The forecaster's toolbox (Continuation_Chap 3)

Forecasting: principles and practice (2nd edition)

book by Rob Hyndman and George Athanasopoulos
slides by Peter Fuleky and modified by Joseph ALBA

Some simple forecasting methods

From Lecture 2:

Some forecasting methods are very simple and surprisingly effective. Here are four methods that we will use as benchmarks for other forecasting methods.

1. Average method
2. Naive method
3. Seasonal naive method
4. Drift method

Continuation of Chapter 3

Transformations and adjustments

Adjusting the historical data can often lead to a simpler forecasting model. Let's look at four kinds of adjustments:

1. mathematical transformations,
2. calendar adjustments,
3. population adjustments, and
4. inflation adjustments.

The purpose of these transformations and adjustments is 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.

Simpler patterns usually lead to more accurate forecasts!

Mathematical transformations

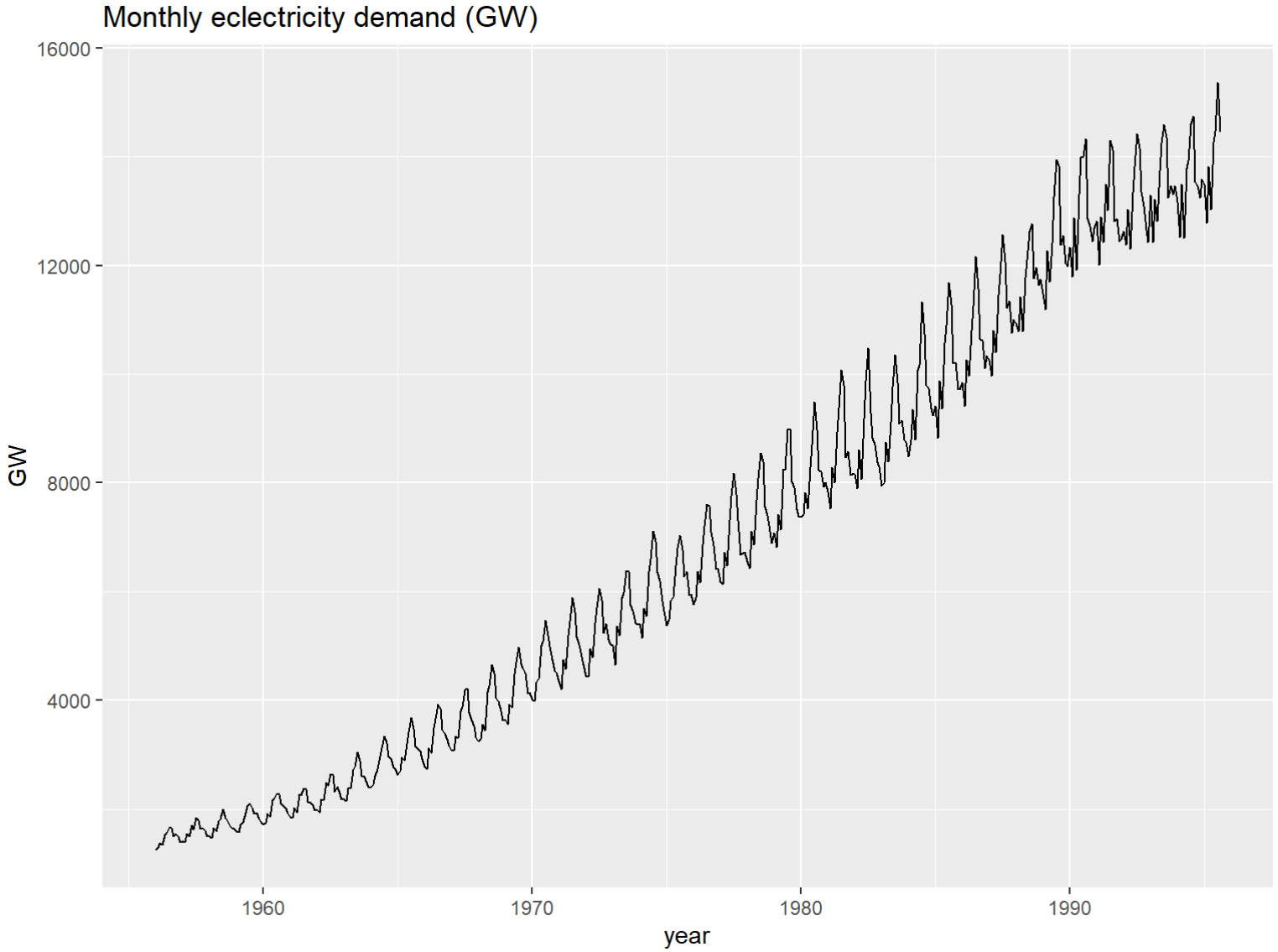
If the data show variation that increases or decreases with the level of the series, then a log-transformation can be useful.

If we denote the original observations as $y_1, \ldots, y_T$ and the transformed observations as $w_1, \ldots, w_T$, then $w_t = \log(y_t)$.

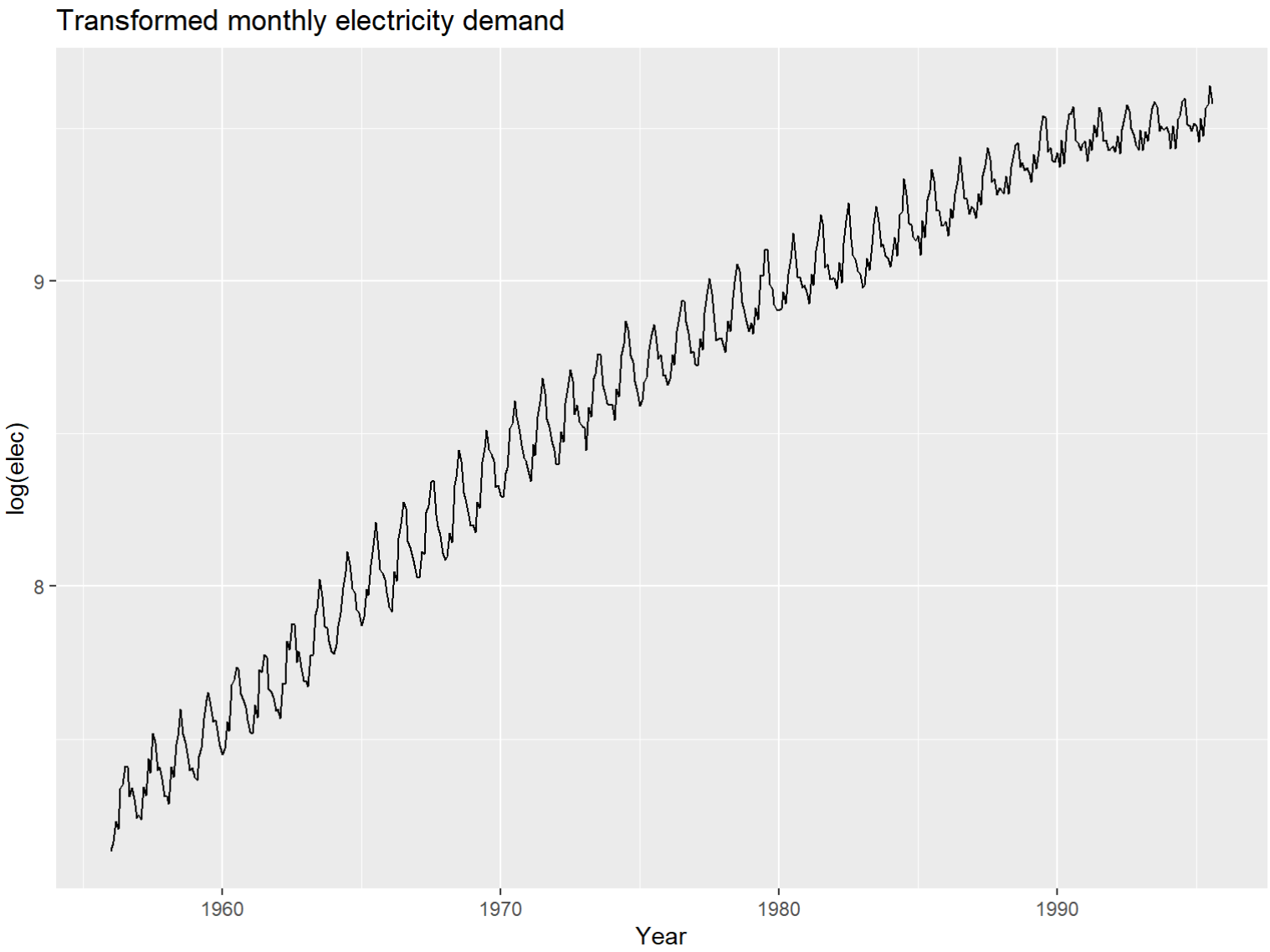
Logarithms are useful because they are interpretable: changes in a log value are relative (or percentage) changes on the original scale.

Another useful feature of log transformations is that they constrain the forecasts to stay positive on the original scale.

```
autoplot(elec) +
ggtitle("Monthly eclectricity demand (GW)") +
  xlab("year") + ylab("GW")
```



```
autoplot(log(elec)) + ylab("log(elec)") +
 xlab("Year") + ggtitle("Transformed monthly electricity demand")
```



A useful family of transformations that includes logarithms and power transformations is the family of "Box-Cox transformations", which depend on the parameter λ and are defined as follows:

$$w_t = \begin{cases} \log(y_t) & \text{if } \lambda = 0; \\ (y_t^\lambda - 1)/\lambda & \text{otherwise.} \end{cases}$$

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 Box-Cox transformation is given by

$$y_t = \begin{cases} \exp(w_t) & \text{if } \lambda = 0; \\ (\lambda w_t + 1)^{1/\lambda} & \text{otherwise.} \end{cases}$$

