MONASH University

MONASH BUSINESS SCHOOL

ETF3231/5231: Business forecasting

Ch5. The forecasters' toolbox

OTexts.org/fpp3/

AACSB ACCREDITED EQUIS ACCREDITED ASSOCIATION AMBA ACCREDITED

Outline

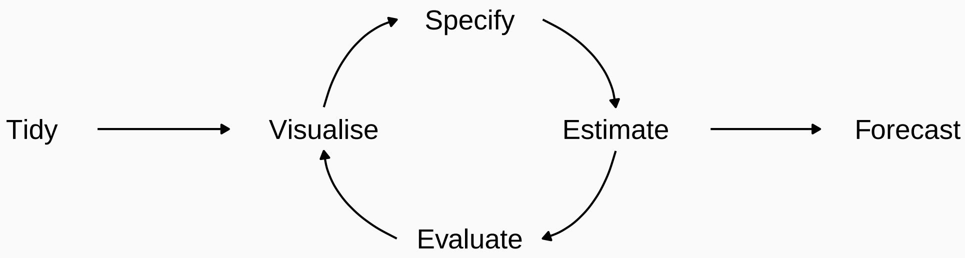
Outline

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



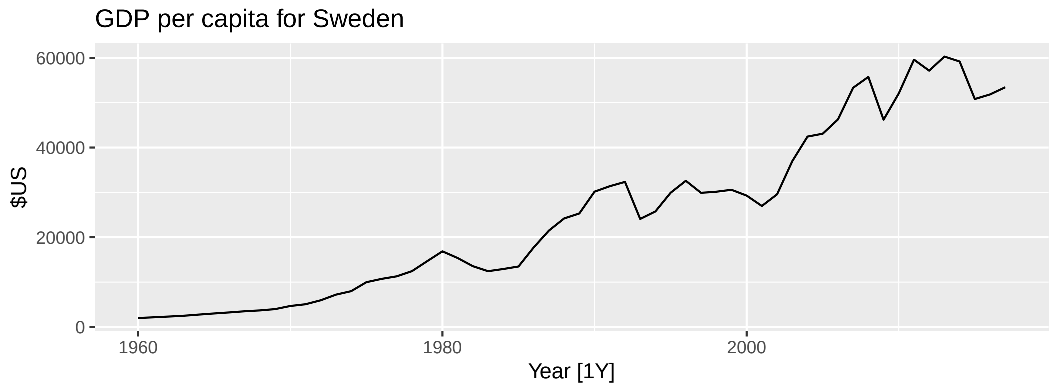
Data preparation (tidy)

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

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

Data visualisation

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



Model estimation

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

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

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

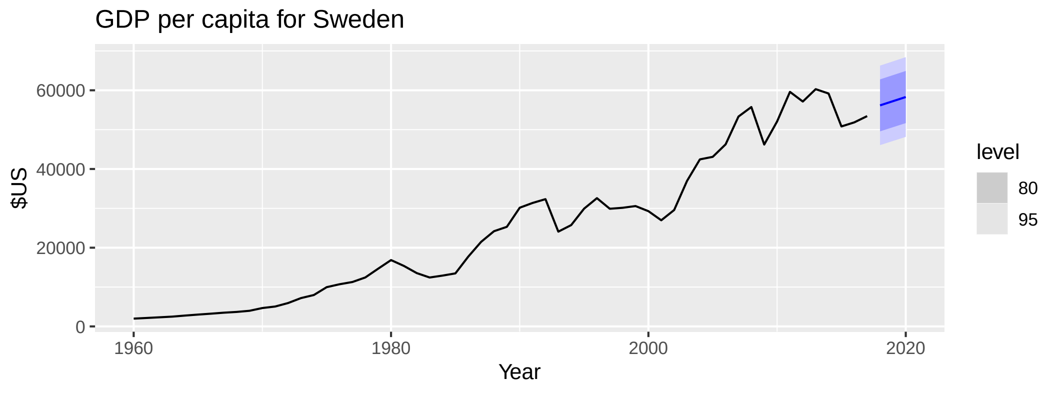
Producing forecasts

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

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

Visualising forecasts

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



Outline

1 A tidy forecasting workflow

2 Some simple forecasting methods

3 Residual diagnostics

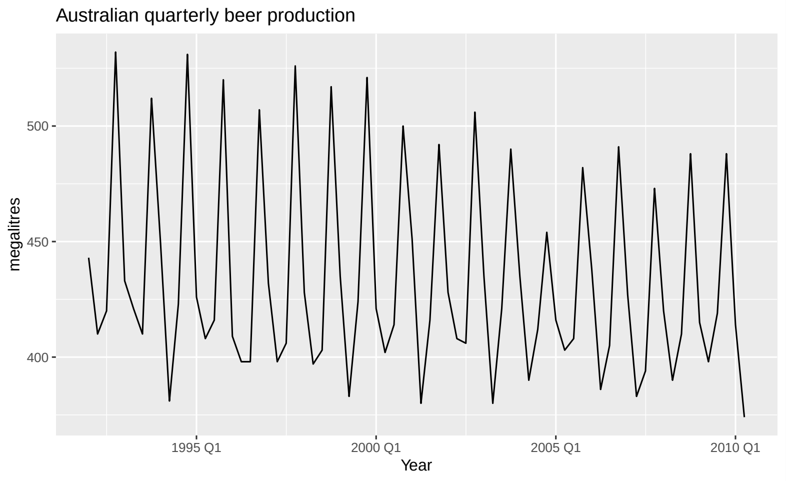
4 Distributional forecasts and prediction intervals

