The **fable** package for doing tidy forecasting in R is now on CRAN. Like [**tsibble**](https://tsibble.tidyverts.org/) and [**feasts**](https://feasts.tidyverts.org/), it is also part of the [tidyverts](https://mail.google.com/tidyverts.org) family of packages for analysing, modelling and forecasting many related time series (stored as tsibbles).

For a brief introduction to tsibbles, see [this post](https://robjhyndman.com/hyndsight/tsibbles/) from last month.

Here we will forecast Australian tourism data by state/region and purpose. This data is stored in the tourism tsibble where Trips contains domestic visitor nights in thousands.

library(tidyverse)

library(tsibble)

library(lubridate)

library(fable)

tourism

## # A tsibble: 24,320 x 5 [1Q]

## # Key: Region, State, Purpose [304]

## Quarter Region State Purpose Trips

##

## 1 1998 Q1 Adelaide South Australia Business 135.

## 2 1998 Q2 Adelaide South Australia Business 110.

## 3 1998 Q3 Adelaide South Australia Business 166.

## 4 1998 Q4 Adelaide South Australia Business 127.

## 5 1999 Q1 Adelaide South Australia Business 137.

## 6 1999 Q2 Adelaide South Australia Business 200.

## 7 1999 Q3 Adelaide South Australia Business 169.

## 8 1999 Q4 Adelaide South Australia Business 134.

## 9 2000 Q1 Adelaide South Australia Business 154.

## 10 2000 Q2 Adelaide South Australia Business 169.

## # … with 24,310 more rows

There are 304 combinations of Region, State and Purpose, each one defining a time series of 80 observations.

To simplify the outputs, we will abbreviate the state names.

tourism <- tourism %>%

mutate(

State = recode(State,

"Australian Capital Territory" = "ACT",

"New South Wales" = "NSW",

"Northern Territory" = "NT",

"Queensland" = "QLD",

"South Australia" = "SA",

"Tasmania" = "TAS",

"Victoria" = "VIC",

"Western Australia" = "WA"

)

)

**Forecasting a single time series**

Although the **fable** package is designed to handle many time series, we will be begin by demonstrating its use on a single time series. For this purpose, we will extract the tourism data for holidays in the Snowy Mountains region of NSW.

snowy <- tourism %>%

filter(

Region == "Snowy Mountains",

Purpose == "Holiday"

)

snowy

## # A tsibble: 80 x 5 [1Q]

## # Key: Region, State, Purpose [1]

## Quarter Region State Purpose Trips

##

## 1 1998 Q1 Snowy Mountains NSW Holiday 101.

## 2 1998 Q2 Snowy Mountains NSW Holiday 112.

## 3 1998 Q3 Snowy Mountains NSW Holiday 310.

## 4 1998 Q4 Snowy Mountains NSW Holiday 89.8

## 5 1999 Q1 Snowy Mountains NSW Holiday 112.

## 6 1999 Q2 Snowy Mountains NSW Holiday 103.

## 7 1999 Q3 Snowy Mountains NSW Holiday 254.

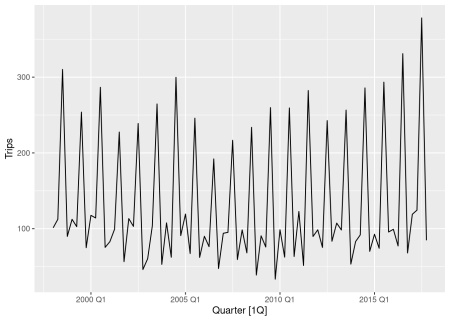
## 8 1999 Q4 Snowy Mountains NSW Holiday 74.9

## 9 2000 Q1 Snowy Mountains NSW Holiday 118.

## 10 2000 Q2 Snowy Mountains NSW Holiday 114.

## # … with 70 more rows

snowy %>% autoplot(Trips)



For this data set, a reasonable benchmark forecast method is the seasonal naive method, where forecasts are set to be equal to the last observed value from the same quarter. Alternative models for this series are ETS and ARIMA models. All these can be included in a single call to the model() function like this.

fit <- snowy %>%

model(

snaive = SNAIVE(Trips ~ lag("year")),

ets = ETS(Trips),

arima = ARIMA(Trips)

)

fit

## # A mable: 1 x 6

## # Key: Region, State, Purpose [1]

## Region State Purpose snaive ets arima

##

## 1 Snowy Mountains NSW Holiday

The returned object is called a “mable” or model table, where each cell corresponds to a fitted model. Because we have only fitted models to one time series, this mable has only one row.

