MONASH University

MONASH BUSINESS SCHOOL

ETC3550 Applied forecasting for business and economics

Ch3. The forecasters’ toolbox

OTexts.org/fpp3/

Outline

1. A tidy forecasting workflow
2. Some simple forecasting methods
3. The workflow in action
4. Transformations
5. Distributional forecasts
6. Evaluating forecast accuracy
7. Time series cross-validation

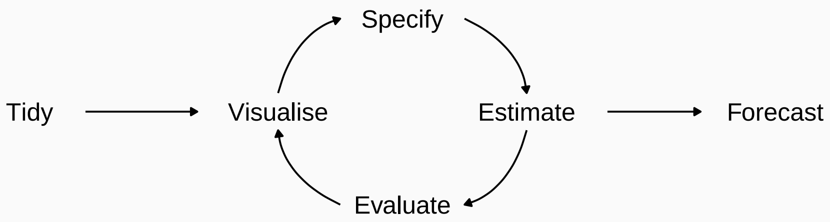
Outline

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 & performance evaluation
6. Producing forecasts

A tidy forecasting workflow



Outline

Some simple forecasting methods

How would you forecast these series?

Some simple forecasting methods

How would you forecast these series?

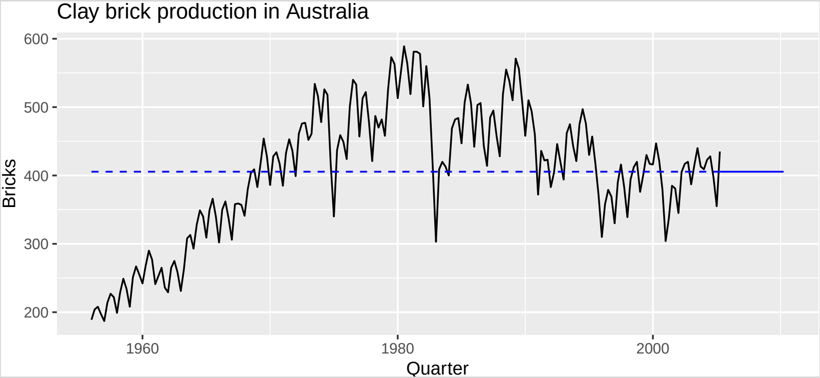
Some simple forecasting methods

How would you forecast these series?

Some simple forecasting methods

`MEAN(y)`: Average method

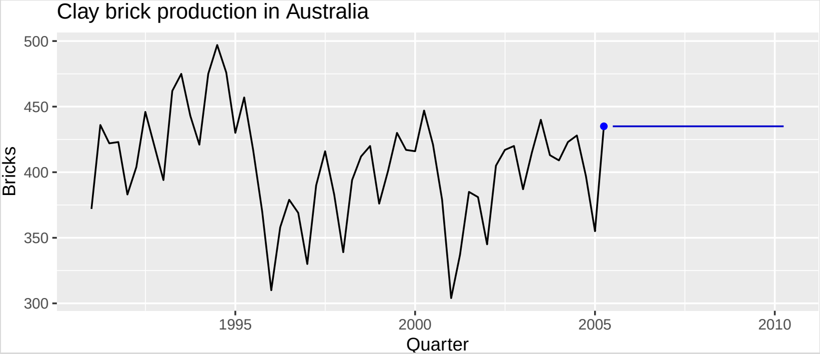
- Forecast of all future values is equal to mean of historical data $\{y_1, \dots, y_T\}$.
- Forecasts: $\hat{y}_{T+h|T} = \bar{y} = (y_1 + \cdots + y_T)/T$



Some simple forecasting methods

`NAIVE(y)`: Naïve method

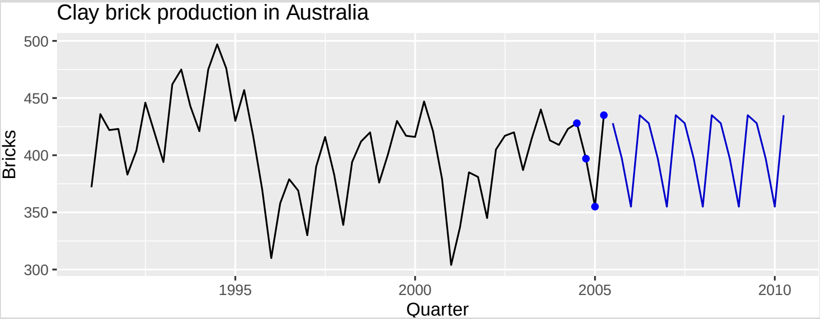
- Forecasts equal to last observed value.
- Forecasts: $\hat{y}_{T+h|T} = y_T$.
- Consequence of efficient market hypothesis.



Some simple forecasting methods

`SNAIVE(y ~ lag(m))`: Seasonal naïve method

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



Some simple forecasting methods

`RW(y ~ drift())`: Drift method

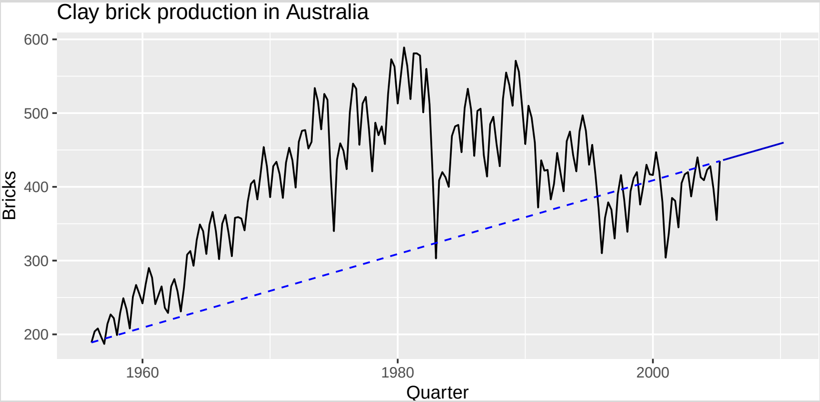
- Forecasts equal to last value plus average change.
- Forecasts:

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

- Equivalent to extrapolating a line drawn between first and last observations.

Some simple forecasting methods

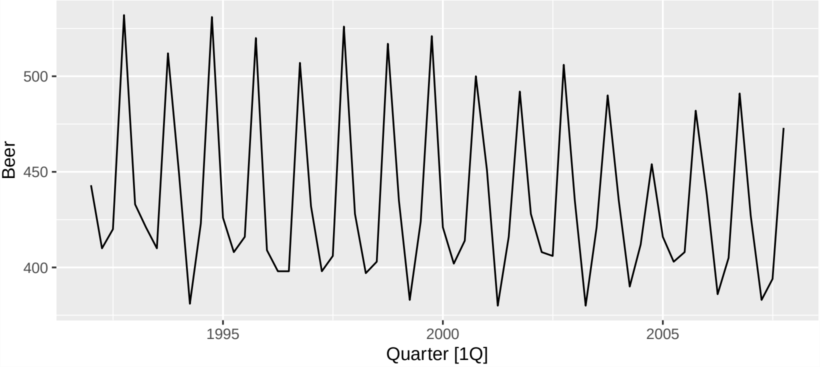
Drift method



Outline

Data preparation and visualisation

```
# Set training data from 1992 to 2007
train <- aus_production %>%
  filter(between(year(Quarter), 1992, 2007))
train %>% autoplot(Beer)
```



Model estimation

The `model()` function trains models to data.

```
# Fit the models
beer_fit <- train %>%
  model(
    Mean = MEAN(Beer),
    `Naïve` = NAIVE(Beer),
    `Seasonal naïve` = SNAIVE(Beer),
    Drift = RW(Beer ~ drift())
  )
```

Model estimation

```r
beer_fit
```

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

A `mable` is a model table, each cell corresponds to a fitted model.