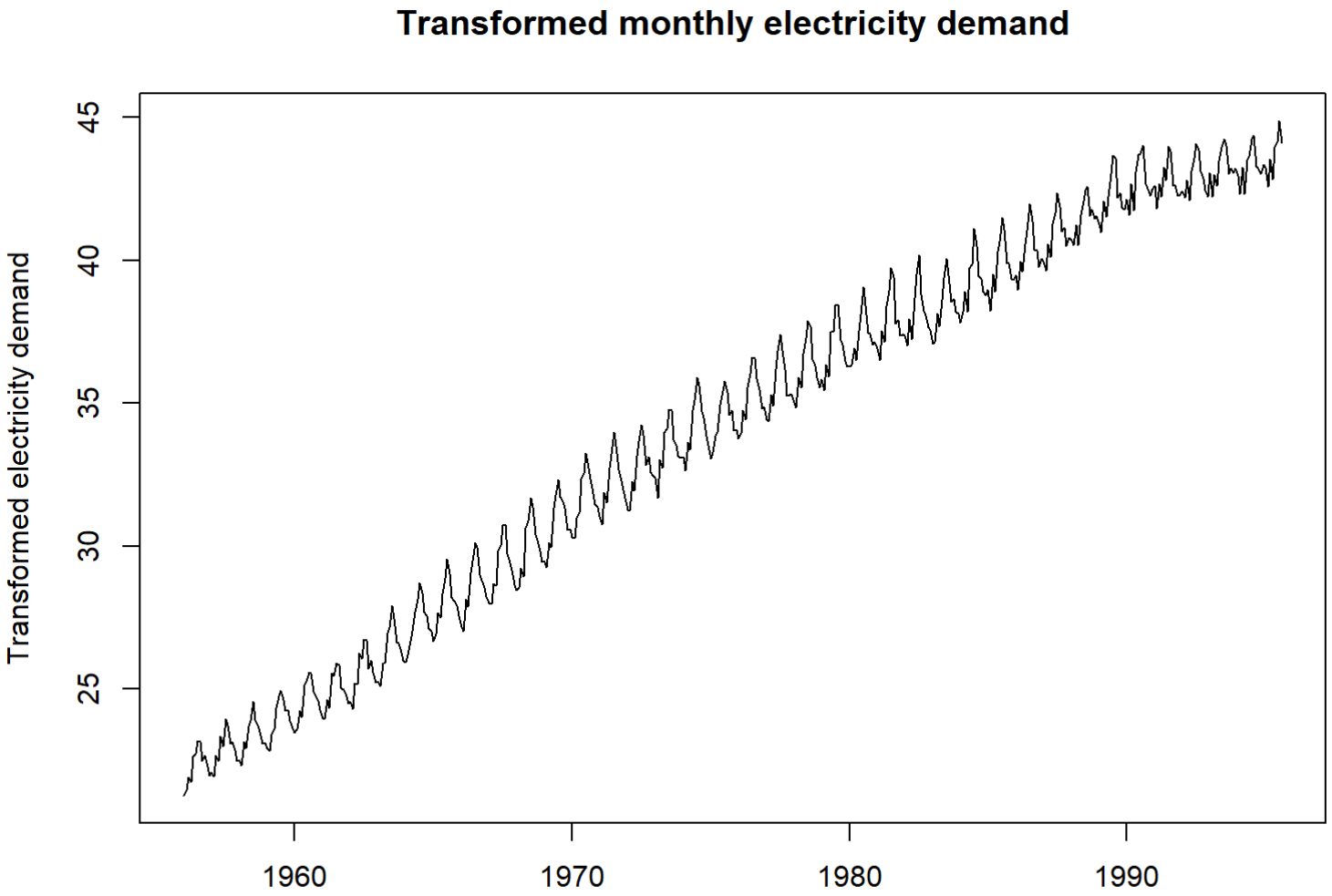
Features of power transformations

- 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 of λ. It makes explanations easier.
- 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.

See the impact of different λ values

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) # = 0.27;
plot(BoxCox(elec,lambda), ylab="Transformed electricity demand",
 xlab="", main="Transformed monthly electricity demand")
```


Transformed monthly electricity demand
Transformed electricity demand
25
30
35
40
45
1960
1970
1980
1990

Bias adjustments

- One issue with using mathematical transformations such as Box-Cox transformations is that the backtransformed forecast will not be the mean of the forecast distribution.
- In fact, it will usually be the *median* of the forecast distribution (assuming that the distribution on the transformed space is symmetric).
- For many purposes, this is acceptable.
- Occasionally the mean forecast is required (e.g., you may wish to add up sales forecasts from various regions to form a forecast for the whole country). But medians do not add up, whereas means do.

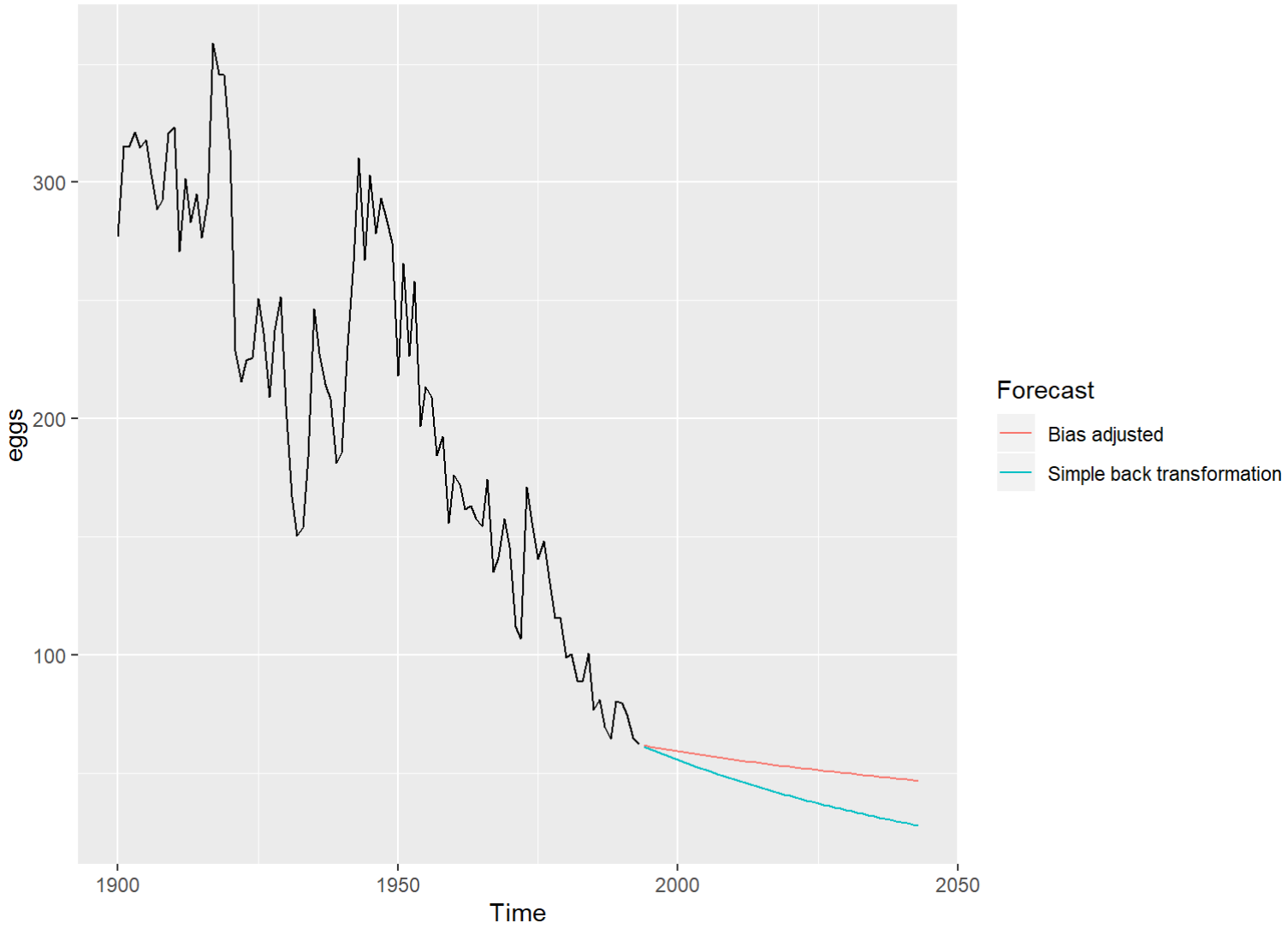
For a Box-Cox transformation, the back-transformed mean is given by

$$y_t = \begin{cases} \exp(w_t)[1 + \frac{\sigma_h^2}{2}] & \text{if } \lambda = 0; \\ (\lambda w_t + 1)^{1/\lambda}[1 + \frac{\sigma_h^2(1-\lambda)}{2(\lambda w_t + 1)^2}] & \text{otherwise.} \end{cases}$$

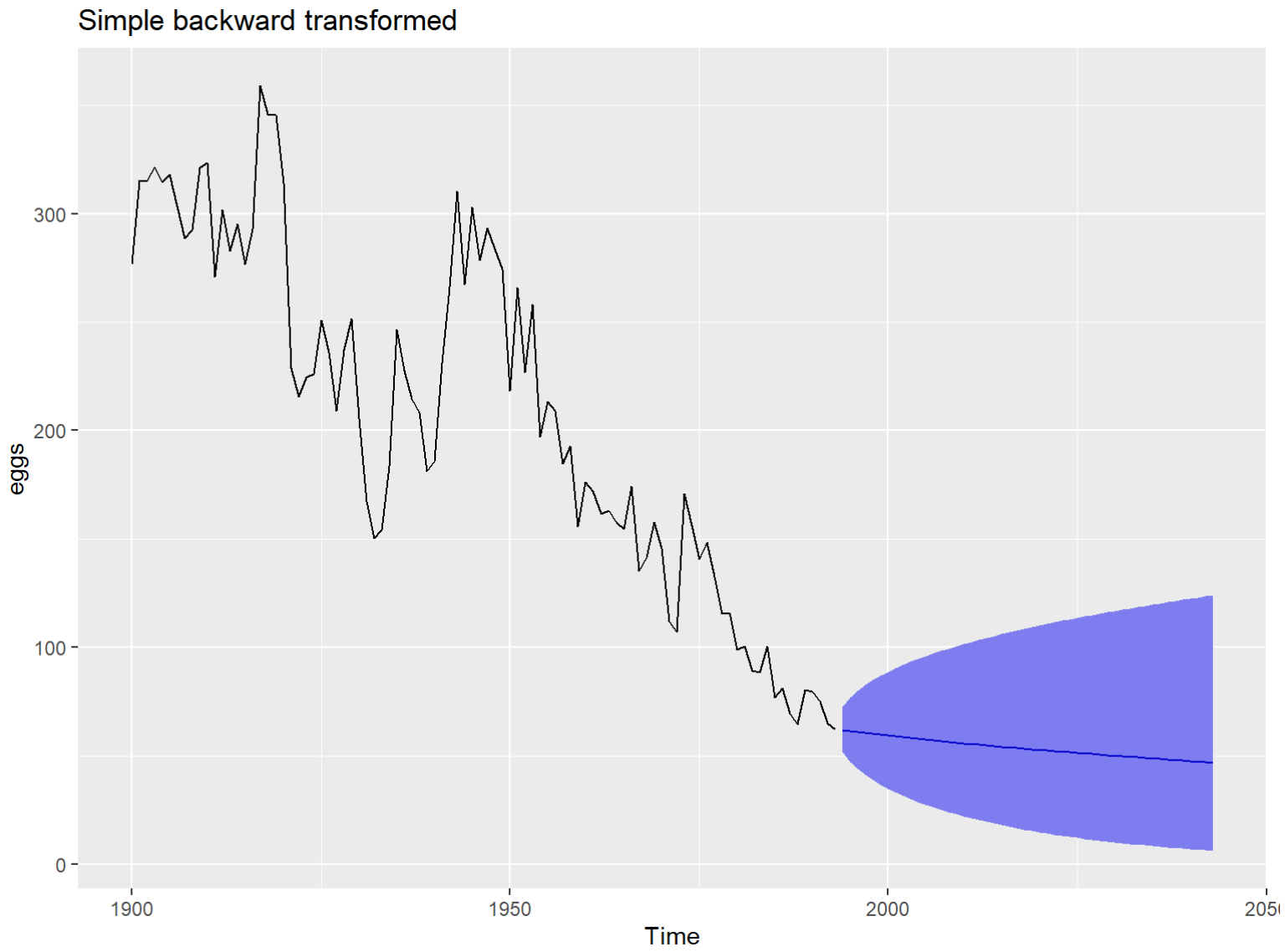
where σ_h^2 is the h-step forecast variance. The larger the forecast variance, the bigger the difference between the mean and the median.

The difference between the simple back-transformed forecast and the mean is called the "bias". Graphically:

```
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) +
forecast::autolayer(fc$mean, series="Simple back transformation") +
forecast::autolayer(fc2$mean, series="Bias adjusted") +
guides(colour=guide_legend(title="Forecast"))
```


eggs
300
200
100
1900
1950
2000
2050
Time
Forecast
Bias adjusted
Simple back transformation