5 Forecasting with transformations

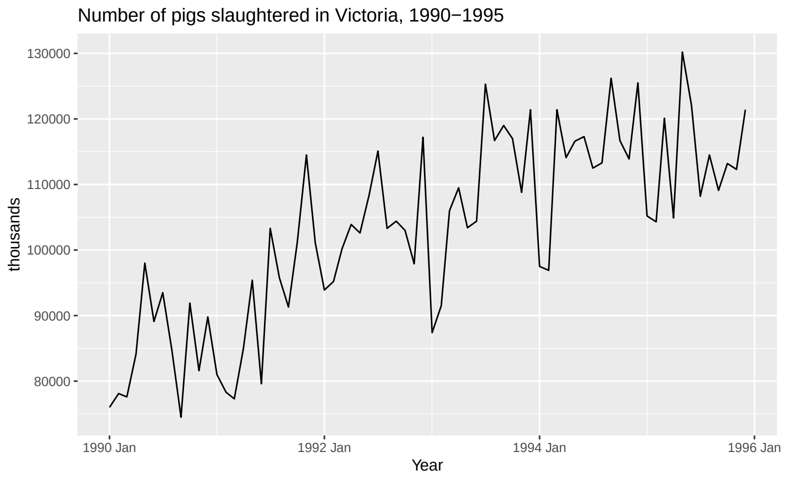
6 Forecasting and decomposition

7 Evaluating forecast accuracy

Some simple forecasting methods



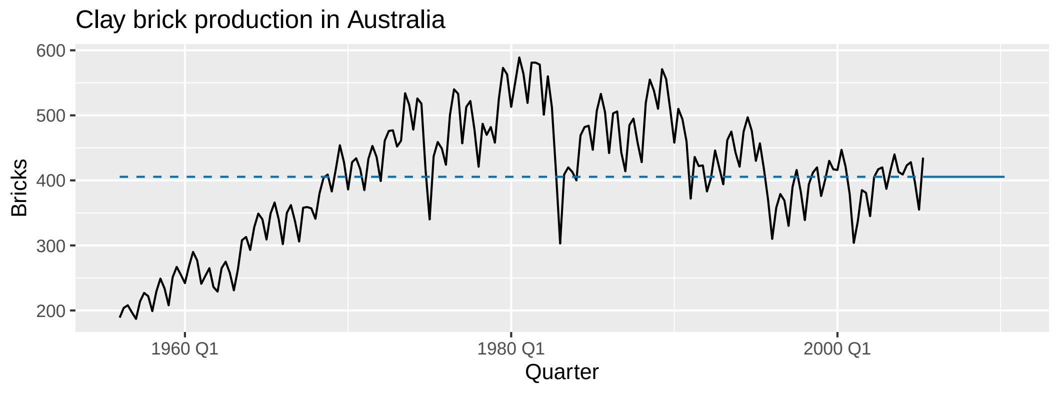
Some simple forecasting methods



Some simple forecasting methods

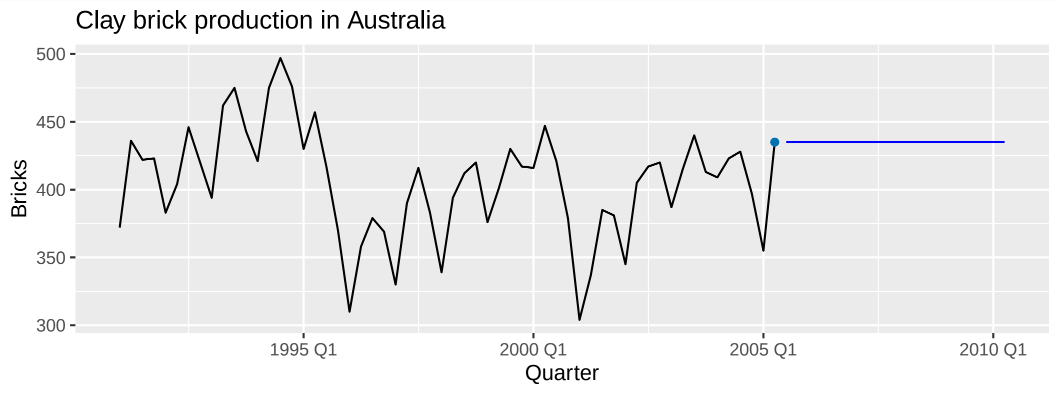
`MEAN(y)`: Average method

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



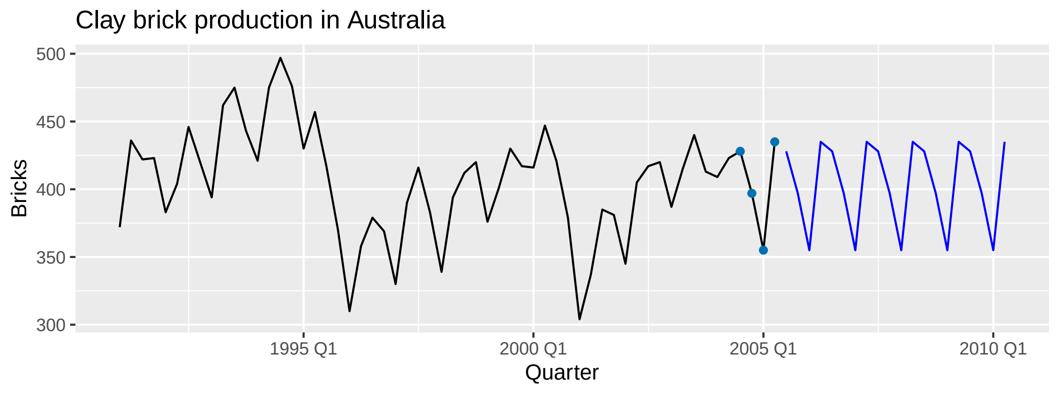
`NAIVE(y)`: Naïve method

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



`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 $(h-1)/m$.



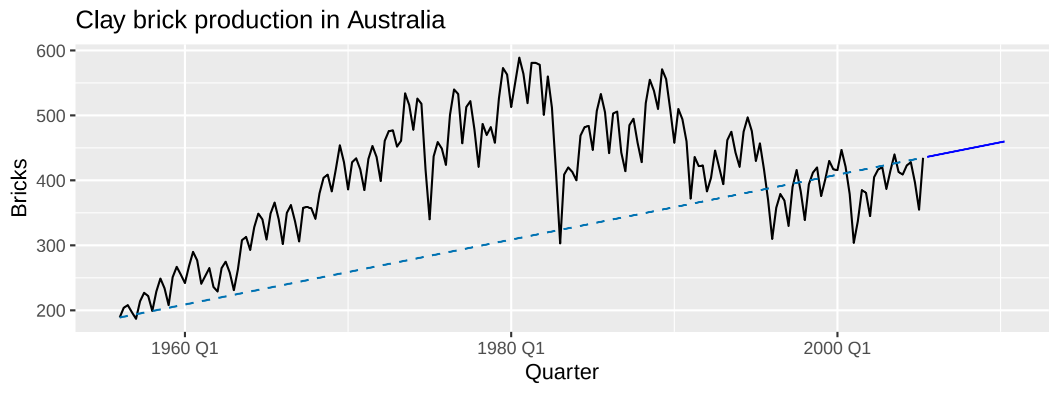
`RW(y ~ drift())`: Drift method

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

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

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

Drift method



Model fitting

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

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

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

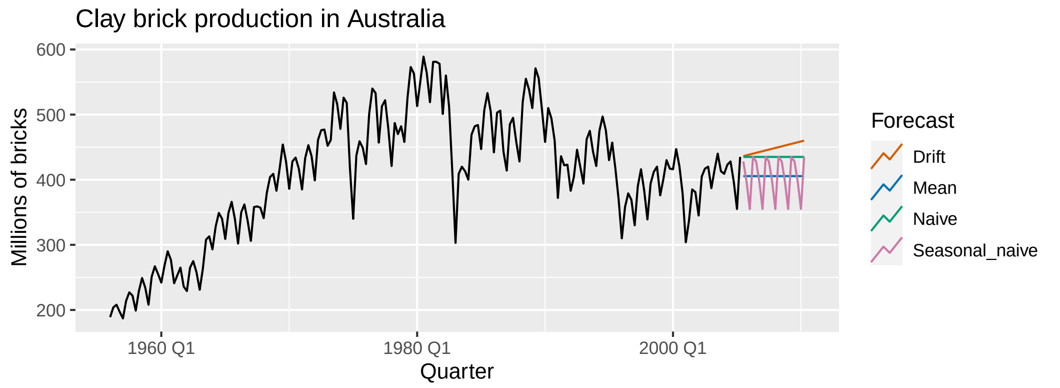
Producing forecasts

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

```
## # A fable: 80 x 4 [1Q]
## # Key:     .model [4]
##   .model         Quarter       Bricks .mean
##   <chr>            <qtr>       <dist> <dbl>
## 1 Seasonal_naive 2005 Q3 N(428, 2336)   428
## 2 Seasonal_naive 2005 Q4 N(397, 2336)   397
## 3 Seasonal_naive 2006 Q1 N(355, 2336)   355
## 4 Seasonal_naive 2006 Q2 N(435, 2336)   435
## # ... with 76 more rows
```

