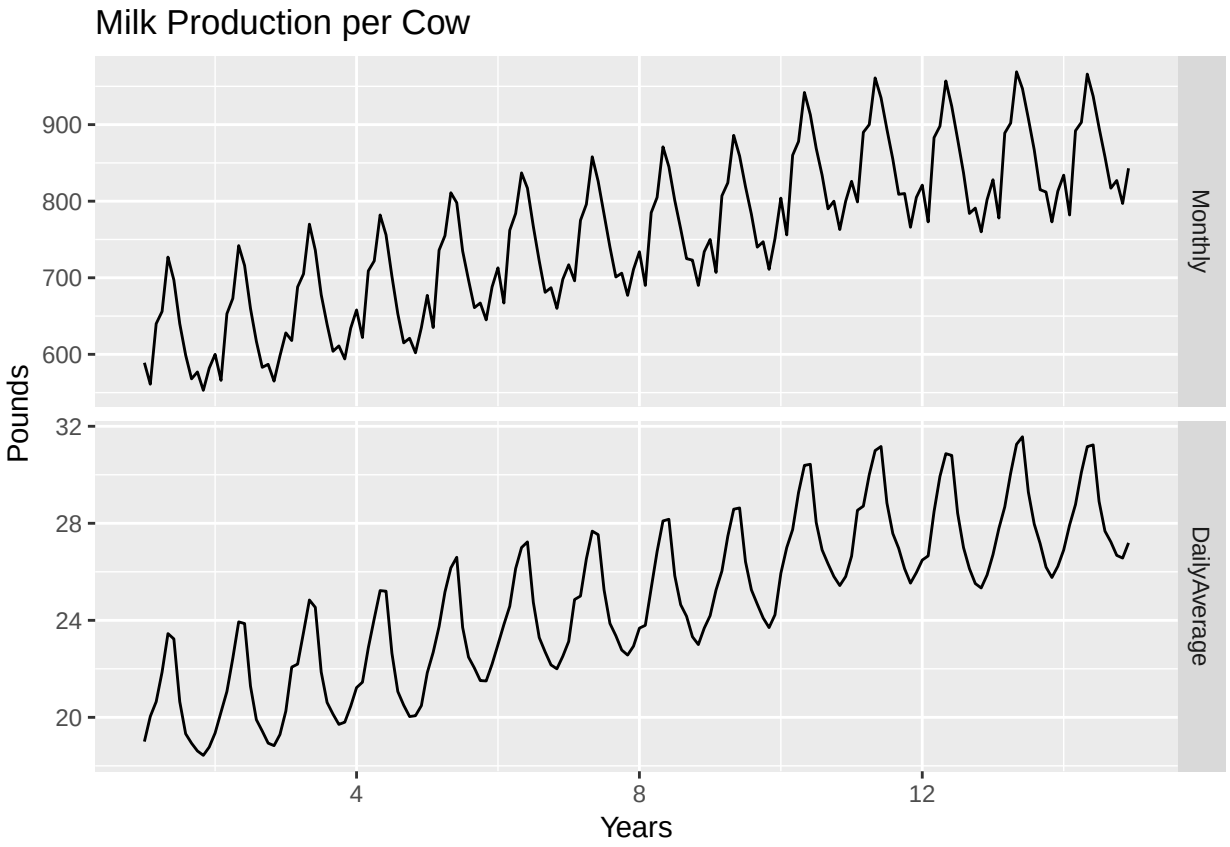
Calendar Adjustments

Some of the variation in seasonal data may be due to simple calendar effects. In these cases, it is easy to remove the variation before fitting a forecasting model.

For example, if we are studying monthly milk production on a farm, there will be variation between the months simply because the different number of days in each month (in addition to the seasonal variation across the year).

```
# monthdays() will compute the number of days in each month or quarter
dframe <- cbind(Monthly = milk, DailyAverage = milk / monthdays(milk))
```

```
autoplot(dframe, facet=TRUE) +
  xlab("Years") + ylab("Pounds") +
  ggtitle("Milk Production per Cow")
```



By looking at the average daily production instead of the average monthly production, we effectively remove the variation due to different month lengths.

Population Adjustments

Any data affected by population changes can be adjusted to give per-capita data. We can consider the data per person, or thousand people, or million people.

Inflation Adjustments

Data which are affected by the value of money are continuously changed by the value of that currency. For example, $200,000 today is worth less than the same amount in the year 2000. Thus we can standardize our monetary measurements in year X dollars.

To make these adjustments, a price index is used. If z_t denotes the price index and y_t denotes the original house price in year t, then $x_t = \frac{y_t}{z_t} * z_{2000}$ gives the adjusted price at year 2000 dollar values.