```
autoplot(fc2) +
  ggtitle("Simple backward transformed")
```



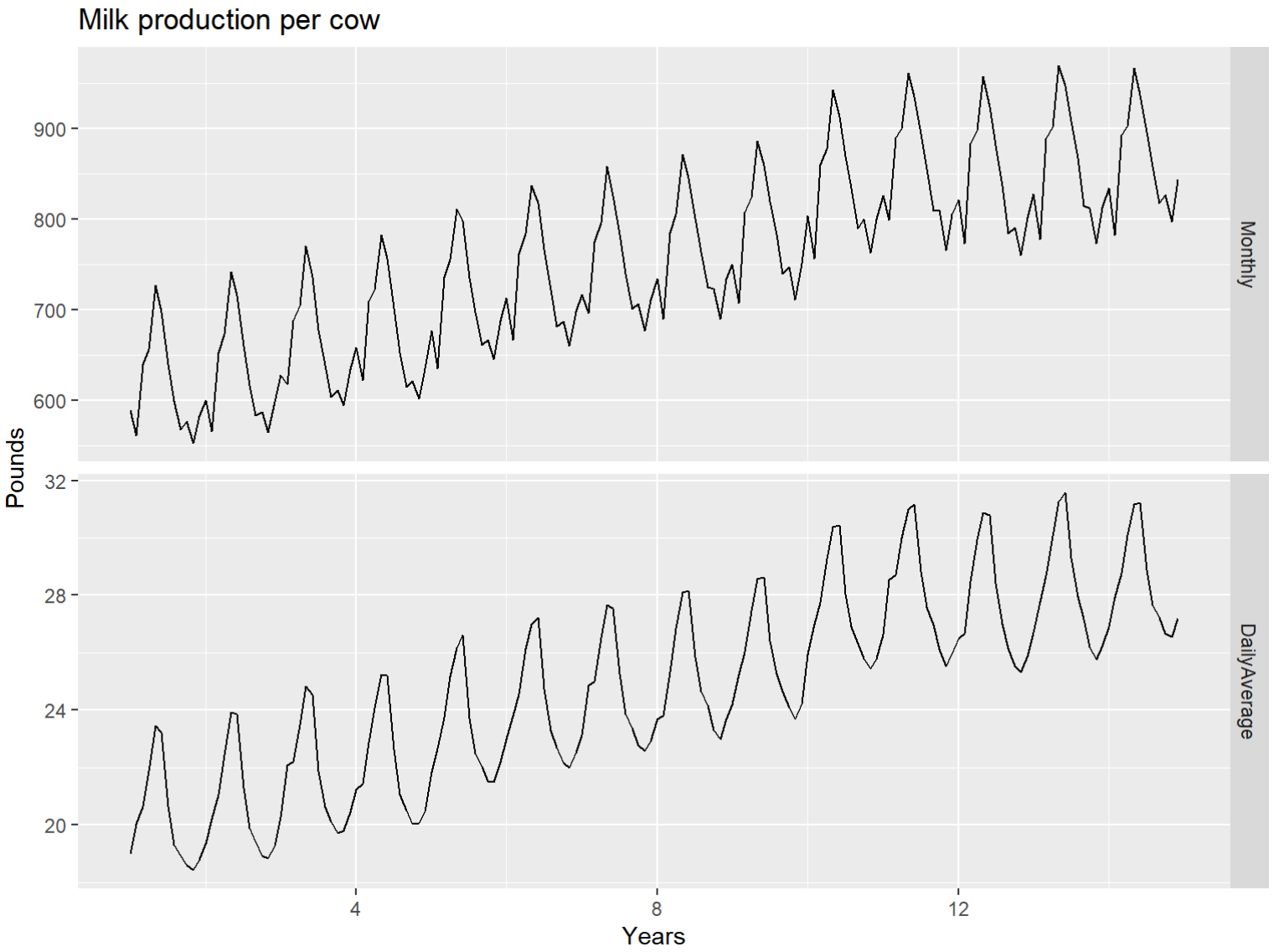
Calendar adjustments

Some variation seen in seasonal data may be due to simple calendar effects. In such cases, it is usually much easier to remove the variation before fitting a forecasting model.

If you are studying monthly milk production on a farm, then there will be variation between the months simply because of the different numbers of days in each month in addition to seasonal variation across the year.

The *monthdays* function will compute the number of days in each month.