Visualising forecasts

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

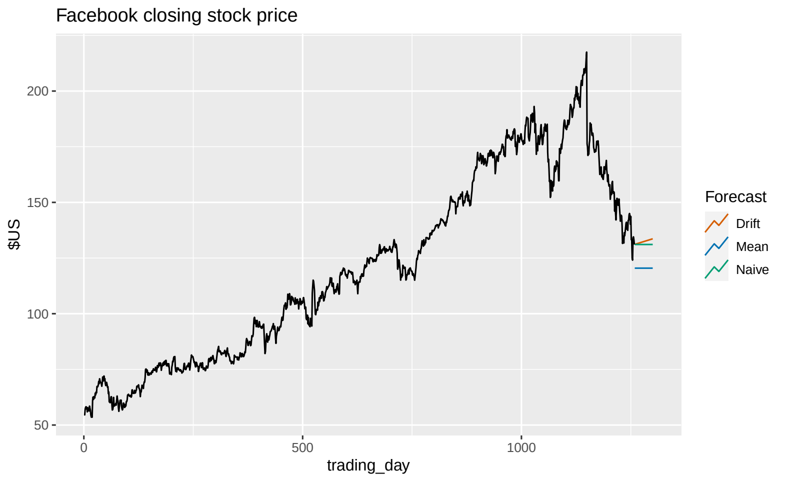


Facebook closing stock price

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

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

Facebook closing stock price



Outline

Fitted values

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

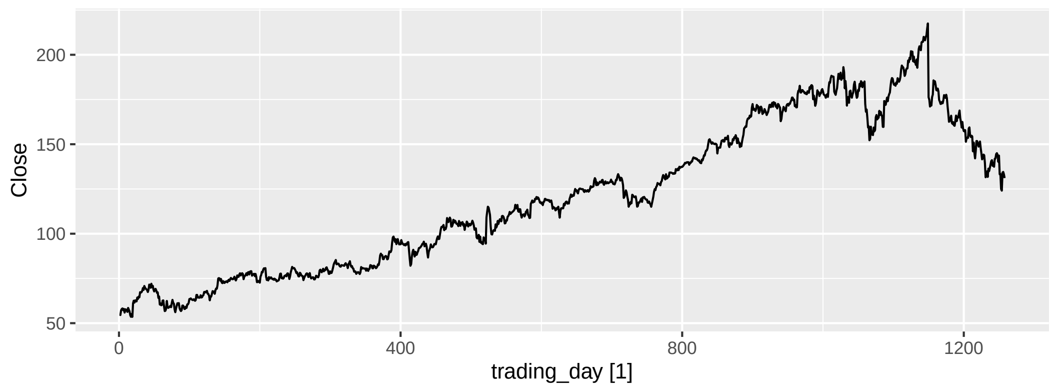
1. $\{e_t\}$ uncorrelated. If they aren't, then information left in residuals that should be used in computing forecasts.
2. $\{e_t\}$ have mean zero. If they don't, then forecasts are biased.

Useful properties (for distributions & prediction intervals)

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

Facebook closing stock price

```
fb_stock <- gafa_stock %>% filter(Symbol == "FB") %>%
  mutate(trading_day = row_number()) %>%
  update_tsibble(index = trading_day, regular = TRUE)
fb_stock %>% autoplot(Close)
```



Facebook closing stock price

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

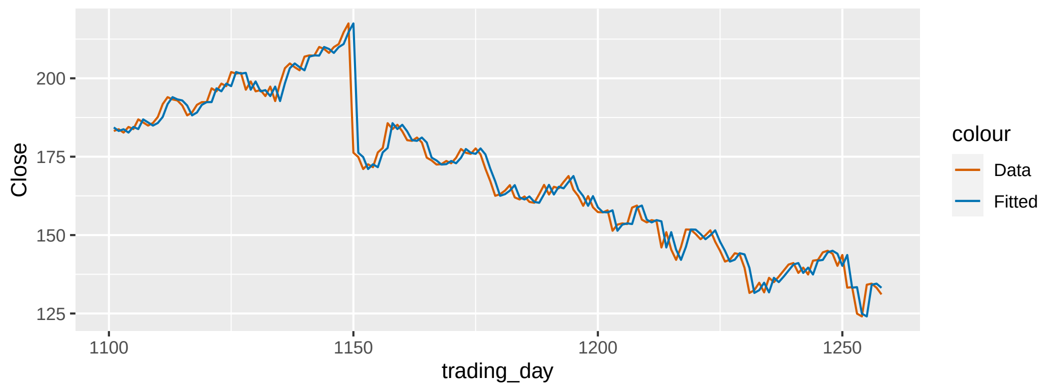
```
## # A tsibble: 1,258 x 7 [1]
## # Key:       Symbol, .model [1]
##    Symbol .model       trading_day Close .fitted .resid .innov
##    <chr>  <chr>              <int> <dbl>   <dbl>  <dbl>  <dbl>
##  1 FB     NAIVE(Close)           1  54.7    NA   NA     NA    
##  2 FB     NAIVE(Close)           2  54.6    54.7 -0.150 -0.150
##  3 FB     NAIVE(Close)           3  57.2    54.6  2.64   2.64 
##  4 FB     NAIVE(Close)           4  57.9    57.2  0.720  0.720
##  5 FB     NAIVE(Close)           5  58.2    57.9  0.310  0.310
##  6 FB     NAIVE(Close)           6  57.2    58.2 -1.01  -1.01 
##  7 FB     NAIVE(Close)           7  57.9    57.2  0.720  0.720
##  8 FB     NAIVE(Close)           8  55.9    57.9 -2.03  -2.03 
##  9 FB     NAIVE(Close)           9  57.7    55.9  1.83   1.83 
## 10 FB     NAIVE(Close)          10  57.6    57.7 -0.140 -0.140
## # ... with 1,248 more rows
```