Mathematical Transformations

If the data shows variation that increases or decreases with the level of the series, then a transformation can be useful. A log transform is often useful.

Other transforms include square roots and cube roots. These are called **power transformations** and can be expressed in the form $w_t = y_t^p$.

Another useful family of transformations that includes both logs and power transformations is the family of **Box-Cox Transformations**.

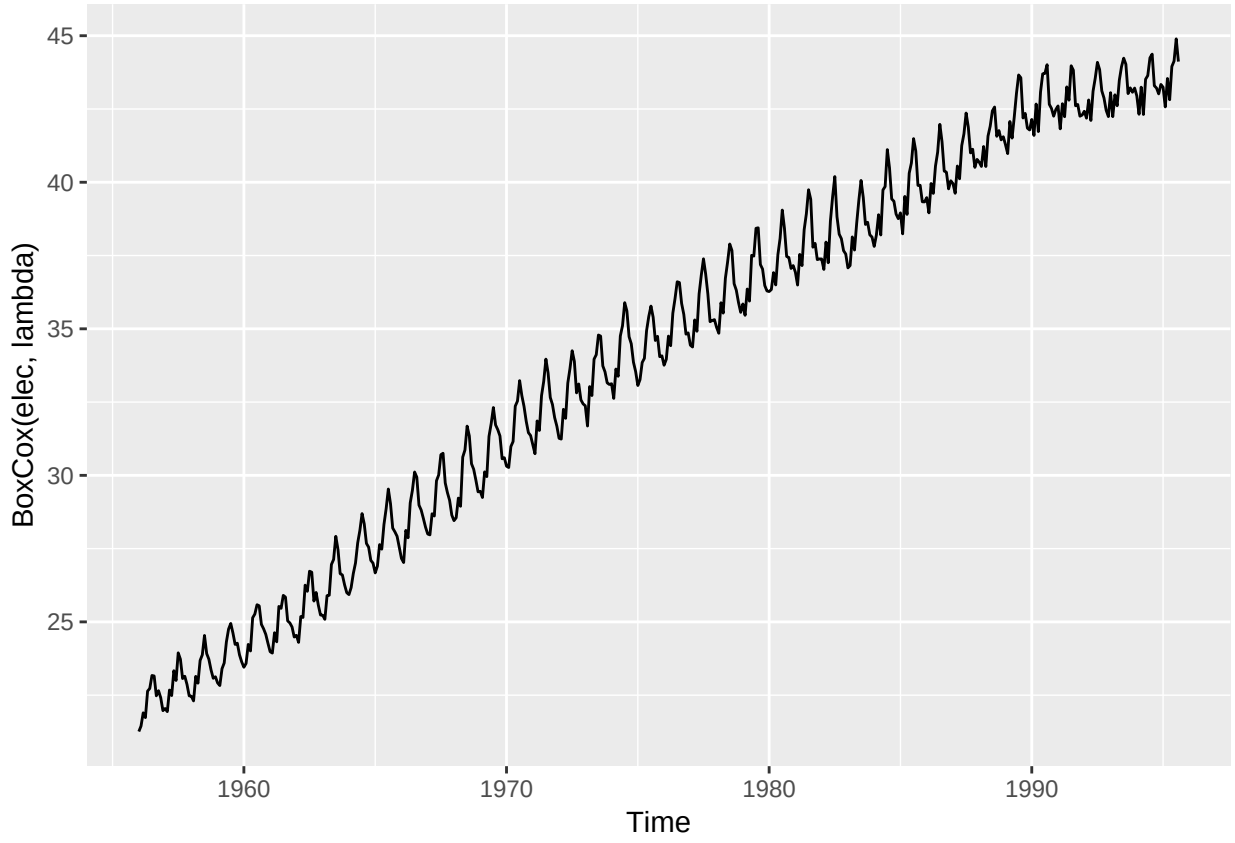
$w_t = \log(y_t)$ if $\lambda = 0$. $w_t = \frac{y_t^\lambda - 1}{\lambda}$ otherwise.

A good value of λ is one which makes the size of the seasonal variation about the same across the whole series.

```
# the BoxCox.lambda() function will choose a value of lambda for you
(lambda <- BoxCox.lambda(elec))
```

```
## [1] 0.2654076
```

```
# plot
autoplot(BoxCox(elec, lambda))
```



Having chosen a transformation, we need to forecast the transformed data. Then, we need to reverse the transformation (or back-transform) to obtain forecasts on the original scale.

The Reverse-BoxCox transformation is given by

$y_t = \exp(w_t)$ if $\lambda = 0$ $y_t = (\lambda w_t + 1)^{\frac{1}{\lambda}}$ otherwise.