Producing forecasts

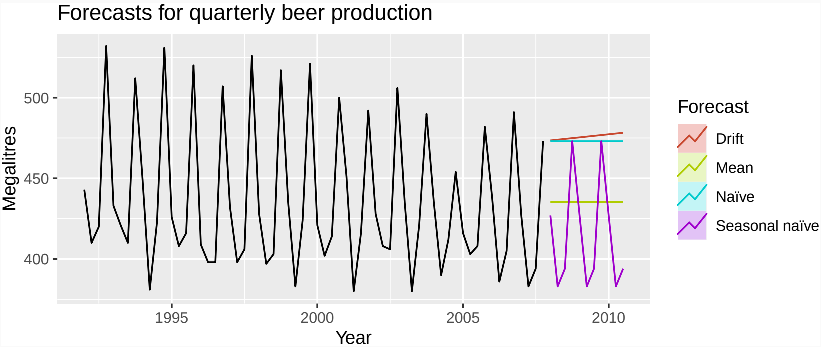
```r
beer_fc <- beer_fit %>%
  forecast(h = 11)
```

```
## # A fable: 44 x 4 [1Q]
## # Key:     .model [4]
##   .model Quarter  Beer .distribution
##   <chr>    <qtr> <dbl> <dist>
## 1 Mean   2008 Q1  435. N(435, 1964)
## 2 Mean   2008 Q2  435. N(435, 1964)
## 3 Mean   2008 Q3  435. N(435, 1964)
## 4 Mean   2008 Q4  435. N(435, 1964)
## # ... with 40 more rows
```

A `fable` is a forecast table with point forecasts and distributions.

Visualising forecasts

```r
beer_fc %>%
  autoplot(train, level = NULL) +
  ggtitle("Forecasts for quarterly beer production") +
  xlab("Year") + ylab("Megalitres") +
  guides(colour=guide_legend(title="Forecast"))
```



Facebook closing stock price

```
# Extract training data
fb_stock <- gafa_stock %>%
  group_by(Symbol) %>%
  mutate(trading_day = row_number()) %>%
  update_tsibble(index=trading_day, regular=TRUE) %>%
  filter(Symbol == "FB",
         between(Date, ymd("2018-01-01"), ymd("2018-09-01")))
# Specify, estimate and forecast
fb_stock %>%
  model(
    Mean = MEAN(Close),
    `Naïve` = NAIVE(Close),
    Drift = RW(Close ~ drift())
  ) %>%
  forecast(h=42) %>%
  autoplot(fb_stock, level = NULL) +
  ggtitle("Facebook closing stock price (daily ending Sep 2018)") +
  xlab("Day") + ylab("") +
  guides(colour=guide_legend(title="Forecast"))
```

Facebook closing stock price

Your turn

- Produce forecasts from the appropriate method for Amazon closing price (`gafa_stock`) and Australian takeaway food turnover (`aus_retail`).
- Plot the results using `autoplot()`.

Outline

Variance stabilization

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

Variance stabilization

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

Denote original observations as $y_1, \dots, y_n$ and transformed observations as $w_1, \dots, w_n$.

Variance stabilization

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

Denote original observations as $y_1, \dots, y_n$ and transformed observations as $w_1, \dots, w_n$.

Mathematical transformations for stabilizing variation

Square root	$w_t = \sqrt{y_t}$	$\downarrow$
Cube root	$w_t = \sqrt[3]{y_t}$	Increasing
Logarithm	$w_t = \log(y_t)$	strength

Variance stabilization

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

Denote original observations as $y_1, \ldots, y_n$ and transformed observations as $w_1, \ldots, w_n$.

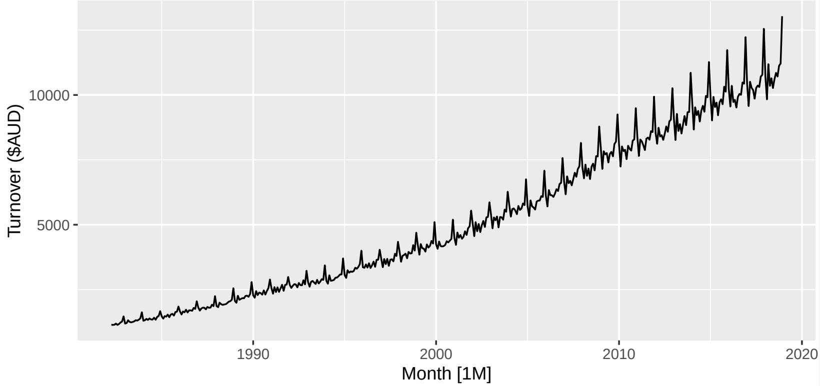
Mathematical transformations for stabilizing variation

Square root	$w_t = \sqrt{y_t}$	$\downarrow$
Cube root	$w_t = \sqrt[3]{y_t}$	Increasing
Logarithm	$w_t = \log(y_t)$	strength

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

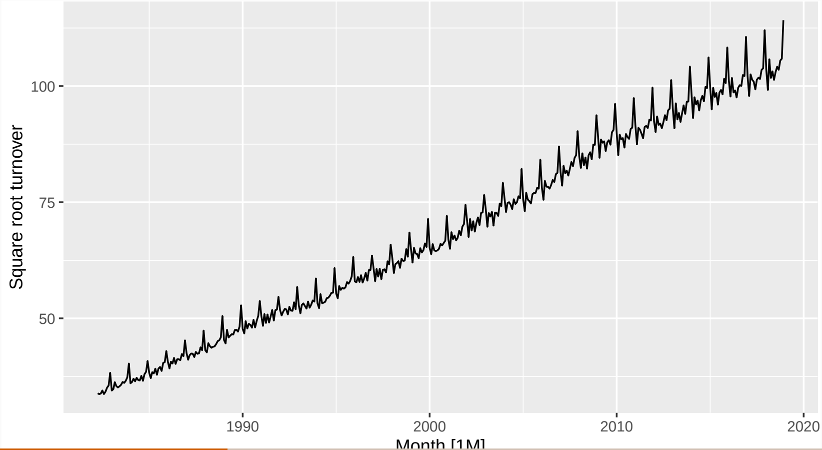
Variance stabilization

```r
food <- aus_retail %>%
  filter(Industry == "Food retailing") %>%
  summarise(Turnover = sum(Turnover))
```



