# Economic Forecasting

## Forecasting with time series data

Sebastian Fossati

University of Alberta | E493 | 2023

# Outline


1 Some simple forecasting methods

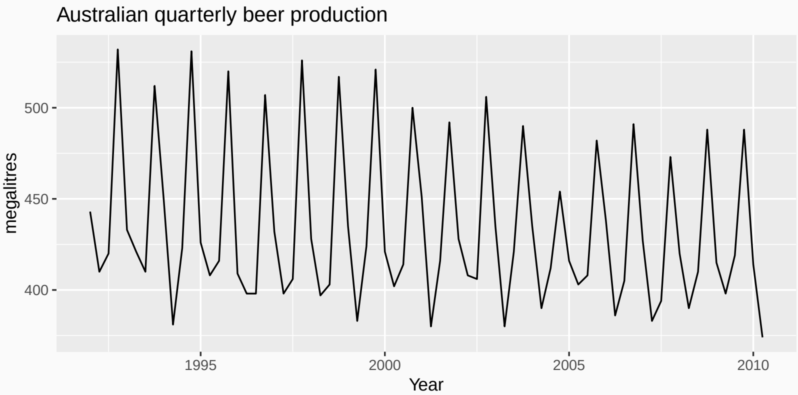
2 Transformations and adjustments

3 Residual diagnostics

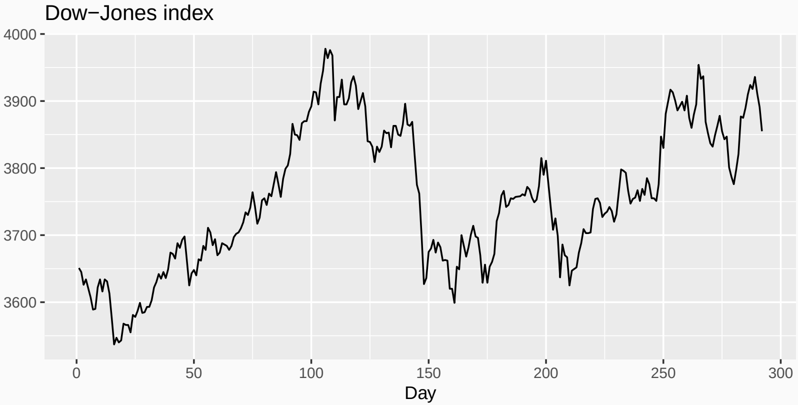
4 Evaluating forecast accuracy

5 Prediction intervals

# How would you forecast this time series?



# How would you forecast this time series?



# Some simple forecasting methods

Some simple forecasting methods are surprisingly effective.

Here are four methods that we will use as benchmarks for other forecasting methods.

1. Average method
2. Naïve method
3. Seasonal naïve method
4. Drift method

## Average method

- forecast of all future values is equal to mean of historical data $\{y_1, \ldots, y_T\}$
- forecasts: $\hat{y}_{T+h|T} = \bar{y} = (y_1 + \cdots + y_T)/T$

```
# y contains the time series
# h is the forecast horizon
meanf(y, h)
```

## Naïve method

- forecasts equal to last observed value
- consequence of efficient market hypothesis
- forecasts: $\hat{y}_{T+h|T} = y_T$

```
# y contains the time series
# h is the forecast horizon
naive(y, h) # alternative rwf(y, h)
```

## Some simple forecasting methods

### Seasonal naïve method

- forecasts equal to last value from same season (eg, the same month of the previous year)
- forecasts: $\hat{y}_{T+h|T} = y_{T+h-m(k+1)}$, where $m$ = seasonal period and $k$ is the integer part of $(h-1)/m$

```
# y contains the time series
# h is the forecast horizon
snaive(y, h)
```

# Some simple forecasting methods

## Drift method

- forecasts equal to last value plus average change
- equivalent to extrapolating a line drawn between first and last observations
- forecasts:

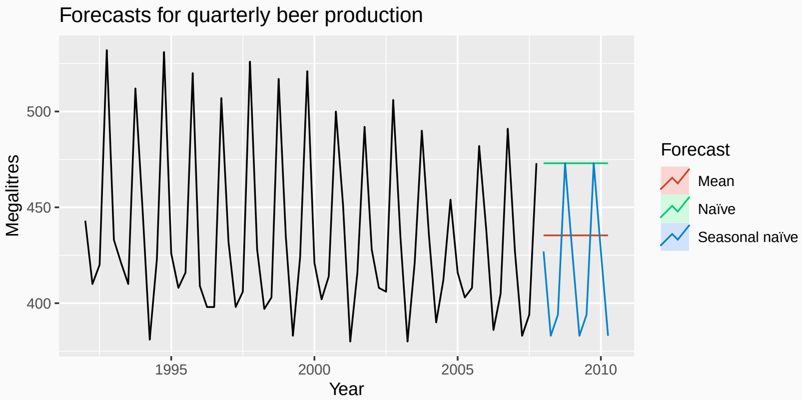
$$\hat{y}_{T+h|T} = y_T + \frac{h}{T-1}\sum_{t=2}^{T}(y_t - y_{t-1})$$

```r
# y contains the time series
# h is the forecast horizon
rwf(y, h, drift = TRUE)
```