Features of Power Transformations

- If some $y_t \leq 0$, no power transformation is possible unless all observations are adjusted by adding a constant to all values.
- Choose a simple value for λ. It makes explanations easier.
- The forecasting results are relatively insensitive to the value of λ
- Often no transformation is needed
- Transformations sometimes make little difference to the forecasts but have a large effect on prediction intervals

Bias Adjustments

One issue with using mathematical transformations like BoxCox is that the back-transformed point forecast will not be the mean, but instead the median, of the forecast distribution.

If we need the mean, for example if we wish to add up sales forecasts from various regions to form a forecast for the whole country, the back-transformed mean is given by:

$y_t = \exp(w_t)[1 + \frac{\sigma_h^2}{2}]$ if $\lambda = 0$ $y_t = (\lambda w_t + 1)^{\frac{1}{\lambda}}[1 + \frac{\sigma_h^2(1-\lambda)}{2(\lambda w_t + 1)^2}]$ otherwise.

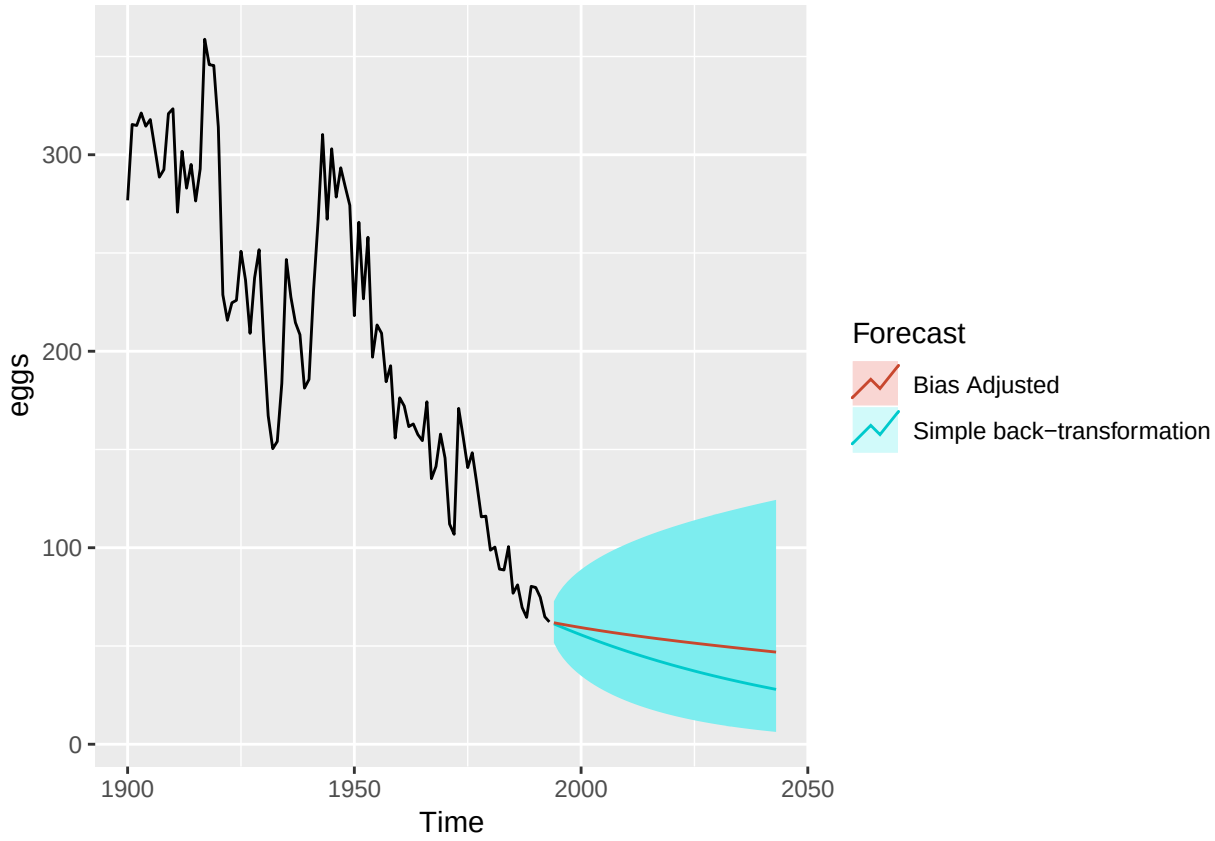
Where σ_h^2 is the h-step forecast variance. The larger the forecast variance, the bigger the difference between mean and the median.