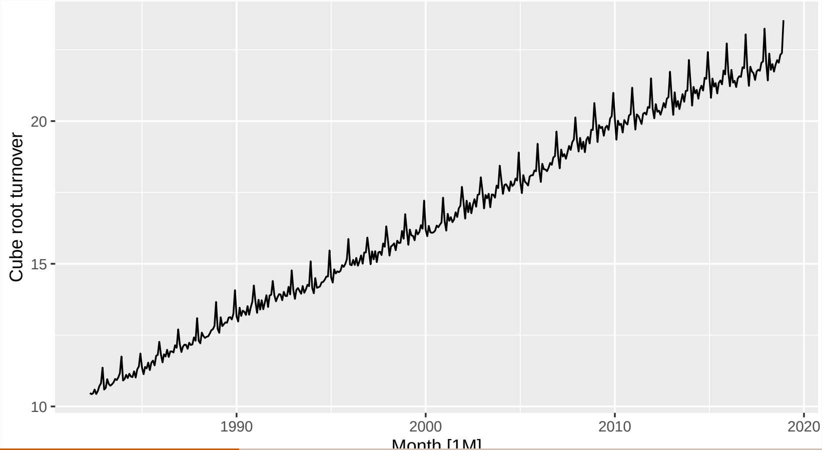
Variance stabilization

```r
food %>% autoplot(sqrt(Turnover)) +
  labs(y = "Square root turnover")
```



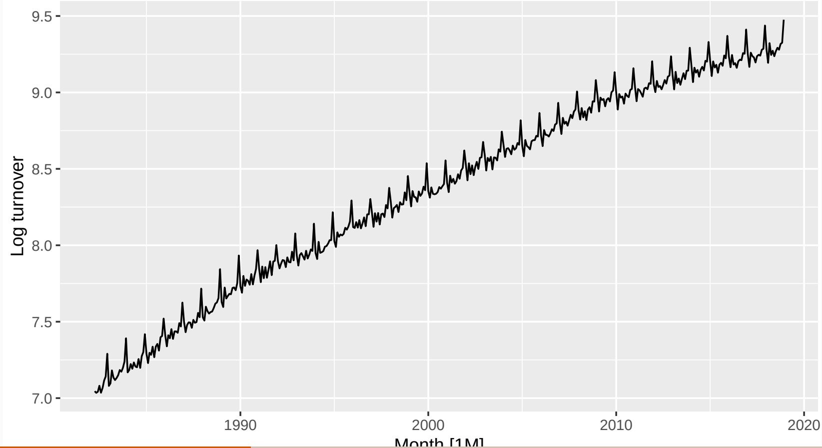
Variance stabilization

```
food %>% autoplot(Turnover^(1/3)) +
  labs(y = "Cube root turnover")
```



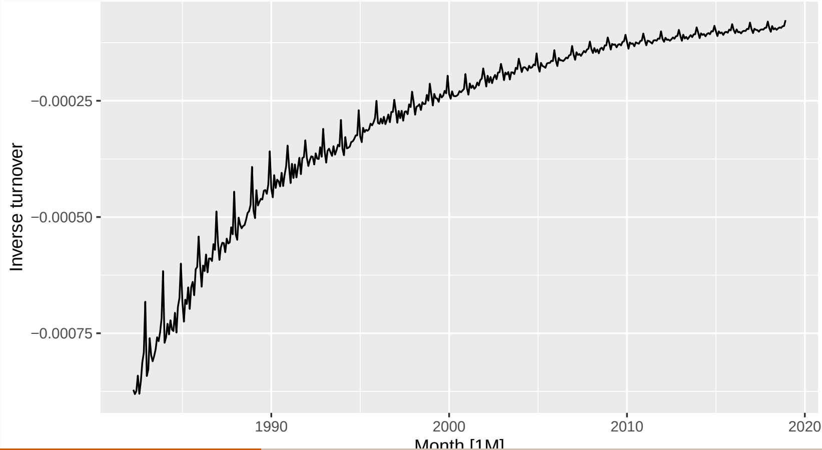
Variance stabilization

```
food %>% autoplot(log(Turnover)) +
  labs(y = "Log turnover")
```



Variance stabilization

```
food %>% autoplot(-1/Turnover) +
  labs(y = "Inverse turnover")
```



Box-Cox transformations

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

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

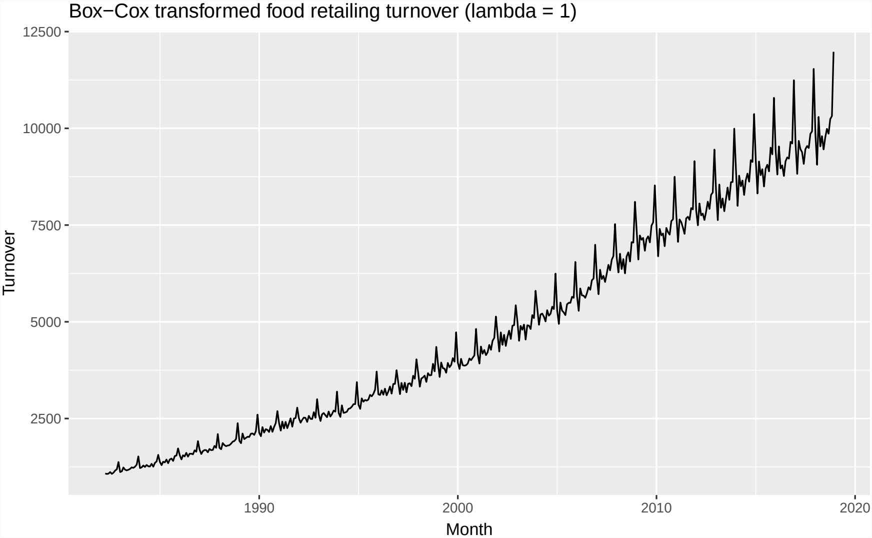
Box-Cox transformations

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

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

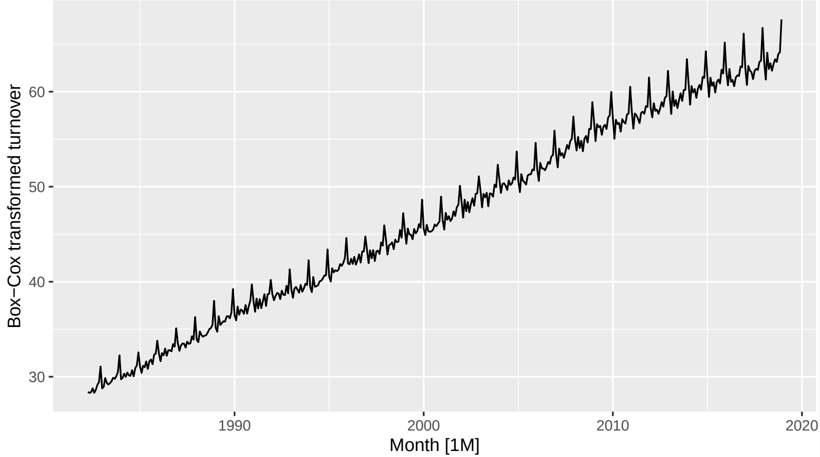
- $\lambda = 1$: (No substantive transformation)
- $\lambda = \frac{1}{2}$: (Square root plus linear transformation)
- $\lambda = 0$: (Natural logarithm)
- $\lambda = -1$: (Inverse plus 1)