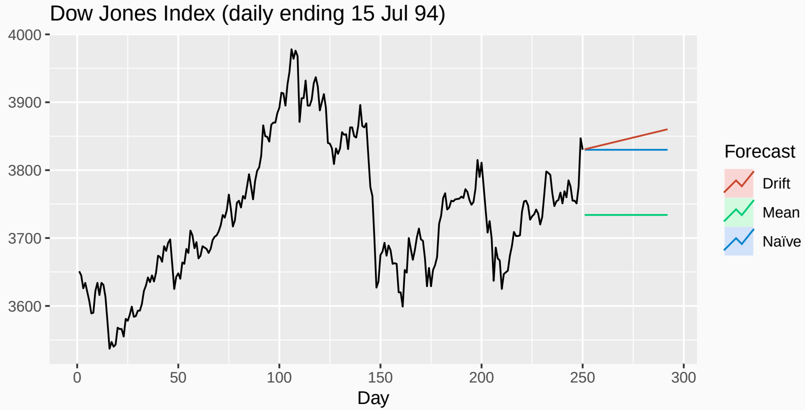
# Example 1: quarterly beer production

```r
# get data
beer2 <- window(ausbeer, start = 1992, end = c(2007,4))
# get forecasts
fcst.m <- meanf(beer2, h = 10)
fcst.n <- naive(beer2, h = 10)
fcst.s <- snaive(beer2, h = 10)
# plot
autoplot(beer2) +
  autolayer(fcst.m, PI = FALSE, series = "Mean") +
  autolayer(fcst.n, PI = FALSE, series = "Naïve") +
  autolayer(fcst.s, PI = FALSE, series = "Seasonal naïve") +
  ggtitle("Forecasts for quarterly beer production") +
  xlab("Year") + ylab("Megalitres") +
  guides(colour = guide_legend(title = "Forecast"))
```

# Example 1: quarterly beer production



# Example 2: Dow Jones Index



# Outline


1 Some simple forecasting methods

2 Transformations and adjustments

3 Residual diagnostics

4 Evaluating forecast accuracy

5 Prediction intervals

# Transformations and adjustments

Adjusting the data can simplify the patterns in the historical data, making the pattern more consistent across the whole sample.

Examples:

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

Simpler patterns usually lead to more accurate forecasts!

# Variance stabilization

If the data show different variation at different levels of the series, then a transformation can be useful.

Logarithms, in particular, are useful because they are more interpretable: changes in a log value are **relative (percent) changes on the original scale.**

# Variance stabilization

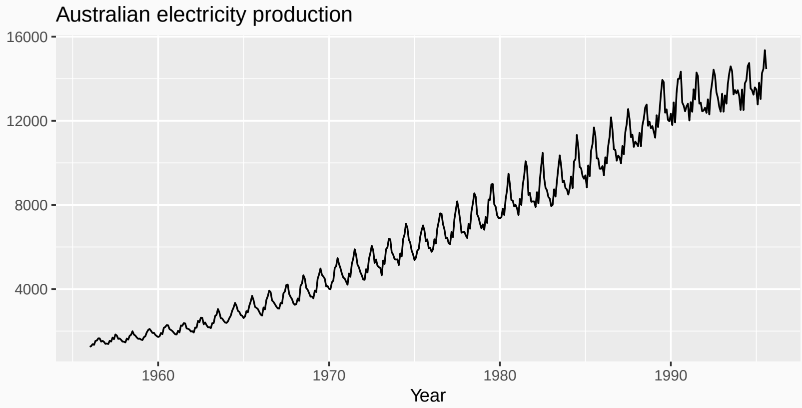
Denote original observations as $y_1, \dots, y_T$ and transformed observations as $w_1, \dots, w_T$.