The difference between the back-transform without the adjustment and the back transform with the adjustment is called the **bias**. When we use the mean rather than the median, we say our point forecasts have been **bias-adjusted**

To consider how much the bias adjustment changes things:

```
fc <- rwf(eggs, drift = TRUE, lambda = 0, h = 50, level = 80)
fc2 <- rwf(eggs, drift = TRUE, lambda = 0, h = 50, level = 80, biasadj = TRUE)

autoplot(eggs) +
  autolayer(fc, series = "Simple back-transformation") +
  autolayer(fc2, series = "Bias Adjusted", PI = FALSE) +
  guides(color = guide_legend(title = "Forecast"))
```



The blue in the figure above shows the forecast medians while the red line shows the forecast means. Notice how the skewed forecast distribution pulls up the point forecast when we use the bias adjustment.

Bias adjustment is not done by default in the forecast package. We must use the arg biasadj=TRUE when we select our boxcox transformation parameter.

3.3 | Residual Diagnostics

Fitted Values

Each observation in a time series can be forecast using all previous observations. These are called **fitted values** and they are denoted by $y_{t|\hat{t}-1}$. This means the forecast of y_t based on observations $y_1, ..., y_{t-1}$. Fitted values always involve one-step forecasts.

Fitted values are not true forecasts because any parameters involved in the forecasting method are estimated using all available observations in a time series, including future observations.

For the average method, the fitted values are given by $\hat{y}_t = \hat{c}$ where $\hat{c}$ is the average computed over all available observations, including those at times after t.

Similarly, for the drift method, the drift parameter is estimated using all available observations. In this case, the fitted values are given by $\hat{y}_t = y_{t-1} + \hat{c}$ where $\hat{c} = \frac{y_T - y_1}{T-1}$.

Residuals

The residuals in a time series model are what is left over after fitting a model. For many TS models, $e_t = y_t - \hat{y}_t$.

A good forecasting model will yeild residuals with the following properties:

1. The residuals are uncorrelated. If there are correlations between residuals, then there is information left in the residuals which should be used in computing forecasts.
2. The residuals have 0 mean. If the residuals have a mean other than 0, then the forecasts are biased. Thankfully, adjusting for bias is easy. If residuals have mean m, the we can add m to all forecasts and the bias problem is solved.

In addition to the 2 properties above, it is useful for residuals to also have these properties:

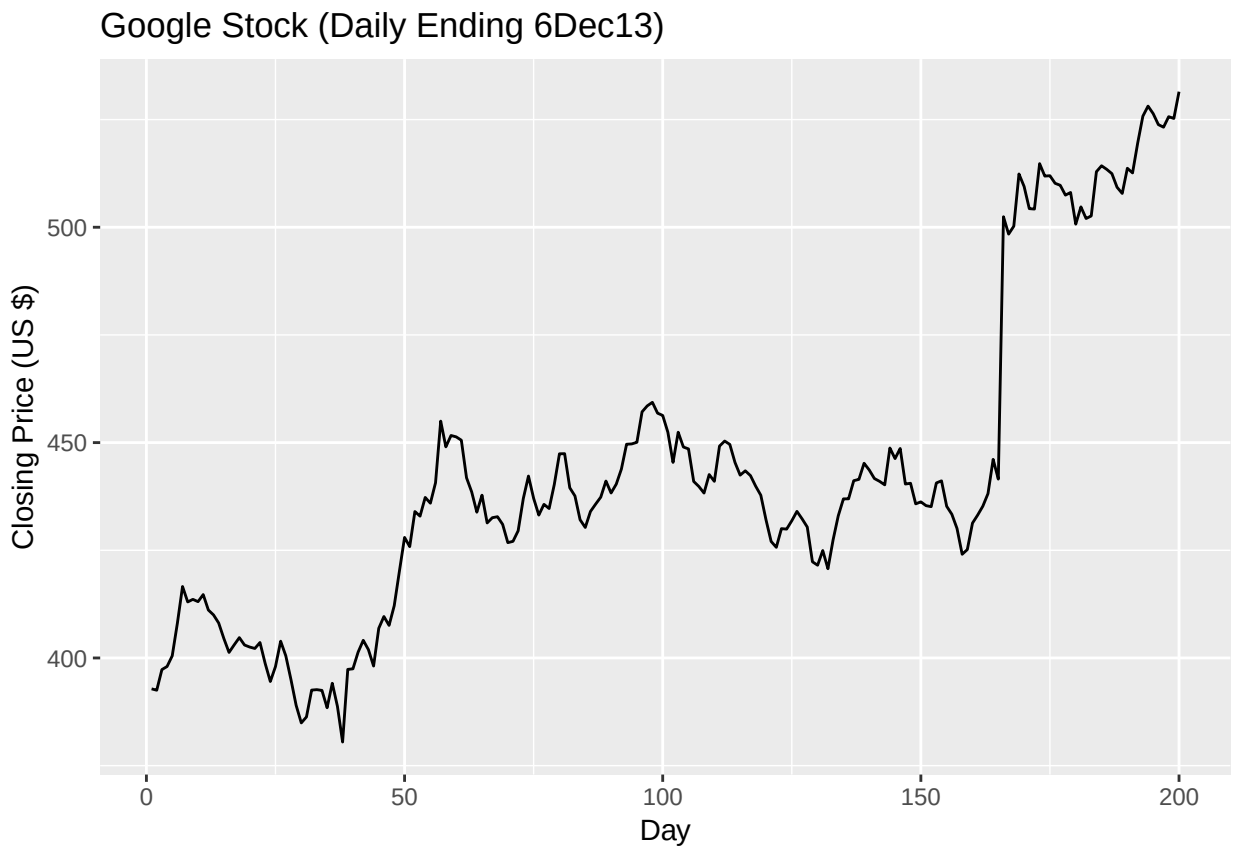
3. The residuals have constant variance
4. The residuals are normally distributed