To forecast all models, we pass the object to the forecast function.

fc <- fit %>%

forecast(h = 12)

fc

## # A fable: 36 x 7 [1Q]

## # Key: Region, State, Purpose, .model [3]

## Region State Purpose .model Quarter Trips .distribution

##

## 1 Snowy Mountains NSW Holiday snaive 2018 Q1 119. N(119, 666)

## 2 Snowy Mountains NSW Holiday snaive 2018 Q2 124. N(124, 666)

## 3 Snowy Mountains NSW Holiday snaive 2018 Q3 378. N(378, 666)

## 4 Snowy Mountains NSW Holiday snaive 2018 Q4 84.7 N( 85, 666)

## 5 Snowy Mountains NSW Holiday snaive 2019 Q1 119. N(119, 1331)

## 6 Snowy Mountains NSW Holiday snaive 2019 Q2 124. N(124, 1331)

## 7 Snowy Mountains NSW Holiday snaive 2019 Q3 378. N(378, 1331)

## 8 Snowy Mountains NSW Holiday snaive 2019 Q4 84.7 N( 85, 1331)

## 9 Snowy Mountains NSW Holiday snaive 2020 Q1 119. N(119, 1997)

## 10 Snowy Mountains NSW Holiday snaive 2020 Q2 124. N(124, 1997)

## # … with 26 more rows

The return object is a “fable” or forecast table with the following characteristics:

* the .model column becomes an additional key;
* the .distribution column contains the estimated probability distribution of the response variable in future time periods;
* the Trips column contains the point forecasts equal to the mean of the probability distribution.

The autoplot() function will produce a plot of all forecasts. By default, level=c(80,95) so 80% and 95% prediction intervals are shown. But to avoid clutter, we will set level=NULL to show no prediction intervals.

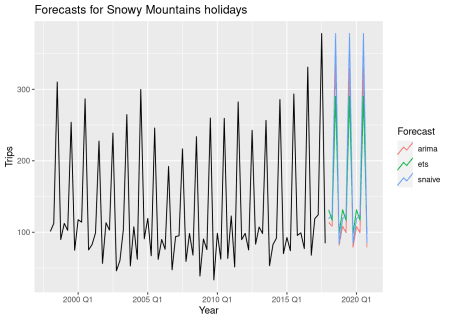
fc %>%

autoplot(snowy, level = NULL) +

ggtitle("Forecasts for Snowy Mountains holidays") +

xlab("Year") +

guides(colour = guide\_legend(title = "Forecast"))



If you want to compute the prediction intervals, the hilo() function can be used:

hilo(fc, level = 95)

## # A tsibble: 36 x 7 [1Q]

## # Key: Region, State, Purpose, .model [3]

## Region State Purpose .model Quarter Trips `95%`

##

## 1 Snowy Mountains NSW Holiday snaive 2018 Q1 119. [ 68.5, 170]95

## 2 Snowy Mountains NSW Holiday snaive 2018 Q2 124. [ 73.9, 175]95

## 3 Snowy Mountains NSW Holiday snaive 2018 Q3 378. [327.6, 429]95

## 4 Snowy Mountains NSW Holiday snaive 2018 Q4 84.7 [ 34.1, 135]95

## 5 Snowy Mountains NSW Holiday snaive 2019 Q1 119. [ 47.5, 191]95

## 6 Snowy Mountains NSW Holiday snaive 2019 Q2 124. [ 53.0, 196]95

## 7 Snowy Mountains NSW Holiday snaive 2019 Q3 378. [306.6, 450]95

## 8 Snowy Mountains NSW Holiday snaive 2019 Q4 84.7 [ 13.1, 156]95

## 9 Snowy Mountains NSW Holiday snaive 2020 Q1 119. [ 31.4, 207]95

## 10 Snowy Mountains NSW Holiday snaive 2020 Q2 124. [ 36.9, 212]95

## # … with 26 more rows

**Forecasting many series**

To scale this up to include all series in the tourism data set requires no more work — we use exactly the same code.

fit <- tourism %>%

model(

snaive = SNAIVE(Trips ~ lag("year")),

ets = ETS(Trips),

arima = ARIMA(Trips)

)

fit

## # A mable: 304 x 6

## # Key: Region, State, Purpose [304]

## Region State Purpose snaive ets arima

##

## 1 Adelaide SA Business

## 4 Adelaide SA Visiting

## 6 Adelaide Hi… SA Holiday

## 7 Adelaide Hi… SA Other

## 8 Adelaide Hi… SA Visiting

## 9 Alice Sprin… NT Business

## 10 Alice Sprin… NT Holiday

## # … with 294 more rows

Now the mable includes models for every combination of keys in the tourism data set.