## Mathematical transformations

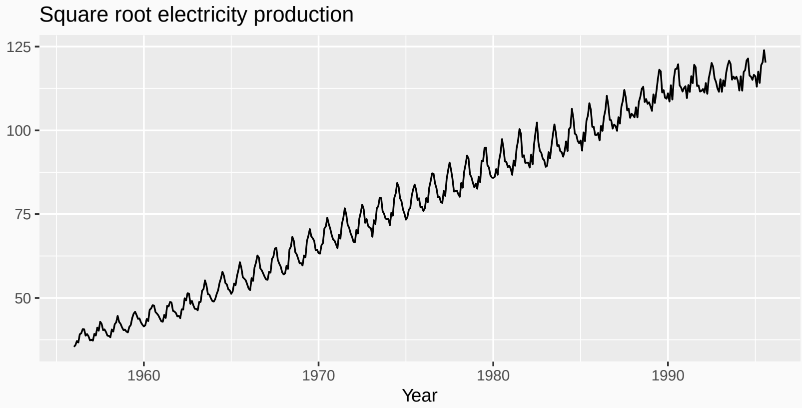
Square root: $w_t = \sqrt{y_t}$

Logarithm: $w_t = \log(y_t)$

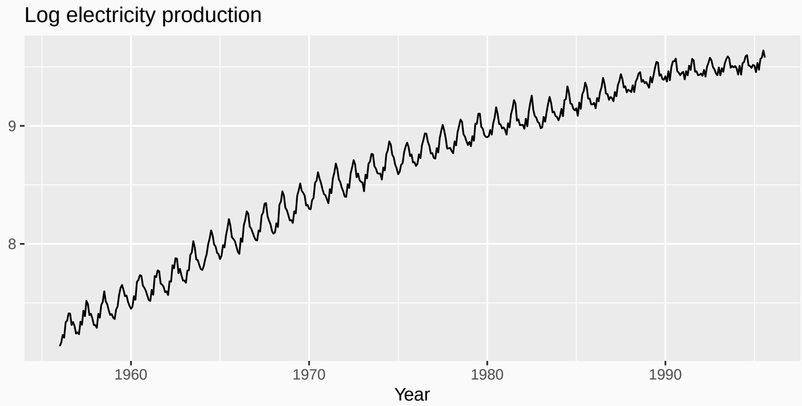
# Variance stabilization



# Variance stabilization



# Variance stabilization



# Box-Cox transformations

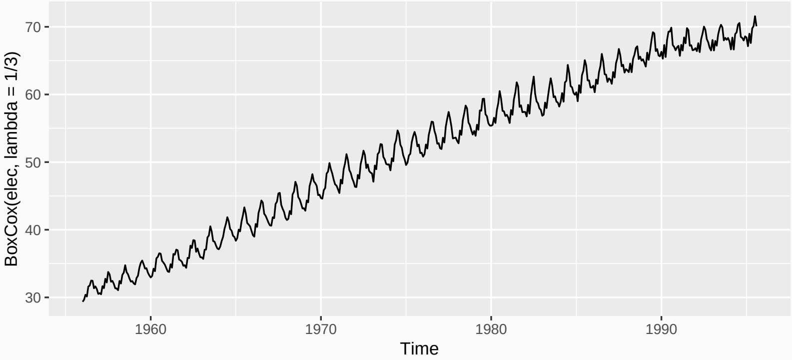
Each of these transformations is close to a member of the family of **Box-Cox transformations**:

$$w_t = \begin{cases} \log(y_t), & \lambda = 0 \\ (y_t^\lambda - 1)/\lambda, & \lambda \neq 0 \end{cases}$$

- $\lambda = 1$: (no substantive transformation)
- $\lambda = \frac{1}{2}$: (square root plus linear transformation)
- $\lambda = 0$: (natural logarithm)

# Box-Cox transformations

```r
# box-cox
autoplot(BoxCox(elec, lambda = 1/3))
```



# Box-Cox transformations

Remarks:

- $y_t^\lambda$ for $\lambda$ close to zero behaves like logs
- if some $y_t \leq 0$, no power transformation is possible unless all $y_t$ adjusted by **adding a constant to all values**
- simple values of $\lambda$ are easier to explain
- often no transformation ($\lambda = 1$) needed
- transformation can have very large effect on PI
- choosing $\lambda = 0$ is a simple way to force forecasts to be positive

# Automated Box-Cox transformations

```
# box-cox selection
BoxCox.lambda(elec)
```

```
## [1] 0.2654
```

- this attempts to balance the seasonal fluctuations and random variation across the series
- a low value of $\lambda$ can give extremely large prediction intervals

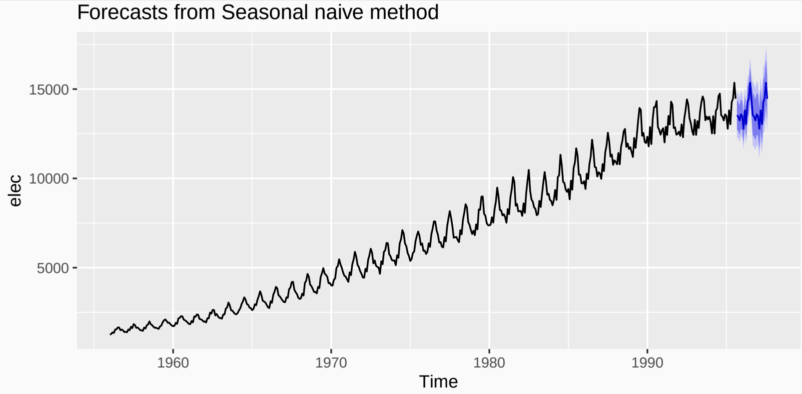
# Back-transformation

We must reverse the transformation (or *back-transform*) to obtain forecasts on the original scale. The reverse Box-Cox transformations are given by

$$y_t = \begin{cases} \exp(w_t), & \lambda = 0 \\ (\lambda W_t + 1)^{1/\lambda}, & \lambda \neq 0 \end{cases}$$

