Cardiff University
Prifysgol Caerdydd
Cardiff Business School
Ysgol Busnes Caerdydd

Forecasting in R

The forecasters' toolbox

Bahman Rostami-Tabar

Outline

1. Learning outcome
2. A tidy forecasting workflow
3. Data preperation(tidy) and visualisation
4. Define the model (specify)
5. Train the model (estimate)
6. Fitted values and Residuals
7. lab session 5
8. Prediction intervals

Outline

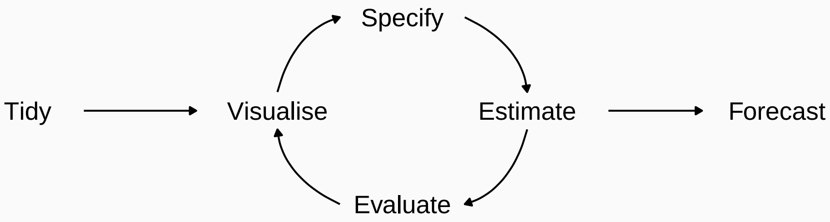
Learning outcome

You should be able to:

1. Discuss general tools that are useful for many different forecasting situations
2. Explain simple forecasting methods (benchmarks)
3. Specify and estimate models using R functions in `fable`
4. Recognise and extract fitted values and residuals
5. Produce point and prediction interval forecasts

Outline

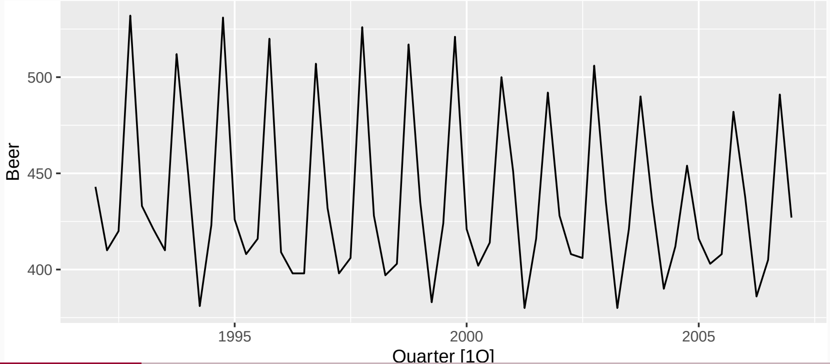
A tidy forecasting workflow



Outline

Data preparation and visualisation

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

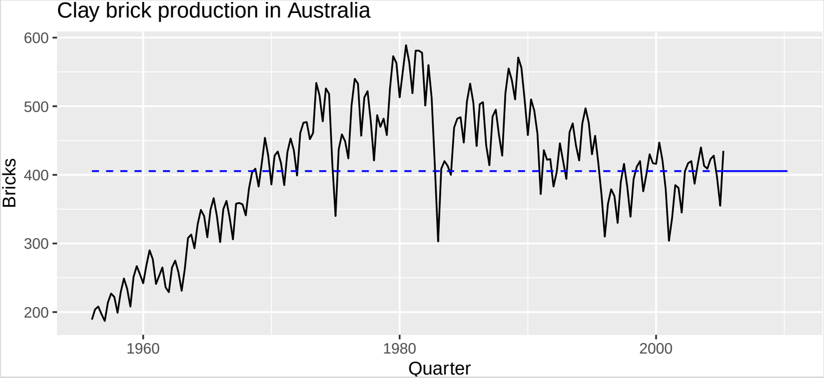


Outline

Some simple forecasting methods

`MEAN(y)`: Average method

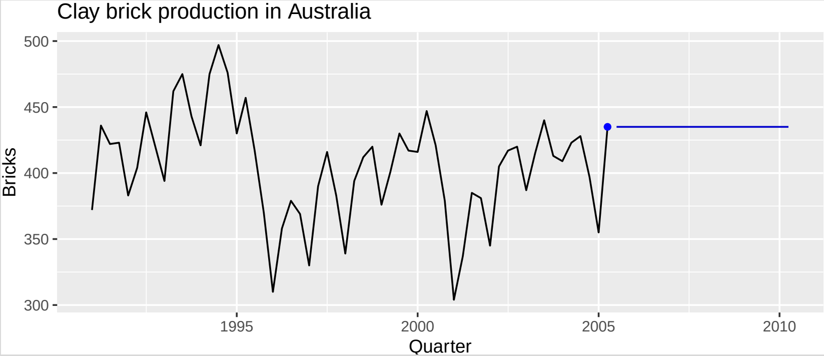
- Forecast of all future values is equal to mean of historical data $\{y_1, \dots, y_T\}$.
- Forecasts: $\hat{y}_{T+h|T} = \bar{y} = (y_1 + \cdots + y_T)/T$