Box-Cox transformations



Box-Cox transformations

```
food %>% autoplot(box_cox(Turnover, 1/3)) +
  labs(y = "Box-Cox transformed turnover")
```



Box-Cox transformations

- y_t^λ for λ close to zero behaves like logs.
- If some $y_t = 0$, then must have $\lambda > 0$
- if some $y_t < 0$, no power transformation is possible unless all y_t adjusted by **adding a constant to all values**.
- Simple values of λ are easier to explain.
- Results are relatively insensitive to λ.
- 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

Box-Cox transformations

```
food %>%
  features(Turnover, features = guerrero)
```

```
## # A tibble: 1 x 1
##   lambda_guerrero
##             <dbl>
## 1          0.0524
```

Box-Cox transformations

```r
food %>%
  features(Turnover, features = guerrero)
```

```
## # A tibble: 1 x 1
##   lambda_guerrero
##             <dbl>
## 1          0.0524
```

- This attempts to balance the seasonal fluctuations and random variation across the series.
- Always check the results.
- A low value of λ 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

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

Modelling with transformations

Transformations used in the left of the formula will be automatically back-transformed. To model log-transformed food retailing turnover, you could use:

```
fit <- food %>%
  model(SNAIVE(log(Turnover) ~ lag("year")))
```

```
## # A mable: 1 x 1
##   `SNAIVE(log(Turnover) ~ lag("year"))`
##   <model>
## 1 <SNAIVE>
```

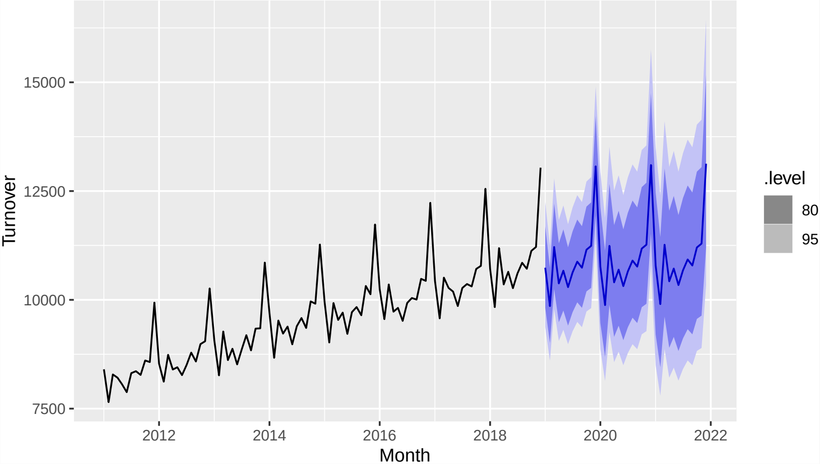
Forecasting with transformations

```r
fc <- fit %>%
  forecast(h = "3 years")
```

```
## # A fable: 36 x 4 [1M]
## # Key:     .model [1]
##   .model                         Month Turnover .distribution
##   <chr>                          <mth>    <dbl> <dist>
## 1 "SNAIVE(log(Turnover) ~  2019 Jan   10738. t(N(9.3, 0.004~
## 2 "SNAIVE(log(Turnover) ~  2019 Feb    9856. t(N(9.2, 0.004~
## 3 "SNAIVE(log(Turnover) ~  2019 Mar   11214. t(N(9.3, 0.004~
## 4 "SNAIVE(log(Turnover) ~  2019 Apr   10378. t(N(9.2, 0.004~
## 5 "SNAIVE(log(Turnover) ~  2019 May   10670. t(N(9.3, 0.004~
## 6 "SNAIVE(log(Turnover) ~  2019 Jun   10292. t(N(9.2, 0.004~
## # ... with 30 more rows
```

Forecasting with transformations

```r
fc %>% autoplot(filter(food,year(Month)>2010))
```



Your turn

Find a transformation that works for the Australian gas production (`aus_production`).

Bias adjustment

- Back-transformed point forecasts are medians.
- Back-transformed PI have the correct coverage.

Bias adjustment

- Back-transformed point forecasts are medians.
- Back-transformed PI have the correct coverage.

Back-transformed means

Let X be have mean μ and variance σ^2.

Let $f(x)$ be back-transformation function, and $Y = f(X)$.

Taylor series expansion about μ:

$$f(X) = f(\mu) + (X - \mu)f'(\mu) + \frac{1}{2}(X - \mu)^2 f''(\mu).$$

Bias adjustment

- Back-transformed point forecasts are medians.
- Back-transformed PI have the correct coverage.

Back-transformed means

Let X be have mean μ and variance σ^2.

Let $f(x)$ be back-transformation function, and $Y = f(X)$.

Taylor series expansion about μ:

$$f(X) = f(\mu) + (X - \mu)f'(\mu) + \frac{1}{2}(X - \mu)^2 f''(\mu).$$

$$\text{E}[Y] = \text{E}[f(X)] = f(\mu) + \tfrac{1}{2}\sigma^2 f''(\mu)$$

Bias adjustment

Box-Cox back-transformation:

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

$$f(x) = \begin{cases} e^x & \lambda = 0; \\ (\lambda x + 1)^{1/\lambda} & \lambda \neq 0. \end{cases}$$

$$f''(x) = \begin{cases} e^x & \lambda = 0; \\ (1 - \lambda)(\lambda x + 1)^{1/\lambda - 2} & \lambda \neq 0. \end{cases}$$

Bias adjustment

Box-Cox back-transformation:

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

$$f(x) = \begin{cases} e^x & \lambda = 0; \\ (\lambda x + 1)^{1/\lambda} & \lambda \neq 0. \end{cases}$$

$$f''(x) = \begin{cases} e^x & \lambda = 0; \\ (1 - \lambda)(\lambda x + 1)^{1/\lambda - 2} & \lambda \neq 0. \end{cases}$$

$$\mathrm{E}[Y] = \begin{cases} e^{\mu}\left[1 + \frac{\sigma^2}{2}\right] & \lambda = 0; \\ (\lambda\mu + 1)^{1/\lambda}\left[1 + \frac{\sigma^2(1-\lambda)}{2(\lambda\mu+1)^2}\right] & \lambda \neq 0. \end{cases}$$

Bias adjustment

```r
eggs <- as_tsibble(fma::eggs)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```r
fit <- eggs %>% model(RW(log(value) ~ drift()))
fc <- fit %>% forecast(h=50)
fc_biased <- fit %>% forecast(h=50, bias_adjust = FALSE)
eggs %>% autoplot(value) +
  autolayer(fc_biased, series="Simple back transformation", level=80) +
  autolayer(fc, series="Bias adjusted", level=NULL) +
  guides(colour=guide_legend(title="Forecast"))
```

```
## Warning: Ignoring unknown parameters: series
```

```
## Warning: Ignoring unknown parameters: series
```

Outline

Forecast distributions

- A forecast $\hat{y}_{T+h|T}$ is (usually) the mean of the conditional distribution $y_{T+h} \mid y_1, \dots, y_T$.
- Most time series models produce normally distributed forecasts.
- The forecast distribution describes the probability of observing any future value.