Facebook closing stock price

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

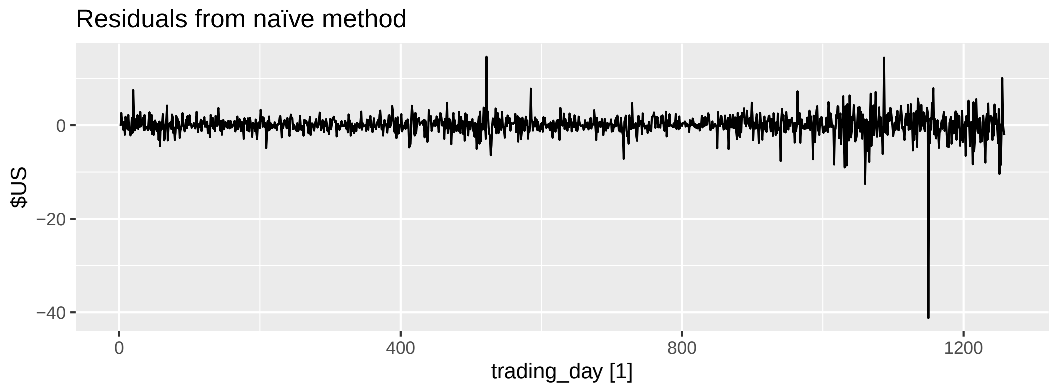
Facebook closing stock price

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



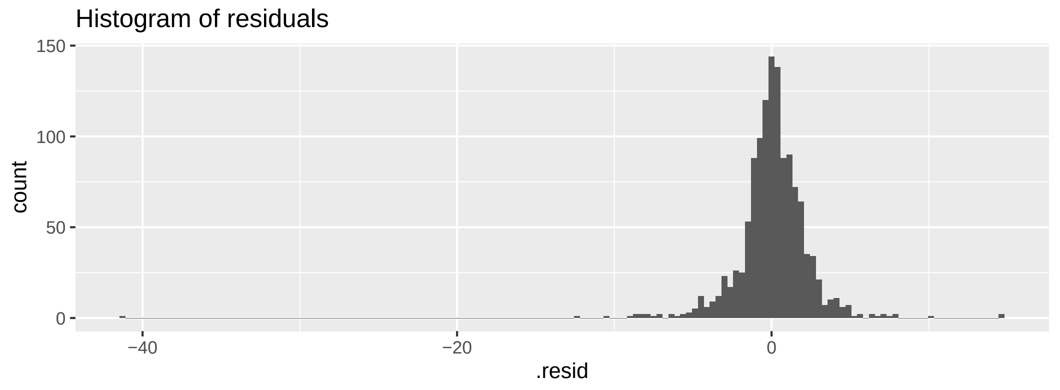
Facebook closing stock price

```
augment(fit) %>%
  autoplot(.resid) +
  labs(y = "$US",
       title = "Residuals from naïve method")
```



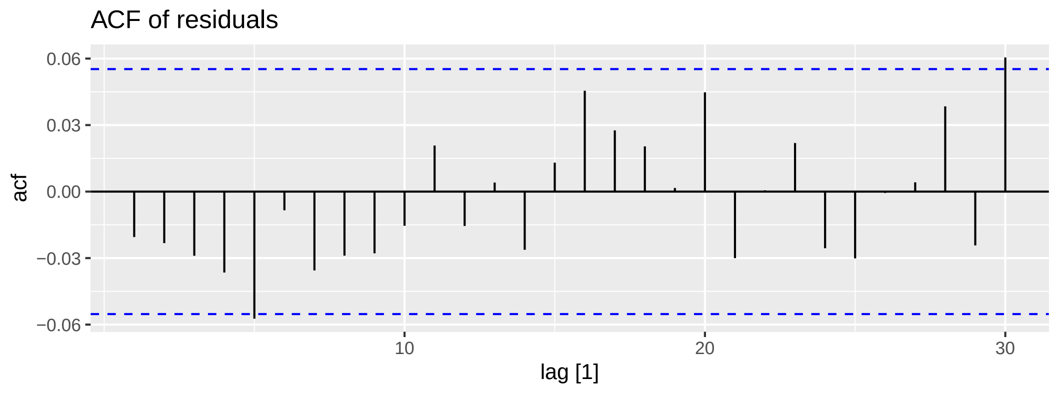
Facebook closing stock price

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



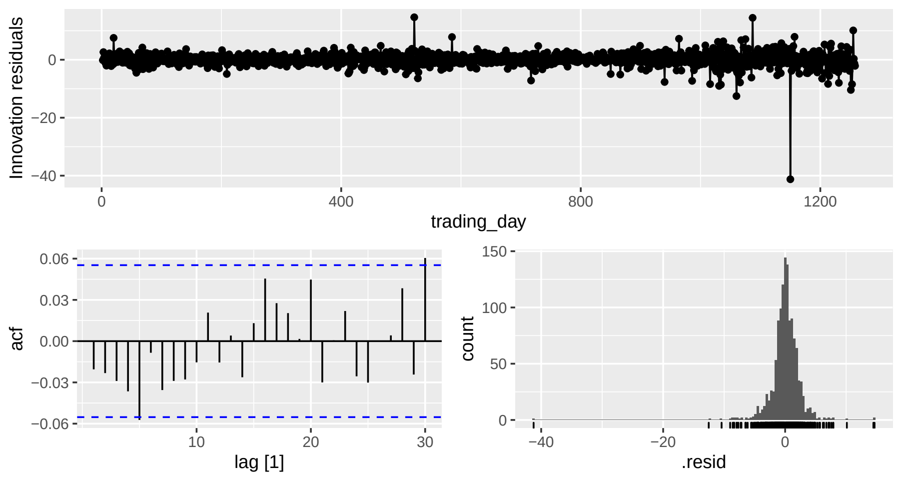
Facebook closing stock price

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



gg_tsresiduals() function

```
gg_tsresiduals(fit)
```



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

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

where ℓ 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

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

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

- My preferences: $\ell = 10$ 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 $(\ell - K)$ degrees of freedom where K = no. parameters in model.
- When applied to raw data, set $K = 0$.
- `lag` = ℓ, `dof` = K

```
augment(fit) %>%
  features(.resid, ljung_box, lag=10, dof=0)
```

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

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: have zero mean, are uncorrelated, normal, with variance = $\hat{\sigma}^2$:

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

```
brick_fc %>% hilo(level = 95)
```

```
## # A tsibble: 80 x 5 [1Q]
## # Key:       .model [4]
##    .model         Quarter        Bricks .mean         `95%`
##    <chr>            <qtr>        <dist> <dbl>        <hilo>
##  1 Seasonal_naive 2005 Q3 N(428, 2336)   428 [333, 523]95
##  2 Seasonal_naive 2005 Q4 N(397, 2336)   397 [302, 492]95
##  3 Seasonal_naive 2006 Q1 N(355, 2336)   355 [260, 450]95
##  4 Seasonal_naive 2006 Q2 N(435, 2336)   435 [340, 530]95
##  5 Seasonal_naive 2006 Q3 N(428, 4672)   428 [294, 562]95
##  6 Seasonal_naive 2006 Q4 N(397, 4672)   397 [263, 531]95
##  7 Seasonal_naive 2007 Q1 N(355, 4672)   355 [221, 489]95
##  8 Seasonal_naive 2007 Q2 N(435, 4672)   435 [301, 569]95
##  9 Seasonal_naive 2007 Q3 N(428, 7008)   428 [264, 592]95
```

Prediction intervals

- Point forecasts often useless without a measure of uncertainty (such as prediction intervals).
- Prediction intervals require a stochastic model (with random errors, etc).
- For most models, prediction intervals get wider as the forecast horizon increases.
- Use `level` argument to control coverage.
- Check residual assumptions before believing them.
- Usually too narrow due to unaccounted uncertainty.