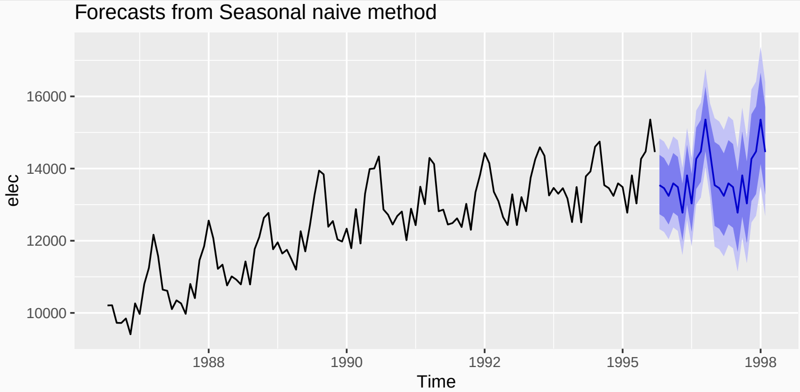
# Back-transformation

```r
# forecasting
fit <- snaive(elec, lambda = 1/3)
autoplot(fit)
```



# Back-transformation

```r
# a closer look
autoplot(fit, include = 120)
```



# Other adjustments

Other adjustments may include:

- calendar adjustments: some variation seen in seasonal data may be due to simple calendar effects (eg, different numbers of days in each month)
- population adjustments: data affected by population changes can be adjusted to give per-capita data
- inflation adjustments: data affected by the value of money are best adjusted before modelling

# Outline


1 Some simple forecasting methods

2 Transformations and adjustments

3 Residual diagnostics

4 Evaluating forecast accuracy

5 Prediction intervals

## Fitted values

The fitted value $\hat{y}_{t|t-1}$ is the forecast of $y_t$ based on observations $y_1, \dots, y_{t-1}$.

- sometimes drop the subscript: $\hat{y}_t \equiv \hat{y}_{t|t-1}$
- not true forecasts if parameters are estimated on all data

For example:

- $\hat{y}_t = \bar{y}$ for average method

# Forecasting residuals

Difference between observed value and its fitted value:

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

### Assumptions

1. $\{e_t\}$ uncorrelated (if they aren't, information left in residuals should be used in computing forecasts)
2. $\{e_t\}$ have mean zero (if they don't, forecasts are biased)

# Forecasting residuals

Difference between observed value and its fitted value:

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

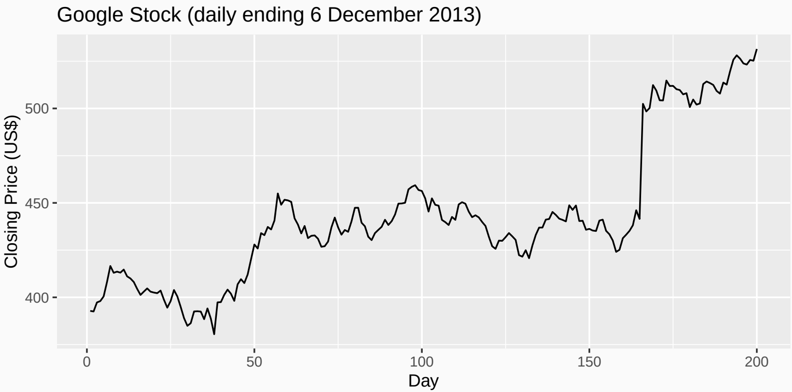
**Assumptions**

1. $\{e_t\}$ uncorrelated (if they aren't, information left in residuals should be used in computing forecasts)
2. $\{e_t\}$ have mean zero (if they don't, forecasts are biased)