These make the calculation of prediction intervals easier.

Example: Forecasting the Google daily stock price

For stock market prices and indexes, the best forecasting method is often the naive method. That is, each forecast is simply equal to the last observed value, or $\hat{y}_t = y_{t-1}$. Then the residuals are simply equal to the difference between consecutive observations: $e_t = y_t - \hat{y}_t = y_t - y_{t-1}$.

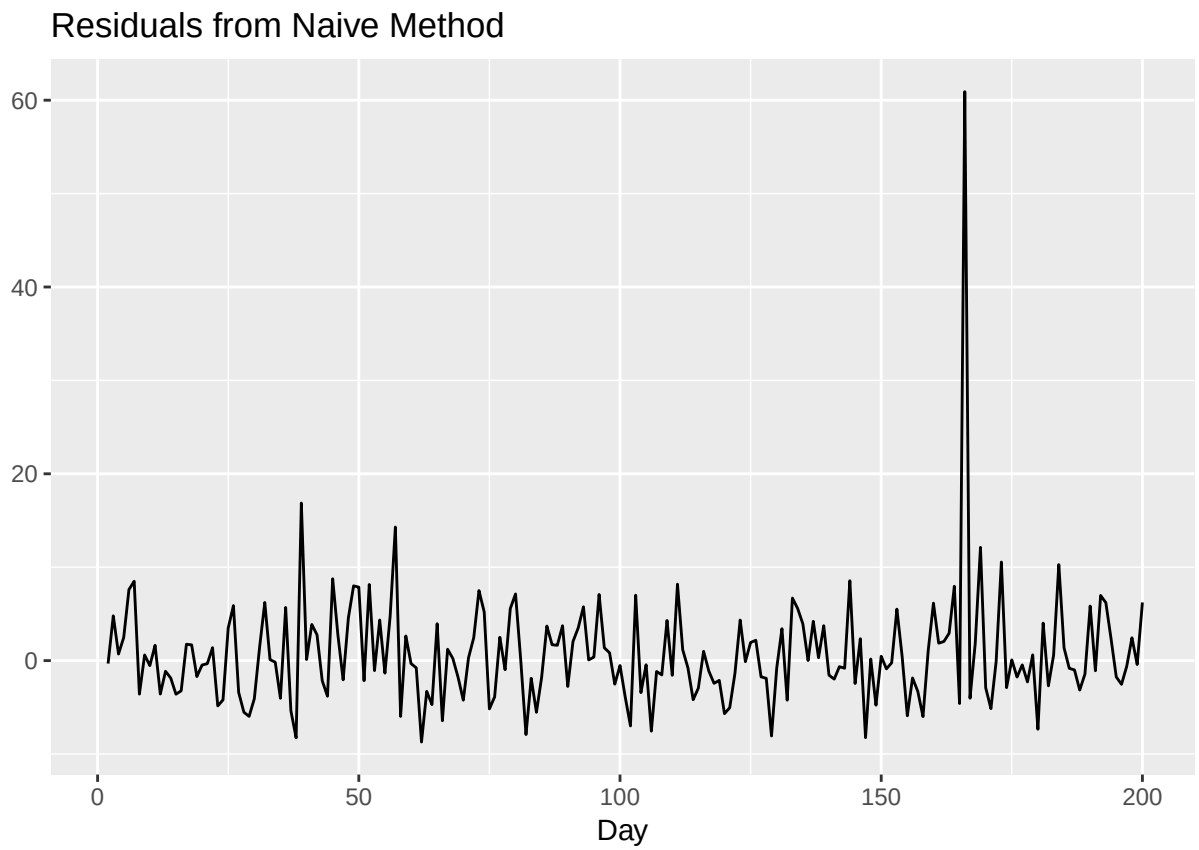
```
autoplot(goog200) +
  xlab("Day") + ylab("Closing Price (US $)") +
  ggtitle("Google Stock (Daily Ending 6Dec13)")
```



In the graph above the large jump at day 166 corresponds to 18Oct13 when the price jumped 12% due to unexpectedly strong third quarter results.