We can extract information about some specific model using the filter, select and report functions.

fit %>%

filter(Region == "Snowy Mountains", Purpose == "Holiday") %>%

select(arima) %>%

report()

## Series: Trips

## Model: ARIMA(1,0,0)(0,1,2)[4]

##

## Coefficients:

## ar1 sma1 sma2

## 0.216 -0.371 -0.190

## s.e. 0.116 0.128 0.116

##

## sigma^2 estimated as 592.9: log likelihood=-350

## AIC=707 AICc=708 BIC=716

When the mable is passed to the forecast() function, forecasts are computed for every model and every key combination.

fc <- fit %>%

forecast(h = "3 years")

fc

## # A fable: 10,944 x 7 [1Q]

## # Key: Region, State, Purpose, .model [912]

## Region State Purpose .model Quarter Trips .distribution

##

## 1 Adelaide SA Business snaive 2018 Q1 129. N(129, 2018)

## 2 Adelaide SA Business snaive 2018 Q2 174. N(174, 2018)

## 3 Adelaide SA Business snaive 2018 Q3 185. N(185, 2018)

## 4 Adelaide SA Business snaive 2018 Q4 197. N(197, 2018)

## 5 Adelaide SA Business snaive 2019 Q1 129. N(129, 4036)

## 6 Adelaide SA Business snaive 2019 Q2 174. N(174, 4036)

## 7 Adelaide SA Business snaive 2019 Q3 185. N(185, 4036)

## 8 Adelaide SA Business snaive 2019 Q4 197. N(197, 4036)

## 9 Adelaide SA Business snaive 2020 Q1 129. N(129, 6054)

## 10 Adelaide SA Business snaive 2020 Q2 174. N(174, 6054)

## # … with 10,934 more rows

Note the use of natural language to specify the forecast horizon. The forecast() function is able to interpret many different time specifications. For quarterly data, h = "3 years" is equivalent to setting h = 12.

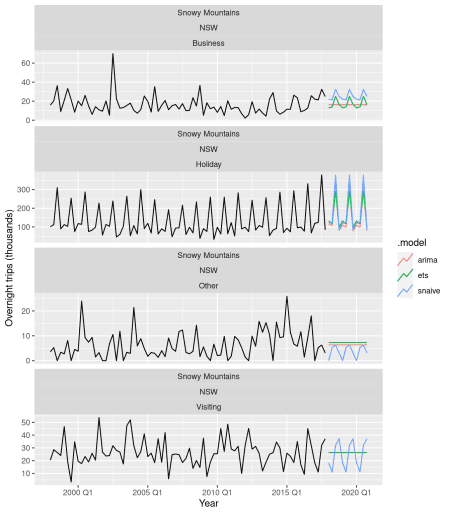
Plots of individual forecasts can also be produced, although filtering is helpful to avoid plotting too many series at once.

fc %>%

filter(Region == "Snowy Mountains") %>%

autoplot(tourism, level = NULL) +

xlab("Year") + ylab("Overnight trips (thousands)")



**Forecast accuracy calculations**

To compare the forecast accuracy of these models, we will create a training data set containing all data up to 2014. We will then forecast the remaining years in the data set and compare the results with the actual values.

train <- tourism %>%

filter(year(Quarter) <= 2014)

fit <- train %>%

model(

ets = ETS(Trips),

arima = ARIMA(Trips),

snaive = SNAIVE(Trips)

) %>%

mutate(mixed = (ets + arima + snaive) / 3)