Forecast distributions

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

Mean:	$\hat{y}_{T+h\|T} \sim N(\bar{y}, (1 + 1/T)\hat{\sigma}^2)$
Naïve:	$\hat{y}_{T+h\|T} \sim N(y_T, h\hat{\sigma}^2)$
Seasonal naïve:	$\hat{y}_{T+h\|T} \sim N(y_{T+h-m(k+1)}, (k + 1)\hat{\sigma}^2)$
Drift:	$\hat{y}_{T+h\|T} \sim N(y_T + \frac{h}{T-1}(y_T - y_1), h\frac{T+h}{T}\hat{\sigma}^2)$

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 forecast variance: $\hat{\sigma}^2$.

Prediction intervals

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

Prediction intervals

```r
fit <- fb_stock %>% model(NAIVE(Close))
forecast(fit)
```

```
## # A fable: 2 x 5 [1]
## # Key:     Symbol, .model [1]
##   Symbol .model    trading_day Close .distribution
##   <chr>  <chr>           <dbl> <dbl> <dist>
## 1 FB     NAIVE(Cl~        3693  176. N(176, 21)
## 2 FB     NAIVE(Cl~        3694  176. N(176, 42)
```

Prediction intervals

```
res_sd <- sqrt(mean(augment(fit)$.resid^2, na.rm = TRUE))
last(fb_stock$Close) + 1.96 * res_sd * c(-1,1)
```

```
## [1] 166.7196 184.7404
```

```
forecast(fit, h = 1) %>%
  transmute(interval = hilo(.distribution, level = 95))
```

```
## # A tsibble: 1 x 4 [1]
## # Key:       Symbol, .model [1]
##   Symbol .model      trading_day                  interval
##   <chr>  <chr>             <dbl>                    <hilo>
## 1 FB     NAIVE(Close)       3693 [166.7198, 184.7402]95
```

Prediction intervals

- Point forecasts are often useless without a measure of uncertainty (such as 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

- Computed automatically from the forecast distribution.
- Use `level` argument to control coverage.
- Check residual assumptions before believing them (we will see this next class).
- Usually too narrow due to unaccounted uncertainty. # Residual diagnostics

Fitted values

- $\hat{y}_{t|t-1}$ is the forecast of y_t based on observations $y_1, \dots, y_t$.
- We call these "fitted values".
- Sometimes drop the subscript: $\hat{y}_t \equiv \hat{y}_{t|t-1}$.
- Often not true forecasts since parameters are estimated on all data.

For example:

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

Forecasting residuals

Residuals in forecasting: difference between observed value and its fitted value: $e_t = y_t - \hat{y}_{t|t-1}$.

Forecasting residuals

Residuals in forecasting: 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, then information left in residuals that should be used in computing forecasts.
2. $\{e_t\}$ have mean zero. If they don't, then forecasts are biased.

Forecasting residuals

Residuals in forecasting: 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, then information left in residuals that should be used in computing forecasts.
2. $\{e_t\}$ have mean zero. If they don't, then forecasts are biased.

Useful properties (for prediction intervals)

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

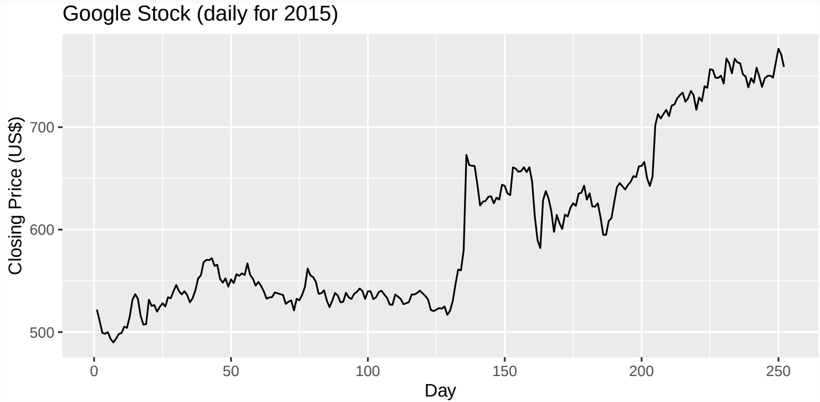
Example: Google stock price

```r
google_2015 <- tsibbledata::gafa_stock %>%
  filter(Symbol == "GOOG", year(Date) == 2015) %>%
  mutate(trading_day = row_number()) %>%
  update_tsibble(index = trading_day, regular = TRUE)
```

```
## # A tsibble: 252 x 9 [1]
## # Key:       Symbol [1]
##    Symbol Date        Open  High   Low Close Adj_Close  Volume
##    <chr>  <date>     <dbl> <dbl> <dbl> <dbl>     <dbl>   <dbl>
##  1 GOOG   2015-01-02  526.  528.  521.  522.      522. 1447600
##  2 GOOG   2015-01-05  520.  521.  510.  511.      511. 2059800
##  3 GOOG   2015-01-06  512.  513.  498.  499.      499. 2899900
##  4 GOOG   2015-01-07  504.  504.  497.  498.      498. 2065100
##  5 GOOG   2015-01-08  495.  501.  488.  500.      500. 3353600
##  6 GOOG   2015-01-09  502.  502.  492.  493.      493. 2069400
##  7 GOOG   2015-01-12  492.  493.  485.  490.      490. 2322400
##  8 GOOG   2015-01-13  496.  500.  490.  493.      493. 2370500
##  9 GOOG   2015-01-14  492.  500.  490.  498.      498. 2235700
## 10 GOOG   2015-01-15  503.  503.  495.  499.      499. 2715800
```

Example: Google stock price

```r
google_2015 %>%
  autoplot(Close) +
    xlab("Day") + ylab("Closing Price (US$)") +
    ggtitle("Google Stock (daily for 2015)")
```



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

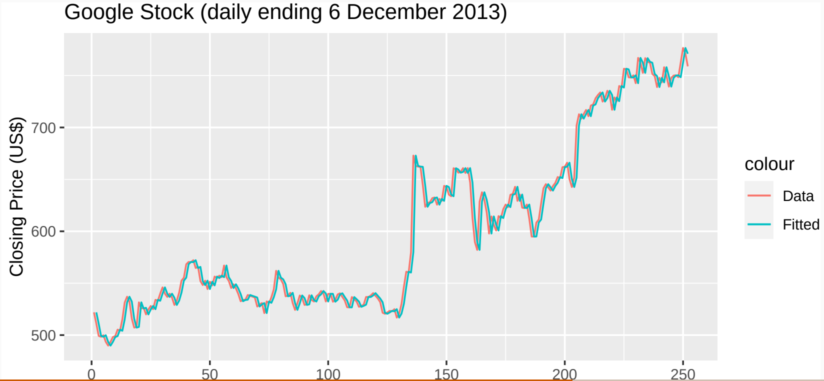
Example: Google stock price

```r
fit <- google_2015 %>% model(NAIVE(Close))
augment(fit)
```

```
## # A tsibble: 252 x 6 [1]
## # Key:       Symbol, .model [1]
##    Symbol .model       trading_day Close .fitted  .resid
##    <chr>  <chr>              <int> <dbl>   <dbl>   <dbl>
##  1 GOOG   NAIVE(Close)           1  522.     NA    NA
##  2 GOOG   NAIVE(Close)           2  511.    522. -10.9
##  3 GOOG   NAIVE(Close)           3  499.    511. -11.8
##  4 GOOG   NAIVE(Close)           4  498.    499.  -0.855
##  5 GOOG   NAIVE(Close)           5  500.    498.   1.57
##  6 GOOG   NAIVE(Close)           6  493.    500.  -6.47
##  7 GOOG   NAIVE(Close)           7  490.    493.  -3.60
##  8 GOOG   NAIVE(Close)           8  493.    490.   3.61
##  9 GOOG   NAIVE(Close)           9  498.    493.   4.66
## 10 GOOG   NAIVE(Close)          10  499.    498.   0.915
## # ... with 242 more rows
```