```
dframe <- cbind(Monthly = milk, DailyAverage=milk/monthdays(milk))
autoplot(dframe, facet=TRUE) + xlab("Years") + ylab("Pounds") +
ggtitle("Milk production per cow")
```


Milk production per cow
Pounds
900
800
700
600
32
28
24
20
Monthly
DailyAverage
4
8
12
Years

The simpler seasonal pattern in the average daily production plot compared to the average monthly production plot is because we effectively removed the variation due to the different month lengths.

Population adjustments

Any data that are affected by population changes can be adjusted to give per-capita data. Consider the data per person (or per thousand or per million people) rather than the total.

- If you are studying the number of hospital beds in a particular region over time, the results are much easier to interpret if you remove the effect of population changes by considering number of beds per thousand people.
- Then you can see if there have been real increases in the number of beds, or whether the increases are entirely due to population increases.
- It is possible for the total number of beds to increase, but the number of beds per thousand people to decrease (population is increasing faster than the number of hospital beds.)
- For most data that are affected by population changes, it is best to use per-capita data rather than the totals.

Inflation adjustments

Data that are affected by the value of money are best adjusted before modelling.

- A $200,000 house this year is not the same as a $200,000 house twenty years ago. For this reason, financial time series are usually adjusted so all values are stated in dollar values from a particular year. For example, the house price data may be stated in year 2000 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 $x_t = y_t/z_t * z_{2000}$ gives the adjusted house price at year 2000 dollar values.
- Price indexes are often constructed by government agencies. For consumer goods, a common price index is the Consumer Price Index (or CPI).

Residual diagnostics

A residual in forecasting is the difference between an observed value and its forecast based on other observations: $e_t = y_t - \hat{y}_t$.

For time series forecasting, a residual is based on one-step forecasts; that is $\hat{y}_t$ is the forecast of y_t based on observations $y_1, \ldots, y_{t-1}$.

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

- 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.
- The residuals have zero mean. If the residuals have a mean other than zero, then the forecasts are biased.

Any forecasting method that does not satisfy these properties can be improved. That does not mean that forecasting methods that satisfy these properties can not be improved.

It is possible to have several forecasting methods for the same data set, all of which satisfy these properties. Checking these properties is important to see if a method is using all available information well, but it is not a good way for selecting a forecasting method.

If either of these two properties is not satisfied, then the forecasting method may 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 address it in the future lecture.

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

- The residuals have constant variance.
- The residuals are normally distributed.

These two properties make the calculation of prediction intervals easier. However, a forecasting method that does not satisfy these properties may not necessarily be improved.

Sometimes applying a transformation such as a logarithm or a square root may assist with these properties, but otherwise there is usually little we can do to ensure your residuals have constant variance and have a normal distribution.

Instead, an alternative approach to finding prediction intervals is necessary and will be discussed later in the semester.

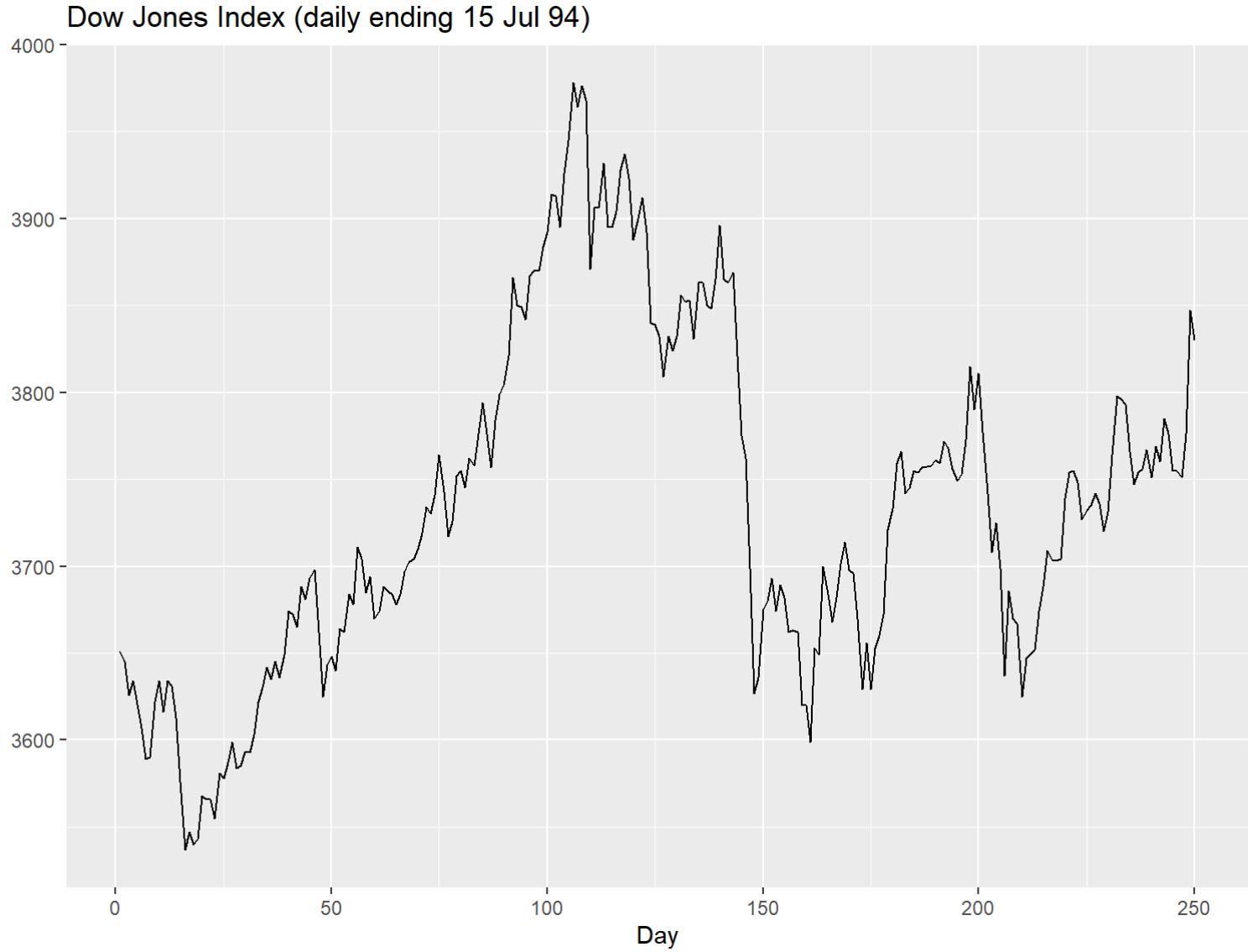
Example: Forecasting the Dow-Jones Index

For stock market indexes, the best forecasting method is often the naive method: 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:

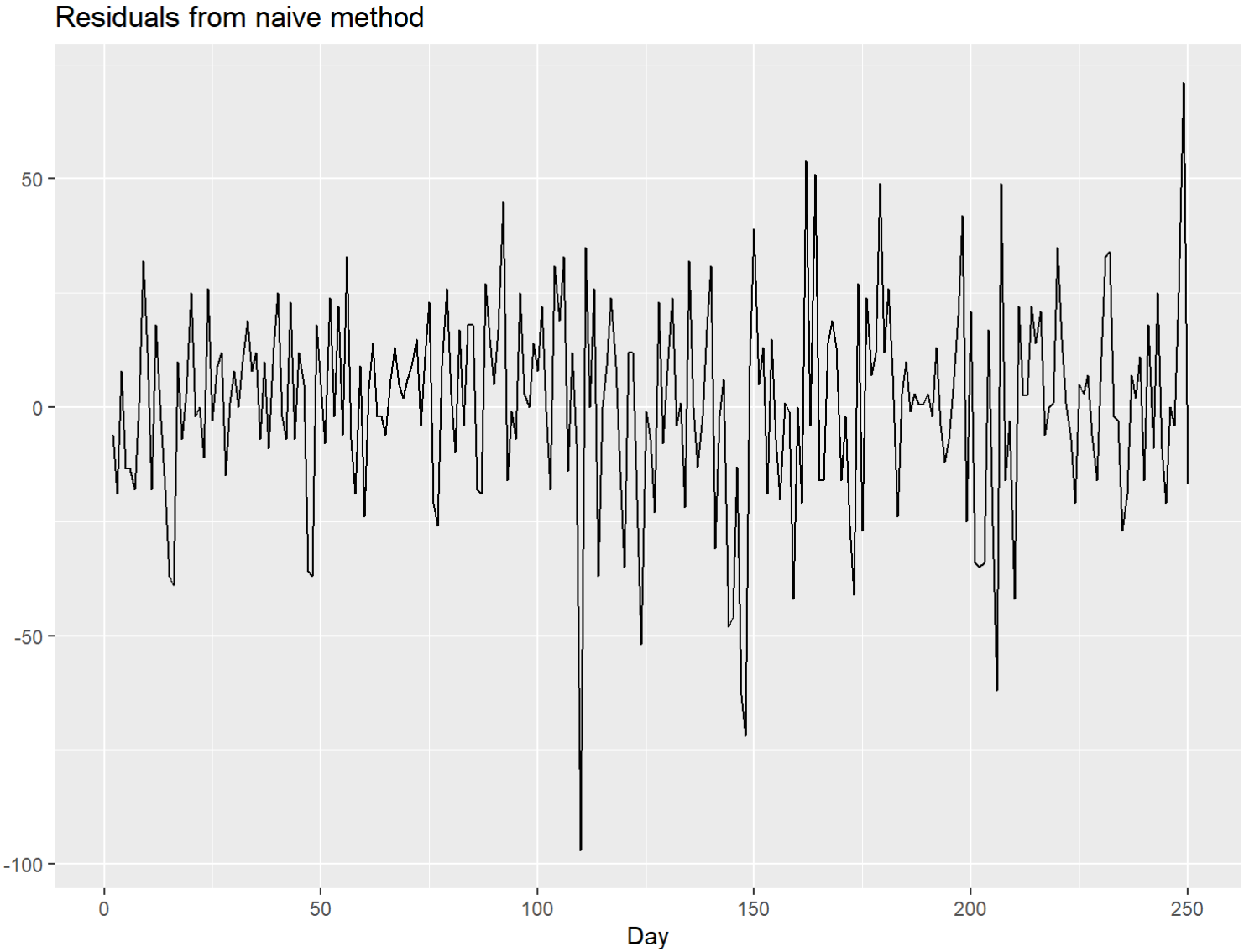
$$e_t = y_t - \hat{y}_t = y_t - y_{t-1}.$$

The following graphs show the Dow Jones Index (DJI), and the residuals obtained from forecasting the DJI with the naive method. These graphs show that the naive method produces forecasts that appear to account for all available information.

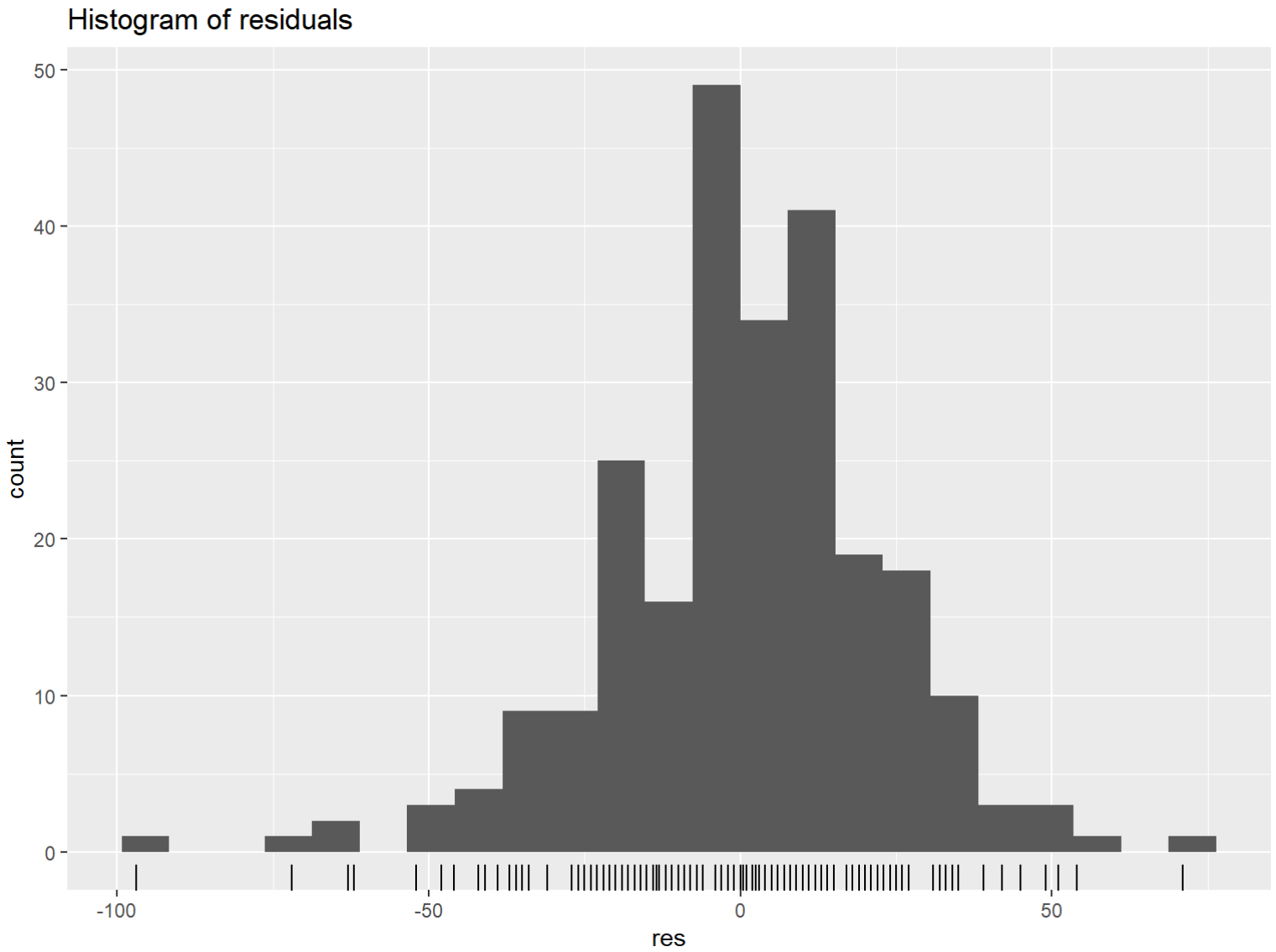
```
dj2 <- window(dj, end=250)
autoplot(dj2) + xlab("Day") + ylab("") +
ggtitle("Dow Jones Index (daily ending 15 Jul 94)")
```



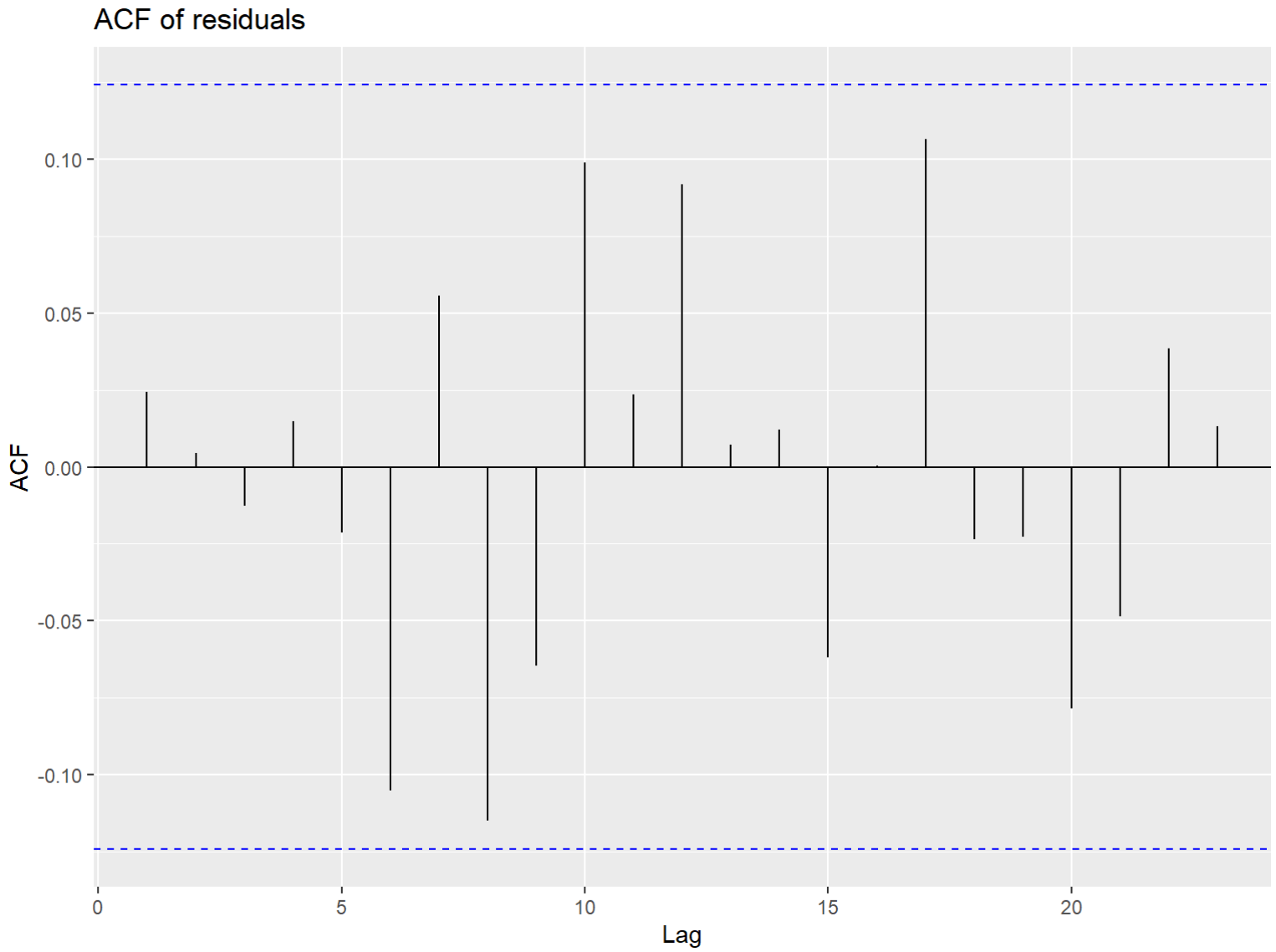
```
res <- residuals(naive(dj2))
autoplot(res) + xlab("Day") + ylab("") +
ggtitle("Residuals from naive method")
```



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



```
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 very 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, so the residual variance can be treated as constant.
- However, the histogram suggests that the residuals may not follow a normal distribution - the left tail looks a little too long. 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 do a more formal test for autocorrelation by considering a whole set of r_k values as a group, rather than treat each one separately.

When we look at the ACF plot to see if each spike is within the required limits, we are implicitly carrying out multiple hypothesis tests, each one with a small probability (5%) 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 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 suitcase 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 number of observations. 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.