Outline

1 A tidy forecasting workflow
2 Some simple forecasting methods
3 Residual diagnostics
4 Distributional forecasts and prediction intervals
5 Forecasting with transformations
6 Forecasting and decomposition
7 Evaluating forecast accuracy

Mathematical transformations

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 $w_1, \dots, w_n$.

Box-Cox transformations

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

- Natural logarithm, particularly useful because they are more interpretable: changes in a log value are relative (percent) changes on the original scale.

Back-transformation

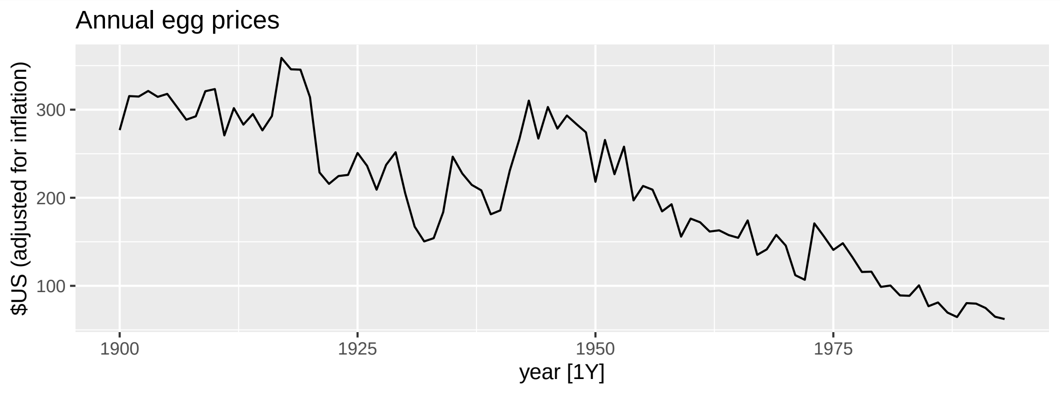
We must reverse the transformation or back-transform to obtain forecasts on the original scale. The reverse Box-Cox transformations are given by

Box-Cox back-transformations

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

Modelling with transformations

```
eggs <- prices %>% filter(!is.na(eggs)) %>% select(eggs)
eggs %>% autoplot() +
  labs(title="Annual egg prices", y="$US (adjusted for inflation)")
```



Modelling with transformations

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

```
fit <- eggs %>%
  model(RW(log(eggs) ~ drift()))
fit
```

```
## # A mable: 1 x 1
##   `RW(log(eggs) ~ drift())`
##                     <model>
## 1             <RW w/ drift>
```

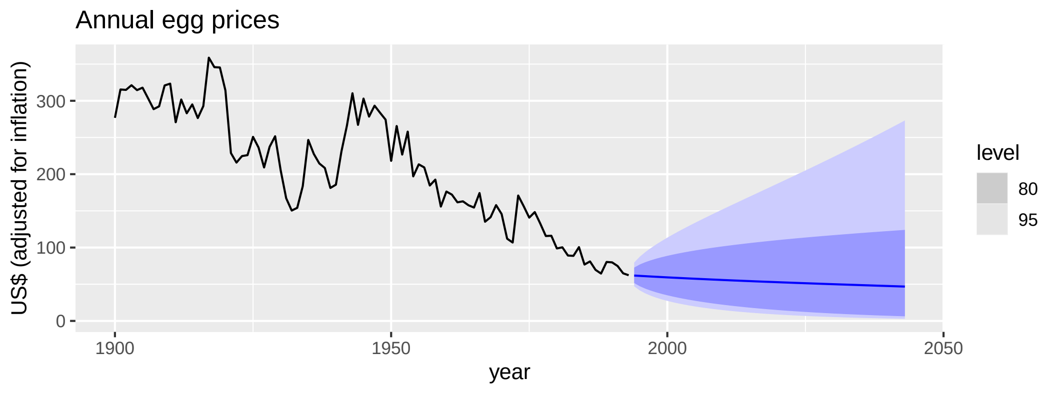
Forecasting with transformations

```r
fc <- fit %>%
  forecast(h = 50)
fc
```

```
## # A fable: 50 x 4 [1Y]
## # Key:     .model [1]
##    .model                       year               eggs .mean
##    <chr>                       <dbl>             <dist> <dbl>
##  1 RW(log(eggs) ~ drift())  1994 t(N(4.1, 0.018))  61.8
##  2 RW(log(eggs) ~ drift())  1995 t(N(4.1, 0.036))  61.4
##  3 RW(log(eggs) ~ drift())  1996 t(N(4.1, 0.054))  61.0
##  4 RW(log(eggs) ~ drift())  1997 t(N(4.1, 0.073))  60.5
##  5 RW(log(eggs) ~ drift())  1998 t(N(4.1, 0.093))  60.1
##  6 RW(log(eggs) ~ drift())  1999    t(N(4, 0.11))  59.7
##  7 RW(log(eggs) ~ drift())  2000    t(N(4, 0.13))  59.3
##  8 RW(log(eggs) ~ drift())  2001    t(N(4, 0.15))  58.9
##  9 RW(log(eggs) ~ drift())  2002    t(N(4, 0.17))  58.6
## 10 RW(log(eggs) ~ drift())  2003    t(N(4, 0.19))  58.2
```

Forecasting with transformations

```
fc %>% autoplot(eggs) +
  labs(title="Annual egg prices",
       y="US$ (adjusted for inflation)")
```



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:

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

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

Bias adjustment

Box-Cox back-transformation:

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

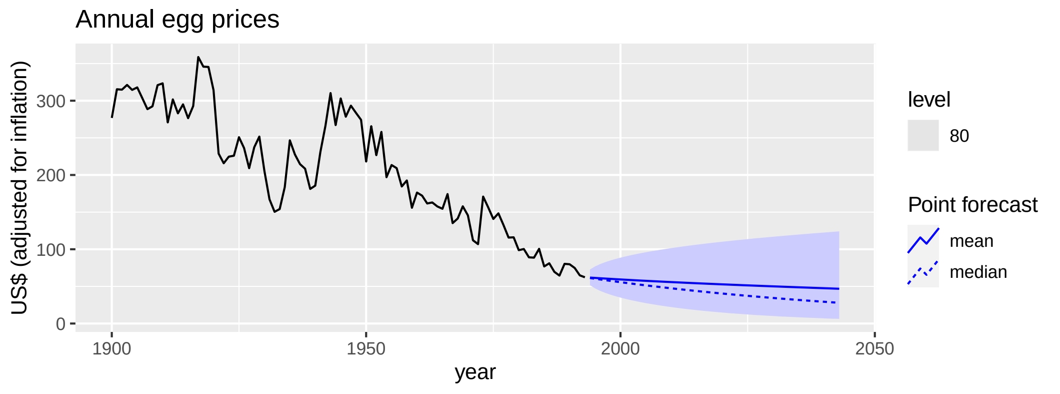
$$f(x) = \begin{cases} e^x & \lambda = 0; \\ (\lambda x + 1)^{1/\lambda} & \lambda \neq 0. \end{cases}$$