**Useful properties** (for prediction intervals)

3. $\{e_t\}$ have constant variance
4. $\{e_t\}$ are normally distributed

# Example: Google stock price



# Example: Google stock price

**Naïve forecast:**

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

# Example: Google stock price

**Naïve forecast:**

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

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

# Example: Google stock price

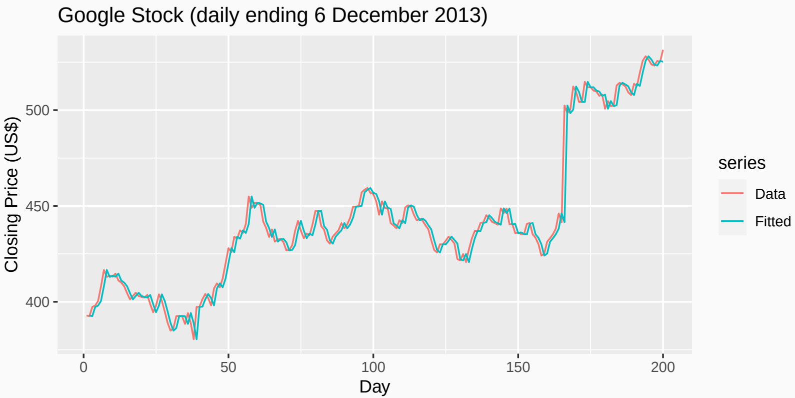
**Naïve forecast:**

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

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

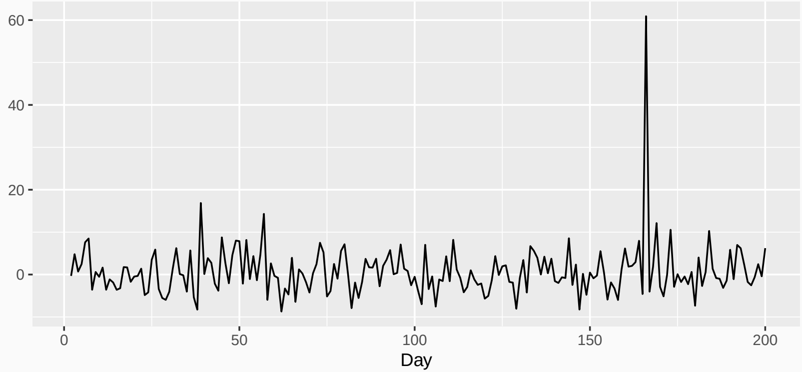
Note: $e_t$ are one-step-forecast residuals.

# Example: Google stock price



# Example: Google stock price

```r
res <- residuals(naive(goog200))
autoplot(res) + xlab("Day") + ylab("")
```



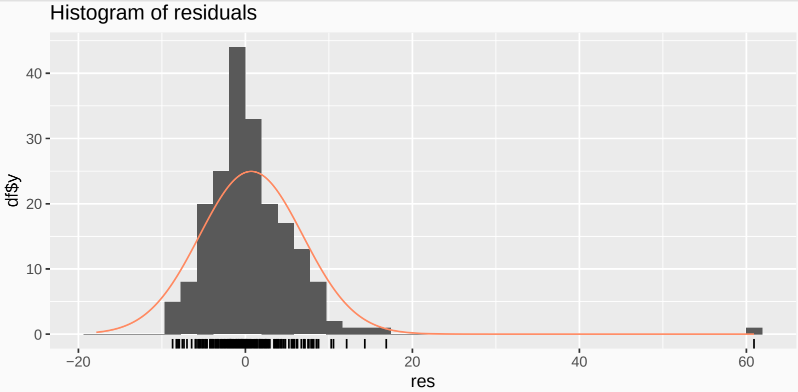
# Properties of residuals

Properties:

- we *expect* residuals to look like white noise (uncorrelated, mean zero, constant variance)
- so a standard residual diagnostic is to check the ACF of the residuals of a forecasting method
- we can also check other properties such as normality

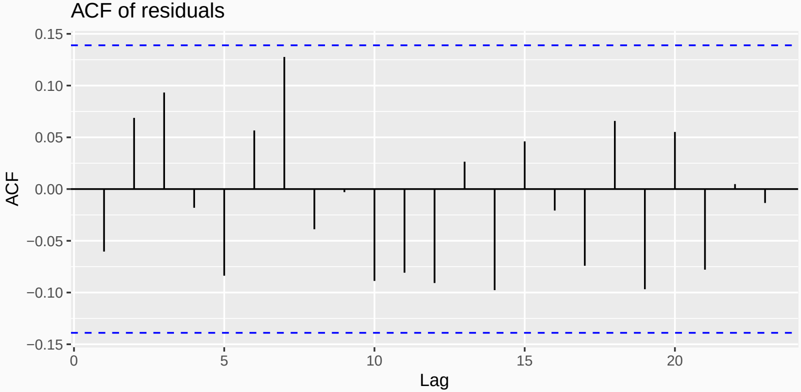
# Example: Google stock price

```
# histogram
gghistogram(res, add.normal = TRUE) + ggtitle("Histogram of residuals")
```



# Example: Google stock price

```r
# acf of residuals
ggAcf(res) + ggtitle("ACF of residuals")
```



# Portmanteau tests

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

## Box-Pierce test

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

where $h$ is max lag being considered and $T$ is number of observations.