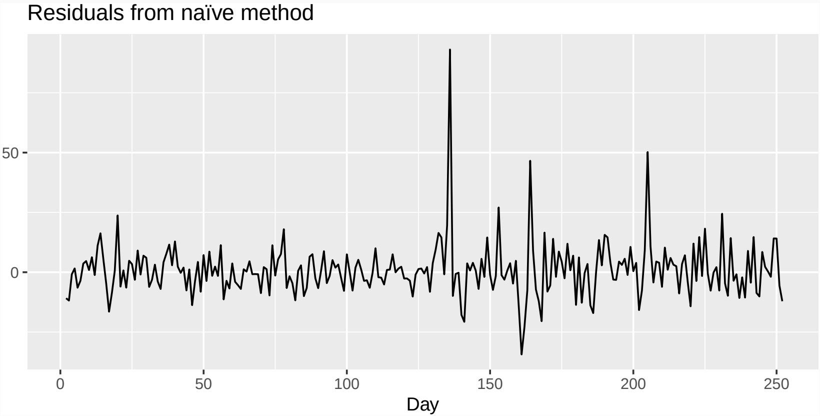
Example: Google stock price

```r
augment(fit) %>%
  ggplot(aes(x = trading_day)) +
    geom_line(aes(y = Close, colour = "Data")) +
    geom_line(aes(y = .fitted, colour = "Fitted")) +
    xlab("Day") + ylab("Closing Price (US$)") +
    ggtitle("Google Stock (daily ending 6 December 2013)")
```



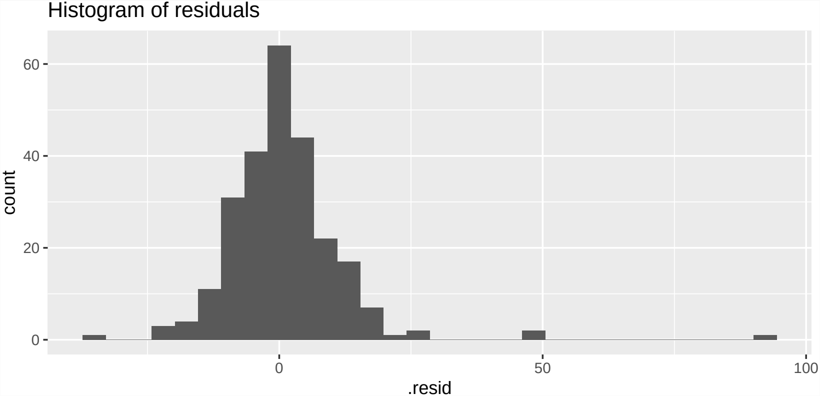
Example: Google stock price

```r
augment(fit) %>%
  autoplot(.resid) + xlab("Day") + ylab("") +
    ggtitle("Residuals from naïve method")
```



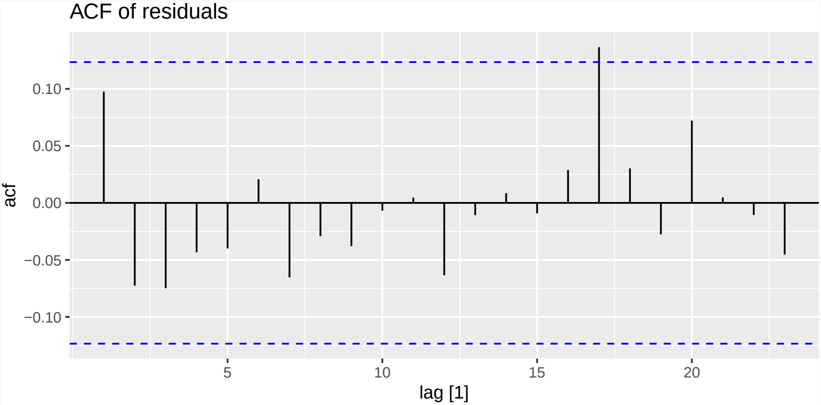
Example: Google stock price

```r
augment(fit) %>%
  ggplot(aes(x = .resid)) +
    geom_histogram(bins = 30) +
    ggtitle("Histogram of residuals")
```



Example: Google stock price

```r
augment(fit) %>% ACF(.resid) %>%
  autoplot() + ggtitle("ACF of residuals")
```



ACF of residuals

- We assume that the residuals are white noise (uncorrelated, mean zero, constant variance). If they aren't, then there is information left in the residuals that should be used in computing forecasts.
- So a standard residual diagnostic is to check the ACF of the residuals of a forecasting method.
- We *expect* these to look like white noise.

Portmanteau tests

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

Portmanteau tests

Consider a *whole set* of 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 values large (positive or negative), Q will be **large**.

Portmanteau tests

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

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.

- My preferences: $h = 10$ for non-seasonal data, $h = 2m$ for seasonal data.
- Better performance, especially in small samples.

Portmanteau tests

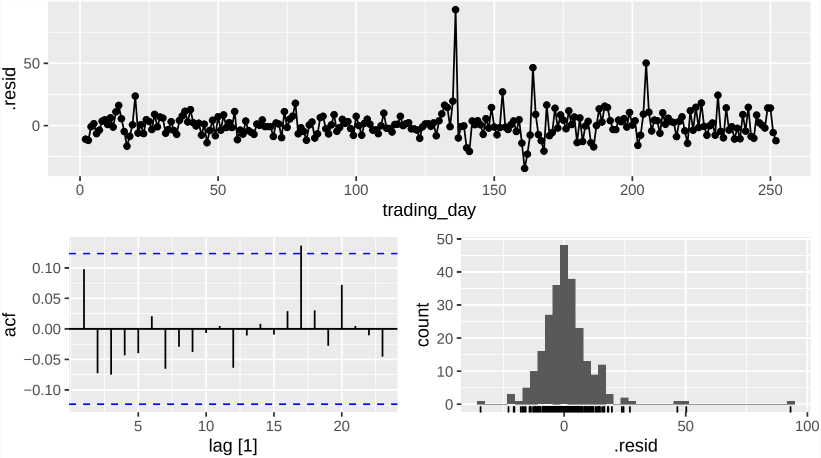
- If data are WN, Q^* has χ^2 distribution with $(h - K)$ degrees of freedom where K = no. parameters in model.
- When applied to raw data, set $K = 0$.

```
# lag=h and fitdf=K
Box.test(augment(fit)$.resid,
  lag = 10, fitdf = 0, type = "Lj")
```

```
##
##  Box-Ljung test
##
## data:  augment(fit)$.resid
## X-squared = 7.9141, df = 10, p-value = 0.6372
```

gg_tsdisplay function

```r
augment(fit) %>%
  gg_tsdisplay(.resid, plot_type = "histogram")
```



Your turn

Compute seasonal naïve forecasts for quarterly Australian beer production from 1992.

```
recent <- aus_production %>% filter(year(Quarter) >= 1992)
fit <- recent %>% model(SNAIVE(Beer))
fit %>% forecast() %>% autoplot(recent)
```

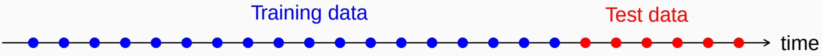
Test if the residuals are white noise.

```
Box.test(augment(fit)$.resid, lag=10, fitdf=0, type="Lj")
augment(fit) %>% gg_tsdisplay(.resid, plot_type = "hist")
```

What do you conclude?