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

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

Bias adjustment

```
fc %>%
  autoplot(eggs, level = 80, point_forecast = lst(mean, median)) +
  labs(title="Annual egg prices",
       y="US$ (adjusted for inflation)")
```



Outline

Forecasting and decomposition

$$y_t = \hat{S}_t + \hat{A}_t$$

- $\hat{A}_t$ is seasonally adjusted component
- $\hat{S}_t$ is seasonal component.

- Forecast $\hat{S}_t$ using SNAIVE.
- Forecast $\hat{A}_t$ using non-seasonal time series method.
- Combine forecasts of $\hat{S}_t$ and $\hat{A}_t$ to get forecasts of original data.

US Retail Employment

```
us_retail_employment <- us_employment %>%
  filter(year(Month) >= 1990, Title == "Retail Trade") %>%
  select(-Series_ID)
us_retail_employment
```

```
## # A tsibble: 357 x 3 [1M]
##         Month Title          Employed
##         <mth> <chr>             <dbl>
##  1 1990 Jan Retail Trade   13256.
##  2 1990 Feb Retail Trade   12966.
##  3 1990 Mar Retail Trade   12938.
##  4 1990 Apr Retail Trade   13012.
##  5 1990 May Retail Trade   13108.
##  6 1990 Jun Retail Trade   13183.
##  7 1990 Jul Retail Trade   13170.
##  8 1990 Aug Retail Trade   13160.
##  9 1990 Sep Retail Trade   13113.
## 10 1990 Oct Retail Trade   13185.
```

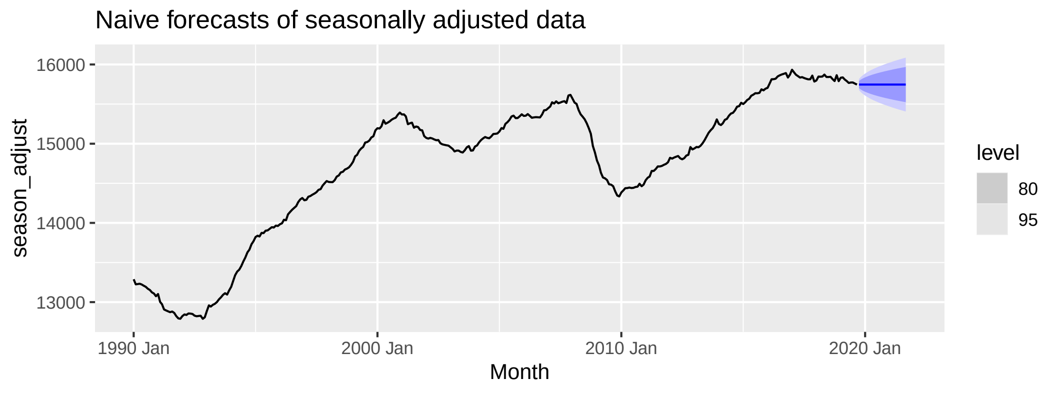
US Retail Employment

```
dcmp <- us_retail_employment %>%
  model(STL(Employed)) %>%
  components() %>% select(-.model)
dcmp
```

```
## # A tsibble: 357 x 6 [1M]
##       Month Employed  trend season_year remainder season_adjust
##       <mth>    <dbl>  <dbl>       <dbl>     <dbl>         <dbl>
##  1 1990 Jan   13256. 13288.       -33.0     0.836        13289.
##  2 1990 Feb   12966. 13269.      -258.    -44.6          13224.
##  3 1990 Mar   12938. 13250.      -290.    -22.1          13228.
##  4 1990 Apr   13012. 13231.      -220.      1.05         13232.
##  5 1990 May   13108. 13211.      -114.     11.3          13223.
##  6 1990 Jun   13183. 13192.       -24.3    15.5          13207.
##  7 1990 Jul   13170. 13172.       -23.2    21.6          13193.
##  8 1990 Aug   13160. 13151.        -9.52   17.8          13169.
##  9 1990 Sep   13113. 13131.       -39.5    22.0          13153.
## 10 1990 Oct   13185. 13110.        61.6    13.2          13124.
```

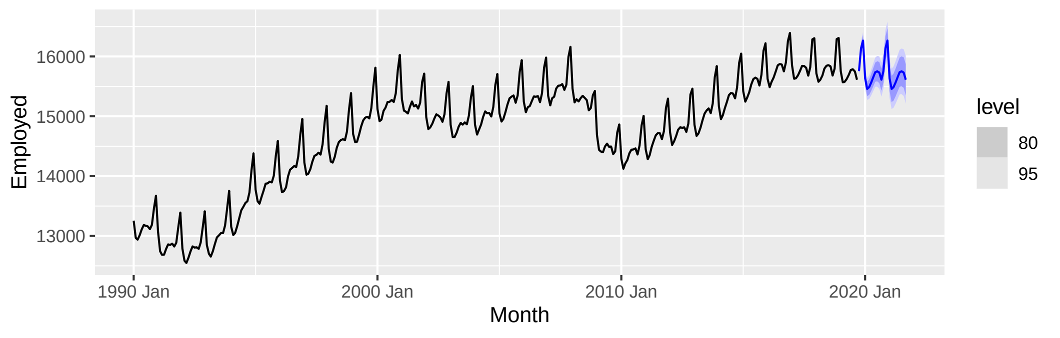
US Retail Employment

```
dcmp %>% model(NAIVE(season_adjust)) %>% forecast() %>%
  autoplot(dcmp) +
  labs(title = "Naive forecasts of seasonally adjusted data")
```



US Retail Employment

```
us_retail_employment %>%
  model(stlf = decomposition_model(
    STL(Employed ~ trend(window = 7), robust = TRUE), NAIVE(season_adjust)
  )) %>%
  forecast() %>% autoplot(us_retail_employment)
```



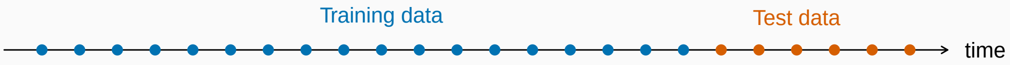
Decomposition models

`decomposition_model()` creates a decomposition model

- You must provide a method for forecasting the `season_adjust` series.
- A seasonal naive method is used by default for the `seasonal` components.
- The variances from both the seasonally adjusted and seasonal forecasts are combined.