- if each $r_k$ close to zero, $Q$ will be **small**
- if some $r_k$ away from zero, $Q$ will be **large**
- about $h = 10$ or $h = 2m$ for seasonal data

## Ljung-Box test

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

where $h$ is max lag being considered and $T$ is number of observations.

- about $h = 10$ or $h = 2m$ for seasonal data
- better performance, especially in small samples

# Portmanteau tests

For the Google example:

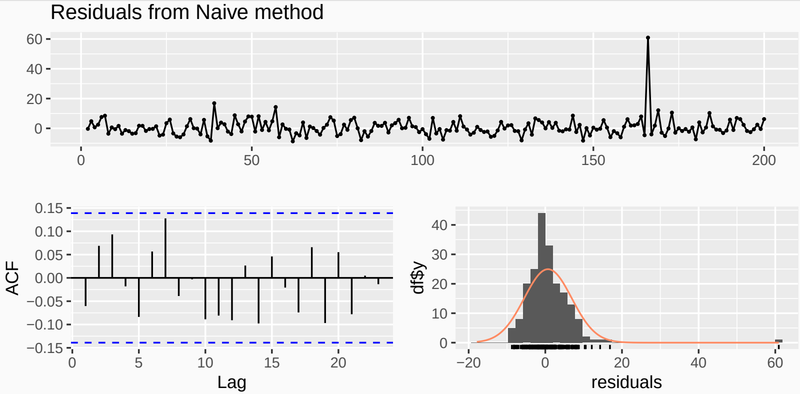
```
# test
Box.test(res, lag = 10, fitdf = 0, type = "Lj")
```

```
##
##  Box-Ljung test
##
## data:  res
## X-squared = 11, df = 10, p-value = 0.4
```

- if data are WN, $Q^*$ has $\chi^2$ distribution with degrees of freedom ($h-$ # parameters in model)
- when applied to raw data, set # parameters = 0

# checkresiduals function

```
# useful function!
checkresiduals(naive(goog200), test = FALSE)
```



# `checkresiduals` function

```r
# useful function!
checkresiduals(naive(goog200), plot = FALSE)
```

```
##
##  Ljung-Box test
##
## data:  Residuals from Naive method
## Q* = 11, df = 10, p-value = 0.4
##
## Model df: 0.   Total lags used: 10
```

# Outline


1 Some simple forecasting methods

2 Transformations and adjustments

3 Residual diagnostics

4 **Evaluating forecast accuracy**

5 Prediction intervals

# Training and test sets

A perfect fit can always be obtained by using a model with enough parameters (over-fitting).

- over-fitting a model to data is just as bad as failing to identify a systematic pattern in the data
- a common practice is to separate the available data into a training set and a test set

# Training and test sets


$$\text{MAE} = \text{mean}(|e_{T+h}|)$$

$$\text{MSE} = \text{mean}(e_{T+h}^2)$$

$$\text{RMSE} = \sqrt{\text{mean}(e_{T+h}^2)}$$

$$\text{MAPE} = 100\text{mean}(|e_{T+h}|/|y_{T+h}|)$$

# Training and test sets


$$\text{MAE} = \text{mean}(|e_{T+h}|)$$

$$\text{MSE} = \text{mean}(e_{T+h}^2)$$

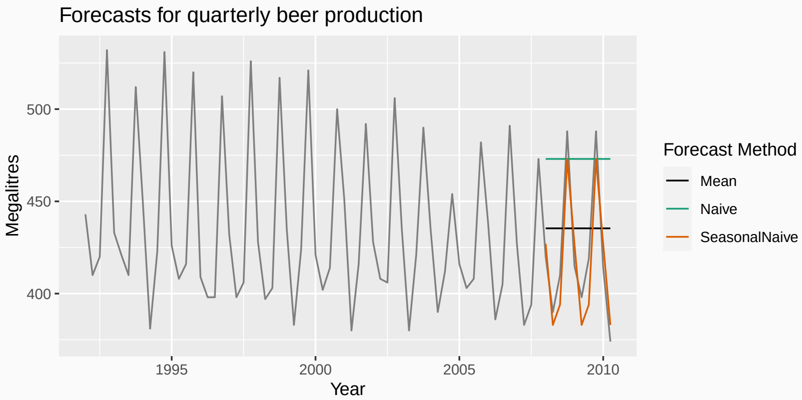
$$\text{RMSE} = \sqrt{\text{mean}(e_{T+h}^2)}$$

$$\text{MAPE} = 100\text{mean}(|e_{T+h}|/|y_{T+h}|)$$

Remarks:

- MAE, MSE, RMSE are all scale dependent
- MAPE is scale independent but is only sensible if $y_t \gg 0$ for all $t$