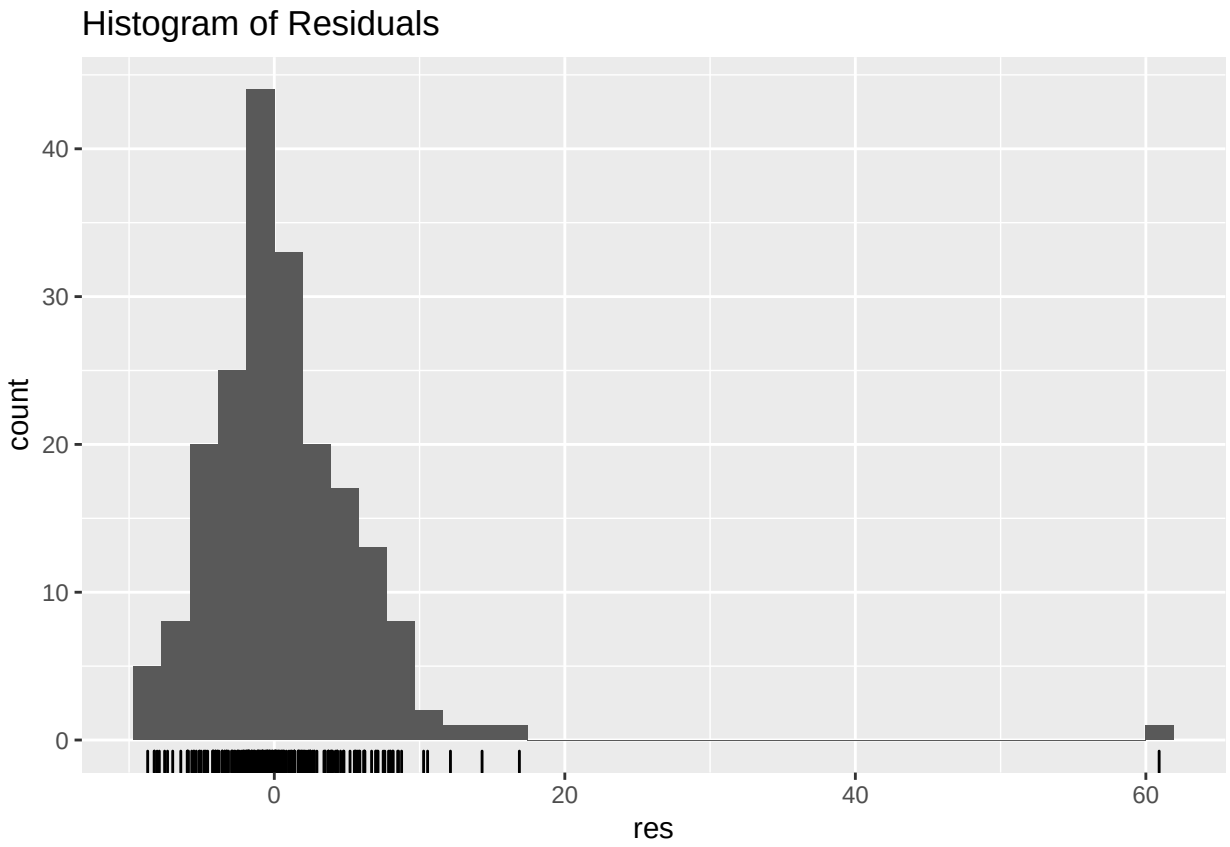
```
# get residuals
res <- residuals(naive(goog200))

# plot
autoplot(res) +
  xlab("Day") + ylab("") +
  ggtitle("Residuals from Naive Method")
```