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

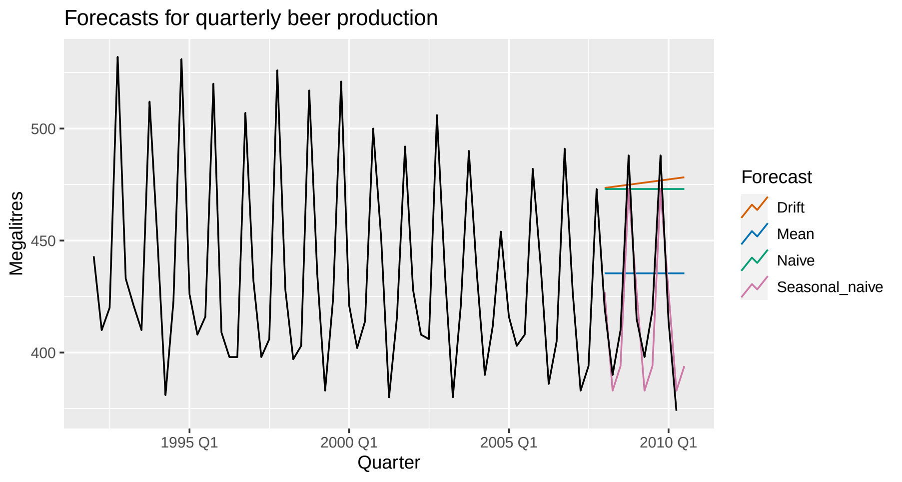
Forecast "error": the difference between an observed value and its forecast.

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

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

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

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

Measures of forecast accuracy

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

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

- 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 $\{y_t\}$.

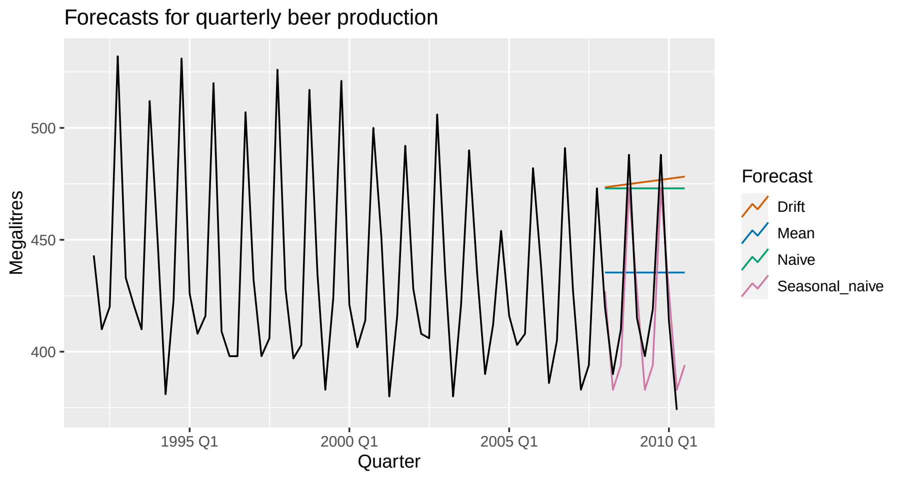
Proposed by Hyndman and Koehler (IJF, 2006).

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



Measures of forecast accuracy

```r
recent_production <- aus_production %>%
  filter(year(Quarter) >= 1992)
train <- recent_production %>%
  filter(year(Quarter) <= 2007)
beer_fit <- train %>%
  model(
    Mean = MEAN(Beer),
    Naive = NAIVE(Beer),
    Seasonal_naive = SNAIVE(Beer),
    Drift = RW(Beer ~ drift())
  )
beer_fc <- beer_fit %>%
  forecast(h = 10)
```

Measures of forecast accuracy

```r
accuracy(beer_fit)
```

```
## # A tibble: 4 x 6
##   .model         .type     RMSE   MAE  MAPE  MASE
##   <chr>          <chr>    <dbl> <dbl> <dbl> <dbl>
## 1 Drift          Training  65.3  54.8 12.2   3.83
## 2 Mean           Training  43.6  35.2  7.89  2.46
## 3 Naive          Training  65.3  54.7 12.2   3.83
## 4 Seasonal_naive Training  16.8  14.3  3.31  1
```

```r
accuracy(beer_fc, recent_production)
```

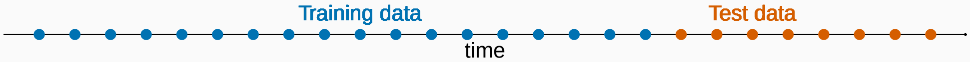
```
## # A tibble: 4 x 6
##   .model         .type  RMSE   MAE  MAPE  MASE
##   <chr>          <chr> <dbl> <dbl> <dbl> <dbl>
## 1 Drift          Test   64.9  58.9 14.6  4.12
## 2 Mean           Test   38.4  34.8  8.28 2.44
## 3 Naive          Test   62.7  57.4 14.2  4.01
## 4 Seasonal_naive Test   14.3  13.4  3.17 0.937
```

Poll: true or false?