Some simple forecasting methods

`NAIVE(y)`: Naïve method

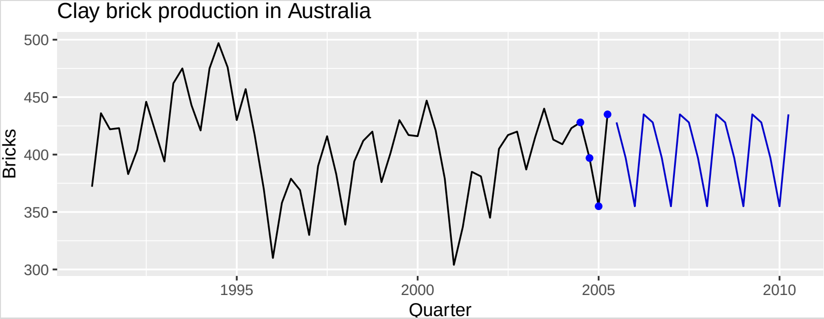
- Forecasts equal to last observed value.
- Forecasts: $\hat{y}_{T+h|T} = y_T$.
- Consequence of efficient market hypothesis.



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

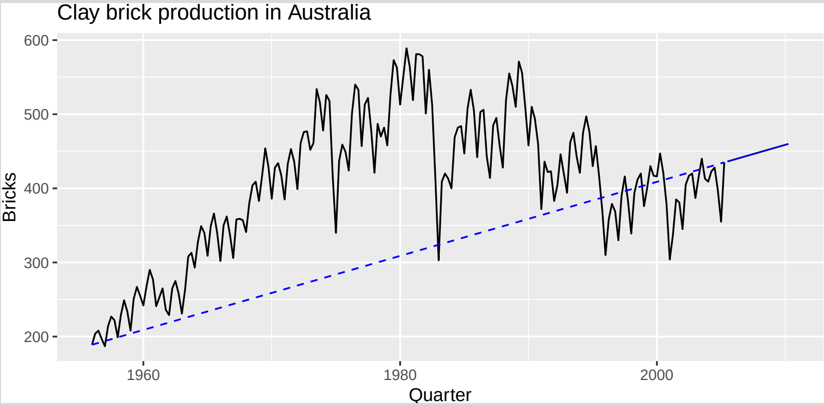
- Forecasts equal to last value plus average change.
- Forecasts:

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

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

Some simple forecasting methods

Drift method



Model specification

- Model specification in fable supports a formula based interface
- A model formula in R is expressed using `response ~ terms`
  - the formula's left side describes the response
  - the right describes terms used to model the response.
- `Attention`: MODEL_NAME is in capital letters, e.g. `SNAIVE`

```
MODEL_NAME(response_variable ~ term1+term2+...)
SNAIVE(Beer ~ lag("year"))
```

Outline

Model estimation: template

The `model()` function trains models to data. - It returns a model table or a `mable` object.

```
# Fit the models
my_mable <- my_data %>%
  model(
    choose_name1 = MODEL_1(response_variable ~ term1+...),
    choose_name2 = MODEL_2(response_variable ~ term1+...),
    choose_name3 = MODEL_3(response_variable ~ term1+...),
    choose_name4 = MODEL_4(response_variable ~ term1+...)
  )
```

Model estimation

```r
# Fit the models
beer_fit <- train %>%
  model(
    mean = MEAN(Beer),
    naive = NAIVE(Beer),
    snaive = SNAIVE(Beer, lag="year"),
    drift = RW(Beer ~ drift())
  )

#beer_fit <- beer_fit %>% stream(new_data),
#we can update the fitted models once we have new data
```

mable: a model object

```
beer_fit
```

```
## # A mable: 1 x 4
##   mean    naive   snaive   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.
- A mable contains
  - a row for each time series
  - a column for each model specification

Extract coefficients from `mable`

```
beer_fit %>% select(snaive) %>% report()
beer_fit %>% tidy()
beer_fit %>% glance()
```

- The `report()` function gives a formatted model-specific display.
- The `tidy()` function is used to extract the coefficients from the models.
- The `glance()` shows a summary from the models.
- We can extract information about some specific model using the `filter()` and `select()`functions.

Producing forecasts

- The `forecast()` function is used to produce forecasts from estimated models.
- **h** can be specified with a number (the number of future observations) or natural language (the length of time to predict).