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

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

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 $(h-K)$ 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 $K=0$.

Authors' suggestions:

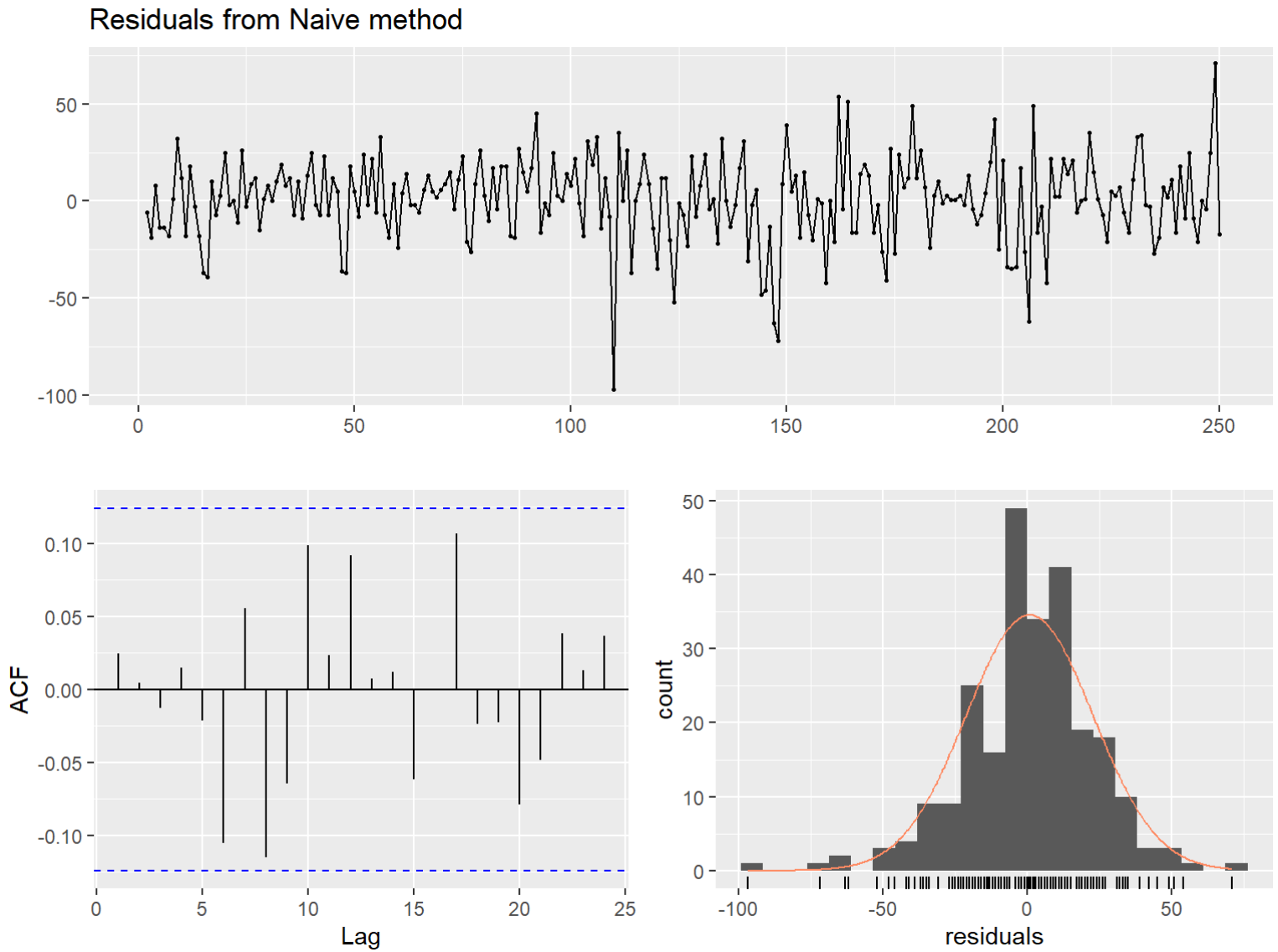
- Use $h = 10$ for non-seasonal data;
- Use $h = 2m$ for seasonal data, with m the period of seasonality;
- However, when h is larger than $T/5$, use $h = T/5$

```
# lag=h and fitdf=K
Box.test(res, lag=10, fitdf=0)
##
##  Box-Pierce test
##
## data:  res
## X-squared = 10.655, df = 10, p-value = 0.385
Box.test(res, lag=10, fitdf=0, type="Lj")
##
##  Box-Ljung test
##
## data:  res
## X-squared = 11.088, df = 10, p-value = 0.3507
```

For the Dow-Jones example, the naive model has no parameters, so $K = 0$. For both Q and Q^*, the results are not significant (i.e., the p-values are relatively large). So the residuals are not distinguishable from white noise series.

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

```
checkresiduals(naive(dj2))
```




```
##
##  Ljung-Box test
##
```

```
## data:  Residuals from Naive method
## Q* = 11.088, df = 10, p-value = 0.3507
##
## Model df: 0.   Total lags used: 10
```

```
##
##  Ljung-Box test
##
## data:  Residuals from Naive method
## Q* = 11.088, df = 10, p-value = 0.3507
##
## Model df: 0.   Total lags used: 10
```

Training and test sets

It is important to evaluate forecast accuracy using genuine forecasts. It is invalid to look at how well a model fits the historical data; the accuracy of forecasts can only be determined by considering how well a model performs on new data that were not used when fitting the model.

When choosing models, it is common to use a portion of the available data for fitting, and use the rest of the data for testing the model. Then the testing data can be used to measure how well the model is likely to forecast on new data.

Training and forecast data

The size of the test set is typically about 20% of the total sample, although this value depends on how long the sample is and how far ahead you want to forecast. The size of the test set should ideally be at least as large as the maximum forecast horizon required.

The following points should be noted.

- A model which fits the data well does not necessarily forecast well.
- A "perfect" fit can always be obtained by using a model with enough parameters.
- Over-fitting a model to data is as bad as failing to identify the systematic pattern in the data.

Some references describe the test set as the "hold-out set" because these data are "held out" of the data used for fitting. Other references call the training set the "in-sample data" and the test set the "out-of-sample data".

Functions to subset a time series

The **window** function introduced in Chapter 2 is very useful when extracting a portion of a time series, such as we need when creating training and test sets. In the window function, we specify the start and/or end of the portion of time series required using time values. For example,

```
window(ausbeer, start=1995)
```

Another useful function is **subset** which allows for more types of subsetting. A great advantage of this function is that it allows the use of indices to choose a subset. For example,

```
subset(ausbeer, start=length(ausbeer)-4*5)
```

extracts the last 5 years of observations from **ausbeer** . It also allows extracting all values for a specific season. For example,

```
subset(ausbeer, quarter = 1)
```

extracts the first quarters for all years.

Finally, head and tail are useful for extracting the first few or last few observations. For example, the last 5 years of ausbeer can also be obtained using

```
tail(ausbeer, 4*5)
```

Forecast errors

A forecast "error" is the difference between an observed value and its forecast. Here "error" does not mean a mistake, it means the unpredictable part of an observation. It can be written as

$$e_{T+h} = y_{T+h} - \hat{y}_{T+h|T}$$

where the training data is given by $\{y_1 \dots y_T\}$ and the test data is given by $\{y_{T+1}, y_{T+2} \dots .\}$.

Note that forecast errors are different from residuals in two ways.

1. Residuals are calculated on the training set while forecast errors are calculated on the test set.

2. Residuals are based on one-step forecasts while forecast errors can involve multi-step forecasts.

We can measure forecast accuracy by summarising the forecast errors in different ways.

Scale dependent forecast accuracy measures

Let y_i denote the ith observation and $\hat{y}_i$ denote a forecast of y_i.

The forecast error is $e_i = y_i - \hat{y}_i$, which is on the same scale as the data. Accuracy measures that are based on e_i are therefore *scale-dependent* and cannot be used to make comparisons between series that are on different scales.

The two most commonly used scale-dependent measures are based on the absolute errors or squared errors:

$$\text{Mean absolute error: MAE} = \text{mean}(|e_i|),$$
$$\text{Root mean squared error: RMSE} = \sqrt{\text{mean}(e_i^2)}.$$

A forecast method that:

- minimizes the MAE will lead to forecasts of the median, while
- minimizing the RMSE will lead to forecasts of the mean.

MAE is easier to interpret but less so for the RMSE. Since RMSE are squared, it gives relatively high weight to large errors. RMSE may be more useful if large errors are undesirable.

Percentage errors

The percentage error is given by $p_i = 100e_i/y_i$.

Percentage errors have the advantage of being scale-independent, and so are frequently used to compare forecast performance between different data sets. The most commonly used measure is:

$$\text{Mean absolute percentage error: MAPE} = \text{mean}(|p_i|).$$

Measures based on percentage errors have the disadvantage of having extreme values for $y_i \approx 0$.

Another problem with percentage errors is that they assume a meaningful zero (not satisfied when measuring the accuracy of temperature forecasts on the Fahrenheit or Celsius scales.)

Scaled errors

Scaled errors are an alternative to using percentage errors when comparing forecast accuracy across series on different scales. The errors can be scaled using the **training** MAE from a simple forecast method.

For a non-seasonal time series, a useful way to define a scaled error uses naive forecasts:

$$q_j = \frac{e_j}{\frac{1}{T-1}\sum_{t=2}^{T}|y_t - y_{t-1}|},$$

where $y_t - y_{t-1}$ is the error of the naive forecast.

Because the numerator and denominator both involve values on the scale of the original data, q_j is independent of the scale of the data.

- A scaled error is less than one if it arises from a better forecast than the average naive forecast computed on the training data.
- Conversely, it is greater than one if the forecast is worse than the average naive forecast computed on the training data.

For seasonal time series, a scaled error can be defined using seasonal naive forecasts:

$$q_j = \frac{e_j}{\frac{1}{T-m}\sum_{t=m+1}^{T} |y_t - y_{t-m}|}.$$

The *mean absolute scaled error* is simply

$$\text{MASE} = \text{mean}(|q_j|).$$

Similarly, the *mean squared scaled error* (MSSE) can be defined where the errors (on both the training data and test data) are absolute values and a *root mean squared scaled error* (RMSSE) can be defined where the errors (on the training data and test data) are squared instead of using absolute values. For example, using the naive forecasts:

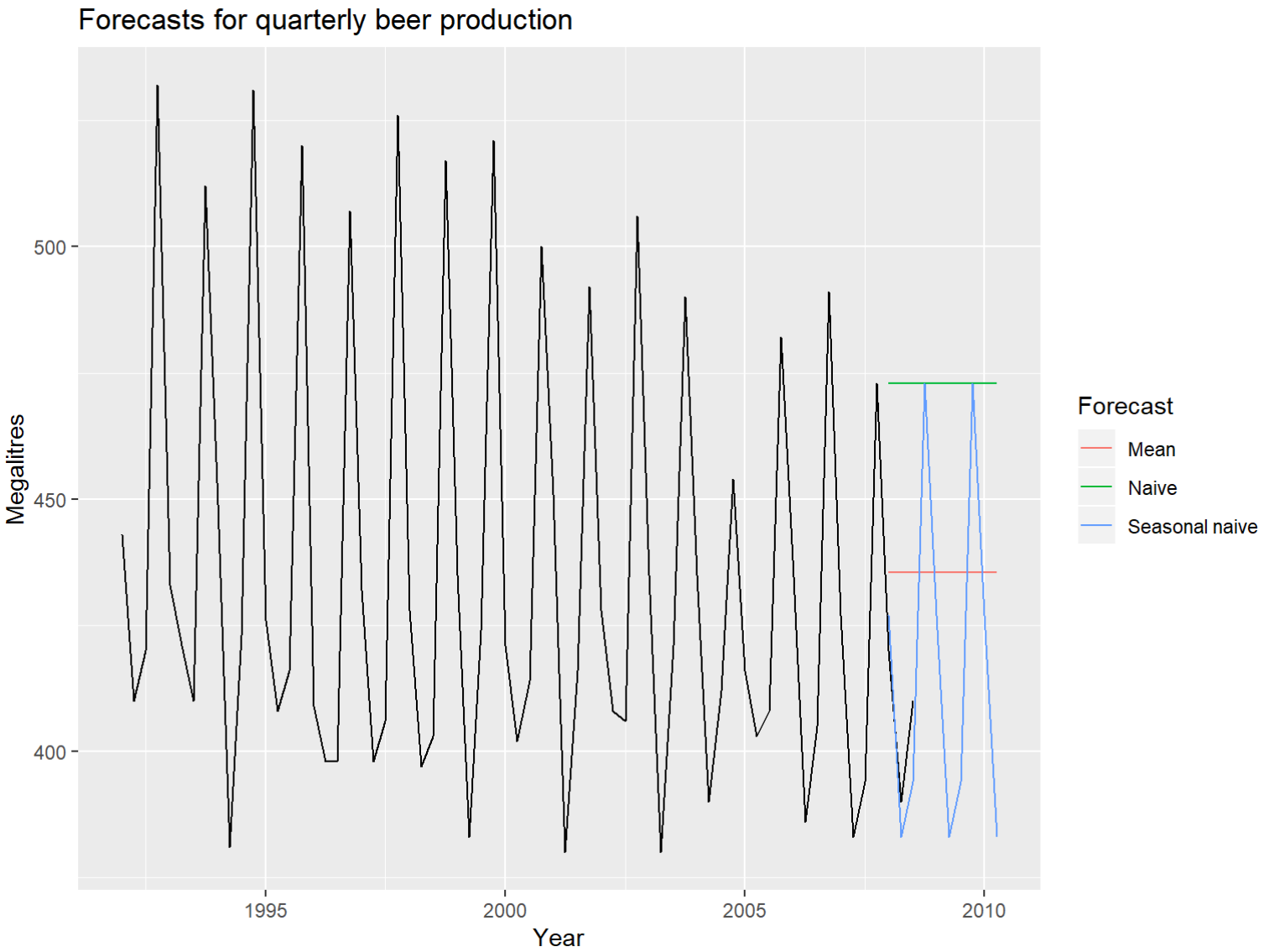
$$q_j = \frac{e_j}{\sqrt{\frac{1}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1})^2}}.$$

$$\text{MSSE} = \text{mean}(q_j^2).$$

$$\text{RMSSE} = (\sqrt{\text{mean}(q_j^2)}).$$

Example 1

```
beer2 <- window(ausbeer,start=1992,end=c(2007,4))
beerfit1 <- meanf(beer2,h=10)
beerfit2 <- rwf(beer2,h=10)
beerfit3 <- snaive(beer2,h=10)
autoplot(window(ausbeer, start=1992)) +
forecast::autolayer(beerfit1$mean, series="Mean") +
forecast::autolayer(beerfit2$mean, series="Naive") +
forecast::autolayer(beerfit3$mean, series="Seasonal naive") +
xlab("Year") + ylab("Megalitres") +
ggtitle("Forecasts for quarterly beer production") +
guides(colour=guide_legend(title="Forecast"))
```


Forecasts for quarterly beer production
Megalitres
500
450
400
1995
2000
2005
2010
Year
Forecast
Mean
Naive
Seasonal naive

```
beer3 <- window(ausbeer, start=2008)
accuracy(beerfit1, beer3)
```

```
##                     ME     RMSE      MAE        MPE     MAPE     MASE
## Training set   0.00000 43.62858 35.23438 -0.9365102 7.886776 2.463942
## Test set     -28.70833 31.30054 28.70833 -7.1614514 7.161451 2.007576
##                    ACF1 Theil's U
## Training set -0.1091511        NA
## Test set     -0.5952381   1.43425
```

```
accuracy(beerfit2, beer3)
```

```
##                      ME     RMSE      MAE         MPE     MAPE     MASE
## Training set   0.4761905 65.31511 54.73016  -0.9162496 12.16415 3.827284
## Test set     -66.3333333 67.49568 66.33333 -16.4223175 16.42232 4.638695
##                    ACF1 Theil's U
## Training set -0.2409829        NA
## Test set     -0.5952381  2.902736
```

```
accuracy(beerfit3, beer3)
```

```
##                      ME     RMSE  MAE        MPE     MAPE      MASE
## Training set -2.133333 16.78193 14.3 -0.5537713 3.313685 1.0000000
## Test set      5.333333 10.86278 10.0  1.3435481 2.454659 0.6993007
##                    ACF1 Theil's U
## Training set -0.28763330        NA
## Test set     -0.01033912 0.5035966
```

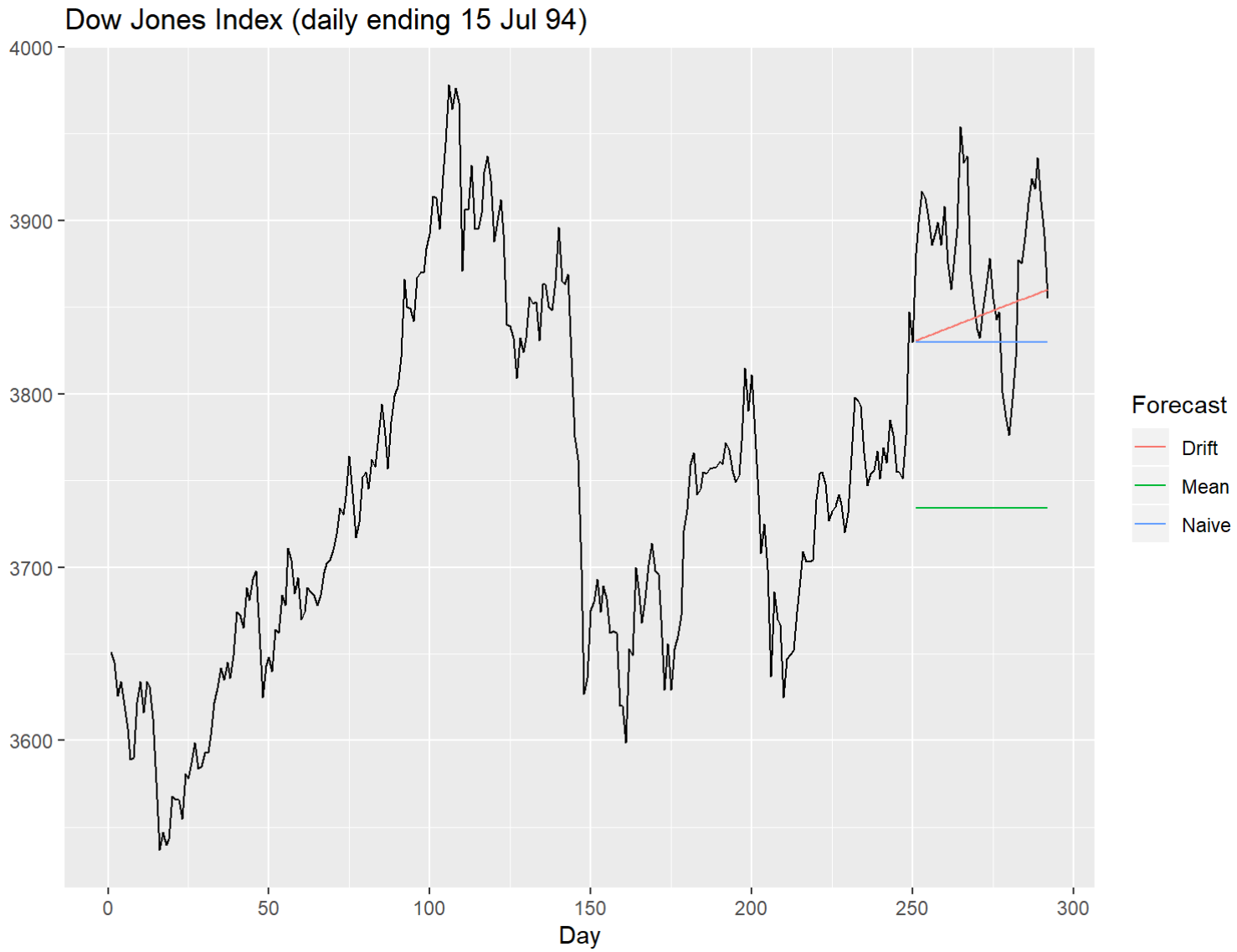
Here are the combined results:

```
beer3 <- window(ausbeer, start=2008)
rbind(accuracy(meanf(beer2,h=10),beer3)[2,c(2,3,5,6)],
accuracy(rwf(beer2,h=10), beer3)[2,c(2,3,5,6)],
accuracy(snaive(beer2,h=10), beer3)[2,c(2,3,5,6)])
```

```
##          RMSE      MAE      MAPE      MASE
## [1,] 31.30054 28.70833  7.161451 2.0075758
## [2,] 67.49568 66.33333 16.422318 4.6386946
## [3,] 10.86278 10.00000  2.454659 0.6993007
```

Example 2

```
dj2 <- window(dj, end=250)
djfc1 <- meanf(dj2, h=42)
djfc2 <- rwf(dj2, h=42)
djfc3 <- rwf(dj2, drift=TRUE, h=42)
autoplot(dj) +
forecast::autolayer(djfc1$mean, series="Mean") +
forecast::autolayer(djfc2$mean, series="Naive") +
forecast::autolayer(djfc3$mean, series="Drift") +
xlab("Day") + ylab("") +
ggtitle("Dow Jones Index (daily ending 15 Jul 94)") +
guides(colour=guide_legend(title="Forecast"))
```


Dow Jones Index (daily ending 15 Jul 94)
4000
3900
3800
3700
3600
0
50
100
150
200
250
300
Day
Forecast
Drift
Mean
Naive

```
dj3 <- window(dj, start=251)
accuracy(djfc1, dj3)
```

```
##                       ME      RMSE       MAE          MPE     MAPE
## Training set 6.553874e-14  98.71439  80.56688 -0.06934572 2.151962
## Test set     1.424185e+02 148.23574 142.41848  3.66304611 3.663046
##                   MASE      ACF1 Theil's U
## Training set 4.920567 0.9719593        NA
## Test set     8.698111 0.8255136  6.072223
```

```
accuracy(djfc2, dj3)
```

```
##                     ME     RMSE      MAE        MPE      MAPE     MASE
## Training set  0.7188755 22.00014 16.37349 0.01749683 0.4380973 1.000000
## Test set     46.4404762 62.02846 54.44048 1.18683463 1.3979371 3.324915
##                    ACF1 Theil's U
## Training set 0.02446257        NA
## Test set     0.82551365   2.54582
```

```
accuracy(djfc3, dj3)
```

```
##                        ME     RMSE      MAE          MPE      MAPE
## Training set 1.278395e-13 21.98839 16.34525 -0.001766862 0.4373707
## Test set     3.098465e+01 53.69767 45.72743  0.787547945 1.1757748
##                   MASE       ACF1 Theil's U
## Training set 0.9982752 0.02446257        NA
## Test set     2.7927719 0.83881869  2.203742
```