https://PollEv.com/georgeathana023

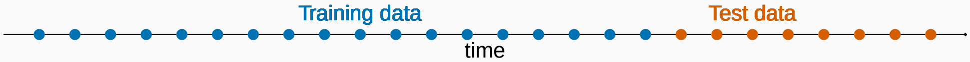
Time series cross-validation

Traditional evaluation

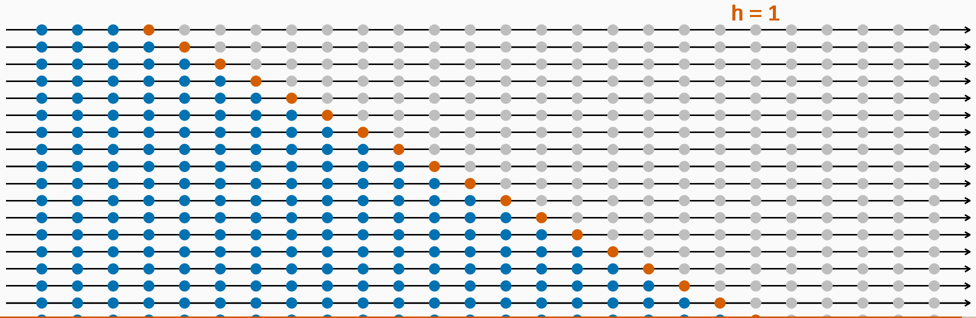


Time series cross-validation

Traditional evaluation

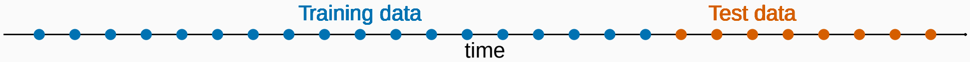


Time series cross-validation

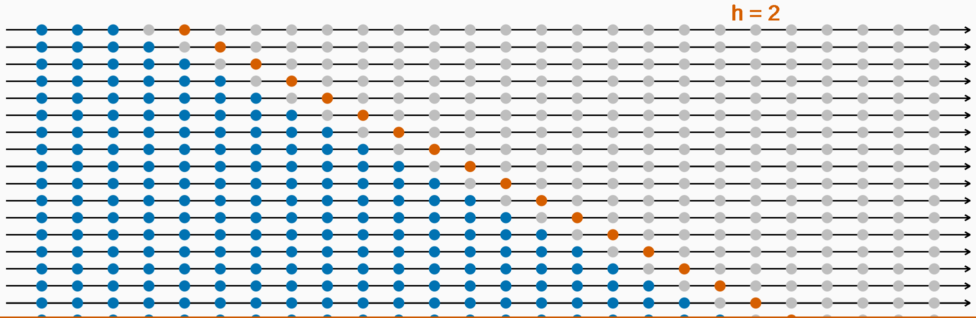


Time series cross-validation

Traditional evaluation

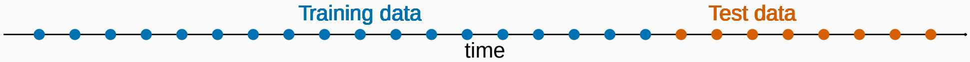


Time series cross-validation

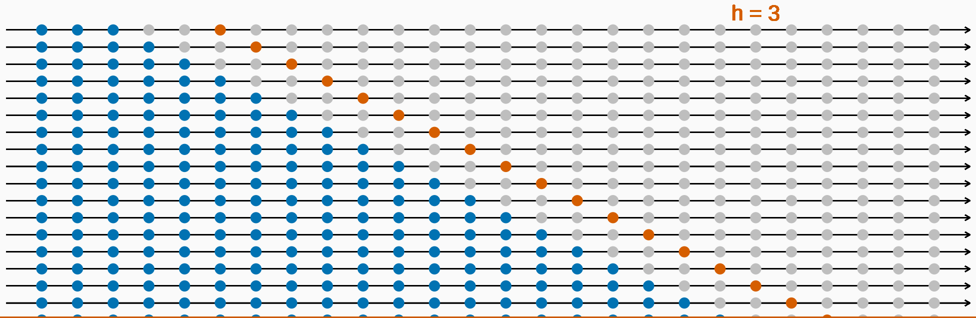


Time series cross-validation

Traditional evaluation

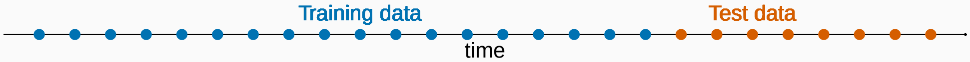


Time series cross-validation

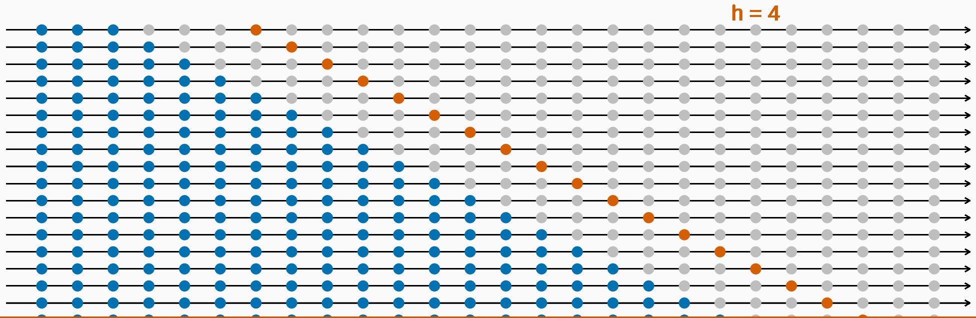


Time series cross-validation

Traditional evaluation

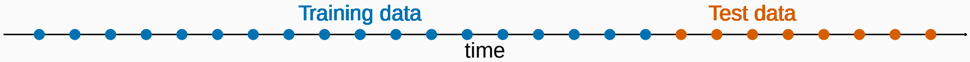


Time series cross-validation

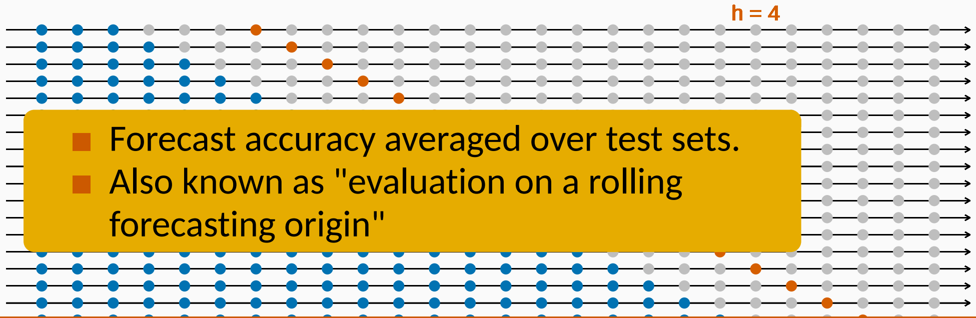


Time series cross-validation

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.

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

```
## # A tsibble: 790,650 x 4 [1]
## # Key:       .id [1,255]
##   Date       Close trading_day   .id
##   <date>     <dbl>       <int> <int>
## 1 2014-01-02  54.7           1     1
## 2 2014-01-03  54.6           2     1
## 3 2014-01-06  57.2           3     1
## 4 2014-01-02  54.7           1     2
## 5 2014-01-03  54.6           2     2
## 6 2014-01-06  57.2           3     2
## 7 2014-01-07  57.9           4     2
## # ... with 790,643 more rows
```

Time series cross-validation

Estimate RW w/ drift models for each window.

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

```
## # A mable: 1,255 x 3
## # Key:     .id, Symbol [1,255]
##      .id Symbol `RW(Close ~ drift())`
##    <int> <chr>                <model>
## 1      1 FB             <RW w/ drift>
## 2      2 FB             <RW w/ drift>
## 3      3 FB             <RW w/ drift>
## 4      4 FB             <RW w/ drift>
## # ... with 1,251 more rows
```

Time series cross-validation

Produce one step ahead forecasts from all models.

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

```
## # A fable: 1,255 x 5 [1]
## # Key:     .id, Symbol [1,255]
##      .id Symbol trading_day      Close .mean
##    <int> <chr>        <dbl>     <dist> <dbl>
## 1      1 FB               4 N(58, 3.9)  58.4
## 2      2 FB               5   N(59, 2)  59.0
## 3      3 FB               6 N(59, 1.5)  59.1
## 4      4 FB               7 N(58, 1.8)  57.7
## # ... with 1,251 more rows
```

Time series cross-validation

```r
# Cross-validated
fc_cv %>% accuracy(fb_stock)
# Training set
fb_stock %>% model(RW(Close ~ drift())) %>% accuracy()
```

	RMSE	MAE	MAPE
Cross-validation	2.418	1.469	1.266
Training	2.414	1.465	1.261

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