# Example 1: quarterly beer production



## Example 1: quarterly beer production

```r
beer2 <- window(ausbeer, start = 1992, end = c(2007,4)) # training set
beer3 <- window(ausbeer, start = 2008) # test set
beerfit1 <- meanf(beer2, h = 10)
beerfit2 <- rwf(beer2, h = 10)
beerfit3 <- snaive(beer2, h = 10)
accuracy(beerfit1, beer3)
accuracy(beerfit2, beer3)
accuracy(beerfit3, beer3)
```

| | RMSE | MAE | MAPE |
|---|---|---|---|
| Mean method | 38.45 | 34.83 | 8.28 |
| Naïve method | 62.69 | 57.40 | 14.18 |
| Seasonal naïve method | 14.31 | 13.40 | 3.17 |

# Time series cross-validation

To evaluate the **out-of-sample** performance of one or more models we can construct a series of $h$-step ahead predictions with the forecasting horizon $h$ fixed.

For example, we can construct a series of $R$ 1-step ahead predictions following these steps:

- estimate the model with $t = 1, \ldots, T$ and construct $\hat{Y}_{T+1|T}$
- re-estimate the model with $t = 1, \ldots, T + 1$ and construct $\hat{Y}_{T+2|T+1}$
- repeat until $t = 1, \ldots, T + R - 1$ and $\hat{Y}_{T+R|T+R-1}$

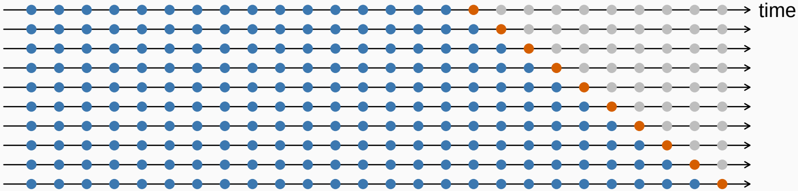
# Time series cross-validation

## Traditional evaluation

# Time series cross-validation

## Traditional evaluation

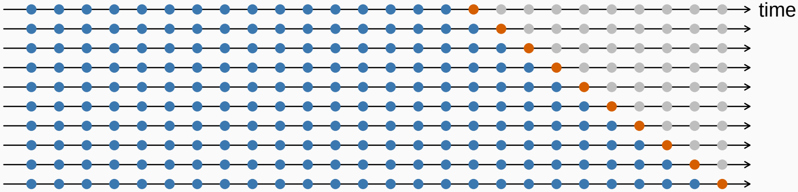

## Time series cross-validation



# Time series cross-validation

## Traditional evaluation


## Time series cross-validation




- forecast accuracy averaged over test sets
- an “evaluation on a rolling forecasting origin”
- or “pseudo out-of-sample forecast evaluation”

# Example 1: quarterly beer production

```r
# data and set up
beer1 <- window(ausbeer, start = 1992,end = c(2009,4))
n.end <- 2004.75 # 2004Q4
# set matrix for storage, 20 obs in test set
pred <- matrix(rep(NA,80),20,4)
# loop
for(i in 1:20){
  tmp0 <- 1992
  tmp1 <- n.end+(i-1)*.25
  tmp <- window(beer1, tmp0, tmp1)
  pred[i,1] <- window(beer1, tmp1+.25, tmp1+.25) # actual
  # compute forecasts
  pred[i,2] <- meanf(tmp, h = 1)$mean
  pred[i,3] <- rwf(tmp, h = 1)$mean
  pred[i,4] <- snaive(tmp, h = 1)$mean
}
```

# Example 1: quarterly beer production

```r
# compute rmse
beerfit1 <- pred[,1] - pred[,2]
beerfit2 <- pred[,1] - pred[,3]
beerfit3 <- pred[,1] - pred[,4]
rmse1 <- sqrt(mean(beerfit1^2, na.rm=TRUE))
rmse2 <- sqrt(mean(beerfit2^2, na.rm=TRUE))
rmse3 <- sqrt(mean(beerfit3^2, na.rm=TRUE))
# display rmse
cbind(rmse1,rmse2,rmse3)
```

```
##      rmse1 rmse2 rmse3
## [1,] 37.41 51.44 13.26
```

- choose forecasting model with the smallest RMSE computed using time series cross-validation