Here the results were combined into a single matrix.

```
dj3 <- window(dj, start=251)
rbind(accuracy(meanf(dj2,h=42), dj3)[2,c(2,3,5,6)],
accuracy(rwf(dj2,h=42), dj3)[2,c(2,3,5,6)],
accuracy(rwf(dj2,drift=TRUE,h=42), dj3)[2,c(2,3,5,6)])
```

```
##           RMSE       MAE     MAPE     MASE
## [1,] 148.23574 142.41848 3.663046 8.698111
## [2,]  62.02846  54.44048 1.397937 3.324915
## [3,]  53.69767  45.72743 1.175775 2.792772
```

Time series Cross-validation

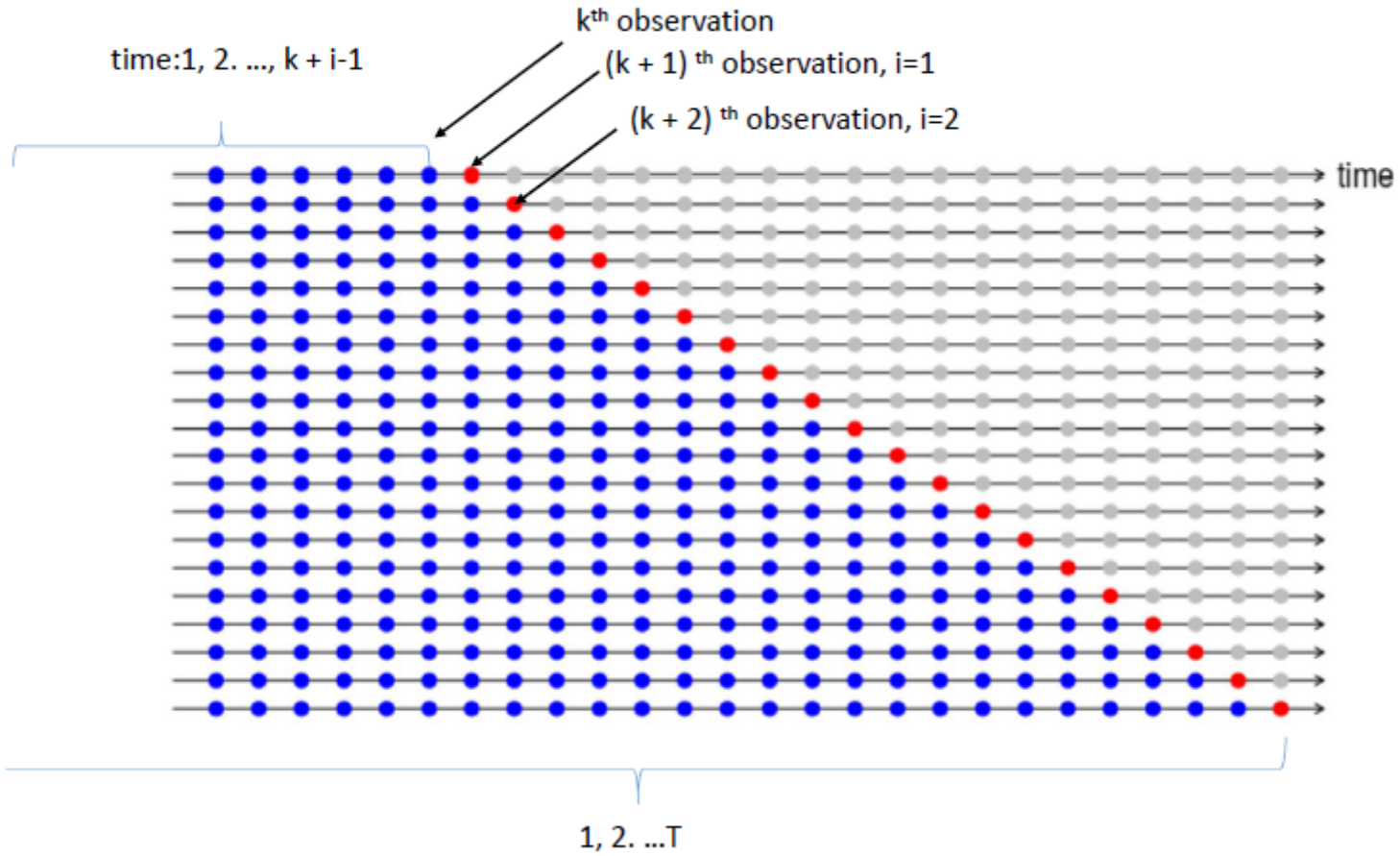
A more sophisticated version of training/test sets is cross-validation. It may not be possible to get a reliable forecast based on a very small training set, so the earliest observations are not considered as test sets. Suppose k observations are required to produce a reliable forecast.

The cross-validation procedure works as follows:

1. Select the observation at time $k+i$ for the test set, and use the observations at times $1, 2, \ldots, k+i-1$ to estimate the forecasting model. Compute the error on the forecast for time $k+i$.
2. Do the above step for $i = 1, 2, \ldots$ until all the observations are used.
3. Compute the forecast accuracy measures based on the errors obtained.

This procedure is sometimes known as a "rolling forecasting origin" because the "origin" $(k+i-1)$ at which the forecast is based rolls forward in time.

The following diagram illustrates the series of training and test sets, where the blue observations form the training sets, and the red observations form the test sets.

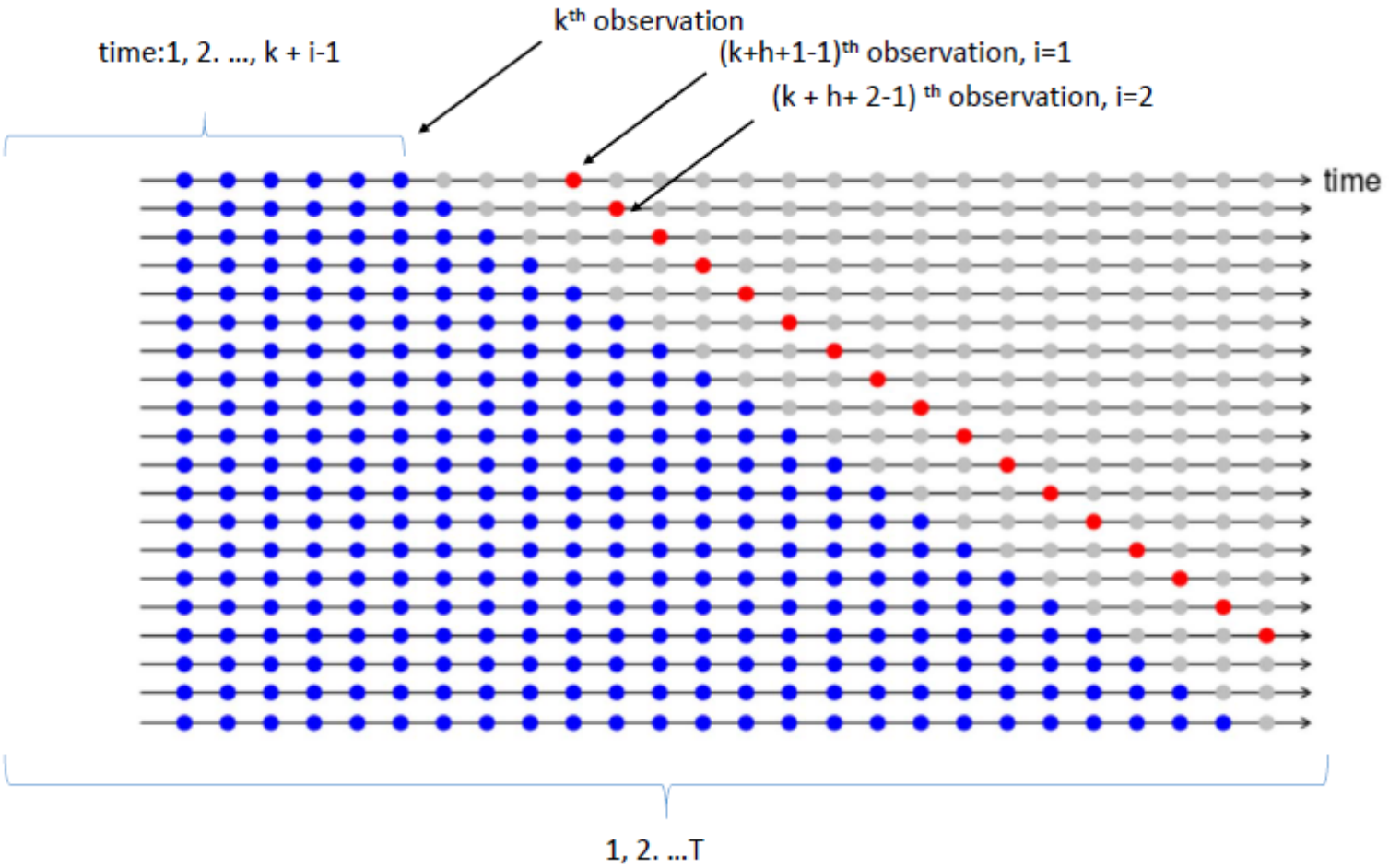


Time series cross validation with h=1.

With time series forecasting, one-step forecasts may not be as relevant as multi-step forecasts. In this case, the cross-validation procedure based on a rolling forecasting origin can be modified to allow multi-step errors to be used. Suppose we are interested in models that produce good h-step-ahead forecasts.

1. Select the observation at time $k+h+i-1$ for the test set, and use the observations at times $1, 2, \ldots, k+i-1$ to estimate the forecasting model. Compute the h-step error on the forecast for time $k+h+i-1$.
2. Do the above step for $i = 1, 2, \ldots$ until all the observations are used.
3. Compute the forecast accuracy measures based on the errors obtained.

When $h = 1$, this gives the same procedure as outlined above.



Time series cross validation with h=4.

Time series cross validation is implemented with the tsCV function. In the following example, we compare the residual RMSE with the RMSE obtained via time series cross-validation.

```
e <- tsCV(dj, rwf, drift=TRUE, h=1)
sqrt(mean(e^2, na.rm=TRUE))
```

```
## [1] 22.68249
```

```
#> [1] 22.7
sqrt(mean(residuals(rwf(dj, drift=TRUE))^2, na.rm=TRUE)) #> [1] 22.5
```

```
## [1] 22.49681
```

As expected, the RMSE from the residuals is smaller, as the corresponding "forecasts" are based on a model fitted to the entire data set, rather than being true forecasts.

A good way to choose the best forecasting model is to find the model with the smallest RMSE computed using time series cross-validation.

The pipe operator %>%

The above commands could also be implemented using the pipe operator in a more intuitive way (from right to left) as follows:

```
dj %>% tsCV(forecastfunction=rwf, drift=TRUE, h=1) -> e
e^2 %>% mean(na.rm=TRUE) %>% sqrt()
```

```
## [1] 22.68249
```

```
dj %>% rwf(drift=TRUE) %>% residuals() -> res
res^2 %>% mean(na.rm=TRUE) %>% sqrt()
```

```
## [1] 22.49681
```

The left hand side of each pipe is passed as the first argument to the function on the right hand side. This is consistent with the way we read from left to right in English. A consequence of using pipes is that all other arguments must be named, which also helps readability. When using pipes, it is natural to use the right arrow assignment -> rather than the left arrow. For example, the third line above can be read as "Take the dj series, pass it to rwf with drift=TRUE , compute the resulting residuals, and store them as res".

The authors use the pipe operator in the 2^{nd} edition for the code to be easier to read. For consistentency, they follow a function with parentheses to differentiate it from other objects, even if it has no arguments eg, use of sqrt() in the code above.