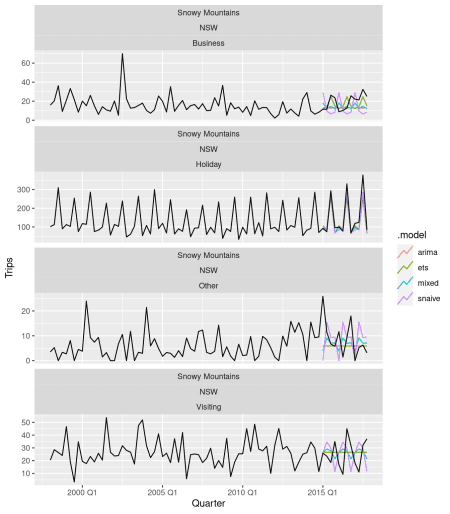
Here we have introduced an ensemble forecast (mixed) which is a simple average of the three fitted models. Note that forecast() will produce distributional forecasts from the ensemble as well, taking into account the correlations between the forecast errors of the component models.

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

fc %>%

filter(Region == "Snowy Mountains") %>%

autoplot(tourism, level = NULL)



Now to check the accuracy, we use the accuracy() function. By default it computes several point forecasting accuracy measures such as MAE, RMSE, MAPE and MASE for every key combination.

accuracy(fc, tourism)

## # A tibble: 1,216 x 12

## .model Region State Purpose .type ME RMSE MAE MPE MAPE MASE

##

## 1 arima Adela… SA Busine… Test 22.5 28.5 25.3 11.9 14.0 0.765

## 2 arima Adela… SA Holiday Test 21.9 34.8 28.0 9.93 14.8 1.31

## 3 arima Adela… SA Other Test 4.71 17.5 14.6 0.529 20.2 1.11

## 4 arima Adela… SA Visiti… Test 32.8 37.1 32.8 13.7 13.7 1.05

## 5 arima Adela… SA Busine… Test 1.31 5.58 3.57 -Inf Inf 1.09

## 6 arima Adela… SA Holiday Test 6.46 7.43 6.46 37.4 37.4 1.14

## 7 arima Adela… SA Other Test 1.35 2.79 1.93 -31.0 99.4 1.71

## 8 arima Adela… SA Visiti… Test 8.37 12.6 10.4 -3.98 72.3 1.35

## 9 arima Alice… NT Busine… Test 9.85 12.2 10.7 34.4 44.3 1.74

## 10 arima Alice… NT Holiday Test 4.80 11.3 9.30 4.46 35.2 1.00

## # … with 1,206 more rows, and 1 more variable: ACF1

But because we have generated distributional forecasts, it is also interesting to look at the accuracy using CRPS (Continuous Rank Probability Scores) and Winkler Scores (for 95% prediction intervals).

fc\_accuracy <- accuracy(fc, tourism,

measures = list(

point\_accuracy\_measures,

interval\_accuracy\_measures,

distribution\_accuracy\_measures

)

)

fc\_accuracy %>%

group\_by(.model) %>%

summarise(

RMSE = mean(RMSE),

MAE = mean(MAE),

MASE = mean(MASE),

Winkler = mean(winkler),

CRPS = mean(CRPS)

) %>%

arrange(RMSE)

## # A tibble: 4 x 6

## .model RMSE MAE MASE Winkler CRPS

##

## 1 mixed 19.8 16.0 0.997 104. 11.4

## 2 ets 20.2 16.4 1.00 128. 11.9

## 3 snaive 21.5 17.3 1.17 121. 12.8

## 4 arima 21.9 17.8 1.07 140. 13.0

In this case, the mixed model is doing best on all accuracy measures.

**Moving from forecast to fable**

Many readers will be familiar with the **forecast** package and will wonder about the differences between **forecast** and **fable**. Here are some of the main differences.

* **fable** is designed for tsibble objects, **forecast** is designed for ts objects.
* **fable** handles many time series at a time, **forecast** handles one time series at a time.
* **fable** can fit multiple models at once, **forecast** fits one model at a time.
* **forecast** produces point forecasts and prediction intervals. **fable** produces point forecasts and distribution forecasts. In **fable**, you can get prediction intervals from the forecast object using hilo() and in plots using autoplot().
* **fable** handles ensemble forecasting easily whereas **forecast** has no facilities for ensembles.
* **fable** has a more consistent interface with every model specified as a formula.
* Automated modelling in **fable** is obtained by simply not specifying the right hand side of the formula. This was shown in the ARIMA() and ETS() functions here.

Subsequent posts will explore other features of the **fable** package.