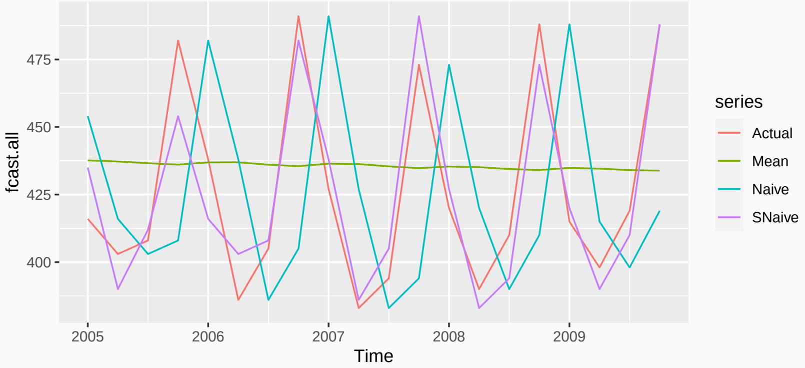
# tsCV function

```
# time series cross-validation function
beerfit1 <- tsCV(beer1, meanf, h = 1)
beerfit2 <- tsCV(beer1, rwf, h = 1)
beerfit3 <- tsCV(beer1, snaive, h = 1)

# compute rmse
rmse1 <- sqrt(mean(beerfit1[52:71]^2))
rmse2 <- sqrt(mean(beerfit2[52:71]^2))
rmse3 <- sqrt(mean(beerfit3[52:71]^2))

# display rmse
cbind(rmse1,rmse2,rmse3)
```

```
##      rmse1 rmse2 rmse3
## [1,] 37.41 51.44 13.26
```

## Example 1: quarterly beer production

```
# organize results
fcast.all <- ts(pred, start = 2005, freq = 4, names = c("Actual","Mean","Naive","SNai
autoplot(fcast.all)
```



# Outline


1 Some simple forecasting methods

2 Transformations and adjustments

3 Residual diagnostics

4 Evaluating forecast accuracy

5 Prediction intervals

# Prediction intervals

A forecast $\hat{y}_{T+h|T}$ is (usually) the mean of the conditional distribution $y_{T+h} \mid y_1, \dots, y_T$.

- a prediction interval gives a region within which we expect $y_{T+h}$ to lie with a specified probability

Assuming forecast errors are normally distributed, then a 95% PI is

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

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

# Prediction intervals

Remarks:

- point forecasts are often useless without prediction intervals
- prediction intervals require a stochastic model (with random errors, etc)
- multi-step forecasts for time series require a more sophisticated approach (with PI getting wider as the forecast horizon increases)

# Prediction intervals

Assume residuals are normal, uncorrelated, sd = $\hat{\sigma}$:

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

**Naïve forecasts:** $\hat{\sigma}_h = \hat{\sigma}\sqrt{h}$

**Seasonal naïve forecasts:** $\hat{\sigma}_h = \hat{\sigma}\sqrt{k + 1}$

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

where $k$ is the integer part of $(h - 1)/m$.

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

# Prediction intervals

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

```
# compute sigma hat
res <- residuals(naive(goog200))
res_sd <- sqrt(mean(res^2, na.rm=TRUE))
# compute 05% interval
c(tail(goog200,1)) + 1.96 * res_sd * c(-1,1)
```

```
## [1] 519.3 543.6
```

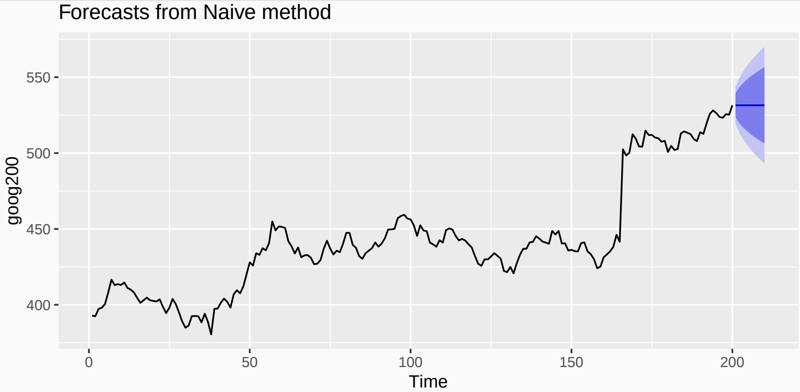
# Prediction intervals

```r
# predictions and prediction intervals
naive(goog200, level = 95)
```

```
##     Point Forecast Lo 95 Hi 95
## 201          531.5 519.3 543.6
## 202          531.5 514.3 548.7
## 203          531.5 510.4 552.6
## 204          531.5 507.1 555.8
## 205          531.5 504.3 558.7
## 206          531.5 501.7 561.3
## 207          531.5 499.3 563.7
## 208          531.5 497.1 565.9
## 209          531.5 495.0 568.0
## 210          531.5 493.0 570.0
```

# Prediction intervals

```r
# plot prediction intervals
autoplot(naive(goog200))
```



# Prediction intervals

Remarks:

- computed automatically using: `naive()`, `snaive()`, `rwf()`, `meanf()`, etc
- use `level` argument to control coverage
- check residual assumptions before believing them
- usually too narrow due to unaccounted uncertainty