Outline

Training and test sets



- A model which fits the training data well will 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 just as bad as failing to identify a systematic pattern in the data.
- The test set must not be used for *any* aspect of model development or calculation of forecasts.
- Forecast accuracy is based only on the test set.

Forecast errors

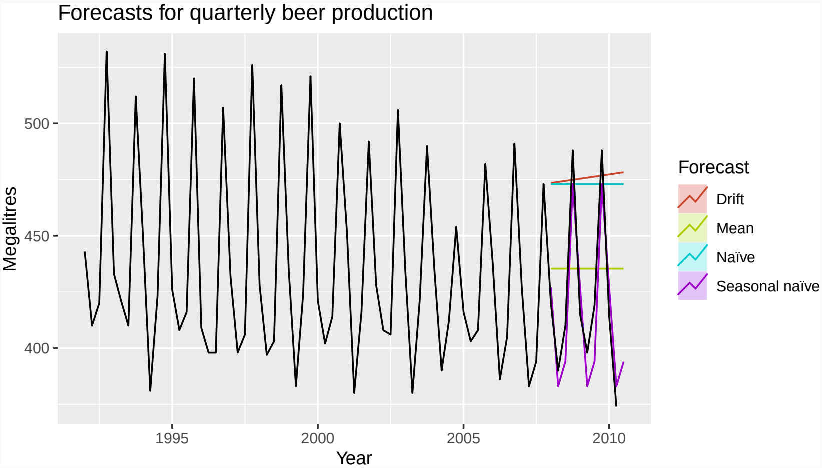
Forecast "error": the difference between an observed value and its forecast.

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

where the training data is given by $\{y_1, \dots, y_T\}$

- Unlike residuals, forecast errors on the test set involve multi-step forecasts.
- These are *true* forecast errors as the test data is not used in computing $\hat{y}_{T+h|T}$.

Measures of forecast accuracy



Measures of forecast accuracy

$$\begin{aligned} y_{T+h} &= (T+h)\text{th observation, } h = 1,\dots,H \\ \hat{y}_{T+h|T} &= \text{its forecast based on data up to time } T. \\ e_{T+h} &= y_{T+h} - \hat{y}_{T+h|T} \end{aligned}$$

$$\begin{aligned} \text{MAE} &= \text{mean}(|e_{T+h}|) \\ \text{MSE} &= \text{mean}(e_{T+h}^2) \qquad\qquad \text{RMSE} = \sqrt{\text{mean}(e_{T+h}^2)} \\ \text{MAPE} &= 100\text{mean}(|e_{T+h}|/|y_{T+h}|) \end{aligned}$$

Measures of forecast accuracy

$$
\begin{aligned}
y_{T+h} &= \text{$(T+h)$th observation, } h = 1, \dots, H \\
\hat{y}_{T+h|T} &= \text{its forecast based on data up to time } T. \\
e_{T+h} &= y_{T+h} - \hat{y}_{T+h|T}
\end{aligned}
$$

$$
\begin{aligned}
\text{MAE} &= \text{mean}(|e_{T+h}|) \\
\text{MSE} &= \text{mean}(e^2_{T+h}) \qquad\qquad \text{RMSE} = \sqrt{\text{mean}(e^2_{T+h})} \\
\text{MAPE} &= 100\text{mean}(|e_{T+h}|/|y_{T+h}|)
\end{aligned}
$$

- MAE, MSE, RMSE are all scale dependent.
- MAPE is scale independent but is only sensible if $y_t \gg 0$ for all t, and y has a natural zero.

Measures of forecast accuracy

Mean Absolute Scaled Error

$$\text{MASE} = \text{mean}(|e_{T+h}|/Q)$$

where Q is a stable measure of the scale of the time series $\{y_t\}$.

Proposed by Hyndman and Koehler (IJF, 2006).

For non-seasonal time series,

$$Q = (T-1)^{-1}\sum_{t=2}^{T}|y_t - y_{t-1}|$$

works well. Then MASE is equivalent to MAE relative to a naïve method.

Measures of forecast accuracy

Mean Absolute Scaled Error

$$\text{MASE} = \text{mean}(|e_{T+h}|/Q)$$

where Q is a stable measure of the scale of the time series $\{y_t\}$.

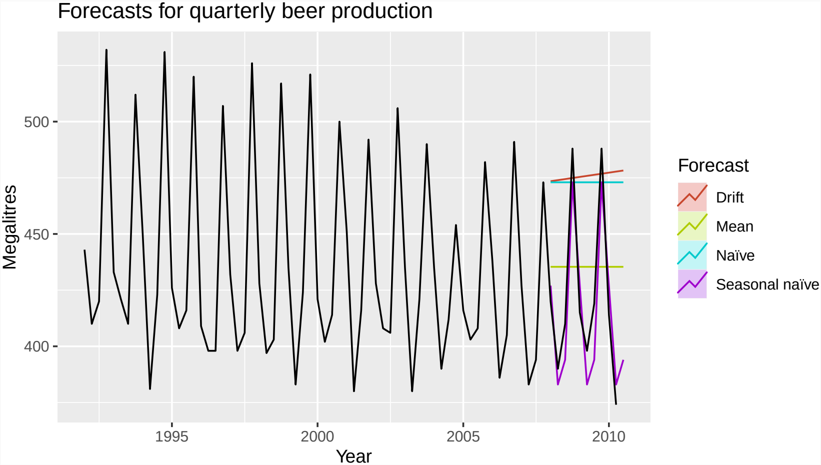
Proposed by Hyndman and Koehler (IJF, 2006).

For seasonal time series,

$$Q = (T-m)^{-1}\sum_{t=m+1}^{T}|y_t - y_{t-m}|$$

works well. Then MASE is equivalent to MAE relative to a seasonal naïve method.