```
beer_fc <- beer_fit %>%
  forecast(h = "3 years")
#h = "3 years" is equivalent to setting h = 12.
```

Producing forecasts

```
## # A fable: 48 x 4 [1Q]
## # Key:      .model [4]
##   .model Quarter  Beer .distribution
##   <chr>     <qtr> <dbl> <dist>
## 1 mean   2007 Q2  436. N(436, 1963)
## 2 mean   2007 Q3  436. N(436, 1963)
## 3 mean   2007 Q4  436. N(436, 1963)
## 4 mean   2008 Q1  436. N(436, 1963)
## # ... with 44 more rows
```

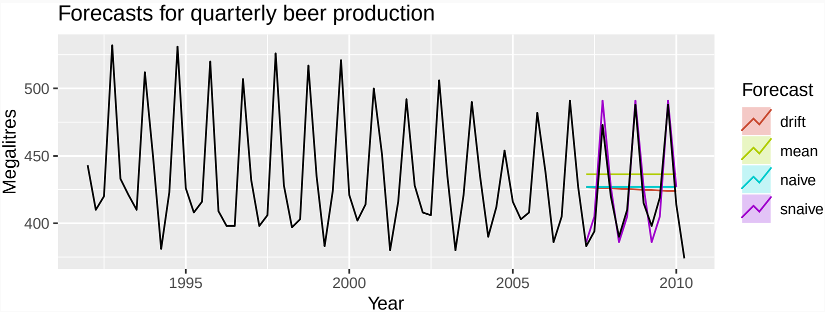
A `fable` is a forecast table with point forecasts and distributions.

Check model performance

Once a model has been fitted, it is importand to check how well it has performed on the data. I come back to this latter.

Visualising forecasts

```
# Plot forecasts against actual values
beer_fc %>%
  autoplot(train, level = NULL) +
    autolayer(filter_index(aus_production, "2007 Q1" ~ .), color = "black") +
    ggtitle("Forecasts for quarterly beer production") +
    xlab("Year") + ylab("Megalitres") +
    guides(colour=guide_legend(title="Forecast"))
```



Outline

Fitted values

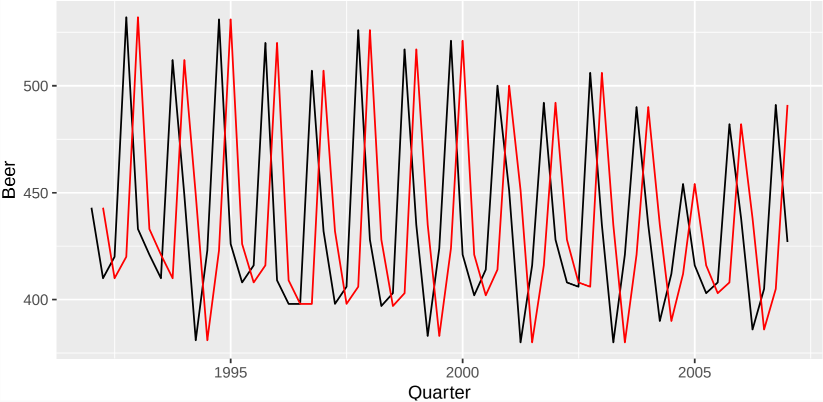
- $\hat{y}_{T|T-1}$ is the forecast of y_T based on observations $y_1, \dots, y_T - 1$.
- We call these “fitted values”.
- Sometimes drop the subscript: $\hat{y}_T \equiv \hat{y}_{T|T-1}$.
- Often not true forecasts since parameters are estimated on all data.

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.
- $\hat{y}_T = y_{T-1}$ for naive method.

Fitted values

```r
beer_fit %>% select(naive) %>% augment() %>%
  ggplot(aes(x=Quarter, y=Beer))+
  geom_line()+
  geom_line(aes(y=.fitted), colour="red")
```



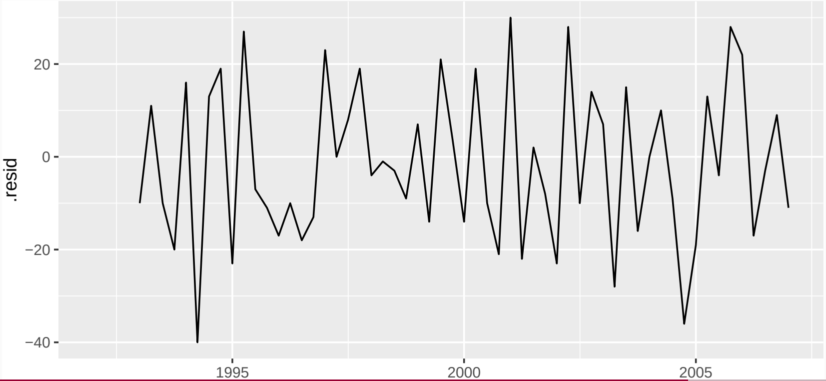
Residuals

- The "residuals" in a time series model are what is left over after fitting a model.
- Residuals are useful in checking whether a model has adequately captured the information in the data.

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

Residuals