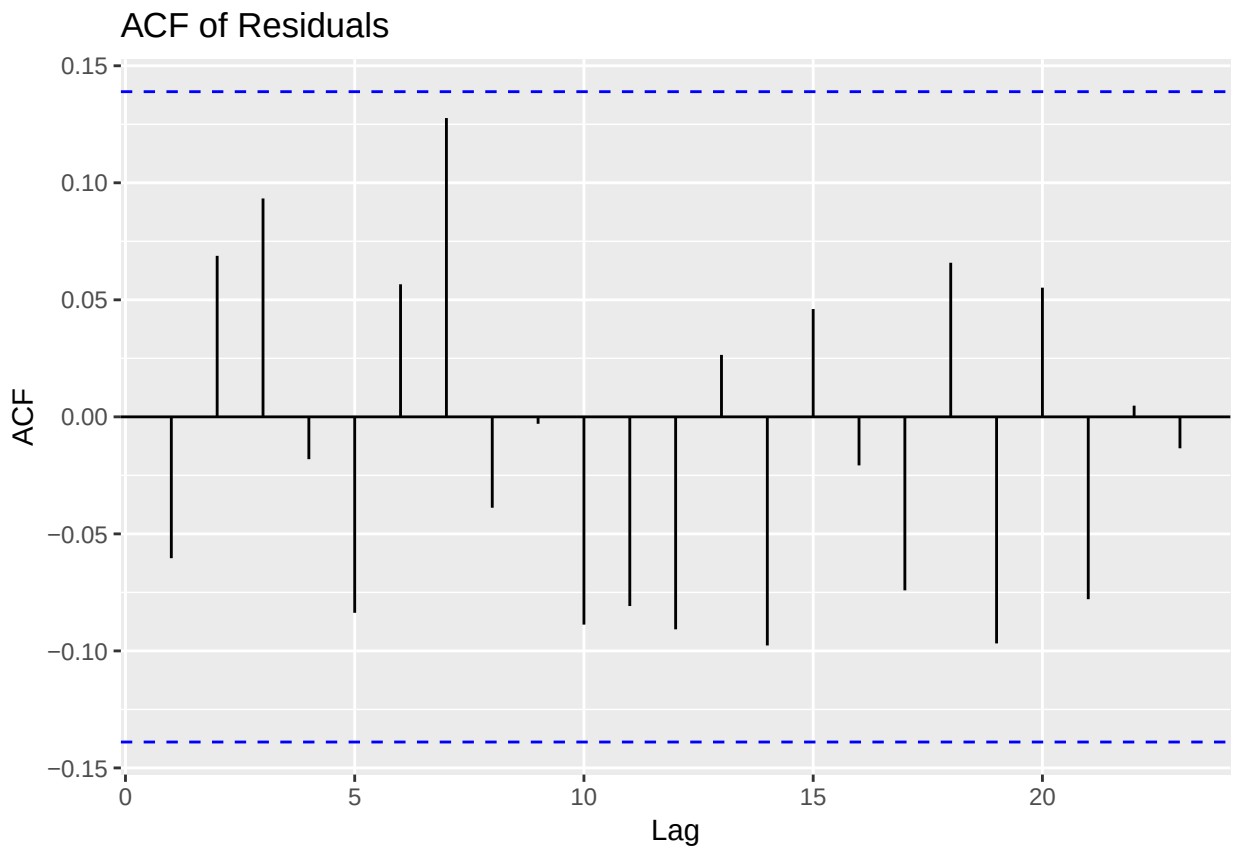



```
gghistogram(res) + ggtitle("Histogram of Residuals")
```

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




```
ggAcf(res) + ggtitle("ACF of Residuals")
```



These graphs show that the naive method produces forecasts that appear to account for all available information. The mean of the residuals is close to 0 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 one outlier, and therefore the residuals variance can be treated as a constant. This can also be seen on the histogram of the residuals, which suggests that the residuals may not be normal - the right tail is too long even when the outlier is ignored.

Consequently, forecasts from this method will probably be quite good, but prediction intervals that are computed assuming a normal distribution may be inaccurate.

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 seperately.

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 tests are done, it is likely that at least one test will give a false positive. This would lead us to believe that the residuals have some remaining autocorrelation, when in fact they do not.

In order to fix this problem, we test whether the first h 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 suitcast containing a number of items.

One such test is the **Box-Pierce test**, based on the following statistic:

$$Q = T \sum_{k=1}^{h} r_k^2$$

where h is the maximum lag being considered and T is the number of observations. If each r_k is close to 0, then Q will be small. If some r_k values are large, then Q will be large. Its suggested to use $h = 10$ for nonseasonal data and $h = 2m$ for seasonal data, where m is the period of seasonality.

This test is not good when h is large, so if these values are larger than $\frac{T}{5}$, then use $h = \frac{T}{5}$.

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

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

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

If the autocorrelations did come from a white noise series, then both Q and Q^* would have a χ^2 distribution with $h - K$ degrees of freedom, where K is the number of parameters in the model.