Measures of forecast accuracy



Training set accuracy

```r
recent_production <- aus_production %>%
  filter(year(Quarter) >= 1992)
train <- recent_production %>% filter(year(Quarter) <= 2007)
beer_fit <- train %>%
  model(
    Mean = MEAN(Beer),
    `Naïve` = NAIVE(Beer),
    `Seasonal naïve` = SNAIVE(Beer),
    Drift = RW(Beer ~ drift())
  )
accuracy(beer_fit)
```

	RMSE	MAE	MAPE	MASE
Mean method	43.62858	35.23438	7.886776	2.463942
Naïve method	65.31511	54.73016	12.164154	3.827284
Seasonal naïve method	16.78193	14.30000	3.313685	1.000000
Drift method	65.31337	54.76795	12.178793	3.829927

Test set accuracy

```
beer_fc <- beer_fit %>%
  forecast(h = 10)
accuracy(beer_fc, recent_production)
```

	RMSE	MAE	MAPE	MASE
Drift method	64.90129	58.87619	14.577487	4.1172161
Mean method	38.44724	34.82500	8.283390	2.4353147
Naïve method	62.69290	57.40000	14.184424	4.0139860
Seasonal naïve method	14.31084	13.40000	3.168503	0.9370629

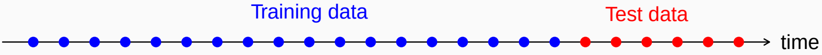
Poll: true or false?

1. Good forecast methods should have normally distributed residuals.
2. A model with small residuals will give good forecasts.
3. The best measure of forecast accuracy is MAPE.
4. If your model doesn't forecast well, you should make it more complicated.
5. Always choose the model with the best forecast accuracy as measured on the test set.

Outline

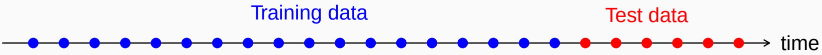
Time series cross-validation

Traditional evaluation

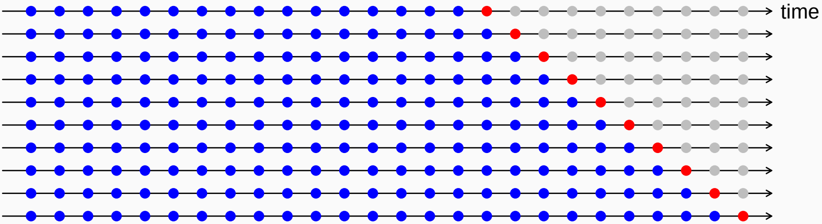


Time series cross-validation

Traditional evaluation

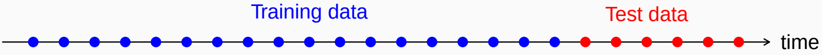


Time series cross-validation

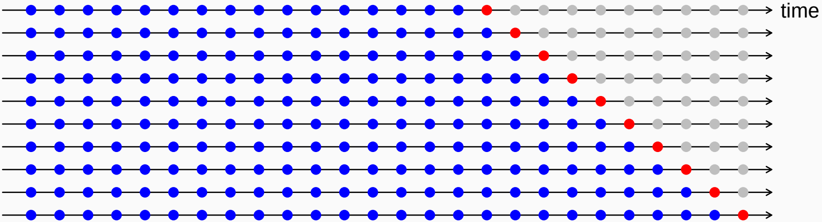


Time series cross-validation

Traditional evaluation



Time series cross-validation



- Forecast accuracy averaged over test sets.
- Also known as "evaluation on a rolling forecasting origin"

Creating the rolling training sets

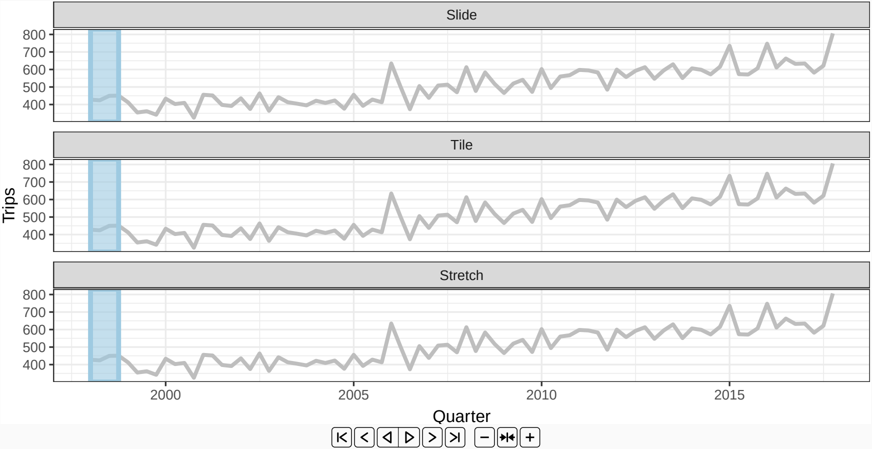
There are three main rolling types which can be used.

- Stretch: extends a growing length window with new data.
- Slide: shifts a fixed length window through the data.
- Tile: moves a fixed length window without overlap.

Three functions to roll a tsibble: `stretch_tsibble()`, `slide_tsibble()`, and `tile_tsibble()`.

For time series cross-validation, stretching windows are most commonly used.

Creating the rolling training sets



Time series cross-validation

Stretch with a minimum length of 3, growing by 1 each step.

```
google_2015_stretch <- google_2015 %>%
  stretch_tsibble(.init = 3, .step = 1) %>%
  filter(.id != max(.id))
```

```
## # A tsibble: 31,623 x 4 [1]
## # Key:       .id [249]
##   Date       Close trading_day   .id
##   <date>     <dbl>       <int> <int>
## 1 2015-01-02  522.           1     1
## 2 2015-01-05  511.           2     1
## 3 2015-01-06  499.           3     1
## 4 2015-01-02  522.           1     2
## 5 2015-01-05  511.           2     2
## 6 2015-01-06  499.           3     2
## 7 2015-01-07  498.           4     2
```

Time series cross-validation

Estimate RW w/ drift models for each window.

```
fit_cv <- google_2015_stretch %>%
  model(RW(Close ~ drift()))
```

```
## # A mable: 249 x 3
## # Key:     .id, Symbol [249]
##      .id Symbol `RW(Close ~ drift())`
##    <int> <chr>  <model>
## 1      1 GOOG   <RW w/ drift>
## 2      2 GOOG   <RW w/ drift>
## 3      3 GOOG   <RW w/ drift>
## 4      4 GOOG   <RW w/ drift>
## # ... with 245 more rows
```

Time series cross-validation

Produce one step ahead forecasts from all models.

```r
fc_cv <- fit_cv %>%
  forecast(h=1)
```

```
## # A fable: 249 x 5 [1]
## # Key:     .id, Symbol [249]
##     .id Symbol trading_day Close .distribution
##   <int> <chr>        <dbl> <dbl> <dist>
## 1     1 GOOG             4  488. N(488, 0.47)
## 2     2 GOOG             5  490. N(490, 37)
## 3     3 GOOG             6  494. N(494, 47)
## 4     4 GOOG             7  488. N(488, 35)
## # ... with 245 more rows
```

Time series cross-validation

```r
# Cross-validated
fc_cv %>% accuracy(google_2015)
# Training set
google_2015 %>% model(NAIVE(Close)) %>% accuracy()
```

	RMSE	MAE	MAPE
Cross-validation	11.26819	7.261240	1.194024
Training	11.18958	7.127985	1.170985

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