Prediction intervals

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

Assuming the forecast errors are uncorrelated and normally distributed, then a simple 95% prediction interval for the next observation in a time series is $\hat{y}_t \pm 1.96\hat{\sigma}$, where $\hat{\sigma}$ is an estimate of the standard deviation of the forecast distribution.

When forecasting one-step ahead, the standard deviation of the forecast distribution is almost the same as the standard deviation of the residuals.

Table 3.1: 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

Multipliers to be used for prediction intervals.

Consider a naive forecast for the Dow-Jones Index. The last value of the observed series is 3830, so the forecast of the next value of the DJI is 3830. The standard deviation of the residuals from the naive method is 21.99. Hence, a 9% prediction interval for the next value of the DJI is

$$3830 \pm 1.96(21.99) = [3787, 3873].$$

Similarly, an 80% prediction interval is given by

$$3830 \pm 1.28(21.99) = [3802, 3858].$$

The use of these multipliers in the formula $\hat{y}_t \pm k\hat{\sigma}$ (where k is the multiplier) assumes that the residuals are normally distributed and uncorrelated. If either of these conditions does not hold, then this method of producing a prediction interval cannot be used.

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. But if we also produce prediction intervals, then it is clear how much uncertainty is associated with each forecast.

To produce a prediction interval, it is necessary to have an estimate of the standard deviation of the forecast distribution.

- For one-step forecasts for time series, the residual standard deviation provides a good estimate of the forecast standard deviation.
- For all other situations, including multi-step forecasts for time series, a more complicated method of calculation is required.

Multi-step prediction intervals

A common feature of prediction intervals is that they increase in length as the forecast horizon increases. The further ahead we forecast, the more uncertainty is associated with the forecast, and so the prediction intervals grow wider. That is, σ_h usually increases with h (although there are some non-linear forecasting methods that 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 ($h = 1$), the residual standard deviation 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, then we can use the following expressions.

Mean forecast: $\hat{\sigma}_h = \hat{\sigma}\sqrt{1+1/T}$

Naive forecast: $\hat{\sigma}_h = \hat{\sigma}\sqrt{h}$

Seasonal Naive forecast: $\hat{\sigma}_h = \hat{\sigma}\sqrt{[(h-1)/m]+1}$

Drift forecast : $\hat{\sigma}_h = \hat{\sigma}\sqrt{[(h(1+h/T)}$

Note that when $h = 1$ and T is large, these all give the same approximate value $\hat{\sigma}$.

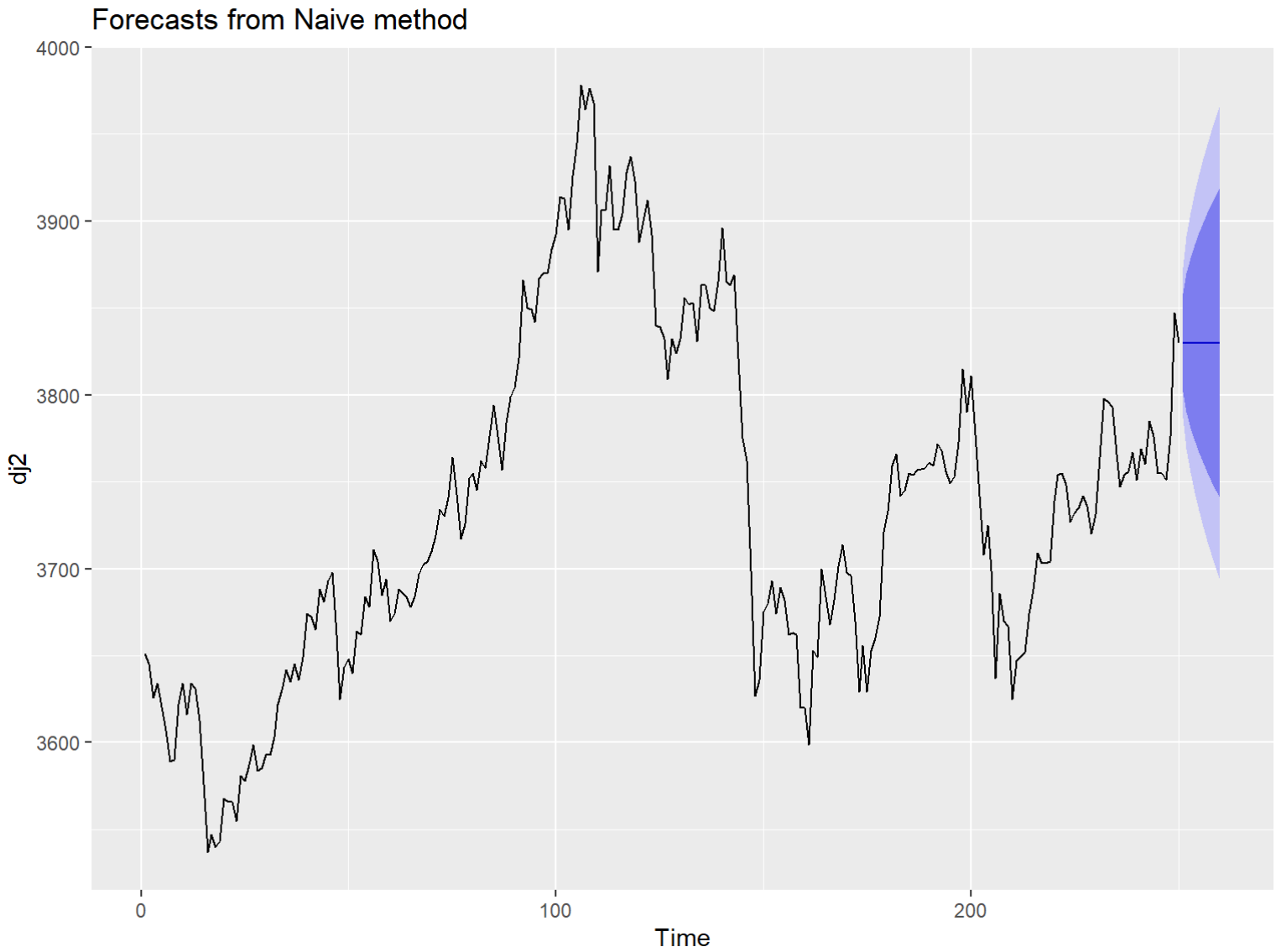
Prediction intervals will be computed for you when using any of the benchmark forecasting methods. For example, here is the output when using the naive method for the Dow-Jones index

```
naive(dj2)
```

```
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 251           3830 3801.806 3858.194 3786.881 3873.119
## 252           3830 3790.127 3869.873 3769.020 3890.980
## 253           3830 3781.166 3878.834 3755.315 3904.685
## 254           3830 3773.611 3886.389 3743.761 3916.239
## 255           3830 3766.956 3893.044 3733.582 3926.418
## 256           3830 3760.938 3899.062 3724.379 3935.621
## 257           3830 3755.405 3904.595 3715.917 3944.083
## 258           3830 3750.254 3909.746 3708.040 3951.960
## 259           3830 3745.417 3914.583 3700.642 3959.358
## 260           3830 3740.842 3919.158 3693.644 3966.356
```

When plotted, the prediction intervals are shown as shaded region, with the strength of colour indicating the probability associated with the interval.

```
autoplot(naive(dj2))
```



Prediction intervals from bootstrapped residuals

When a normal distribution for the forecast errors is an unreasonable assumption, one alternative is to use bootstrapping, which only assumes that the forecast errors are uncorrelated.

A forecast error is defined as $e_t = y_t - \hat{y}_{t|t-1}$. We can re-write this as

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

the next observation of the time series could be simulated as

$$y_{T+1} = \hat{y}_{T+1|T} + e_{T+1}$$

where $\hat{y}_{T+1|T}$ is the one-step forecast and e_{T+1} is the unknown future error. Assuming future errors will be similar to past errors, we can replace e_{T+1} by sampling from the collection of errors we have seen in the past (i.e., the residuals). Adding the new simulated observation to our data set, we can repeat the process to obtain

$$y_{T+2} = \hat{y}_{T+2|T+1} + e_{T+2}$$

Doing this repeatedly, we obtain many possible futures. Then we can compute prediction intervals by calculating percentiles 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.

To generate such intervals, we can simply add the *bootstrap* argument to our forecasting functions. For example:

```
naive(dj2, bootstrap=TRUE)
```

```
##     Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 251           3830 3804.281 3854.281 3783.281 3874.281
## 252           3830 3790.562 3866.562 3764.562 3887.562
## 253           3830 3781.843 3875.843 3748.843 3901.843
## 254           3830 3773.124 3883.124 3738.466 3912.124
## 255           3830 3766.589 3890.406 3728.406 3923.406
## 256           3830 3760.687 3896.687 3718.687 3932.516
## 257           3830 3752.968 3902.468 3709.310 3937.968
## 258           3830 3749.249 3907.249 3702.249 3948.249
## 259           3830 3745.530 3913.030 3695.701 3954.530
## 260           3830 3742.811 3916.811 3689.982 3959.470
```

In this case, they are very similar (but not identical) to the prediction intervals based on the normal distribution.

Prediction intervals with transformations

If a transformation has been used, then the prediction interval should be computed on the transformed scale, and the end points back-transformed to give a prediction interval on the original scale. This approach preserves the probability coverage of the prediction interval, although it will no longer be symmetric around the point forecast.

The back-transformation of prediction intervals is done automatically using the functions in the *forecast* package in R, provided you have used the *lambda* argument when computing the forecasts.

The forecast package in R

forecast package in R is loaded automatically when you load the fpp2 package. For the details of the forecast package, please seach Google for forecast in R (Rdocumentation).

Many functions, including meanf , naive , snaive and rwf , produce output in the form of a forecast object (i.e., an object of class forecast). This allows other functions (such as autoplot) to work consistently across a range of forecasting models.

Objects of class forecast contain information about the forecasting method, the data used, the point forecasts obtained, prediction intervals, residuals and fitted values. There are several functions designed to work with these objects including autoplot , summary and print.

The following list shows some of the functions that produce forecast objects. meanf() naive(), snaive() rwf() croston() stlf() ses() holt() , hw() splinef() thetaf() forecast()

forecast() function

So far we have used functions which produce a forecast object directly. But a more common approach, will be to fit a model to the data, and then use the forecast() function to produce forecasts from that model. The forecast() function works with many different types of input. It generally takes a time series or time series model as its main argument, and produces forecasts appropriately. It always returns objects of class forecast .

If the first argument is of class ts , it returns forecasts from the automatic ETS algorithm discussed in Chapter 7.

Here is a simple example, applying forecast() to the ausbeer data:

```
forecast(ausbeer)
```

```
##         Point Forecast    Lo 80    Hi 80    Lo 95    Hi 95
## 2008 Q4       480.4201 458.0566 502.7836 446.2181 514.6221
## 2009 Q1       423.2123 402.9617 443.4628 392.2417 454.1828
## 2009 Q2       385.7490 366.6696 404.8283 356.5696 414.9283
## 2009 Q3       402.1541 381.4857 422.8225 370.5445 433.7637
## 2009 Q4       479.6591 452.2863 507.0319 437.7961 521.5222
## 2010 Q1       422.5416 397.3980 447.6853 384.0877 460.9955
## 2010 Q2       385.1374 361.1592 409.1157 348.4659 421.8090
## 2010 Q3       401.5163 375.2894 427.7431 361.4058 441.6267
```

This works if you have no idea what sort of model to use. But by the end of this course, you should not need to use forecast() in this "blind" fashion. Instead, you will fit a model appropriate to the data, and then use forecast() to produce forecasts from that model.

Next Topic: Chapter 5 - time series regression models (excluding section 5.7)