If they are calculated from raw data (rather than the residuals from a model), then set $K = 0$.

For the google stock price example, the naive model has no parameters, so $K = 0$ in that case also:

```
# box pierce test
Box.test(res, lag=10, fitdf = 0)
```

```
##
##  Box-Pierce test
##
## data:  res
## X-squared = 10.611, df = 10, p-value = 0.3886
```

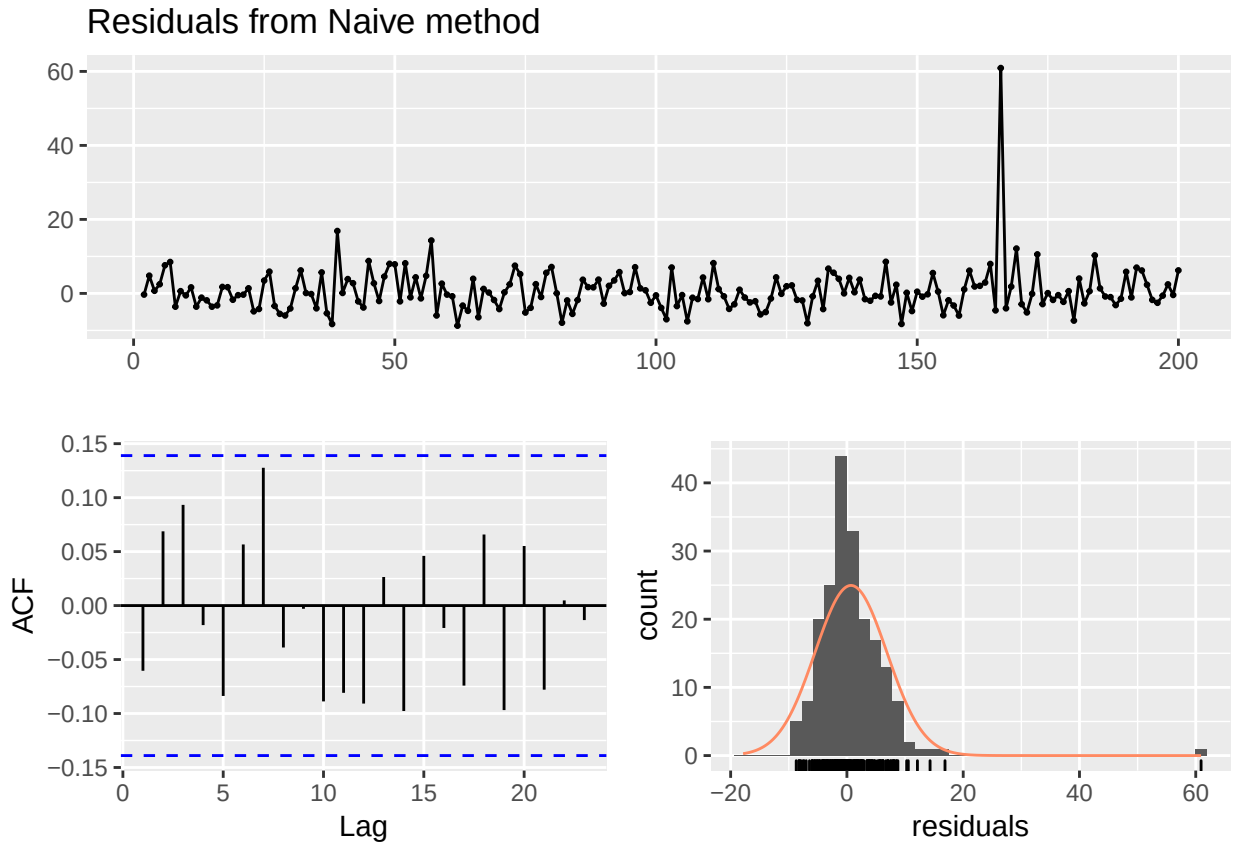
```
# box ljung test
Box.test(res, lag=10, fitdf = 0, type = "Lj")
```

```
##
##  Box-Ljung test
##
## data:  res
## X-squared = 11.031, df = 10, p-value = 0.3551
```

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

All of these methods are packaged into one R function checkresiduals() which will produce a time plot, ACF plot, and a histogram of the residuals with an overlaid normal distribution for comparison, and do a Ljung-Box test with the correct DOF

```
checkresiduals(naive(goog200))
```




```
##
##  Ljung-Box test
##
## data:  Residuals from Naive method
## Q* = 11.031, df = 10, p-value = 0.3551
##
## Model df: 0.   Total lags used: 10
```