```
#beer_fit %>% fitted
#augment() fucntion gets residuals and fitted values
beer_fit %>% select(snaive) %>% augment() %>%
  ggplot(aes(x=Quarter, y=.resid))+
  geom_line()
```



Extract fitted values and residuals

```r
beer_fit %>% augment()
```

```
## # A tsibble: 244 x 5 [1Q]
## # Key:       .model [4]
##    .model Quarter  Beer .fitted .resid
##    <chr>    <qtr> <dbl>   <dbl>  <dbl>
##  1 mean   1992 Q1   443    436.   6.70
##  2 mean   1992 Q2   410    436. -26.3
##  3 mean   1992 Q3   420    436. -16.3
##  4 mean   1992 Q4   532    436.  95.7
##  5 mean   1993 Q1   433    436.  -3.30
##  6 mean   1993 Q2   421    436. -15.3
##  7 mean   1993 Q3   410    436. -26.3
##  8 mean   1993 Q4   512    436.  75.7
##  9 mean   1994 Q1   449    436.  12.7
## 10 mean   1994 Q2   381    436. -55.3
```

Outline

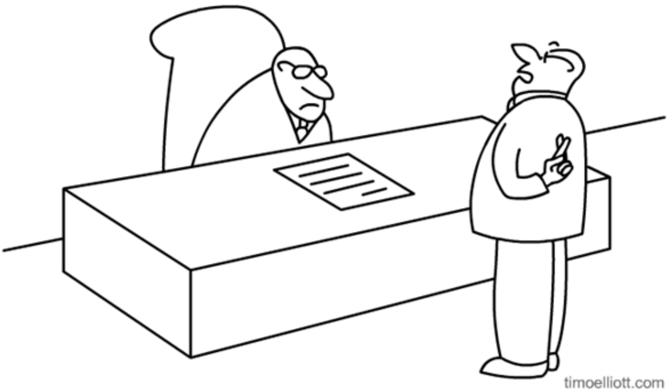
lab session 5

- Produce forecasts from the bechmark methods for daily A&E series for 42 days
- Plot the results using `autoplot()`.
  - Use `filter_index()` to show the plot from 2016
- Use `augment()` to extract fitted values for snaive method
- Extract residuals for mean method

Outline

Importance of providing interval forecast

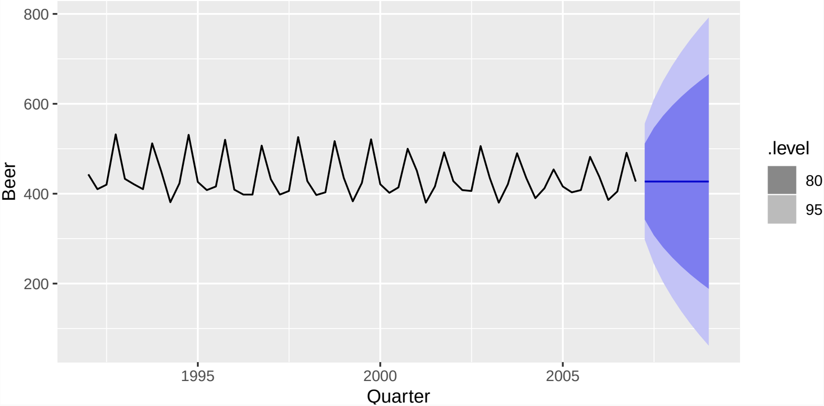
Point forecasts are often useless without a measure of uncertainty



"Yes sir, you can absolutely trust those numbers"

Prediction intervals

- A prediction interval gives a region within which we expect y_{T+h} to lie with a specified probability
- It consists of an upper and a lower limit between which the future value is expected to lie



Prediction intervals

- Assuming forecast errors are normally distributed, then a c% PI is:

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

where the multiplier c depends on the coverage probability and $\hat{\sigma}_h$ is the st dev of the h-step distribution.

Prediction intervals

- Forecast intervals can be extracted using the `hilo()` function
- Use `level` argument to control coverage.

```
fit <- train %>% model(NAIVE(Beer))
forecast(fit) %>% hilo(level = c(80, 95))
```

```
## # A tsibble: 8 x 5 [1Q]
## # Key:       .model [1]
##   .model      Quarter  Beer              80%                95%
##   <chr>         <qtr> <dbl>           <hilo>             <hilo>
## 1 NAIVE(Be~   2007 Q2   427 [342.5627, 511~ [297.86430, 556~
## 2 NAIVE(Be~   2007 Q3   427 [307.5876, 546~ [244.37454, 609~
## 3 NAIVE(Be~   2007 Q4   427 [280.7503, 573~ [203.33041, 650~
```